YAYASAN PRIMA AGUS TEKNIK

TEKNOLOGI KRIPTOGRAFI MODERN

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

TEKNOLOGI KRIPTOGRAFI MODERN

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8642-17-5 (PDF)

TEKNOLOGI KRIPTOGRAFI MODERN

Penulis :
Dr. Joseph Teguh Santoso, S.Kom., M.Kom

ISBN : 978-623-8642-17-5

Editor :
Muhammad Sholikan, M.Kom

Penyunting :
Dr. Mars Caroline Wibowo. S.T., M.Mm.Tech

Desain Sampul dan Tata Letak :
Irdha Yunianto, S.Ds., M.Kom

Penebit :
Yayasan Prima Agus Teknik Bekerja sama dengan
Universitas Sains & Teknologi Komputer (Universitas STEKOM)

Anggota IKAPI No: 279 / ALB / JTE / 2023

Redaksi :
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

Distributor Tunggal :
Universitas STEKOM
Jl. Majapahit no 605 Semarang
Telp. (024) 6723456
Fax. 024-6710144
Email : info@stekom.ac.id

KATA PENGANTAR

Puji Syukur kehadirat Tuhan Yang Maha Esa, karena berkat dan anugrah-Nya, penulis dapat menyelesaikan buku yang berjudul ***"Teknologi Kriptografi Modern",*** dengan baik dan maksimal. Kriptografi adalah ilmu yang mempelajari tentang cara mengamankan data atau pesan dengan menggunakan algoritma dan teknik khusus agar hanya dapat dibaca atau diinterpretasikan oleh penerima yang diinginkan, sementara orang lain tidak dapat membaca atau mengubah pesan tersebut. Kriptografi berfokus pada penggunaan metode matematika dan ilmu komputer untuk mengamankan komunikasi dan data dalam bentuk yang dapat dienkripsi dan didekripsi.

Kriptografi memberikan manfaat yang signifikan, termasuk keamanan data, integritas data, authenticity, confidentiality, dan non-repudiation. Keamanan data memastikan bahwa data yang dikirimkan melalui jaringan atau disimpan dalam sistem komputer tidak dapat dibaca oleh pihak yang tidak berhak. Integritas data memastikan bahwa data yang diterima adalah data yang asli dan tidak terubah oleh pihak lain. Authenticity memastikan bahwa penerima data adalah penerima yang diinginkan dan tidak ada pihak lain yang dapat meniru identitasnya. Confidentiality memastikan bahwa data yang dikirimkan hanya dapat dibaca oleh penerima yang diinginkan, sehingga mencegah akses oleh pihak yang tidak berhak. Non-repudiation memastikan bahwa penerima data tidak dapat menyangkal bahwa mereka telah menerima data tersebut.Mempelajari kriptografi dapat membuka peluang karir di bidang keamanan informasi, teknologi informasi, dan pengembangan perangkat lunak. Kriptografi juga digunakan dalam berbagai bidang seperti e-commerce, jaringan komputer, komunikasi, dan keamanan nasional. Selain itu, mempelajari kriptografi dapat memungkinkan pengembangan inovasi baru dalam bidang keamanan informasi dan teknologi.

Tujuan buku ini adalah membuat prinsip-prinsip dasar kriptografi modern dapat diakses oleh mahasiswa ilmu komputer, teknik elektro, atau matematika; bagi para profesional yang ingin memasukkan kriptografi ke dalam sistem atau perangkat lunak yang mereka kembangkan; dan bagi siapa pun yang memiliki tingkat kematangan matematika dasar yang tertarik untuk memahami bidang yang menakjubkan ini.

Demikian buku ini dibuat supaya dapat menjadi sumber ilmu yang berguna bagi mahasiswa maupun Masyarakat umum. Terima Kasih.

Semarang, Juli 2024
Penulis

Dr. Joseph Teguh Santoso, S.Kom., M.Kom.

DAFTAR ISI

BAB 1
PERKENALAN

1.1 KRIPTOGRAFI DAN KRIPTOGRAFI MODERN

Kamus Bahasa Inggris Oxford yang Ringkas mendefinisikan kriptografi sebagai "*seni menulis atau memecahkan kode*." Hal ini akurat secara historis, namun tidak mencakup luasnya bidang tersebut atau landasan ilmiahnya saat ini. Definisi ini hanya berfokus pada kode-kode yang telah digunakan selama berabad-abad untuk memungkinkan komunikasi rahasia. Namun kriptografi saat ini mencakup lebih dari itu: ia berkaitan dengan mekanisme untuk memastikan integritas, teknik pertukaran kunci rahasia, protokol untuk otentikasi pengguna, lelang dan pemilihan elektronik, uang digital, dan banyak lagi. Tanpa berusaha memberikan karakterisasi yang lengkap, kita dapat mengatakan bahwa kriptografi modern melibatkan studi teknik matematika untuk mengamankan informasi digital, sistem, dan komputasi terdistribusi terhadap serangan musuh.

Definisi kamus juga menyebut kriptografi sebagai seni. Hingga akhir abad ke-20, kriptografi sebagian besar merupakan sebuah seni. Membangun kode yang baik, atau memecahkan kode yang sudah ada, bergantung pada kreativitas dan pemahaman yang berkembang tentang cara kerja kode. Hanya ada sedikit teori yang dapat diandalkan dan, untuk waktu yang lama, tidak ada definisi yang berfungsi mengenai apa yang dimaksud dengan kode yang baik. Dimulai pada tahun 1970an dan 1980an, gambaran kriptografi berubah secara radikal. Sebuah teori yang kaya mulai muncul, memungkinkan studi mendalam tentang kriptografi sebagai ilmu dan disiplin matematika. Perspektif ini, pada gilirannya, mempengaruhi cara berpikir para peneliti tentang bidang keamanan komputer yang lebih luas.

Perbedaan lain yang sangat penting antara kriptografi klasik (katakanlah, sebelum tahun 1980an) dan kriptografi modern berkaitan dengan penerapannya. Secara historis, konsumen utama kriptografi adalah organisasi militer dan pemerintah. Saat ini, kriptografi ada dimana-mana! Jika Anda pernah mengautentikasi diri Anda dengan mengetikkan kata sandi, membeli sesuatu dengan kartu kredit melalui Internet, atau mengunduh pembaruan terverifikasi untuk sistem operasi Anda, Anda pasti pernah menggunakan kriptografi. Dan semakin banyak pemrogram dengan pengalaman yang relatif sedikit diminta untuk "mengamankan" aplikasi yang mereka tulis dengan menggabungkan mekanisme kriptografi.

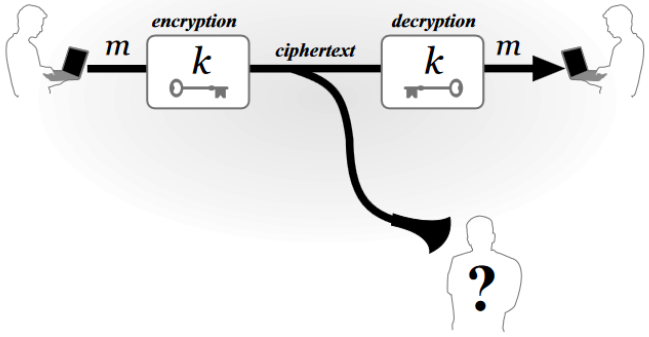
Singkatnya, kriptografi telah berubah dari seperangkat alat heuristik yang berkaitan dengan memastikan komunikasi rahasia bagi militer menjadi ilmu yang membantu mengamankan sistem bagi masyarakat biasa di seluruh dunia. Ini juga berarti bahwa kriptografi telah menjadi topik sentral dalam ilmu komputer.

1.2 PENGATURAN ENKRIPSI KUNCI PRIBADI

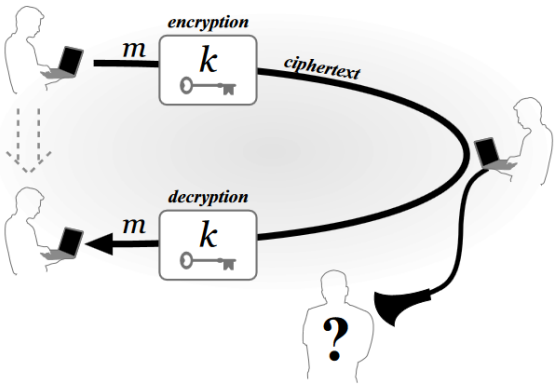
Kriptografi klasik berkaitan dengan perancangan dan penggunaan kode (juga disebut cipher) yang memungkinkan dua pihak berkomunikasi secara diam-diam di hadapan seorang

penyadap yang dapat memantau semua komunikasi di antara mereka. Dalam bahasa modern, kode disebut skema enkripsi dan itulah terminologi yang akan kita gunakan di sini. Keamanan semua skema enkripsi klasik bergantung pada sebuah rahasia sebuah kunci yang dibagikan oleh pihak-pihak yang berkomunikasi sebelumnya dan tidak diketahui oleh penyadap. Skenario ini dikenal sebagai pengaturan kunci pribadi (atau kunci bersama/rahasia), dan enkripsi kunci pribadi hanyalah salah satu contoh primitif kriptografi yang digunakan dalam pengaturan ini. Sebelum menjelaskan beberapa skema enkripsi historis, kita membahas enkripsi kunci pribadi secara lebih umum.

Dalam pengaturan enkripsi kunci pribadi, dua pihak berbagi kunci dan menggunakan kunci ini ketika mereka ingin berkomunikasi secara diam-diam. Satu pihak dapat mengirim pesan, atau teks biasa, ke pihak lain dengan menggunakan kunci bersama untuk mengenkripsi (atau "mengacak") pesan tersebut dan dengan demikian memperoleh teks tersandi yang dikirimkan ke penerima. Penerima menggunakan kunci yang sama untuk mendekripsi (atau "menguraikan") ciphertext dan memulihkan pesan aslinya. Perhatikan bahwa kunci yang sama digunakan untuk mengubah teks biasa menjadi teks tersandi dan sebaliknya; itulah mengapa hal ini juga dikenal sebagai pengaturan kunci simetris, dimana kesimetriannya terletak pada kenyataan bahwa kedua belah pihak memegang kunci yang sama yang digunakan untuk enkripsi dan dekripsi. Hal ini berbeda dengan enkripsi asimetris atau kunci publik (diperkenalkan pada Bab 10), dimana enkripsi dan dekripsi menggunakan kunci yang berbeda.



Gambar 1.1 Salah Satu Pengaturan Umum Kriptografi Kunci Pribadi (Di Sini, Enkripsi): Dua Pihak Berbagi Kunci Yang Mereka Gunakan Untuk Berkomunikasi Dengan Aman.



Gambar 1.2 Pengaturan Umum Kriptografi Kunci Pribadi Lainnya (Sekali Lagi, Enkripsi): Satu Pengguna Menyimpan Data Dengan Aman Dari Waktu Ke Waktu.

Seperti telah disebutkan, tujuan enkripsi adalah untuk menjaga agar teks biasa tetap tersembunyi dari penyadap yang dapat memantau saluran komunikasi dan mengamati teks tersandi. Kita akan membahas hal ini secara lebih rinci nanti di bab ini, dan menghabiskan banyak waktu di Bab 2 dan 3 untuk mendefinisikan tujuan ini secara formal.

Ada dua aplikasi kanonik kriptografi kunci pribadi. Yang pertama, ada dua pihak berbeda yang terpisah dalam ruang, misalnya, seorang pekerja di New York berkomunikasi dengan rekannya di California; lihat Gambar 1.2. Kedua pengguna ini diasumsikan telah dapat berbagi kunci dengan aman sebelum komunikasi mereka. (Perhatikan bahwa jika satu pihak hanya mengirimkan kunci ke pihak lain melalui saluran komunikasi publik, maka penyadap akan mendapatkan kuncinya juga!) Seringkali hal ini mudah dilakukan dengan mengadakan pertemuan fisik di lokasi yang aman untuk berbagi kunci sebelum mereka memisahkan; dalam contoh yang baru saja diberikan, rekan kerja mungkin mengatur untuk berbagi kunci ketika mereka berdua berada di kantor New York. Dalam kasus lain, berbagi kunci dengan aman lebih sulit dilakukan. Untuk beberapa bab berikutnya kita hanya berasumsi bahwa berbagi kunci adalah mungkin; kami akan meninjau kembali masalah ini di Bab 10.

Penerapan kriptografi kunci pribadi yang kedua secara luas melibatkan pihak yang sama berkomunikasi dengan dirinya sendiri sepanjang waktu. (Lihat Gambar 1.2.) Pertimbangkan, misalnya, enkripsi disk, di mana pengguna mengenkripsi beberapa teks biasa dan menyimpan teks sandi yang dihasilkan di hard drive mereka; pengguna yang sama akan kembali lagi di lain waktu untuk mendekripsi ciphertext dan memulihkan data asli. Hard drive di sini berfungsi sebagai saluran komunikasi di mana penyerang mungkin menguping dengan mendapatkan akses ke hard drive dan membaca isinya. “Berbagi” kunci sekarang menjadi hal yang sepele, meskipun pengguna masih memerlukan cara yang aman dan dapat diandalkan untuk mengingat/menyimpan kunci untuk digunakan di lain waktu.

Sintaks enkripsi. Secara formal, skema enkripsi kunci pribadi didefinisikan dengan menentukan ruang pesan bersama dengan tiga algoritma: prosedur untuk menghasilkan kunci (Gen), prosedur untuk mengenkripsi (Enc), dan prosedur untuk mendekripsi (Des). Ruang pesan mendefinisikan kumpulan pesan “legal”, yaitu pesan yang didukung oleh skema. Algoritme memiliki fungsi sebagai berikut:

1. Algoritma pembangkitan kunci Gen adalah algoritma probabilistik yang menghasilkan kunci k yang dipilih menurut beberapa distribusi.
2. Algoritma enkripsi Enc mengambil masukan kunci k dan pesan m dan mengeluarkan teks sandi c. Kami menyatakan dengan $\mathrm{Enc}_k(m)$ enkripsi teks biasa m menggunakan kunci k.
3. Algoritma dekripsi Dec mengambil masukan kunci k dan teks tersandi c dan mengeluarkan teks biasa m. Kami menunjukkan dekripsi ciphertext c menggunakan kunci k dengan $\mathrm{Dec}_k(c)$. Skema enkripsi harus memenuhi persyaratan kebenaran berikut: untuk setiap kunci k keluaran Gen dan setiap pesan $m \in M$, berlaku bahwa

$$Dec_k\big(Enc_k(m)\big) = m.$$

Dengan kata lain: mengenkripsi pesan dan kemudian mendekripsi teks sandi yang dihasilkan (menggunakan kunci yang sama) akan menghasilkan pesan asli.

Himpunan semua kunci yang mungkin dihasilkan oleh algoritma pembangkitan kunci disebut ruang kunci dan dilambangkan dengan K. Hampir selalu, Gen hanya memilih kunci seragam dari ruang kunci; pada kenyataannya, kita dapat berasumsi tanpa kehilangan gambaran umum bahwa memang demikianlah masalahnya (lihat Latihan 2.1).

Meninjau pembahasan kita sebelumnya, skema enkripsi dapat digunakan oleh dua pihak yang ingin berkomunikasi sebagai berikut. Pertama, Gen dijalankan untuk mendapatkan kunci k yang dibagikan oleh para pihak. Kemudian, ketika salah satu pihak ingin mengirimkan teks biasa m ke pihak lain, ia menghitung $c := Enc_k(m)$ dan mengirimkan hasil ciphertext c melalui saluran publik ke pihak lain. Setelah menerima c, pihak lain menghitung $m := Dec_k(C)$ untuk memulihkan teks biasa asli.

Prinsip Keys dan Kerckhoffs. Seperti yang sudah dijelaskan di atas, jika musuh yang melakukan penyadapan mengetahui algoritma Dec serta kunci k yang digunakan oleh kedua pihak yang berkomunikasi, maka musuh tersebut akan mampu mendekripsi teks sandi apa pun yang dikirimkan oleh pihak-pihak tersebut. Karena alasan inilah pihak-pihak yang berkomunikasi harus berbagi kunci k dengan aman dan merahasiakan k dari orang lain. Mungkin mereka juga harus merahasiakan algoritma dekripsi Des? Oleh karena itu, bukankah lebih baik bagi mereka untuk merahasiakan semua rincian skema enkripsinya?

Pada akhir abad ke-19, Auguste Kerckhoffs berpendapat sebaliknya dalam sebuah makalah yang ditulisnya yang menjelaskan beberapa prinsip desain sandi militer. Salah satu prinsip terpenting, yang sekarang dikenal sebagai prinsip Kerckhoffs, adalah:

"Metode sandi tidak harus dirahasiakan, dan harus bisa jatuh ke tangan musuh tanpa ketidaknyamanan."

Artinya, skema enkripsi harus dirancang agar aman meskipun penyadap mengetahui semua detail skema tersebut, selama penyerang tidak mengetahui kunci yang digunakan. Dengan kata lain, keamanan tidak boleh bergantung pada kerahasiaan skema enkripsi; sebaliknya, prinsip Kerckhoffs menuntut keamanan hanya bergantung pada kerahasiaan kunci.

Ada tiga argumen utama yang mendukung prinsip Kerckhoffs. Yang pertama adalah jauh lebih mudah bagi para pihak untuk menjaga kerahasiaan kunci pendek dibandingkan merahasiakan algoritma (yang lebih rumit) yang mereka gunakan. Hal ini terutama berlaku jika kita membayangkan penggunaan enkripsi untuk mengamankan komunikasi antara semua pasangan karyawan di suatu organisasi. Kecuali masing-masing pasangan pihak menggunakan algoritme unik mereka sendiri, beberapa pihak akan mengetahui algoritme yang digunakan oleh pihak lain. Informasi tentang algoritme enkripsi mungkin dibocorkan oleh salah satu karyawan ini (misalnya, setelah dipecat), atau diperoleh oleh penyerang menggunakan

rekayasa balik. Singkatnya, tidak realistis untuk berasumsi bahwa algoritma enkripsi akan tetap dirahasiakan.

Kedua, jika informasi rahasia yang dibagikan oleh pihak-pihak yang jujur terbongkar, maka akan lebih mudah bagi mereka untuk mengganti kunci dibandingkan mengganti skema enkripsi. (Pertimbangkan untuk memperbarui file dibandingkan menginstal program baru.) Selain itu, menghasilkan rahasia acak baru adalah hal yang relatif mudah, padahal merancang skema enkripsi baru akan menjadi upaya yang sangat besar. Terakhir, untuk penerapan skala besar, akan jauh lebih mudah bagi pengguna untuk mengandalkan algoritme/perangkat lunak enkripsi yang sama (dengan kunci berbeda) dibandingkan setiap orang menggunakan algoritme khusus mereka sendiri. (Hal ini berlaku bahkan untuk satu pengguna yang berkomunikasi dengan beberapa pihak berbeda.) Faktanya, skema enkripsi sebaiknya distandarisasi sehingga (1) kompatibilitas dipastikan secara default dan (2) pengguna akan menggunakan skema enkripsi yang telah mendapat sorotan publik dan tidak ditemukan kelemahannya.

Saat ini prinsip Kerckhoffs dipahami sebagai anjuran agar desain kriptografi dipublikasikan sepenuhnya, sangat berbeda dengan gagasan “keamanan karena ketidakjelasan” yang menyatakan bahwa menjaga kerahasiaan algoritma akan meningkatkan keamanan. Sangat berbahaya jika menggunakan algoritma yang bersifat “buatan sendiri” (yaitu algoritma yang tidak terstandarisasi dan dirancang secara rahasia oleh suatu perusahaan). Sebaliknya, desain yang dipublikasikan harus melalui tinjauan publik dan oleh karena itu cenderung lebih kuat. Pengalaman bertahun-tahun menunjukkan bahwa sangat sulit untuk membangun skema kriptografi yang baik. Oleh karena itu, keyakinan kita terhadap keamanan suatu skema akan jauh lebih tinggi jika skema tersebut telah dipelajari secara ekstensif (oleh para ahli selain perancang skema) dan tidak ditemukan kelemahannya. Meski terdengar sederhana dan jelas, prinsip desain kriptografi terbuka (yaitu prinsip Kerckhoffs) telah diabaikan berulang kali dan mengakibatkan bencana. Untungnya, saat ini terdapat cukup sistem kripto yang aman, terstandarisasi, dan tersedia secara luas sehingga tidak ada alasan untuk menggunakan sistem kripto lainnya.

1.3 SANDI SEJARAH DAN KRIPTANALISISNYA

Dalam studi kita tentang kriptografi “klasik”, kita akan memeriksa beberapa skema enkripsi historis dan menunjukkan bahwa skema tersebut tidak aman. Tujuan utama kami dalam menyajikan materi ini adalah

(1) untuk menyoroti kelemahan pendekatan “ad hoc” terhadap kriptografi, dan dengan demikian memotivasi pendekatan modern dan ketat yang akan diambil dalam sisa buku ini, dan;
(2) untuk menunjukkan bahwa pendekatan sederhana untuk mencapai enkripsi yang aman kemungkinan besar tidak akan berhasil. Selanjutnya, kami akan menyajikan beberapa prinsip utama kriptografi yang terinspirasi oleh kelemahan skema historis ini.

Pada bagian ini, karakter teks biasa ditulis dalam huruf kecil dan karakter teks sandi ditulis dalam HURUF BESAR untuk kejelasan tipografi.

sandi Caesar, salah satu sandi tertua yang tercatat, yang dikenal sebagai sandi Caesar, dijelaskan dalam *De Vita Caesarum, Divus Iulius* ("Kehidupan Para Kaisar, Julius yang Didewakan"), yang ditulis sekitar tahun 110 M:

Ada juga surat-suratnya kepada Cicero, serta kepada teman-teman karibnya mengenai urusan pribadi, dan pada surat-surat terakhir itu, jika ia mempunyai sesuatu yang rahasia untuk dikatakan, ia menulisnya dalam sandi, yakni dengan mengubah urutan surat-suratnya. alfabet, sehingga tidak ada satu kata pun yang dapat terucap. . .

Julius Caesar melakukan enkripsi dengan menggeser huruf abjad 3 tempat ke depan: a diganti dengan D, b dengan E, dan seterusnya. Di akhir alfabet, huruf-hurufnya melingkari sehingga z diganti dengan C, y dengan B, dan x dengan A. Misalnya, enkripsi pesan memulai serangan sekarang, dengan spasi dihilangkan, menghasilkan:

EHJLQWKHDWWDFNQRZ.

Masalah langsung dengan sandi ini adalah metode enkripsinya telah diperbaiki; tidak ada kunci. Dengan demikian, siapapun yang mengetahui bagaimana Caesar mengenkripsi pesannya akan dapat mendekripsinya dengan mudah.

Menariknya, varian sandi yang disebut ROT-13 (dengan pergeseran 13 tempat, bukan 3) masih digunakan hingga saat ini di berbagai forum online. Dapat dipahami bahwa hal ini tidak memberikan keamanan kriptografi apa pun; ini digunakan hanya untuk memastikan bahwa teks (misalnya, spoiler film) tidak dapat dipahami kecuali pembaca pesan secara sadar memilih untuk mendekripsi pesan tersebut.

Pergeseran cipher dan prinsip ruang kunci yang memadai. Shift cipher dapat dilihat sebagai varian kunci dari Caesar's cipher. Secara khusus, dalam shift cipher, kunci k adalah angka antara 0 dan 25. Untuk mengenkripsi, huruf digeser seperti pada Caesar's cipher, namun sekarang sebanyak k tempat. Memetakan ini ke sintaks enkripsi yang dijelaskan sebelumnya, ruang pesan terdiri dari rangkaian huruf Inggris dengan panjang sembarang dengan tanda baca, spasi, dan angka dihilangkan, dan tanpa perbedaan antara huruf besar dan kecil. Algoritma Gen mengeluarkan kunci seragam $k\ 0, \ldots, 25$; algoritma Enc mengambil kunci k dan teks biasa dan menggeser setiap huruf dari teks biasa ke depan posisi k (melilit di akhir alfabet); dan algoritma Des mengambil kunci k dan teks sandi dan menggeser setiap huruf teks sandi ke belakang posisi k.

Deskripsi yang lebih matematis diperoleh dengan menyamakan alfabet bahasa Inggris dengan himpunan $\{0, \ldots, 25\}$ (jadi a = 0, b = 1, dst.). Ruang pesan M kemudian merupakan barisan bilangan bulat berhingga dari himpunan ini. Enkripsi pesan $m = m_1 \cdot \cdot \cdot m_A$ (di mana $m_i \in \{0, \ldots, 25\}$) menggunakan kunci k diberikan oleh

$$Enc_k(m_1 \ldots . m_\varphi) = c_1 \ldots c_{\varphi,} \, dimana \, c_1 = [(m_i + k) Mod \, 26]$$

(Notasi [a mod N] menunjukkan sisa a setelah pembagian dengan N , dengan 0 ≤ [a mod N] < N . Kita mengacu pada proses pemetaan a ke [a mod N] sebagai modulo reduksi N ; kita akan membahas lebih banyak tentang ini di awal Bab 8.) Dekripsi ciphertext c = $c_1 \cdot \cdot \cdot c_A$ menggunakan kunci k diberikan oleh

$$\mathrm{Dec}_k(\mathrm{c}_1 \ldots . \mathrm{c}_\varphi) = \mathrm{m}_1 \ldots \mathrm{m}_{\varphi,} \text{ dimana } \mathrm{m}_i = [(c_i + \mathrm{k}) \mathrm{Mod}\ 26]$$

Apakah sandi shift aman? Sebelum melanjutkan membaca, cobalah mendekripsi ciphertext berikut yang dihasilkan menggunakan shift cipher dan kunci rahasia k:

OVDTHUFWVZZPISLRLFZHYLAOLYL

Apakah mungkin memulihkan pesan tanpa mengetahui k? Sebenarnya itu sepele! Alasannya adalah hanya ada 26 kemungkinan kunci. Jadi seseorang dapat mencoba mendekripsi ciphertext menggunakan setiap kunci yang mungkin dan dengan demikian memperoleh daftar 26 kandidat plaintext. Teks biasa yang benar pasti ada dalam daftar ini; terlebih lagi, jika ciphertextnya "cukup panjang" maka plaintext yang benar kemungkinan besar akan menjadi satu-satunya kandidat dalam daftar yang "masuk akal". (Yang terakhir ini belum tentu benar, namun akan selalu benar. Sekalipun tidak benar, serangan ini mempersempit kumpulan teks biasa yang potensial menjadi paling banyak 26 kemungkinan.) Dengan memindai daftar kandidat, hal ini dapat dilakukan dengan mudah. untuk memulihkan teks biasa asli.

Serangan yang melibatkan percobaan setiap kunci yang mungkin disebut serangan brute force atau serangan pencarian menyeluruh. Jelasnya, agar skema enkripsi aman, skema tersebut tidak boleh rentan terhadap serangan semacam itu. Pengamatan ini dikenal sebagai prinsip ruang kunci yang memadai:

Setiap skema enkripsi yang aman harus memiliki ruang kunci yang cukup besar untuk membuat serangan pencarian menyeluruh menjadi tidak mungkin dilakukan.

Kita dapat memperdebatkan berapa banyak upaya yang membuat suatu tugas "tidak mungkin dilakukan", dan penentuan kelayakan yang tepat bergantung pada sumber daya penyerang potensial dan lamanya waktu pengirim dan penerima ingin memastikan kerahasiaan komunikasi mereka. Saat ini, penyerang dapat menggunakan superkomputer, puluhan ribu komputer pribadi, atau unit pemrosesan grafis (GPU) untuk mempercepat serangan brute force. Oleh karena itu, untuk melindungi dari serangan-serangan tersebut, ruang kunci harus berukuran sangat besar katakanlah, setidaknya berukuran 270, dan bahkan lebih besar lagi jika kita mengkhawatirkan keamanan jangka panjang terhadap penyerang yang memiliki dana besar.

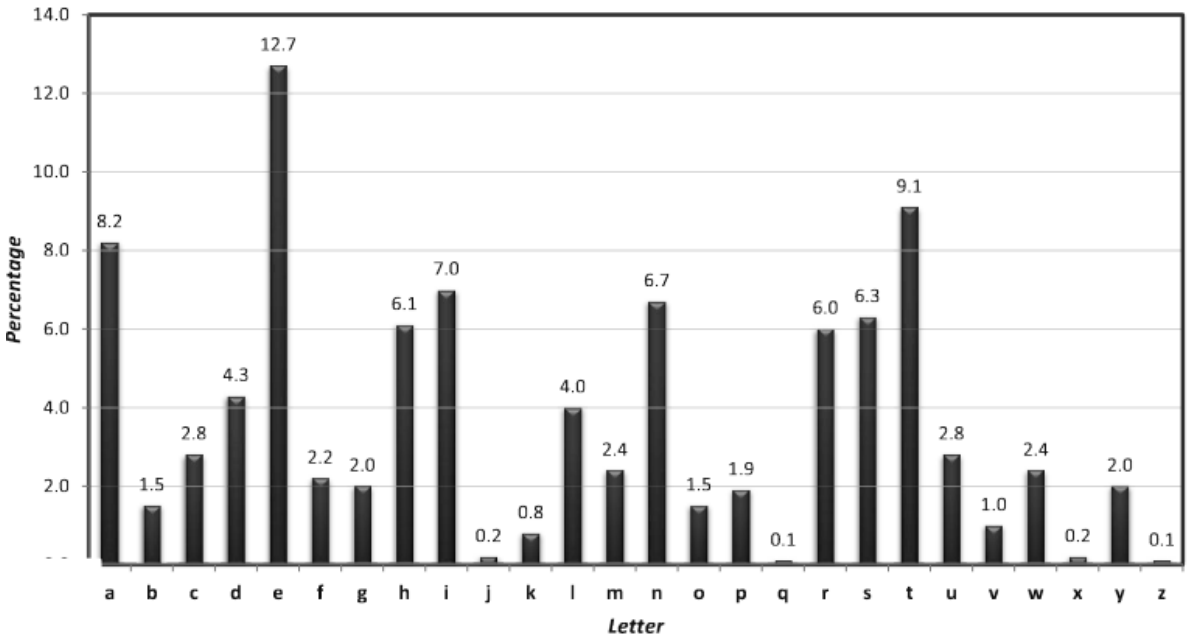
Prinsip ruang kunci yang memadai memberikan kondisi yang diperlukan untuk keamanan, namun tidak cukup. Contoh berikutnya menunjukkan hal ini. Sandi substitusi

mono-abjad. Pada shift cipher, kunci mendefinisikan peta dari setiap huruf alfabet (plaintext) ke beberapa huruf alfabet (ciphertext), dimana peta tersebut merupakan pergeseran tetap yang ditentukan oleh kunci. Dalam sandi substitusi mono-abjad, kuncinya juga mendefinisikan peta berdasarkan alfabet, tetapi peta tersebut sekarang diperbolehkan untuk berubah-ubah hanya dengan batasan satu-ke-satu sehingga dekripsi dapat dilakukan. Ruang kunci dengan demikian terdiri dari semua bijeksi, atau permutasi, alfabet. Misalnya, kunci yang mendefinisikan permutasi berikut

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
X	E	U	A	D	N	B	K	V	M	R	O	C	Q	F	S	Y	H	W	G	L	Z	I	J	P	T

(di mana peta ke X, dll.) akan mengenkripsi pesan tellhimaboutme ke GDOOKVCXEFLGCD. Nama sandi ini berasal dari fakta bahwa kunci tersebut mendefinisikan substitusi (tetap) untuk masing-masing karakter teks biasa.

Dengan asumsi alfabet Inggris yang digunakan, ruang kuncinya berukuran $26! = 26\ 25\ 24\ 2\ 1$, atau sekitar 288, dan serangan brute force tidak mungkin dilakukan. Namun, ini tidak berarti sandi tersebut aman! Faktanya, seperti yang akan kami tunjukkan selanjutnya, skema ini mudah dipatahkan meskipun memiliki ruang kunci yang besar.



Gambar 1.3 Rata-Rata Frekuensi Huruf Untuk Teks Berbahasa Inggris.

Asumsikan teks berbahasa Inggris sedang dienkripsi (yaitu, teks tersebut merupakan tulisan bahasa Inggris yang benar secara tata bahasa, bukan hanya teks yang ditulis menggunakan karakter alfabet Inggris). Sandi substitusi mono-abjad kemudian dapat diserang dengan memanfaatkan pola statistik bahasa Inggris. (Tentu saja, serangan yang sama dapat diterapkan pada bahasa apa pun.) Serangan tersebut bergantung pada fakta bahwa:

1. Untuk kunci apa pun, pemetaan setiap huruf adalah tetap, sehingga jika e dipetakan ke D, maka setiap kemunculan e di plaintext akan menghasilkan kemunculan D di ciphertext.
2. Diketahui distribusi frekuensi masing-masing huruf dalam bahasa Inggris (lihat Gambar 1.3). Tentu saja, teks yang sangat pendek mungkin menyimpang dari distribusi ini, tetapi bahkan teks yang hanya terdiri dari beberapa kalimat cenderung memiliki distribusi yang mendekati rata-rata.

Serangan tersebut bekerja dengan mentabulasikan distribusi frekuensi karakter dalam ciphertext, yaitu mencatat A muncul 11 kali, B muncul 4 kali, dan seterusnya. Frekuensi ini kemudian dibandingkan dengan frekuensi huruf yang diketahui dalam teks bahasa Inggris normal. Seseorang kemudian dapat menebak bagian pemetaan yang ditentukan oleh kunci berdasarkan frekuensi yang diamati. Misalnya, karena e adalah huruf yang paling sering muncul dalam bahasa Inggris, orang dapat menebak bahwa karakter yang paling sering muncul dalam ciphertext berhubungan dengan karakter plaintext e, dan seterusnya. Beberapa tebakan mungkin salah, namun cukup banyak tebakan yang benar sehingga memungkinkan dekripsi yang relatif cepat (terutama memanfaatkan pengetahuan bahasa Inggris lainnya, seperti fakta bahwa u biasanya mengikuti q, dan h kemungkinan besar muncul di antara t dan e). Kami menyimpulkan bahwa meskipun sandi substitusi mono-abjad memiliki ruang kunci yang besar, namun tetap tidak aman.

Tidak mengherankan jika sandi substitusi mono-abjad dapat dengan cepat dipecahkan, karena teka-teki berdasarkan sandi ini muncul di surat kabar (dan dipecahkan oleh beberapa orang sebelum minum kopi pagi!). Kami menyarankan Anda mencoba menguraikan ciphertext berikut ini akan meyakinkan Anda betapa mudahnya serangan itu dilakukan. (Gunakan Gambar 1.3 untuk membantu Anda.)

```
JGRMQOYGHMVBJWRWQFPWHGFFDQGFPFZRKBEEBJIZQQOCIBZKLFAFGQVFZFWWE
OGWOPFGFHWOLPHLRLOLFDMFGQWBLWBWQOLKFWBYLBLYLFSFLJGRMQBOLWJVFP
FWQVHQWFFPQOQVFPQOCFPOGFWFJIGFQVHLHLROQVFGWJVFPFOLFHGQVQVFILE
OGQILHQFQGIQVVOSFAFGBWQVHQWIJVWJVFPFWHGFIWIHZZRQGBABHZQOCGFHX
```

Serangan yang ditingkatkan pada shift cipher. Kita dapat menggunakan tabel frekuensi huruf untuk memberikan serangan yang lebih baik pada shift cipher. Serangan kami sebelumnya terhadap shift cipher memerlukan dekripsi ciphertext menggunakan setiap kunci yang mungkin, dan kemudian memeriksa kunci mana yang menghasilkan teks biasa yang "masuk akal." Kelemahan dari pendekatan ini adalah agak sulit untuk diotomatisasi, karena sulit bagi komputer untuk memeriksa apakah teks biasa yang diberikan "masuk akal." (Kami tidak mengklaim bahwa hal ini tidak mungkin dilakukan, karena serangan dapat diotomatisasi menggunakan kamus kata-kata bahasa Inggris yang valid. Kami hanya mengklaim bahwa mengotomatisasi bukanlah hal yang mudah.) Selain itu, mungkin ada beberapa kasus kita akan melihatnya nanti di mana karakter teks biasa didistribusikan seperti teks berbahasa Inggris

meskipun teks biasa itu sendiri bukan bahasa Inggris yang valid, dalam hal ini pemeriksaan teks biasa yang “masuk akal” tidak akan berhasil.

Kami sekarang menggambarkan serangan yang tidak mengalami kelemahan ini. Seperti sebelumnya, kaitkan huruf alfabet bahasa Inggris dengan $0, \ldots, 25$. Misalkan pi, dengan $0 \quad P_i \quad 1$, menunjukkan frekuensi huruf ke-i dalam teks bahasa Inggris normal (mengabaikan spasi, tanda baca, dll.). Perhitungan menggunakan Gambar 1.3 memberikan

$$\sum_{i=0}^{25} P_i^2 \approx 0.065. \qquad (1.1)$$

Sekarang, katakanlah kita diberikan beberapa ciphertext dan biarkan qi menunjukkan frekuensi huruf ke-i dari alfabet dalam ciphertext ini; yaitu, q_i hanyalah jumlah kemunculan huruf ke-i dari alfabet dalam teks tersandi dibagi dengan panjang teks tersandi. Jika kuncinya adalah k, maka pi kira-kira sama dengan q_{i+k} untuk semua i, karena huruf ke-i dipetakan ke huruf ke$-(i + k)$. (Kami menggunakan $i+k$ daripada $[i+k \mod 26]$ yang lebih rumit).) Jadi, jika kita menghitung

$$I_j \stackrel{\text{def}}{=} \sum_{i=0}^{25} P_i \cdot q_{i+j}$$

untuk setiap nilai $j \in \{0, \ldots, 25\}$, maka kita berharap untuk menemukan bahwa $I_k \approx 0,065$ (di mana k adalah kunci sebenarnya), sedangkan $I_j \text{ untuk } j \neq k$ akan berbeda dari $0,065$. Hal ini menyebabkan serangan pemulihan kunci yang mudah diotomatisasi: hitung I_j untuk semua j, lalu keluarkan nilai k yang I_k nya paling dekat dengan $0,065$.

Sandi *Vigen`ere* (pergeseran poli-abjad). Serangan statistik terhadap sandi substitusi mono-abjad dapat dilakukan karena kunci tersebut mendefinisikan pemetaan tetap yang diterapkan huruf demi huruf pada teks biasa. Serangan seperti itu dapat digagalkan dengan menggunakan sandi substitusi poli-abjad di mana kuncinya mendefinisikan pemetaan yang diterapkan pada blok karakter teks biasa. Di sini, misalnya, sebuah kunci mungkin memetakan blok 2 karakter ab ke DZ sementara memetakan ac ke TY; perhatikan bahwa karakter teks biasa a tidak dipetakan ke karakter teks sandi tetap. Sandi substitusi poli-abjad “memuluskan” distribusi frekuensi karakter dalam teks tersandi dan mempersulit analisis statistik.

Sandi Vigen`ere, yang merupakan kasus khusus di atas, juga disebut sandi pergeseran poli-abjad, bekerja dengan menerapkan beberapa contoh sandi geser yang independen secara berurutan. Kuncinya sekarang dipandang sebagai rangkaian huruf; enkripsi dilakukan dengan menggeser setiap karakter teks biasa sesuai jumlah yang ditunjukkan oleh karakter kunci berikutnya, membungkus kunci bila diperlukan. (Ini berubah menjadi shift cipher jika kuncinya memiliki panjang 1.) Misalnya, enkripsi pesan tellhimaboutme menggunakan key cafe akan berfungsi sebagai berikut:

Teks biasa	Beritahu dia tentang aku
Kunci (Berulang):	Cafecafecafeca
Teks sandi:	VEQPJIREDOZXOE

(Kuncinya tidak harus berupa kata dalam bahasa Inggris.) Ini sama persis dengan mengenkripsi kata pertama, kelima, kesembilan, . . . karakter dengan shift cipher dan kunci c; yang kedua, keenam, kesepuluh,. . . karakter dengan kunci a; yang ketiga, ketujuh,. . . karakter dengan f; dan yang keempat, kedelapan, . . . karakter dengan e. Perhatikan bahwa dalam contoh di atas l dipetakan sekali ke Q dan sekali ke P. Selanjutnya, karakter ciphertext E kadang-kadang diperoleh dari e dan kadang-kadang dari a. Dengan demikian, frekuensi karakter dari ciphertext "dihaluskan", sesuai keinginan.

Jika kuncinya cukup panjang, memecahkan sandi ini tampak sulit. Memang benar bahwa sistem ini dianggap oleh banyak orang sebagai sistem yang "tidak dapat dipecahkan," dan meskipun sistem ini ditemukan pada abad ke-16, serangan sistematis terhadap skema ini baru dilakukan ratusan tahun kemudian.

Menyerang sandi Vigen`ere. Pengamatan pertama dalam menyerang sandi Vigenere adalah jika panjang kunci diketahui maka menyerang sandi tersebut relatif mudah. Secara spesifik, katakanlah panjang kunci, disebut juga periode, adalah t. Tuliskan kunci k sebagai $k = k_1 \cdot \cdot \cdot k_t$ dimana setiap ki adalah huruf alfabet. Sebuah ciphertext yang diamati $c = c_1c_2$ dapat dibagi menjadi t bagian dimana setiap bagian dapat dilihat telah dienkripsi menggunakan shift cipher. Khususnya, untuk semua $j \in \{1, \ldots, t\}$ $k$arakter teks tersandi

$$C_j, C_{j+t}, C_{j+2t}, \ldots$$

semua dihasilkan dengan menggeser karakter teks biasa yang sesuai sebanyak posisi k_j. Kami menyebut urutan karakter di atas sebagai aliran $ke-j$. Yang tersisa hanyalah menentukan, untuk masing-masing t aliran, yang mana dari 26 kemungkinan pergeseran yang digunakan. Hal ini tidak sepele seperti dalam kasus shift cipher, karena tidak mungkin lagi mencoba pergeseran yang berbeda dalam upaya untuk menentukan kapan dekripsi suatu aliran "masuk akal." (Ingat bahwa aliran tidak berhubungan dengan huruf-huruf yang berurutan dalam teks biasa.) Selain itu, mencoba menebak seluruh kunci k sekaligus akan memerlukan pencarian brute force melalui 26^t kemungkinan berbeda, yang tidak mungkin dilakukan untuk t besar. Meskipun demikian, kita masih dapat menggunakan analisis frekuensi huruf untuk menganalisis setiap aliran secara independen. Yaitu, untuk setiap aliran, kami mentabulasikan frekuensi setiap karakter cipherteks dan kemudian memeriksa yang mana dari 26 kemungkinan pergeseran yang menghasilkan distribusi probabilitas yang "benar" untuk aliran tersebut. Karena ini dapat dilakukan secara independen untuk setiap aliran (yaitu, untuk setiap karakter kunci), serangan ini memerlukan waktu 26^t daripada waktu 26^t.

Pendekatan yang lebih berprinsip dan lebih mudah diotomatisasi adalah dengan menggunakan metode yang lebih baik untuk menyerang shift cipher yang telah dibahas sebelumnya. Serangan tersebut tidak bergantung pada pemeriksaan teks biasa yang "masuk akal", namun hanya mengandalkan distribusi frekuensi karakter dalam teks biasa. Salah satu dari pendekatan di atas memberikan serangan yang berhasil ketika panjang kunci diketahui. Bagaimana jika panjang kunci tidak diketahui?

Perhatikan terlebih dahulu bahwa selama panjang maksimum T dari kunci tersebut tidak terlalu besar, kita cukup mengulangi serangan di atas sebanyak T kali (untuk setiap kemungkinan nilai $t \in 1, \ldots, T$). Hal ini menghasilkan paling banyak T kandidat teks biasa yang berbeda, di antaranya teks biasa yang sebenarnya kemungkinan besar akan mudah diidentifikasi. Jadi panjang kunci yang tidak diketahui bukanlah kendala yang serius.

Ada juga cara yang lebih efisien untuk menentukan panjang kunci dari ciphertext yang diamati. Salah satunya adalah dengan menggunakan metode Kasiski yang diterbitkan pada pertengahan abad ke-19. Langkah pertama di sini adalah mengidentifikasi pola berulang dengan panjang 2 atau 3 dalam ciphertext. Ini kemungkinan besar merupakan hasil dari bigram atau trigram tertentu yang sering muncul di teks biasa. Misalnya, pertimbangkan kata umum "the." Kata ini akan dipetakan ke karakter ciphertext yang berbeda-beda, bergantung pada posisinya dalam plaintext. Namun jika muncul dua kali pada posisi relatif yang sama, maka akan dipetakan ke karakter ciphertext yang sama. Untuk plaintext yang cukup panjang, ada kemungkinan besar bahwa "the" akan dipetakan berulang kali ke karakter ciphertext yang sama.

Perhatikan contoh nyata berikut dengan manik-manik kunci (spasi telah ditambahkan untuk kejelasan):

Plaintext:	Laki – laki dan perempuan mengembil surat dari kantor pos
Kunci:	Bea dsb ead sbe adsbe adsbeadsb ead sbeads bead sbe adsb eadsbe
Ciphertext:	ULE PSO ENG LII WREBR RHLSMEYWE XHH DFXTHJ GVOP LII PRKU SFIADI

Kata the terkadang dipetakan ke ULE, terkadang ke LII, dan terkadang ke XHH. Namun, ini dipetakan dua kali ke LII, dan dalam teks yang cukup panjang kemungkinan besar akan dipetakan beberapa kali ke masing-masing kemungkinan tersebut. Pengamatan Kasiski adalah bahwa jarak antara kemunculan berulang tersebut (dengan asumsi hal tersebut tidak terjadi secara kebetulan) pasti merupakan kelipatan periode. (Dalam contoh di atas, periodenya adalah 5 dan jarak antara dua kemunculan LII adalah 30, yaitu 6 kali periode.) Oleh karena itu, pembagi persekutuan terbesar dari jarak antar barisan yang berulang (dengan asumsi keduanya tidak kebetulan) akan menghasilkan panjang kunci t atau kelipatannya.

Pendekatan alternatif, yang disebut metode indeks kebetulan, lebih metodis dan karenanya lebih mudah untuk diotomatisasi. Ingatlah bahwa jika panjang kunci adalah t, maka karakter ciphertextnya pada aliran pertama semuanya dihasilkan dari enkripsi menggunakan shift yang sama.

$$C_1, C_{1+t}, C_{1+2t}, \ldots$$

Artinya frekuensi karakter dalam urutan ini diharapkan identik dengan frekuensi karakter teks bahasa Inggris standar dalam beberapa urutan yang bergeser. Secara lebih rinci: misalkan qi menunjukkan frekuensi pengamatan huruf bahasa Inggris ke-i dalam aliran ini; ini hanyalah jumlah kemunculan huruf ke-i dalam alfabet dibagi dengan jumlah total huruf dalam aliran. Jika pergeseran yang digunakan di sini adalah j (yaitu, jika karakter pertama k1 pada kunci sama dengan j), maka untuk semua i kita mengharapkan $q_{i+j} \approx p_i$, dimana pi adalah frekuensi huruf ke-i dalam alfabet standar teks bahasa inggris. (Sekali lagi, kita menggunakan q_{i+j} sebagai pengganti $q_{[i+j \bmod 26]}$.) Namun ini berarti barisan $q_0, \ldots, q_{25}$ hanyalah barisan $p_0, \ldots, p_{25}$ bergeser *j* tempat. Akibatnya (lih. Persamaan (1.1)):

$$\sum_{i=0}^{25} q_i^2 \approx \sum_{i=0}^{25} p_i^2 \approx 0.065$$

Hal ini mengarah pada cara yang bagus untuk menentukan panjang kunci t. Untuk $\tau = 1, 2, \ldots$, lihatlah barisan karakter ciphertext $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$ dan tabulasikan $q_0, \ldots, q_{25}$ untuk urutan ini. Kemudian hitung

$$s_T \stackrel{\text{def}}{=} \sum_{i=0}^{25} q_i^2 \cdot$$

Ketika $\tau = t$ kita mengharapkan $S_\tau \approx 0{,}065$, seperti dibahas di atas. Di sisi lain, jika τ bukan kelipatan t, kita perkirakan semua karakter akan muncul dengan probabilitas yang kira-kira sama dalam barisan $c_1, c_{1+\tau}, c_{1+2\tau}, \ldots$, jadi kami mengharapkan $q_i \approx 1/26$ untuk semua i. Dalam hal ini kita akan memperoleh

$$S_T \approx \sum_{i=0}^{25} \left(\frac{1}{26}\right)^2 \approx 0.038.$$

Nilai τ terkecil yang $S_\tau \approx 0{,}065$ kemungkinan besar merupakan panjang kunci. Seseorang dapat memvalidasi lebih lanjut dugaan τ dengan melakukan penghitungan serupa menggunakan aliran kedua $c_2, c_{2+\tau}, c_{2+2\tau}, \ldots$, dll.

Panjang ciphertext dan serangan kriptanalitik. Serangan terhadap sandi Vigen`ere di atas memerlukan teks sandi yang lebih panjang dibandingkan serangan pada skema sebelumnya. Misalnya, metode indeks kebetulan memerlukan c_1, c_{1+t}, c_{1+2t} (dimana t adalah panjang kunci sebenarnya) yang cukup panjang untuk memastikan bahwa frekuensi yang diamati sesuai dengan yang diharapkan; ciphertext itu sendiri harus kira-kira t kali lebih besar. Demikian pula, serangan yang kami tunjukkan pada cipher substitusi mono-abjad memerlukan cipherteks yang lebih panjang dibandingkan serangan pada cipher shift (yang dapat bekerja untuk enkripsi bahkan untuk satu kata). Hal ini menggambarkan bahwa kunci

yang lebih panjang, secara umum, memerlukan kriptanalis untuk memperoleh lebih banyak cipherteks agar dapat melakukan serangan. (Memang benar, sandi Vigen`ere dapat terbukti aman jika kuncinya sepanjang yang dienkripsi. Kita akan melihat fenomena serupa di bab berikutnya.)

1.4 PRINSIP KRIPTOGRAFI MODERN

Seperti yang sudah jelas dari bagian sebelumnya, kriptografi secara historis lebih merupakan suatu seni daripada ilmu pengetahuan. Skema dirancang secara ad hoc dan dievaluasi berdasarkan kompleksitas atau kepintaran yang dirasakan. Sebuah skema akan dianalisis untuk melihat apakah ada serangan yang dapat ditemukan; jika demikian, skema tersebut akan "ditambal" untuk menggagalkan serangan tersebut, dan prosesnya akan berulang. Meskipun mungkin ada kesepakatan bahwa beberapa skema tidak aman (seperti yang terlihat dari serangan yang sangat merusak), tidak ada kesepakatan mengenai persyaratan apa yang harus dipenuhi oleh skema yang "aman", dan tidak ada bukti bahwa skema tertentu tidak aman.

Selama beberapa dekade terakhir, kriptografi telah berkembang menjadi suatu ilmu pengetahuan. Skema sekarang dikembangkan dan dianalisis dengan cara yang lebih sistematis, dengan tujuan akhir adalah untuk memberikan bukti yang kuat bahwa suatu konstruksi aman. Untuk mengartikulasikan bukti-bukti tersebut, pertama-tama kita memerlukan definisi formal yang menjelaskan dengan tepat apa arti "aman"; definisi seperti itu berguna dan menarik. Ternyata, sebagian besar bukti kriptografi bergantung pada asumsi yang saat ini belum terbukti mengenai kekerasan algoritmik dari permasalahan matematika tertentu; asumsi apa pun harus dibuat eksplisit dan dinyatakan secara tepat. Penekanan pada definisi, asumsi, dan bukti membedakan kriptografi modern dari kriptografi klasik; kita membahas ketiga prinsip ini secara lebih rinci di bagian berikut.

Prinsip 1 - Definisi Formal

Salah satu kontribusi utama kriptografi modern adalah pengakuan bahwa definisi formal keamanan sangat penting untuk desain, studi, evaluasi, dan penggunaan kriptografi primitif yang tepat. Definisi formal memberikan pemahaman tersebut dengan memberikan gambaran jelas mengenai ancaman apa saja yang berada dalam cakupannya dan jaminan keamanan apa yang diinginkan. Dengan demikian, definisi dapat membantu memandu desain skema kriptografi. Memang benar, lebih baik memformalkan apa yang diperlukan sebelum proses desain dimulai, daripada memberikan definisi post facto setelah desain selesai. Pendekatan yang terakhir ini berisiko berakhirnya fase desain ketika kesabaran para perancang telah habis (bukannya ketika tujuan telah tercapai), atau dapat mengakibatkan konstruksi mencapai lebih dari yang dibutuhkan sehingga mengorbankan efisiensi. Definisi juga menawarkan cara untuk mengevaluasi dan menganalisis apa yang dikonstruksi.

Dengan adanya definisi, seseorang dapat mempelajari skema yang diusulkan untuk melihat apakah skema tersebut mencapai jaminan yang diinginkan; dalam beberapa kasus, seseorang bahkan dapat membuktikan suatu konstruksi aman dengan menunjukkan bahwa konstruksi tersebut memenuhi definisi. Di sisi lain, definisi dapat digunakan untuk

menunjukkan secara meyakinkan bahwa suatu skema tertentu tidak aman, sejauh skema tersebut tidak memenuhi definisi tersebut. Secara khusus, perlu diingat bahwa serangan di bagian sebelumnya tidak secara otomatis menunjukkan bahwa skema apa pun yang ditampilkan di sana adalah "tidak aman." Misalnya, serangan terhadap sandi Vigen`ere berasumsi bahwa teks bahasa Inggris yang dienkripsi cukup panjang, namun sandi Vigen`ere dapat "aman" jika teks bahasa Inggris pendek, atau teks terkompresi (yang kira-kira memiliki frekuensi huruf yang seragam), dienkripsi? Sulit untuk mengatakannya tanpa definisi formal.

Definisi memungkinkan perbandingan skema yang bermakna. Seperti yang akan kita lihat, ada beberapa cara (yang valid) untuk mendefinisikan keamanan; pilihan yang "benar" bergantung pada konteks penggunaan skema tersebut. Suatu skema yang memenuhi definisi yang lebih lemah mungkin lebih efisien dibandingkan skema lain yang memenuhi definisi yang lebih kuat; dengan definisi yang tepat kita dapat mengevaluasi dengan tepat trade-off antara kedua skema tersebut. Sejalan dengan itu, definisi memungkinkan penggunaan skema yang aman. Pertimbangkan pertanyaan dalam memutuskan skema enkripsi mana yang akan digunakan untuk beberapa aplikasi yang lebih besar. Cara yang tepat untuk mengatasi masalah ini adalah dengan terlebih dahulu memahami gagasan keamanan apa yang diperlukan untuk aplikasi tersebut, dan kemudian menemukan skema enkripsi yang memenuhi gagasan tersebut. Manfaat sampingan dari pendekatan ini adalah modularitas: perancang dapat "menukar" satu skema enkripsi dan menggantinya dengan skema enkripsi lain (yang juga memenuhi definisi keamanan yang diperlukan) tanpa harus khawatir akan memengaruhi keamanan aplikasi secara keseluruhan.

Menulis definisi formal memaksa seseorang untuk berpikir tentang apa yang penting bagi masalah yang dihadapi dan sifat apa yang tidak relevan. Melalui proses tersebut sering kali terungkap seluk-beluk masalah yang tidak terlihat jelas pada pandangan pertama. Kami mengilustrasikannya selanjutnya untuk kasus enkripsi.

- ☞ Contoh: enkripsi aman. Kesalahan yang sering terjadi adalah berpikir bahwa definisi formal tidak diperlukan, atau tidak mudah untuk diberikan, karena "setiap orang memiliki gagasan intuitif tentang apa yang dimaksud dengan keamanan." Ini bukan kasusnya. Sebagai contoh, kami mempertimbangkan kasus enkripsi. (Pembaca mungkin ingin berhenti sejenak di sini untuk memikirkan bagaimana mereka akan secara formal mendefinisikan apa yang dimaksud dengan keamanan skema enkripsi.) Meskipun kami menunda definisi formal enkripsi aman ke dua bab berikutnya, di sini kami menjelaskan secara informal apa itu enkripsi aman. definisi harus menangkap.

Secara umum, definisi keamanan memiliki dua komponen: jaminan keamanan (atau, dari sudut pandang penyerang, apa yang dimaksud dengan keberhasilan serangan terhadap skema) dan model ancaman. Jaminan keamanan mendefinisikan skema apa yang dimaksudkan untuk mencegah penyerang melakukan hal tersebut, sedangkan model ancaman menggambarkan kekuatan musuh, yaitu tindakan apa yang diasumsikan dapat dilakukan oleh penyerang.

Mari kita mulai dengan yang pertama. Apa yang harus dijamin oleh skema enkripsi yang aman? Berikut beberapa pemikirannya: Seharusnya mustahil bagi penyerang untuk

memulihkan kuncinya. Kita telah mengamati sebelumnya bahwa jika seorang penyerang dapat menentukan kunci yang digunakan bersama oleh dua pihak menggunakan suatu skema, maka skema tersebut tidak aman. Namun, sangat mudah untuk membuat skema dimana pemulihan kunci tidak mungkin dilakukan, namun skema tersebut jelas-jelas tidak aman. Pertimbangkan, misalnya, skema dimana $\mathrm{Enc}_k(m) = m$. Ciphertext tidak membocorkan informasi tentang kunci tersebut (sehingga kunci tidak dapat dipulihkan jika cukup panjang) namun pesan terkirim dengan jelas! Oleh karena itu, kami melihat bahwa ketidakmampuan untuk memulihkan kunci tidak cukup untuk keamanan. Hal ini masuk akal: tujuan enkripsi adalah untuk melindungi pesan; kuncinya adalah sarana untuk mencapai hal ini, namun kuncinya tidak penting.

Seharusnya tidak mungkin bagi penyerang untuk memulihkan seluruh teks biasa dari teks tersandi. Definisi ini lebih baik, namun masih jauh dari memuaskan. Secara khusus, definisi ini akan menganggap skema enkripsi aman jika teks sandinya mengungkapkan 90% teks biasa, selama 10% teks biasa tetap sulit untuk diketahui. Hal ini jelas tidak dapat diterima di sebagian besar aplikasi enkripsi umum; misalnya, saat mengenkripsi database gaji, kami akan kecewa jika 90% gaji karyawan terungkap!

Seharusnya tidak mungkin bagi penyerang untuk memulihkan karakter teks biasa dari teks tersandi. Definisi ini tampaknya bagus, namun masih belum cukup. Kembali ke contoh mengenkripsi database gaji, kami tidak akan menganggap skema enkripsi aman jika skema tersebut mengungkapkan apakah gaji seorang karyawan lebih dari atau kurang dari Rp.10.000.000, bahkan jika skema tersebut tidak mengungkapkan digit tertentu dari gaji karyawan tersebut. Demikian pula, kami tidak ingin skema enkripsi mengungkapkan apakah karyawan A menghasilkan lebih banyak daripada karyawan B.

Masalah lainnya adalah bagaimana memformalkan apa artinya bagi musuh untuk "memulihkan karakter teks biasa." Bagaimana jika penyerang menebak dengan tepat, berdasarkan keberuntungan atau informasi eksternal, bahwa angka paling signifikan dari gaji seseorang adalah 0? Tentu saja hal ini tidak membuat skema enkripsi menjadi tidak aman, sehingga definisi apa pun yang masuk akal harus mengesampingkan perilaku tersebut sebagai serangan yang berhasil.

Jawaban yang "benar" adalah apapun informasi yang dimiliki penyerang, teks tersandi tidak boleh membocorkan informasi tambahan tentang teks biasa yang mendasarinya. Definisi informal ini mencakup seluruh permasalahan yang diuraikan di atas. Perhatikan secara khusus bahwa ini tidak mencoba untuk mendefinisikan informasi apa tentang teks biasa yang "bermakna"; itu hanya mengharuskan tidak ada informasi yang bocor. Hal ini penting, karena ini berarti bahwa skema enkripsi yang aman cocok untuk semua aplikasi potensial yang memerlukan kerahasiaan.

Apa yang hilang di sini adalah rumusan definisi matematis yang tepat. Bagaimana kita menangkap pengetahuan penyerang sebelumnya tentang teks biasa? Dan apa maksudnya (tidak) membocorkan informasi? Kita akan kembali ke pertanyaan-pertanyaan ini dalam dua bab berikutnya; lihat khususnya Definisi 2.3 dan 3.12. Sekarang kita telah menetapkan sasaran keamanan, tinggal menentukan model ancaman. Hal ini menentukan "kekuatan" apa yang

diasumsikan dimiliki oleh penyerang, namun tidak membatasi strategi musuh. Ini merupakan perbedaan yang penting: kita menentukan apa yang kita asumsikan tentang kemampuan musuh, namun kita tidak berasumsi apa pun tentang bagaimana musuh menggunakan kemampuan tersebut. Tidak mungkin untuk memperkirakan strategi apa yang mungkin digunakan dalam suatu serangan, dan sejarah telah membuktikan bahwa upaya untuk melakukan hal tersebut pasti akan gagal.Ada beberapa opsi yang masuk akal untuk model ancaman dalam konteks enkripsi; yang standar, dalam rangka meningkatkan kekuatan penyerang, adalah:

- **Serangan hanya ciphertext**: Ini adalah serangan paling dasar, dan mengacu pada skenario di mana musuh hanya mengamati ciphertext (atau beberapa ciphertext) dan mencoba untuk menentukan informasi tentang plaintext (atau plaintext) yang mendasarinya. Ini adalah model ancaman yang secara implisit kami asumsikan ketika membahas skema enkripsi klasik di bagian sebelumnya.
- **Serangan teks biasa yang dikenal**: Di sini, musuh dapat mempelajari satu atau lebih pasangan teks biasa/teks sandi yang dihasilkan menggunakan beberapa kunci. Tujuan dari musuh kemudian adalah untuk menyimpulkan informasi tentang teks biasa yang mendasari beberapa teks sandi lain yang dihasilkan menggunakan kunci yang sama. Semua skema enkripsi klasik yang telah kita lihat mudah dibobol menggunakan serangan teks biasa; kami meninggalkan demonstrasi sebagai latihan.
- **Serangan teks biasa terpilih**: Dalam serangan ini, musuh dapat memperoleh pasangan teks biasa/teks sandi (seperti di atas) untuk teks biasa pilihannya.
- **Serangan teks sandi terpilih**: Jenis serangan terakhir adalah serangan di mana musuh juga dapat memperoleh (beberapa informasi tentang) dekripsi teks sandi pilihannya, misalnya, apakah dekripsi beberapa teks sandi yang dipilih oleh penyerang menghasilkan pesan bahasa Inggris yang valid - Sage. Tujuan musuh, sekali lagi, adalah untuk mempelajari informasi tentang teks biasa yang mendasari beberapa teks tersandi lainnya (yang dekripsinya tidak dapat diperoleh secara langsung oleh musuh).

Tidak satu pun dari model ancaman ini yang secara inheren lebih baik dibandingkan model ancaman lainnya; yang tepat untuk digunakan bergantung pada lingkungan di mana skema enkripsi diterapkan. Dua jenis serangan pertama adalah yang paling mudah dilakukan. Dalam serangan ciphertext-only, satu-satunya hal yang perlu dilakukan musuh adalah menguping saluran komunikasi publik yang digunakan untuk mengirim pesan terenkripsi. Dalam serangan teks biasa yang diketahui, diasumsikan bahwa musuh juga mendapatkan teks sandi yang sesuai dengan teks biasa yang diketahui. Hal ini sering kali mudah dilakukan karena tidak semua pesan terenkripsi bersifat rahasia, setidaknya tidak selamanya. Sebagai contoh sederhana, dua pihak selalu mengenkripsi pesan “halo” setiap kali mereka mulai berkomunikasi. Sebagai contoh yang lebih kompleks, enkripsi dapat digunakan untuk menjaga kerahasiaan laporan pendapatan triwulanan hingga tanggal rilisnya; dalam hal ini, siapa pun yang menguping teks sandi nantinya akan mendapatkan teks biasa yang sesuai. Dalam dua serangan terakhir, musuh diasumsikan dapat memperoleh enkripsi dan/atau dekripsi teks biasa/teks sandi pilihannya. Hal ini mungkin tampak aneh pada awalnya, dan kami menunda

pembahasan lebih rinci mengenai serangan ini, dan kepraktisannya, ke Bagian 3.4 (untuk serangan teks biasa yang dipilih) dan Bagian 3.7 (untuk serangan teks sandi yang dipilih).

Prinsip 2 – Asumsi yang Tepat

Kebanyakan konstruksi kriptografi modern tidak dapat dibuktikan aman tanpa syarat; pembuktian seperti itu memerlukan penyelesaian pertanyaan-pertanyaan dalam teori kompleksitas komputasi yang tampaknya masih jauh dari terjawab saat ini. Akibat dari keadaan yang tidak menguntungkan ini adalah bukti keamanan biasanya bergantung pada asumsi. Kriptografi modern mengharuskan asumsi semacam itu dibuat secara eksplisit dan tepat secara matematis. Pada tingkat paling dasar, ini hanya karena bukti keamanan matematis memerlukan hal ini. Tapi ada alasan lain juga:

(1) **Validasi asumsi**: Berdasarkan sifatnya, asumsi adalah pernyataan yang tidak terbukti namun justru dianggap benar. Untuk memperkuat keyakinan kita terhadap suatu asumsi, maka asumsi tersebut perlu dipelajari. Semakin banyak asumsi tersebut diperiksa dan diuji tanpa dibantah, semakin yakin kita bahwa asumsi tersebut benar. Lebih jauh lagi, studi terhadap suatu asumsi dapat memberikan bukti validitasnya dengan menunjukkan bahwa asumsi tersebut tersirat dalam beberapa asumsi lain yang juga diyakini secara luas. Jika asumsi yang diandalkan tidak dinyatakan secara tepat, maka asumsi tersebut tidak dapat dipelajari dan (berpotensi) dibantah. Jadi, prasyarat untuk meningkatkan keyakinan kita terhadap suatu asumsi adalah memiliki pernyataan yang tepat tentang apa yang sebenarnya diasumsikan.

(2) **Perbandingan skema**: Seringkali dalam kriptografi kita dihadapkan pada dua skema yang keduanya dapat dibuktikan memenuhi beberapa definisi, masing-masing didasarkan pada asumsi yang berbeda. Dengan asumsi semua hal sama, skema manakah yang sebaiknya dipilih? Jika asumsi yang mendasari skema pertama lebih lemah dibandingkan asumsi yang mendasari skema kedua (yaitu asumsi kedua menyiratkan asumsi pertama), maka skema pertama lebih disukai karena bisa jadi asumsi kedua salah. padahal asumsi pertama benar. Jika asumsi yang digunakan oleh kedua skema tidak dapat dibandingkan, maka aturan umumnya adalah memilih skema yang didasarkan pada asumsi yang telah dipelajari dengan lebih baik dan memiliki keyakinan yang lebih besar.

(3) **Memahami asumsi-asumsi yang diperlukan**: Skema enkripsi mungkin didasarkan pada beberapa blok bangunan yang mendasarinya. Jika kemudian ditemukan kelemahan pada blok penyusunnya, bagaimana kita dapat mengetahui apakah skema enkripsi masih aman? Jika asumsi-asumsi mendasar mengenai landasan telah dibuat jelas sebagai bagian dari pembuktian keamanan skema, maka kita hanya perlu memeriksa apakah asumsi-asumsi yang diperlukan dipengaruhi oleh kelemahan-kelemahan baru yang ditemukan.

Pertanyaan yang terkadang muncul adalah: daripada membuktikan suatu skema aman berdasarkan asumsi lain, mengapa tidak berasumsi saja bahwa konstruksi itu sendiri aman? Dalam beberapa kasus misalnya, ketika sebuah skema telah berhasil menahan serangan selama bertahun-tahun hal ini mungkin merupakan pendekatan yang masuk akal. Namun

pendekatan ini tidak pernah disukai, dan sangat berbahaya jika ada skema baru yang diperkenalkan. Alasan di atas membantu menjelaskan alasannya. Pertama, asumsi yang telah diuji selama beberapa tahun lebih baik daripada asumsi baru yang bersifat ad hoc yang diperkenalkan bersamaan dengan konstruksi baru. Kedua, terdapat preferensi umum terhadap asumsi-asumsi yang lebih sederhana untuk dinyatakan, karena asumsi-asumsi tersebut lebih mudah untuk dipelajari dan (berpotensi) dibantah. Jadi, misalnya, asumsi bahwa suatu masalah matematika sulit dipecahkan lebih mudah dipelajari dan dievaluasi dibandingkan asumsi bahwa skema enkripsi memenuhi definisi keamanan yang kompleks. Keuntungan lain dari mengandalkan asumsi "tingkat yang lebih rendah" (daripada hanya mengasumsikan suatu konstruksi aman) adalah bahwa asumsi tingkat rendah ini biasanya dapat digunakan dalam konstruksi lain. Terakhir, asumsi tingkat rendah dapat memberikan modularitas. Pertimbangkan skema enkripsi yang keamanannya bergantung pada properti yang diasumsikan dari salah satu blok penyusunnya. Jika blok penyusun yang mendasarinya ternyata tidak memenuhi asumsi yang disebutkan, skema enkripsi masih dapat dibuat menggunakan komponen lain yang diyakini memenuhi persyaratan yang diperlukan.

Prinsip 3 – Bukti Keamanan

Dua prinsip yang dijelaskan di atas memungkinkan kami mencapai tujuan kami dalam memberikan bukti yang kuat bahwa suatu konstruksi memenuhi definisi tertentu berdasarkan asumsi tertentu. Bukti tersebut sangat penting dalam konteks kriptografi di mana terdapat penyerang yang secara aktif mencoba "mematahkan" suatu skema. Bukti keamanan memberikan jaminan yang kuat relatif terhadap definisi dan asumsi bahwa tidak ada penyerang yang akan berhasil; ini jauh lebih baik daripada mengambil pendekatan yang tidak berprinsip atau heuristik terhadap masalah tersebut. Tanpa bukti bahwa tidak ada musuh dengan sumber daya tertentu yang dapat mematahkan suatu skema, kita hanya mempunyai intuisi bahwa inilah masalahnya. Pengalaman menunjukkan bahwa intuisi dalam kriptografi dan keamanan komputer adalah sebuah bencana. Ada banyak sekali contoh skema yang belum terbukti namun gagal, kadang-kadang segera dan kadang-kadang bertahun-tahun setelah dikembangkan.

Ringkasan: Pendekatan Keamanan yang Ketat vs. Ad Hoc

Ketergantungan pada definisi, asumsi, dan bukti merupakan pendekatan ketat terhadap kriptografi yang berbeda dari pendekatan informal kriptografi klasik. Sayangnya, solusi-solusi yang tidak berprinsip dan bersifat "*on-the-cuff*" masih dirancang dan diterapkan oleh mereka yang ingin mendapatkan solusi cepat atas suatu masalah, atau oleh mereka yang tidak memiliki pengetahuan. Kami berharap buku ini dapat memberikan kontribusi pada kesadaran akan pendekatan yang ketat dan pentingnya pendekatan ini dalam mengembangkan skema yang terbukti aman.

Keamanan yang Dapat Dibuktikan dan Keamanan Dunia Nyata

Sebagian besar kriptografi modern kini bertumpu pada landasan matematika yang kuat. Namun bukan berarti bidang ini tidak lagi menjadi bagian dari seni. Pendekatan yang ketat memberikan ruang bagi kreativitas dalam mengembangkan definisi yang sesuai dengan aplikasi dan lingkungan kontemporer, dalam mengusulkan asumsi matematika baru atau

merancang primitif baru, dan dalam membangun skema baru dan membuktikan keamanannya. Tentu saja, akan selalu ada seni menyerang sistem kripto yang digunakan, bahkan jika sistem tersebut terbukti aman. Kami memperluas poin ini selanjutnya.

Pendekatan yang diambil oleh kriptografi modern telah merevolusi bidang ini, dan membantu memberikan kepercayaan terhadap keamanan skema kriptografi yang diterapkan di dunia nyata. Namun penting untuk tidak melebih-lebihkan arti dari bukti keamanan. Bukti keamanan selalu berhubungan dengan definisi yang dipertimbangkan dan asumsi yang digunakan. Jika jaminan keamanan tidak sesuai dengan kebutuhan, atau model ancaman tidak mampu menangkap kemampuan sebenarnya dari musuh, maka bukti yang ada mungkin tidak relevan. Begitu pula jika asumsi yang diandalkan ternyata salah, maka pembuktian keamanannya tidak ada artinya.

Kesimpulannya adalah bahwa keamanan yang dapat dibuktikan dari suatu skema tidak selalu berarti keamanan skema tersebut di dunia nyata.4 Meskipun beberapa orang memandang hal ini sebagai kelemahan dari keamanan yang dapat dibuktikan, kami memandang hal ini secara optimis sebagai gambaran kekuatan dari suatu skema. mendekati. Untuk menyerang skema yang terbukti aman di dunia nyata, cukup memusatkan perhatian pada definisinya (yaitu, mengeksplorasi bagaimana definisi yang diidealkan berbeda dari lingkungan dunia nyata di mana skema tersebut diterapkan) atau asumsi yang mendasarinya (yaitu, untuk lihat apakah mereka tahan). Pada gilirannya, tugas para kriptografer adalah terus menyempurnakan definisi mereka agar lebih sesuai dengan dunia nyata, dan menyelidiki asumsi-asumsi mereka untuk menguji validitasnya. Keamanan yang dapat dibuktikan tidak mengakhiri pertarungan lama antara penyerang dan pembela, namun memberikan kerangka kerja yang membantu mengubah peluang yang menguntungkan pihak bertahan.

BAB 2
ENKRIPSI RAHASIA SEMPURNA

Pada bab sebelumnya kami menyajikan skema enkripsi historis dan menunjukkan bagaimana skema tersebut dapat dipecahkan dengan sedikit usaha komputasi. Dalam bab ini, kita melihat skema enkripsi ekstrim lainnya dan mempelajari skema enkripsi yang terbukti aman bahkan terhadap musuh dengan kekuatan komputasi tak terbatas. Skema seperti ini disebut sangat rahasia. Selain mendefinisikan gagasan tersebut secara ketat, kami juga akan mengeksplorasi kondisi di mana kerahasiaan sempurna dapat dicapai.

Materi dalam bab ini, dalam arti tertentu, lebih berkaitan dengan dunia kriptografi "klasik" daripada dunia kriptografi "modern". Selain fakta bahwa semua materi yang diperkenalkan di sini dikembangkan sebelum revolusi kriptografi yang terjadi pada pertengahan tahun 1970an dan 1980an, konstruksi yang kita pelajari dalam bab ini hanya mengandalkan prinsip pertama dan ketiga yang diuraikan di Bagian 1.4. Artinya, definisi matematis yang tepat digunakan dan bukti yang kuat diberikan, namun tidak perlu bergantung pada asumsi komputasi yang belum terbukti. Jelas bermanfaat untuk menghindari asumsi-asumsi seperti itu; namun kita akan melihat bahwa melakukan hal tersebut memiliki keterbatasan yang melekat. Jadi, selain berfungsi sebagai dasar yang baik untuk memahami prinsip-prinsip yang mendasari kriptografi modern, hasil dari bab ini juga membenarkan penerapan ketiga prinsip yang disebutkan di atas.

Dimulai dengan bab ini, kita akan mendefinisikan skema keamanan dan menganalisis menggunakan eksperimen probabilistik yang melibatkan algoritma yang membuat pilihan acak; contoh dasar diberikan dengan pihak-pihak yang berkomunikasi memilih kunci acak. Jadi, sebelum kembali ke topik kriptografi, kita akan membahas secara singkat masalah menghasilkan keacakan yang cocok untuk aplikasi kriptografi. Sepanjang buku ini, kita hanya akan berasumsi bahwa pihak-pihak mempunyai akses terhadap pasokan bit acak yang independen dan tidak memihak dalam jumlah yang tidak terbatas. Dalam praktiknya, dari mana asal bit acak ini? Pada prinsipnya, seseorang dapat menghasilkan sejumlah kecil bit acak dengan tangan, misalnya dengan melempar koin. Namun pendekatan seperti ini sangat tidak nyaman dan tidak dapat diperluas skalanya.

Pembuatan bilangan acak modern berlangsung dalam dua langkah. Pertama, "kumpulan" data dengan entropi tinggi dikumpulkan. (Untuk tujuan kita, definisi formal entropi tidak diperlukan, dan cukup menganggap entropi sebagai ukuran ketidakpastian.) Selanjutnya, data dengan entropi tinggi ini diproses untuk menghasilkan rangkaian bit yang hampir independen dan tidak bias. Langkah kedua ini diperlukan karena data dengan entropi tinggi belum tentu seragam.

Untuk langkah pertama, diperlukan sumber data yang tidak dapat diprediksi. Ada beberapa cara untuk memperoleh data tersebut. Salah satu tekniknya adalah dengan mengandalkan input eksternal, misalnya penundaan antar kejadian jaringan, waktu akses hard-disk, penekanan tombol atau pergerakan mouse yang dilakukan oleh pengguna, dan

sebagainya. Data tersebut kemungkinan besar tidak seragam, namun jika pengukuran yang dilakukan cukup, kumpulan data yang dihasilkan diharapkan memiliki entropi yang memadai. Pendekatan yang lebih canggih yang dirancang dengan menggabungkan pembangkitan angka acak secara lebih erat ke dalam sistem pada tingkat perangkat keras juga telah digunakan. Hal ini bergantung pada fenomena fisik seperti kebisingan termal/tembakan atau peluruhan radioaktif. Intel baru-baru ini mengembangkan prosesor yang menyertakan generator angka acak digital pada chip prosesor dan menyediakan instruksi khusus untuk mengakses bit acak yang dihasilkan (setelah bit tersebut diproses dengan tepat untuk menghasilkan bit yang independen dan tidak bias, seperti yang akan dibahas selanjutnya).

Pemrosesan yang diperlukan untuk "memuluskan" data dengan entropi tinggi untuk mendapatkan bit (yang hampir) seragam adalah proses yang tidak sepele. Di sini kami hanya memberikan contoh sederhana untuk memberikan gambaran tentang apa yang dilakukan. Bayangkan kumpulan entropi tinggi kita dihasilkan dari serangkaian pelemparan koin yang bias, di mana "kepala" muncul dengan probabilitas p dan "ekor" dengan probabilitas 1 p. (Namun, kami berasumsi bahwa hasil lemparan koin tidak bergantung pada semua lemparan koin lainnya. Dalam praktiknya, asumsi ini biasanya tidak valid.) Hasil dari 1.000 lemparan koin tentunya memiliki entropi yang tinggi, namun tidak mendekati seragam. Kita dapat memperoleh distribusi yang seragam dengan mempertimbangkan pelemparan koin secara berpasangan: jika kita melihat kepala diikuti oleh ekor maka kita menghasilkan "0", dan jika kita melihat ekor diikuti oleh kepala maka kita menghasilkan "1". (Jika kita melihat dua kepala atau dua ekor berturut-turut, kita tidak menghasilkan apa-apa, dan langsung beralih ke pasangan berikutnya.) Probabilitas bahwa setiap pasangan menghasilkan "0" adalah p (1 p), yang sama persis dengan probabilitas bahwa setiap pasangan menghasilkan "1", dan dengan demikian kita memperoleh keluaran yang terdistribusi secara merata dari kumpulan entropi tinggi awal kita.

Cara bit acak dihasilkan harus hati-hati, dan penggunaan generator angka acak yang buruk sering kali membuat sistem kriptografi yang baik rentan terhadap serangan. Seseorang harus menggunakan generator angka acak yang dirancang untuk penggunaan kriptografi, daripada generator angka acak "tujuan umum", yang tidak cocok untuk aplikasi kriptografi. Secara khusus, fungsi rand() di pustaka C stdlib.h tidak aman secara kriptografis, dan menggunakannya dalam pengaturan kriptografi dapat menimbulkan konsekuensi yang berbahaya.

2.1 DEFINISI

Kita mulai dengan mengingat dan memperluas sintaksis yang telah diperkenalkan pada bab sebelumnya. Skema enkripsi ditentukan oleh tiga algoritma Gen, Enc, dan Dec, serta spesifikasi ruang pesan (terbatas) dengan $\|M\| > 1^1$ Algoritma pembangkitan kunci Gen adalah algoritma probabilistik yang mengeluarkan kunci k yang dipilih berdasarkan beberapa distribusi. Kita menyatakan dengan K ruang kunci (yang terbatas), yaitu himpunan semua kunci yang mungkin yang dapat dihasilkan oleh Gen. Algoritma enkripsi Enc mengambil kunci $k \in K$ dan pesan $m \in M$ sebagai masukan, dan menghasilkan teks sandi c . Kami sekarang

mengizinkan algoritma enkripsi menjadi probabilistik (sehingga $\mathrm{Enc}_k(m)$ mungkin mengeluarkan ciphertext yang berbeda ketika dijalankan beberapa kali), dan kami menulis c $\mathrm{Enc}_k(m)$ untuk menunjukkan kemungkinan proses probabilistik dimana pesan m dienkripsi menggunakan kunci k untuk memberikan cipherteks c. (Jika Enc bersifat deterministik, kita dapat menekankan hal ini dengan menulis $c := Enc_k(m)$. Ke depan, terkadang kita juga menggunakan notasi $x \rightarrow S$ untuk menunjukkan pemilihan x yang seragam dari himpunan S.) Kita misalkan C menyatakan himpunan semua kemungkinan cipherteks yang dapat dihasilkan oleh $\mathrm{Enc}_k(m)$, untuk semua kemungkinan pilihan $k \in K$ dan $m \in M$ (dan untuk semua pilihan acak Enc jika diacak). Algoritma dekripsi Dec mengambil masukan berupa kunci $k \in K$ dan teks tersandi $c \in C$ dan mengeluarkan pesan $m \in M$. Kita asumsikan kebenarannya sempurna, artinya untuk semua $k \in K, m \in M$, dan teks tersandi apa pun yang dihasilkan oleh $\mathrm{Enc}_k(m)$, menyatakan bahwa $\mathrm{Dec}_k(c) = m$ dengan probabilitas 1. Ketepatan sempurna menyiratkan bahwa kita dapat berasumsi bahwa Des bersifat deterministik tanpa kehilangan keumumannya, karena $\mathrm{Dec}_k(c)$ harus memberikan keluaran yang sama setiap kali dijalankan. Kita kemudian akan menulis $m := \mathrm{Dec}_k(c)$ untuk menunjukkan proses mendekripsi ciphertext c menggunakan kunci k untuk menghasilkan pesan m.

Dalam definisi dan teorema di bawah, kita mengacu pada distribusi probabilitas pada K, M, dan C. Distribusi berakhir adalah distribusi yang ditentukan dengan menjalankan Gen dan mengambil hasilnya. (Hampir selalu terjadi bahwa Gen memilih sebuah kunci secara seragam dan, pada kenyataannya, kita dapat mengasumsikan hal ini tanpa kehilangan keumumannya). Kita membiarkan K menjadi variabel acak yang menunjukkan nilai keluaran kunci oleh Gen; jadi, untuk $k \in K$ apa pun, $\Pr[K = k]$ menunjukkan probabilitas bahwa keluaran kunci oleh Gen sama dengan k. Demikian pula, kita membiarkan M menjadi variabel acak yang menunjukkan pesan yang dienkripsi, jadi $\Pr[M = m]$ menunjukkan probabilitas bahwa pesan tersebut mengambil nilai m . Distribusi probabilitas pesan tidak ditentukan oleh skema enkripsi itu sendiri, namun mencerminkan kemungkinan pesan berbeda yang dikirim oleh pihak-pihak yang menggunakan skema tersebut, serta ketidakpastian musuh tentang apa yang akan dikirim. Sebagai contoh, musuh mungkin mengetahui bahwa pesannya akan diserang hari ini atau tidak akan diserang. Musuh bahkan mungkin mengetahui (dengan cara lain) bahwa dengan probabilitas 0,7 pesan tersebut akan menjadi perintah untuk menyerang dan dengan probabilitas 0,3 pesan tersebut akan menjadi perintah untuk tidak menyerang. Dalam kasus ini, kita mempunyai Pr[M = serang hari ini] = 0,7 dan $\Pr[M = \text{jangan serang}] = 0,3$.

K dan M diasumsikan independen, yaitu apa yang dikomunikasikan oleh para pihak tidak bergantung pada kunci yang mereka bagikan. Hal ini masuk akal, antara lain, karena distribusi pada K ditentukan oleh skema enkripsi itu sendiri (karena ditentukan oleh Gen), sedangkan distribusi pada M bergantung pada konteks di mana skema enkripsi tersebut digunakan. Memperbaiki skema enkripsi dan distribusi pada M menentukan distribusi pada ruang cipherteks C yang diberikan dengan memilih kunci $k \in K$ (menurut Gen) dan pesan $m \in M$ (menurut distribusi yang diberikan), dan kemudian menghitung ciphertext $c \leftarrow$

$Enc_k(m)$. Kita misalkan C menjadi variabel acak yang menunjukkan ciphertext yang dihasilkan dan karenanya, untuk $c \in C$, tulis $\Pr[C = c]$ untuk menunjukkan probabilitas bahwa ciphertext sama dengan nilai tetap c.

Contoh 2.1

Kita mengerjakan contoh sederhana untuk shift cipher (lih. Bagian 1.3). Di sini, menurut definisi, kita mempunyai $K = \{0, \ldots, 25\}$ dengan $Pr[K = k] = 1/26$ untuk setiap $k \in K$. Katakanlah kita diberikan distribusi berikut pada M:

$$\Pr[\mathrm{M} = \mathrm{a}] = 0.7 \text{ dan } \Pr[\mathrm{M} = \mathrm{z}] = 0.3.$$

Berapa probabilitas bahwa ciphertextnya adalah B? Hanya ada dua cara terjadinya hal ini: $M = a \; dan \; K = 1, atau \; M = z \; dan \; K = 2$. Dengan independensi M dan K, kita peroleh

$$\Pr[\mathrm{M} = \mathrm{a} \wedge \mathrm{K} = 1] = \Pr[\mathrm{M} = \mathrm{a}] \,.\, \Pr[\mathrm{K} = 1] = 0.7.\left(\frac{1}{26}\right).$$

Demikian pula,

$$\Pr[\mathrm{M} = \mathrm{z} \wedge \mathrm{K} = 2] = 0.3.\left(\frac{1}{26}\right).$$

Karena itu,

$$\Pr[\mathrm{C} = \mathrm{B}] = \Pr[\mathrm{M} = \mathrm{a} \wedge \mathrm{K} = 1] + \Pr\,[\mathrm{M} = \mathrm{z} \wedge \mathrm{K} = 2]$$

Kita juga dapat menghitung probabilitas bersyarat. Misalnya, berapa probabilitas pesan a dienkripsi, mengingat kita mengamati teks tersandi B? Menggunakan Teorema Bayes (Teorema A.8) yang kita miliki

$$\Pr[\mathrm{M} = \mathrm{a}|\mathrm{C} = \mathrm{B}] = \frac{\Pr[\mathrm{C} = \mathrm{B}|\mathrm{M} = \mathrm{a}] \,.\, \Pr\,[\mathrm{M} = \mathrm{a}]}{\Pr\,[\mathrm{C} = \mathrm{B}]} = \frac{0.7.\Pr\,[\mathrm{C} = \mathrm{B}|\mathrm{M} = \mathrm{a}]}{1/2}$$

Perhatikan bahwa $Pr[C = B \,|M = a] = 1/26$, karena jika $M = a$ maka satu-satunya cara $C = B$ dapat terjadi adalah jika $K = 1$ (yang terjadi dengan probabilitas 1/26). Kita menyimpulkan bahwa $Pr[M = a \mid C = B] = 0{,}7$.

Contoh 2.2

Pertimbangkan shift cipher lagi, tetapi dengan distribusi berikut pada M:

$$\Pr[\mathrm{M} = \mathrm{kom}] = 0.5, \Pr[\mathrm{M} = \mathrm{ann}] = 0.2, \Pr[\mathrm{M} = \mathrm{boo}] = 0.3.$$

Berapa peluang terambilnya `C = DQQ`? Satu-satunya cara ciphertext ini dapat terjadi adalah jika `M = ann` dan `K = 3`, atau `M = boo` dan `K = 2`, yang terjadi dengan probabilitas `0.2 1/26 + 0.3 1/26 = 1/52.`

Jadi berapa probabilitas bahwa ann dienkripsi, dengan syarat mengamati DQQ ciphertext? Perhitungan seperti di atas menggunakan Teorema Bayes menghasilkan `Pr[M = ann | C = DQQ] = 0,4.`

Kerahasiaan yang sempurna. Kami sekarang siap untuk mendefinisikan gagasan kerahasiaan sempurna. Kita membayangkan musuh yang mengetahui distribusi probabilitas; artinya, musuh mengetahui kemungkinan pesan yang berbeda akan dikirim. Musuh ini juga mengetahui skema enkripsi yang digunakan; satu-satunya hal yang tidak diketahui musuh adalah kunci yang dibagikan oleh para pihak. Sebuah pesan dipilih oleh salah satu pihak yang jujur dan dienkripsi, dan teks sandi yang dihasilkan dikirimkan ke pihak lain.

Musuh dapat menguping komunikasi para pihak, dan dengan demikian mengamati teks sandi ini. (Artinya, ini adalah serangan ciphertext-only, dimana penyerang hanya mendapatkan satu ciphertext.) Agar suatu skema benar-benar rahasia, pengamatan ciphertext ini seharusnya tidak berpengaruh pada pengetahuan musuh mengenai pesan sebenarnya yang dikirimkan; dengan kata lain, probabilitas a posteriori bahwa suatu pesan m terkirim, dikondisikan pada ciphertext yang diamati, seharusnya tidak berbeda dengan probabilitas a priori bahwa m akan terkirim. Artinya, teks tersandi tidak mengungkapkan apa pun tentang teks biasa yang mendasarinya, dan musuh sama sekali tidak mengetahui apa pun tentang teks biasa yang dienkripsi. Secara formal:

DEFINISI 2.3 Skema enkripsi (Gen, Enc, Des) dengan ruang pesan M benar-benar rahasia jika untuk setiap distribusi probabilitas pada M, setiap pesan $m \in M$, dan setiap ciphertext $c \in C$ yang mana $Pr[C = c] > 0$:

$$\Pr[\mathrm{M} = \mathrm{m} | \mathrm{C} = \mathrm{c}] = \Pr[\mathrm{M} = \mathrm{m}].$$

(Persyaratan bahwa $Pr[C = c] > 0$ adalah persyaratan teknis yang diperlukan untuk mencegah pengkondisian pada kejadian dengan probabilitas nol.)

Kami sekarang memberikan rumusan yang setara dengan kerahasiaan sempurna. Secara informal, rumusan ini mensyaratkan bahwa distribusi probabilitas teks tersandi tidak bergantung pada teks biasa, yaitu, untuk dua pesan m, m' distribusi teks tersandi ketika m dienkripsi harus sama dengan distribusi teks tersandi. teks tersandi ketika m' dienkripsi. Secara formal, untuk setiap $m, m' \in M$, dan setiap $c \in C$,

$$\Pr[\mathrm{Enc}_K(\mathrm{m}) = \mathrm{c}] = \Pr[\mathrm{Enc}_K(\mathrm{m}') = \mathrm{c}] \qquad (2.1)$$

(di mana probabilitasnya melebihi pilihan K dan keacakan Enc). Hal ini menyiratkan bahwa ciphertext tidak mengandung informasi tentang plaintext, dan tidak mungkin membedakan enkripsi m dari enkripsi m', karena distribusi pada ciphertext adalah sama dalam setiap kasus.

LEMMA 2.4 Skema enkripsi (Gen, Enc, Des) dengan ruang pesan M benar-benar rahasia jika dan hanya jika Persamaan (2.1) berlaku untuk setiap $m, m' \in M$ dan setiap $c \in C$.

BUKTI Kami menunjukkan bahwa jika kondisi yang dinyatakan terpenuhi, maka skema tersebut benar-benar rahasia; implikasi sebaliknya diserahkan pada Latihan 2.4. Perbaiki

distribusi pada , pesan m, dan teks sandi c yang mana $Pr[C = c] > 0$. Jika $Pr[M = m] = 0$ maka kita punya

$$\Pr[\mathrm{M} = \mathrm{m}|\mathrm{C} = \mathrm{c}] = 0 = \Pr[\mathrm{M} = \mathrm{m}]$$

Jadi, asumsikan $Pr[M = m] > 0$. Perhatikan dulu hal itu

$$\Pr[\mathrm{C} = \mathrm{c}|\mathrm{M} = \mathrm{m}] = \Pr[\Pr[\mathrm{Enc}_K(\mathrm{M}) = \mathrm{c}|\mathrm{M} = \mathrm{m}] = \Pr[\mathrm{Enc}_K(\mathrm{m}) = \mathrm{c}],$$

dimana persamaan pertama menurut definisi variabel acak C, dan persamaan kedua karena kita mengkondisikan kejadian M sama dengan m. Himpunan $\delta_d \stackrel{\text{def}}{=} \Pr[\mathrm{Enc}_{\mathrm{k}}(m) = c] = \Pr[C = c \mid M = m]$. Jika kondisi lemma berlaku, maka untuk setiap m' kita mempunyai $\Pr[\mathrm{Enc}_K(m') = c] = \Pr[C = c \mid M = m'] = \delta_c$. Dengan menggunakan Teorema Bayes (lihat Lampiran A.3), kita mendapatkan

$$\begin{aligned}\Pr[\mathrm{M} = \mathrm{m}|\mathrm{C} = \mathrm{c}] &= \frac{\Pr[\mathrm{C} = \mathrm{c}|\mathrm{M} = \mathrm{m}] \,.\, \Pr[\mathrm{M} = \mathrm{m}]}{\Pr[\mathrm{C} = \mathrm{c}]} \\ &= \frac{\Pr[\mathrm{C} = \mathrm{c}|\mathrm{M} = \mathrm{m}] \,.\, \Pr[\mathrm{M} = \mathrm{m}]}{\sum_{\mathrm{m}' \in \mathcal{M}} \Pr[\mathrm{C} = \mathrm{c}|\mathrm{M} = \mathrm{m}'] \,.\, \Pr[\mathrm{M} = \mathrm{m}']} = \frac{\delta_{\mathrm{c}} . \Pr[\mathrm{M} = \mathrm{m}']}{\sum_{\mathrm{m}' \in \mathcal{M}} \delta_{\mathrm{c}} \Pr[\mathrm{M} = \mathrm{m}']} \\ &= \frac{\Pr[\mathrm{M} = \mathrm{m}]}{\sum_{\mathrm{m}' \in \mathcal{M}} \delta_{\mathrm{c}} \Pr[\mathrm{M} = \mathrm{m}']} = \Pr[\mathrm{M} = \mathrm{m}],\end{aligned}$$

dimana penjumlahannya selesai $m' \in M$ dengan $\Pr[M = m'] \neq 0$. Kita simpulkan bahwa untuk setiap $m \in M$ dan $c \in C$ dimana $\Pr[C = c] > 0$, maka $Pr[M = m \mid C = c] = Pr[M = m]$, sehingga skema ini sangat rahasia.

Ketidakmampuan membedakan yang sempurna (bermusuhan). Kami menyimpulkan bagian ini dengan menyajikan definisi lain yang setara tentang kerahasiaan sempurna. Definisi ini didasarkan pada eksperimen yang melibatkan musuh yang secara pasif mengamati teks sandi dan kemudian mencoba menebak pesan mana yang mungkin dienkripsi. Kami memperkenalkan gagasan ini karena ini akan menjadi titik awal untuk mendefinisikan keamanan komputasi pada bab berikutnya. Memang benar, sepanjang sisa buku ini kita akan sering menggunakan eksperimen semacam ini untuk mendefinisikan keamanan.

Dalam konteks ini, kami mempertimbangkan percobaan berikut: musuh A pertama-tama menetapkan dua pesan arbitrer $m_0, m_1 \in M$. Salah satu dari dua pesan ini dipilih secara seragam secara acak dan dienkripsi menggunakan kunci acak; teks sandi yang dihasilkan diberikan kepada A. Terakhir, menghasilkan "tebakan" pesan mana yang dienkripsi; berhasil jika tebakannya benar. Skema enkripsi tidak dapat dibedakan jika tidak ada musuh A yang dapat berhasil dengan probabilitas lebih baik dari 1/2. (Perhatikan bahwa, untuk skema enkripsi apa pun, dapat berhasil dengan probabilitas 1/2 dengan mengeluarkan tebakan yang seragam; syaratnya adalah tidak ada penyerang yang dapat melakukan lebih baik dari ini.)

Kami menekankan bahwa tidak ada batasan yang ditempatkan pada kekuatan komputasi dari A.

Secara formal, misalkan Π = (Gen, Enc, Dec) adalah skema enkripsi dengan ruang pesan M. Misalkan A adalah musuh, yang secara formal hanya merupakan algoritma (stateful).
Kami mendefinisikan eksperimen $\mathrm{PrivK}_{\mathrm{A},\Pi}$ sebagai berikut:

- Eksperimen ketidakmungkinan membedakan $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi}$:
  1. Musuh A mengeluarkan sepasang pesan $m_0, m_1 \in M$.
  2. Kunci k dihasilkan menggunakan Gen, dan bit seragam $b \in \{0,1\}$ dipilih. Ciphertext $c \rightarrow Enc_k(m_b)$ dihitung dan diberikan kepada A. Kita menyebut c sebagai tantangan ciphertext.
  3. A mengeluarkan sedikit b'.
  4. Keluaran percobaan didefinisikan 1 jika $b' = b$,
  dan 0 sebaliknya. Kita tuliskan $PrivK^{eav}_{\mathcal{A},\Pi}$= 1 jika keluaran percobaannya adalah 1 dan dalam hal ini kita katakan A berhasil.

Seperti disebutkan sebelumnya, adalah hal yang mudah untuk berhasil dengan probabilitas 1/2 dengan mengeluarkan tebakan acak. Ketidakmampuan membedakan yang sempurna mensyaratkan bahwa mustahil bagi A untuk berbuat lebih baik.

DEFINISI 2.5 Skema enkripsi Π = (Gen, Enc, Des) dengan ruang pesan M tidak dapat dibedakan jika untuk setiap A dinyatakan bahwa

$$Pr[PrivK^{eav}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$$

Lemma berikut menyatakan bahwa Definisi 2.5 setara dengan Definisi 2.3.

LEMMA 2.6 Skema enkripsi Π sangat rahasia jika dan hanya jika skema tersebut tidak dapat dibedakan secara sempurna.

Contoh 2.7

Kami menunjukkan bahwa sandi Vigen`ere tidak dapat dibedakan secara sempurna, setidaknya untuk parameter tertentu. Secara konkret, misalkan $\Pi$ menunjukkan sandi Vigen`ere untuk ruang pesan string dua karakter, dan periode dipilih secara seragam dalam `{1, 2}`. Untuk menunjukkan bahwa Π tidak dapat dibedakan secara sempurna, kami menunjukkan $\mathrm{Pr}[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi} = 1] > \frac{1}{2}$

Musuh A melakukan:

1. Keluaran $m_0 = aa$ dan $m_1 = ab$.
2. Setelah menerima tantangan ciphertext $c = c_1c_2$, lakukan hal berikut: `if` $c_1 = c_2$ `output 0`; lain keluaran `1`.

Perhitungan $\mathrm{Pr}[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\Pi} = 1]$membosankan tetapi mudah.

$$\begin{aligned}\Pr[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\prod} = 1 &= \frac{1}{2}.\Pr\left[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\prod} = 1 \middle| \mathrm{b}\ 0\right] + \frac{1}{2}.\Pr\left[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A}\prod} = 1 \middle| \mathrm{b} = 1\right] \\ &= \frac{1}{2}.\Pr[\mathcal{A}\ \text{outputs}\ 0|\mathrm{b} = 0] + \frac{1}{2}.\Pr\ [\mathcal{A}\text{outputs}\ 1|\mathrm{b} = 1 \\ &= \frac{1}{2}.\Pr[\mathcal{A}\ \text{outputs}\ 0|\mathrm{b} = 0] + \frac{1}{2}.\Pr[\mathcal{A}\ \text{outputs}\ 1|\mathrm{b} = 1], \qquad (2{,}2)\end{aligned}$$

dimana b adalah bit seragam yang menentukan pesan mana yang akan dienkripsi. A menghasilkan 0 jika dan hanya jika dua karakter $ciphertext\ c = c_1c_2$ sama. Ketika $b = 0$ (jadi `m0 = aa` dienkripsi) maka $c_1 = c_2$ jika salah satu (1) kunci periode 1 dipilih, atau (2) kunci periode 2 dipilih, dan kedua karakter kunci tersebut sama. Yang pertama terjadi dengan probabilitas 1 , dan yang kedua terjadi dengan probabilitas $\frac{1}{2}.\frac{1}{26}$.

Jadi

$$\Pr[\mathcal{A}\ outputs\ 1|b = 1] = 1 - \Pr[\mathcal{A}\ outputs\ 0|b = 1] = 1 - \frac{1}{2}.\frac{1}{26} \approx 0.98$$

Memasukkan ke Persamaan (2.2) lalu memberikan

$$\Pr\left[\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A}\prod} = 1\right] = \frac{1}{2}.\left(\frac{1}{2} + \frac{1}{2}.\frac{1}{26} + 1 - \frac{1}{2}.\frac{1}{26}\right) = 0.75 > \frac{1}{2},$$

dan skema ini tidak bisa dibedakan secara sempurna.

2.2 PAPAN SEKALI PAKAI

Pada tahun 1917, Vernam mematenkan skema enkripsi rahasia yang sekarang disebut one-time pad. Pada saat Vernam mengusulkan skema tersebut, tidak ada bukti bahwa skema tersebut benar-benar rahasia; pada kenyataannya, belum ada gagasan mengenai apa itu kerahasiaan yang sempurna. Namun, sekitar 25 tahun kemudian, Shannon memperkenalkan definisi kerahasiaan sempurna dan menunjukkan bahwa one-time pad mencapai tingkat keamanan tersebut.

Dalam menjelaskan skema ini kita membiarkan `a` $\oplus$ `b` menunjukkan bitwise eksklusif-atau (`XOR`) dari dua string biner a dan b (yaitu, jika $a = a_1 \ldots a_\ell$ dan $b = b_1 \cdots b_\ell$ adalah string l-bit, maka $a \oplus b$ adalah string ℓ -bit yang diberikan oleh $a_1 \oplus b_1$, $a_\ell \oplus b_\ell$). Dalam skema enkripsi one-time pad, kuncinya adalah string seragam yang panjangnya sama dengan pesan; ciphertext dihitung hanya dengan melakukan `XOR` pada kunci dan pesan. Definisi formal diberikan sebagai Konstruksi 2.8. Sebelum membahas keamanan, pertama-tama kita verifikasi kebenarannya: untuk setiap kunci k dan setiap pesan m yang berisi $\mathrm{Dec}_k\left(\mathrm{Enc}_k(m)\right) = k \oplus k \oplus m = m$, sehingga one-time pad merupakan skema enkripsi yang valid.

Seseorang dapat dengan mudah membuktikan kerahasiaan sempurna dari one-time pad menggunakan Lemma 2.4 dan fakta bahwa ciphertext didistribusikan secara seragam

terlepas dari pesan apa yang dienkripsi. Kami memberikan bukti berdasarkan langsung pada definisi aslinya.

TEOREMA 2.9 Skema enkripsi one-time pad benar-benar rahasia.
BUKTI Pertama-tama kita hitung $Pr[C = c \mid M = m']$ untuk sembarang $c \in C$ dan $m' \in$ M. Untuk pad satu kali,

$$\Pr[C = c|M = m'] = \Pr[Enc_K(m') = c] = \Pr[m' \oplus K = c] = \Pr[K = m' \oplus c] = 2^{-\ell},$$

dimana persamaan akhir berlaku karena kunci K adalah string ℓ-bit yang seragam. Perbaiki distribusi apa pun di M. Untuk $c \in C$ apa pun, kita punya

$$\Pr[C = c] = \sum_{m' \in \mathcal{M}} \Pr[C = c|M = m'] \,.\, \Pr[M = m'] = 2^{-\ell}. \sum_{m' \in \mathcal{M}} \Pr[M = m'] = 2^{-\ell},$$

dimana jumlahnya melebihi $m' \in M$ dengan $Pr[M = m'] \neq 0$. Teorema Bayes menghasilkan:

$$\Pr[C = c] = \sum_{m' \in \mathcal{M}} \Pr[C = c|M = m'] \,.\, \Pr[M = m'] = 2^{-\ell}. \sum_{m' \in \mathcal{M}} \Pr[M = m'] = 2^{-\ell}$$

Kami menyimpulkan bahwa catatan sekali pakai itu benar-benar rahasia.

One-time pad digunakan oleh beberapa badan intelijen nasional pada pertengahan abad ke-20 untuk mengenkripsi lalu lintas sensitif. Mungkin yang paling terkenal adalah "telepon merah" yang menghubungkan Gedung Putih dan Kremlin selama Perang Dingin dilindungi menggunakan enkripsi one-time pad, di mana pemerintah AS dan Uni Soviet akan bertukar kunci yang sangat panjang menggunakan kurir tepercaya yang membawa tas kerja berisi kertas. karakter acak mana yang ditulis.

Sekalipun demikian, enkripsi one-time pad jarang digunakan lagi karena sejumlah kelemahan yang dimilikinya. Yang paling menonjol adalah bahwa kuncinya adalah sepanjang pesannya. Hal ini membatasi kegunaan skema untuk mengirimkan pesan yang sangat panjang (karena mungkin sulit untuk berbagi dan menyimpan kunci yang sangat panjang dengan aman), dan menjadi masalah jika para pihak tidak dapat melakukannya. memprediksi sebelumnya (batas atas) berapa panjang pesan tersebut.

Terlebih lagi, one-time pad seperti namanya hanya aman jika digunakan satu kali (dengan kunci yang sama). Meskipun kami belum mendefinisikan pengertian kerahasiaan ketika beberapa pesan dienkripsi, mudah untuk melihat bahwa mengenkripsi lebih dari satu pesan dengan kunci yang sama akan membocorkan banyak informasi. Secara khusus, katakanlah dua pesan m, m' dienkripsi menggunakan kunci k yang sama. Musuh yang memperoleh $c = m \oplus k$ dan $c' = m' \oplus k$ dapat menghitung

$$c \oplus c' = (m \oplus k) \oplus (m' \oplus k) = m \oplus m'$$

dan dengan demikian mempelajari eksklusivitas-atau dari dua pesan tersebut atau, secara ekuivalen, di mana tepatnya kedua pesan tersebut berbeda. Meskipun hal ini mungkin tidak tampak terlalu signifikan, hal ini cukup untuk mengesampingkan klaim kerahasiaan sempurna untuk mengenkripsi dua pesan menggunakan kunci yang sama. Terlebih lagi, jika pesan-pesan tersebut sesuai dengan teks bahasa alami, maka dengan adanya dua pesan eksklusif atau cukup panjang maka analisis frekuensi dapat dilakukan (seperti pada bab sebelumnya, meskipun lebih kompleks) dan memulihkan pesan-pesan itu sendiri. Sebuah contoh sejarah yang menarik mengenai hal ini diberikan oleh proyek VENONA, dimana AS dan Inggris mampu mendekripsi ciphertext yang dikirim oleh Uni Soviet yang secara keliru dienkripsi dengan bagian berulang dari one-time pad selama beberapa dekade.

2.3 BATASAN KERAHASIAAN SEMPURNA

Kami mengakhiri bagian sebelumnya dengan mencatat beberapa kelemahan skema enkripsi one-time pad. Di sini, kami menunjukkan bahwa kelemahan ini tidak spesifik pada skema tersebut, namun merupakan keterbatasan yang melekat pada kerahasiaan yang sempurna. Secara khusus, kami membuktikan bahwa setiap skema enkripsi yang sangat rahasia harus memiliki ruang kunci yang setidaknya sama besarnya dengan ruang pesan. Jika semua kunci memiliki panjang yang sama, dan ruang pesan terdiri dari semua string dengan panjang tertentu, ini berarti bahwa kunci tersebut setidaknya sepanjang pesan. Secara khusus, panjang kunci dari one-time pad sudah optimal. (Keterbatasan lainnya—yakni, bahwa kunci hanya dapat digunakan satu kali juga melekat jika diperlukan kerahasiaan yang sempurna; lihat Latihan 2.13.)

TEOREMA 2.10 Jika (Gen, Enc, Des) adalah skema enkripsi rahasia sempurna dengan ruang pesan M dan ruang kunci K, maka $|K| \geq |M|$.

BUKTI Kami menunjukkan bahwa jika maka $|K| < |M|$ maka skema tersebut tidak dapat dirahasiakan sepenuhnya. Asumsikan maka $|K| < |M|$. Pertimbangkan distribusi seragam dan misalkan c adalah teks tersandi yang muncul dengan probabilitas bukan nol. Misalkan (c) adalah himpunan semua pesan yang mungkin dapat didekripsi dari c; itu adalah

$$\mathcal{M}(c) \stackrel{\text{def}}{=} \{m | m = Dec_K(c) for\ some\ k \in K\}$$

Jelas $|M(c)| \leq |K|$. (Ingatlah bahwa kita mungkin menganggap Dec bersifat deterministik.) Jika $|K| < |M|$, ada beberapa $m' \in M$ sehingga $m' \notin M(c)$. Tapi kemudian

$$\Pr[M = m' | C = c] = 0 \neq \Pr[M = m'],$$

jadi skema ini tidak sepenuhnya rahasia.

Kerahasiaan sempurna dengan kunci yang lebih pendek? Teorema di atas menunjukkan keterbatasan yang melekat pada skema yang mencapai kerahasiaan sempurna. Meski begitu, ada kalanya individu mengklaim bahwa mereka telah mengembangkan skema enkripsi baru yang "tidak dapat dipecahkan" dan mencapai keamanan one-time pad tanpa menggunakan kunci selama data tersebut dienkripsi. Bukti di atas menunjukkan bahwa klaim tersebut tidak mungkin benar; siapa pun yang membuat klaim seperti itu hanya tahu sedikit tentang kriptografi atau berbohong secara terang-terangan.

2.4 TEOREMA SHANNON

Dalam karyanya tentang kerahasiaan sempurna, Shannon juga memberikan karakterisasi skema enkripsi yang sangat rahasia. Karakterisasi ini mengatakan bahwa, dalam kondisi tertentu, algoritma pembangkitan kunci Gen harus memilih kunci secara seragam dari kumpulan semua kunci yang mungkin (seperti pada one-time pad); terlebih lagi, untuk setiap pesan m dan ciphertext c terdapat pemetaan kunci unik m ke c (sekali lagi, seperti pada one-time pad). Selain menarik, teorema ini juga merupakan alat yang berguna untuk membuktikan (atau menyangkal) kerahasiaan sempurna dari skema yang disarankan. Kami membahas ini lebih lanjut setelah pembuktian.

Teorema yang dinyatakan di sini mengasumsikan$|M| = |K| = |C|$, artinya himpunan teks biasa, kunci, dan teks tersandi semuanya memiliki ukuran yang sama. Kita telah melihat bahwa untuk kerahasiaan yang sempurna kita harus memiliki $|K| \geq |M|$. Sangat mudah untuk melihat bahwa dekripsi yang benar memerlukan $|C| \geq |M|$. Oleh karena itu, dalam beberapa hal, skema enkripsi dengan $|M| = |K| = |C|$ adalah "optimal."

TEOREMA 2.11 (Teorema Shannon) Misalkan (Gen, Enc, Des) adalah skema enkripsi dengan ruang pesan, yang mana $|M| = |K| = |C|$.

1. Setiap kunci $k \in K$ dipilih dengan probabilitas (sama) $1/|K|$ dengan algoritma Gen.
2. Untuk setiap $m \in M$ dan setiap $c \in C$, terdapat kunci unik $k \in K$ sehingga $\mathrm{Enc}_k(m)$ mengeluarkan c.

BUKTI Intuisi di balik pembuktian adalah sebagai berikut. Untuk melihat bahwa kondisi yang disebutkan menyiratkan kerahasiaan yang sempurna, perhatikan bahwa kondisi 2 berarti bahwa setiap ciphertext c bisa menjadi hasil enkripsi setiap kemungkinan plaintext m, karena ada beberapa kunci k yang memetakan m ke c. Karena ada kunci yang unik, dan setiap kunci dipilih dengan probabilitas yang sama, kerahasiaan sempurna mengikuti seperti pada one-time pad. Untuk arah lain, kerahasiaan sempurna segera menyiratkan bahwa untuk setiap m dan c setidaknya ada satu pemetaan kunci m ke c. Fakta bahwa $|M| = |K| = |C|$ berarti, bahwa untuk setiap m dan c terdapat tepat satu kunci tersebut. Mengingat hal ini, setiap kunci harus dipilih dengan probabilitas yang sama atau kerahasiaan sempurna akan gagal dipertahankan. Bukti formal berikut ini.

Kami berasumsi untuk kesederhanaan bahwa Enc bersifat deterministik. (Dapat ditunjukkan bahwa hal ini tidak menghilangkan keumumannya di sini.) Pertama-tama kita buktikan bahwa jika skema enkripsi memenuhi kondisi 1 dan 2, maka skema tersebut benar-

benar rahasia. Pembuktiannya pada dasarnya sama dengan pembuktian kerahasiaan sempurna untuk one-time pad, jadi kita akan membahasnya secara relatif singkat. Perbaiki sembarang $c \in C$ dan $m \in M$. Misalkan k adalah kunci unik, yang dijamin oleh kondisi 2, dengan $\mathrm{Enc}_k(m) = c$. Kemudian,

$$\Pr[C = c|M = m] = \Pr[K = k] = 1/|K|,$$

dimana persamaan akhir dipenuhi oleh kondisi 1. Jadi

$$\Pr[C = c] = \sum_{m \in \mathcal{M}} \Pr[Enc_K(m) = c] \,.\, \Pr[M = m] = 1/|K|$$

Hal ini berlaku untuk setiap distribusi di atas M. Jadi, untuk setiap distribusi di atas M, setiap $m \in M$ dengan $Pr[M = m] \neq 0$, dan setiap $c \in C$, kita mempunyai:

$$\Pr[M = m|C = c] = \frac{\Pr[C = c|M = m] \cdot \Pr[M = m]}{\Pr[C = c]} = \frac{\Pr[Enc_K(m) = c] \cdot \Pr[M = m]}{\Pr[C = c]}$$
$$= \frac{|K|^{-1} \cdot \Pr[M = m]}{|K|^{-1}} = \Pr[M = m],$$

dan skema ini sangat rahasia.

Untuk arah kedua, asumsikan skema enkripsi benar-benar rahasia; kami menunjukkan bahwa kondisi 1 dan 2 berlaku. Perbaiki sembarang $c \in C$. Harus ada pesan m^* yang mana $\Pr[\mathrm{Enc}_k(m^*) = c] \neq 0$. Lemma 2.4 kemudian mengimplikasikan bahwa $\Pr[\mathrm{Enc}_k(m) = c] \neq 0$. untuk setiap $m \in M$. Dengan kata lain, jika kita misalkan $M = \{m_1, m_2, \ldots\}$, maka untuk setiap $m_1 \in M$ kita mempunyai himpunan kunci $K_i \subset K$ yang tidak kosong sehingga $Enc_k(m_i) = c$ jika dan hanya jika $k \in K_i$. Terlebih lagi, ketika $i \neq j$ maka K_i dan K_j harus terpisah-pisah atau kebenarannya tidak dapat dipertahankan. Sejak $|K| = |M|$, kita melihat bahwa setiap K_i hanya berisi satu kunci k_i, seperti yang disyaratkan oleh kondisi 2. Sekarang, Lemma 2.4 menunjukkan bahwa untuk sembarang $m_i, m_j \in M$ kita mempunyai

$$\Pr[K = k_i] = \Pr[Enc_K(m_i) = c] = \Pr\left[Enc_K\left(m_j\right) = c\right] = \Pr[K = k_j] \cdot$$

Karena ini berlaku untuk semua $1 \leq i, j \leq |M| = |K|$, dan $k_i \neq k_j$ untuk $i \neq j$, ini berarti setiap kunci dipilih dengan probabilitas $1/|K|$, seperti yang disyaratkan oleh kondisi 1.

Teorema Shannon berguna untuk menentukan apakah suatu skema tertentu benar-benar rahasia. Kondisi 1 mudah untuk diperiksa, dan kondisi 2 dapat didemonstrasikan (atau dikontradiksi) tanpa harus menghitung probabilitas apa pun (berbeda dengan bekerja dengan Definisi 2.3 secara langsung). Sebagai contoh, kerahasiaan sempurna dari one-time pad adalah hal yang mudah untuk dibuktikan menggunakan teorema Shannon. Namun kami tekankan bahwa teorema tersebut hanya berlaku jika $|M| = |K| = |C|$.

BAB 3
ENKRIPSI KUNCI PRIBADI

Pada bab sebelumnya kita melihat beberapa keterbatasan mendasar dari kerahasiaan sempurna. Dalam bab ini kita memulai studi kita tentang kriptografi modern dengan memperkenalkan gagasan kerahasiaan komputasi yang lebih lemah (namun cukup). Kami kemudian akan menunjukkan bagaimana definisi ini dapat digunakan untuk melewati hasil ketidakmungkinan yang ditunjukkan sebelumnya dan, khususnya, bagaimana kunci pendek (misalnya, panjang 128 bit) dapat digunakan untuk mengenkripsi banyak pesan panjang (misalnya, total gigabyte). Sepanjang perjalanan kita akan mempelajari gagasan dasar tentang keacakan semu (pseudorandomness), yang menangkap gagasan bahwa sesuatu dapat "terlihat" benar-benar acak meskipun sebenarnya tidak. Konsep kuat ini mendasari sebagian besar kriptografi modern, dan juga memiliki penerapan dan implikasi di luar bidang ini.

3.1 KEAMANAN KOMPUTASI

Pada Bab 2 kami memperkenalkan gagasan kerahasiaan sempurna. Meskipun kerahasiaan yang sempurna merupakan tujuan yang berharga, namun hal ini juga tidak perlu kuat. Kerahasiaan yang sempurna mensyaratkan bahwa sama sekali tidak ada informasi tentang pesan terenkripsi yang bocor, bahkan kepada penyadap dengan kekuatan komputasi tak terbatas. Namun, untuk semua tujuan praktis, skema enkripsi masih dianggap aman jika hanya membocorkan sejumlah kecil informasi kepada penyadap dengan kekuatan komputasi terbatas. Misalnya, skema yang membocorkan informasi dengan probabilitas paling banyak 2^{-60} kepada penyadap yang menginvestasikan upaya komputasi hingga 200 tahun pada superkomputer tercepat yang ada sudah memadai untuk aplikasi apa pun di dunia nyata. Definisi keamanan yang memperhitungkan batasan komputasi pada penyerang, dan memungkinkan kemungkinan kegagalan yang kecil, disebut komputasi, untuk membedakannya dari gagasan (seperti kerahasiaan sempurna) yang bersifat teori informasi. Keamanan komputasi sekarang menjadi cara de facto di mana keamanan didefinisikan untuk semua tujuan kriptografi.

Kami menekankan bahwa meskipun kami menyerah dalam mendapatkan keamanan yang sempurna, hal ini tidak berarti kami mengabaikan pendekatan matematis yang ketat. Definisi dan bukti masih penting, dan satu-satunya perbedaan adalah kita sekarang mempertimbangkan definisi keamanan yang lebih lemah (namun tetap bermakna).

Keamanan komputasi menggabungkan dua kelonggaran yang berhubungan dengan gagasan teori informasi tentang keamanan (dalam kasus enkripsi, kedua kelonggaran ini diperlukan untuk melampaui batasan kerahasiaan sempurna yang dibahas dalam bab sebelumnya):

1. Keamanan hanya dijamin terhadap musuh efisien yang berjalan selama jangka waktu tertentu. Artinya, dengan waktu yang cukup (atau sumber daya komputasi yang mencukupi), penyerang mungkin dapat melanggar keamanan. Jika kita dapat

membuat sumber daya yang diperlukan untuk memecahkan skema ini lebih besar daripada yang tersedia bagi penyerang yang realistis, maka secara praktis skema tersebut tidak dapat dipecahkan.

2. Musuh berpotensi berhasil (yaitu, keamanan berpotensi gagal) dengan probabilitas yang sangat kecil. Jika kita dapat membuat kemungkinan ini cukup kecil, kita tidak perlu khawatir.

Untuk memperoleh teori yang bermakna, kita perlu mendefinisikan secara tepat relaksasi-relaksasi di atas. Ada dua pendekatan umum untuk melakukan hal ini: pendekatan konkrit dan pendekatan asimtotik. Hal ini dijelaskan selanjutnya.

Pendekatan Konkret

Pendekatan konkrit terhadap keamanan komputasi mengukur keamanan skema kriptografi dengan secara eksplisit membatasi probabilitas keberhasilan maksimum dari setiap musuh (yang diacak) yang berjalan selama jangka waktu tertentu atau, lebih tepatnya, menginvestasikan sejumlah upaya komputasi tertentu. Dengan demikian, definisi konkrit keamanan kira-kira berbentuk sebagai berikut:

"Suatu skema dikatakan aman (t, ε) jika ada musuh yang berjalan dalam waktu paling lama t berhasil memutus skema dengan probabilitas paling banyak ε."

(Tentu saja, pernyataan di atas hanya berfungsi sebagai acuan umum, dan agar pernyataan di atas masuk akal, kita perlu mendefinisikan dengan tepat apa yang dimaksud dengan "mematahkan" skema yang dimaksud.) Sebagai contoh, seseorang mungkin memiliki skema dengan menjamin bahwa tidak ada musuh yang berjalan paling lama 200 tahun menggunakan superkomputer tercepat yang tersedia yang dapat berhasil memecahkan skema dengan probabilitas lebih baik dari 2^{-60}. Atau, mungkin lebih mudah untuk mengukur waktu berjalan dalam kaitannya dengan siklus CPU, dan untuk membangun skema sedemikian rupa sehingga tidak ada musuh yang menggunakan paling banyak 2^{80} siklus yang dapat mematahkan skema dengan probabilitas lebih baik dari 2^{-60}. Penting untuk memahami nilai t yang besar dan nilai ε yang kecil yang merupakan ciri khas skema kriptografi modern.

Contoh 3.1

Skema enkripsi kunci pribadi modern umumnya diasumsikan memberikan keamanan yang hampir optimal dalam pengertian berikut: ketika kunci memiliki panjang n— sehingga ruang kunci memiliki ukuran 2^n— musuh berjalan dalam waktu t (diukur dalam, katakanlah, siklus komputer) berhasil memutus skema dengan probabilitas paling banyak $ct/2^n$ untuk beberapa konstanta tetap c. (Ini hanya berhubungan dengan pencarian brute force pada ruang kunci, dan mengasumsikan tidak ada pra-pemrosesan yang dilakukan.)

Dengan asumsi c = 1 untuk kesederhanaan, kunci dengan panjang n= 60 memberikan keamanan yang memadai terhadap musuh yang menggunakan komputer desktop. Memang benar, pada prosesor 4 GHz (yang mengeksekusi 4×10^9 siklus per detik) 2^{60} siklus CPU memerlukan $2^{60}/(4\times10^9)$ detik, atau sekitar 9 tahun. Namun, superkomputer tercepat pada saat penulisan ini dapat menjalankan sekitar 2×10^{16} operasi floating point per detik, dan 260

operasi tersebut hanya memerlukan sekitar 1 menit pada mesin tersebut. Mengambil n = 80 akan menjadi pilihan yang lebih bijaksana; bahkan komputer yang baru disebutkan akan membutuhkan waktu sekitar 2 tahun untuk menjalankan 2^{80} operasi. (Angka-angka di atas hanya untuk tujuan ilustrasi; dalam praktiknya c > 1, dan beberapa faktor lainnya seperti waktu yang diperlukan untuk akses memori dan kemungkinan komputasi paralel pada jaringan komputer secara signifikan mempengaruhi kinerja serangan brute-force .)

Namun saat ini, panjang kunci yang direkomendasikan mungkin adalah n = 128. Selisih antara 2^{80} dan 2^{128} merupakan faktor perkalian dari 2^{48}. Untuk mengetahui seberapa besarnya, perhatikan bahwa menurut perkiraan fisikawan, jumlah detik karena Big Bang berada di urutan 2^{58}.

Jika probabilitas penyerang berhasil memulihkan pesan terenkripsi dalam satu tahun adalah paling banyak 2^{-60}, maka kemungkinan besar pengirim dan penerima akan terkena sambaran petir dalam jangka waktu yang sama. Suatu peristiwa yang terjadi setiap seratus tahun sekali dapat diperkirakan terjadi dengan probabilitas 2^{-30} pada detik tertentu. Sesuatu yang terjadi dengan probabilitas 2^{-60} pada detik tertentu memiliki kemungkinan 2^{30} kali lebih kecil, dan diperkirakan akan terjadi kira-kira sekali setiap 100 miliar tahun.

Pendekatan konkrit penting dalam praktiknya, karena jaminan konkrit adalah hal yang pada akhirnya diminati oleh pengguna skema kriptografi. Namun, jaminan konkrit yang tepat sulit diberikan. Selain itu, seseorang harus berhati-hati dalam menafsirkan klaim keamanan yang konkrit. Misalnya, klaim bahwa tidak ada musuh yang berjalan selama 5 tahun dapat mematahkan skema tertentu dengan probabilitas lebih baik daripada ε menimbulkan pertanyaan: jenis daya komputasi apa (misalnya, PC desktop, superkomputer, jaringan ratusan komputer) yang diasumsikan oleh hal ini? ? Apakah hal ini memperhitungkan kemajuan daya komputasi di masa depan (yang menurut Hukum Moore, meningkat dua kali lipat setiap 18 bulan)? Apakah perkiraan tersebut mengasumsikan penggunaan algoritma "*off-the-shelf*", atau implementasi perangkat lunak khusus yang dioptimalkan untuk serangan tersebut? Lebih jauh lagi, jaminan seperti itu tidak menjelaskan apa pun tentang kemungkinan keberhasilan musuh yang mencalonkan diri selama 2 tahun (selain fakta bahwa jaminan tersebut paling banyak ε) dan tidak menjelaskan apa pun tentang kemungkinan keberhasilan musuh yang mencalonkan diri selama 10 tahun.

Pendekatan Asimptotik

Seperti disebutkan di atas, terdapat beberapa kesulitan teknis dan teoritis dalam menggunakan pendekatan keamanan konkret. Persoalan-persoalan ini harus ditangani dalam praktiknya, namun ketika keamanan nyata bukan merupakan permasalahan yang mendesak, maka akan lebih mudah jika kita menggunakan pendekatan asimtotik terhadap keamanan; inilah pendekatan yang diambil dalam buku ini. Pendekatan ini, yang berakar pada teori kompleksitas, memperkenalkan parameter keamanan bernilai integer (dilambangkan dengan n) yang membuat parameterisasi skema kriptografi serta semua pihak yang terlibat (yaitu, pihak yang jujur dan juga penyerang). Ketika pihak yang jujur menginisialisasi sebuah skema (yaitu, ketika mereka menghasilkan kunci), mereka memilih beberapa nilai n untuk parameter keamanan; untuk keperluan diskusi ini, parameter keamanan dapat dianggap sesuai dengan

panjang kunci. Parameter keamanan diasumsikan diketahui oleh musuh yang menyerang skema tersebut, dan kita sekarang melihat waktu berjalannya musuh, serta probabilitas keberhasilannya, sebagai fungsi dari parameter keamanan dan bukan sebagai angka konkrit. Kemudian:

1. Kami menyamakan "musuh yang efisien" dengan algoritma acak (yaitu probabilistik) yang berjalan dalam polinomial waktu dalam n. Ini berarti ada beberapa polinomial p sehingga musuh berjalan paling lama p(n) ketika parameter keamanannya adalah n. Kami juga mensyaratkan untuk efisiensi di dunia nyata bahwa partai-partai yang jujur dapat mencalonkan diri dalam jangka waktu yang banyak, meskipun kami menekankan bahwa pihak lawan mungkin jauh lebih kuat (dan berkuasa lebih lama dibandingkan) partai-partai yang jujur.
2. Kami menyamakan gagasan "probabilitas keberhasilan yang kecil" dengan probabilitas keberhasilan yang lebih kecil dibandingkan polinomial invers dalam n (lihat Definisi 3.4). Probabilitas seperti ini disebut dapat diabaikan. Biarkan PPT berarti "waktu polinomial probabilistik." Definisi keamanan asimtotik kemudian mengambil bentuk umum berikut:
   - ☞ Suatu skema dianggap aman jika ada musuh PPT yang berhasil melanggar skema tersebut dengan probabilitas yang paling kecil.
   - ☞ Gagasan tentang keamanan ini tidak menunjukkan gejala karena keamanan bergantung pada perilaku skema untuk nilai n yang cukup besar. Contoh berikut memperjelas hal ini.

Contoh 3.2

Katakanlah kita memiliki skema yang aman secara asimtotik. Maka mungkin saja musuh yang berlari selama n^3 menit dapat berhasil "mematahkan skema" dengan probabilitas $2^{40} \cdot 2^{-n}$ (yang merupakan fungsi n yang dapat diabaikan). Ketika $n \leq 40$ ini berarti bahwa musuh yang berlari selama 40^3 menit (sekitar 6 minggu) dapat merusak skema dengan probabilitas 1, sehingga nilai n tersebut tidak terlalu berguna. Bahkan untuk n = 50, musuh yang berlari selama 50^3 menit (sekitar 3 bulan) dapat merusak skema dengan probabilitas sekitar 1/1000, yang mungkin tidak dapat diterima. Di sisi lain, ketika n = 500, musuh yang berjalan selama 200 tahun akan mematahkan skema tersebut hanya dengan probabilitas sekitar 2^{-500}.

Seperti yang ditunjukkan pada contoh sebelumnya, kita dapat melihat parameter keamanan sebagai mekanisme yang memungkinkan pihak-pihak yang jujur untuk "menyesuaikan" keamanan suatu skema ke tingkat yang diinginkan. (Meningkatkan parameter keamanan juga meningkatkan waktu yang dibutuhkan untuk menjalankan skema, serta panjang kunci, sehingga pihak yang jujur ingin menetapkan parameter keamanan sekecil mungkin untuk mempertahankan diri dari kelas serangan yang mereka khawatirkan. tentang.) Melihat parameter keamanan sebagai panjang kunci, hal ini kira-kira sesuai dengan fakta bahwa waktu yang diperlukan untuk serangan pencarian menyeluruh tumbuh secara eksponensial sepanjang kunci. Kemampuan untuk "meningkatkan keamanan" dengan meningkatkan parameter keamanan memiliki konsekuensi praktis yang penting, karena hal ini

memungkinkan pihak yang jujur untuk mempertahankan diri dari peningkatan daya komputasi. Contoh berikut memberikan gambaran bagaimana hal ini dapat diterapkan dalam praktiknya.

Contoh 3.3

Mari kita lihat dampak ketersediaan komputer yang lebih cepat terhadap keamanan dalam praktiknya. Katakanlah kita memiliki skema kriptografi di mana pihak yang jujur menjalankan selama $10^6 \cdot n^2$ siklus, dan musuh yang menjalankan selama $10^8 \cdot n^4$ siklus dapat berhasil "mematahkan" skema dengan probabilitas paling banyak 2−n/2. (Angka-angka tersebut dimaksudkan untuk mempermudah penghitungan, dan tidak dimaksudkan untuk menyesuaikan dengan skema kriptografi yang ada.)

Katakanlah semua pihak menggunakan komputer 2 GHz dan pihak yang jujur menetapkan n = 80. Kemudian pihak yang jujur berjalan selama $10^6 \cdot 6400$ siklus, atau 3,2 detik, dan musuh yang berjalan selama $10^8 \cdot (80)^4$ siklus, atau kira-kira 3 minggu, dapat rusak skema dengan probabilitas hanya 2^{-40}.

Katakanlah komputer 8 GHz tersedia, dan semua pihak melakukan upgrade. Pihak yang jujur dapat meningkatkan n menjadi 160 (yang memerlukan pembuatan kunci baru) dan mempertahankan waktu berjalan 3,2 detik (yaitu, $10^6 \cdot 160^2$ siklus pada $8 \cdot 10^9$ siklus/detik). Sebaliknya, musuh kini harus berlari selama lebih dari 8 juta detik, atau lebih dari 13 minggu, untuk mencapai probabilitas keberhasilan 2^{-80}. Dampak dari komputer yang lebih cepat adalah mempersulit pekerjaan musuh.

Bahkan ketika menggunakan pendekatan asimtotik, penting untuk diingat bahwa, pada akhirnya, ketika sistem kriptografi diterapkan dalam praktik, jaminan keamanan yang nyata akan diperlukan. (Bagaimanapun juga, seseorang harus memutuskan suatu nilai n.) Akan tetapi, seperti yang ditunjukkan oleh contoh di atas, pada umumnya klaim keamanan asimtotik dapat diterjemahkan ke dalam ikatan keamanan konkret untuk nilai n yang diinginkan.

Pendekatan Asimptotik secara Detail

Sekarang kita membahas secara lebih formal pengertian "algoritme waktu polinomial" dan "probabilitas keberhasilan yang dapat diabaikan".

Algoritma yang efisien. Didefinisikan sebagai suatu algoritma menjadi efisien jika dijalankan dalam waktu polinomial. Algoritme A berjalan dalam waktu polinomial jika terdapat polinomial p sehingga, untuk setiap masukan x {0, 1}*, komputasi A(x) berakhir paling banyak dalam langkah *p(x).* (Di sini, x menunjukkan panjang string x.) Seperti disebutkan sebelumnya, kami hanya tertarik pada musuh yang waktu berjalannya polinomial dalam parameter keamanan n. Karena kita mengukur waktu berjalan suatu algoritma berdasarkan panjang masukannya, terkadang kita menyediakan algoritma dengan parameter keamanan yang ditulis dalam bentuk unary (misalnya, 1^n, atau string sebanyak n) sebagai masukan. Pihak-pihak (atau, lebih tepatnya, algoritme yang mereka jalankan) dapat mengambil masukan lain selain parameter keamanan misalnya, sebuah pesan untuk dienkripsi dan kami mengizinkan waktu berjalan mereka menjadi polinomial dalam (total) panjang masukan mereka.

Secara default, kami mengizinkan semua algoritme bersifat probabilistik (atau acak). Algoritme semacam itu mungkin "melempar koin" pada setiap langkah pelaksanaannya; ini adalah cara metaforis untuk mengatakan bahwa algoritma dapat mengakses bit acak yang tidak bias pada setiap langkah. Hal yang sama, kita dapat melihat algoritma acak sebagai algoritma yang, selain inputnya, diberikan pita acak yang terdistribusi secara seragam dengan panjang yang cukup yang bit-bitnya dapat digunakan, sesuai kebutuhan, sepanjang eksekusinya. Kami mempertimbangkan algoritma acak secara default karena dua alasan. Pertama, keacakan sangat penting dalam kriptografi (misalnya, untuk memilih kunci acak dan sebagainya) sehingga pihak yang jujur harus bersifat probabilistik; mengingat hal ini, wajar jika membiarkan musuh juga bersifat probabilistik. Kedua, pengacakan bersifat praktis dan sejauh yang kami tahu memberi penyerang kekuatan tambahan. Karena tujuan kami adalah memodelkan semua serangan yang realistis, kami lebih memilih definisi komputasi efisien yang lebih liberal.

Probabilitas keberhasilan dapat diabaikan. Fungsi yang dapat diabaikan adalah fungsi yang secara asimtotik lebih kecil dibandingkan fungsi polinomial invers mana pun. Secara formal:

DEFINISI 3.4 *Suatu fungsi f dari bilangan asli ke bilangan real non-negatif dapat diabaikan jika untuk setiap polinomial positif p terdapat N sehingga untuk semua bilangan bulat n > N berlaku* $f(n) < \frac{1}{p(n)}$.

Singkatannya, pernyataan di atas juga dinyatakan sebagai berikut: untuk setiap polinomial p dan semua nilai n yang cukup besar maka $f(n) < \frac{1}{p(n)}$. Rumusan ekuivalen di atas adalah mensyaratkan bahwa untuk semua konstanta c terdapat N sehingga untuk semua n > N dinyatakan bahwa $f(n) < n^{-c}$. Kami biasanya menyatakan fungsi yang dapat diabaikan secara sembarang dengan pengabaian.

Contoh 3.5

Fungsi $2^{-n}, 2^{-\sqrt{n}}, dan\, n^{-\log n}$ semuanya dapat diabaikan. Namun, keduanya mendekati nol pada tingkat yang sangat berbeda. Misalnya, kita dapat melihat nilai minimum n yang setiap fungsinya lebih kecil dari $1/n^5$:

1. Menyelesaikan $2^{-n} < n^{-5}$ kita mendapatkan $n > 5\, log\, n$. Nilai bilangan bulat terkecil dari n yang dimilikinya adalah $n = 23$.
2. Menyelesaikan $2^{-\sqrt{n}} < n^{-5}$ kita mendapatkan $n > 25\, log^2\, n$. Nilai bilangan bulat terkecil dari n yang dimilikinya adalah $n \approx 3500$.
3. Menyelesaikan $n^{-\log n} < n^{-5}$ kita mendapatkan $log\, n > 5$. Nilai bilangan bulat terkecil dari n yang dapat dipegang adalah $n = 33$.

Dari penjelasan di atas Anda mungkin mendapat kesan bahwa $n^{-\log n}$ mendekati nol lebih cepat daripada $2^{-\sqrt{n}}$. Namun, hal ini tidak benar; untuk semua $n > 65536$ dinyatakan bahwa $2^{-\sqrt{n}} < n^{-\log n}$. Namun demikian, hal ini menunjukkan bahwa untuk nilai n dalam

ratusan atau ribuan, probabilitas keberhasilan adversarial sebesar $n^{-\log n}$ lebih disukai daripada probabilitas keberhasilan adversarial sebesar $2^{-\sqrt{n}}$.

Keuntungan teknis dari bekerja dengan probabilitas keberhasilan yang dapat diabaikan adalah bahwa mereka mematuhi properti penutupan tertentu. Berikut ini adalah latihan yang mudah.

PROPOSISI 3.6 Misalkan negl_1 dan negl_2 merupakan fungsi yang dapat diabaikan. Kemudian,

1. Fungsi negl_3 yang didefinisikan oleh $\mathrm{negl}_3(\mathrm{n}) = \mathrm{negl}_1(\mathrm{n}) + \mathrm{negl}_2(\mathrm{n})$ dapat diabaikan.
2. Untuk polinomial positif p, fungsi negl_4 yang didefinisikan oleh $\mathrm{negl}_4(\mathrm{n}) = \mathrm{p}(\mathrm{n}) \cdot \mathrm{negl}_1(\mathrm{n})$dapat diabaikan.

Bagian kedua dari proposisi di atas menyiratkan bahwa jika suatu peristiwa tertentu terjadi dengan probabilitas yang dapat diabaikan dalam suatu eksperimen tertentu, maka peristiwa tersebut terjadi dengan probabilitas yang dapat diabaikan bahkan jika eksperimen tersebut diulang berkali-kali secara polinomial. (Hal ini bergantung pada ikatan gabungan; lihat Proposisi A.7.) Misalnya, probabilitas bahwa n koin yang adil membalik semua "kepala" yang muncul dapat diabaikan. Ini berarti bahwa meskipun kita mengulangi percobaan pelemparan n koin secara polinomial berkali-kali, kemungkinan bahwa percobaan tersebut menghasilkan n kepala masih dapat diabaikan.

Akibat wajar dari bagian kedua proposisi di atas adalah jika suatu fungsi g tidak dapat diabaikan, maka fungsi $f(n) \stackrel{\text{def}}{=} g(n)/p(n)$ polinomial positif p juga tidak dapat diabaikan.

Keamanan Asimptotik

Definisi keamanan apa pun terdiri dari dua bagian: definisi mengenai apa yang dianggap sebagai "kerusakan" skema, dan spesifikasi mengenai kekuatan musuh. Kekuatan musuh dapat berhubungan dengan banyak isu (misalnya, dalam kasus enkripsi, apakah kita mengasumsikan serangan hanya ciphertext atau serangan teks biasa yang dipilih). Namun, jika menyangkut kekuatan komputasi musuh, mulai sekarang kami akan memodelkan musuh sebagai efisien dan dengan demikian hanya mempertimbangkan strategi permusuhan yang dapat diterapkan dalam waktu polinomial probabilistik. Definisi juga akan selalu dirumuskan sedemikian rupa sehingga penembusan yang terjadi dengan probabilitas yang dapat diabaikan tidak dianggap signifikan. Dengan demikian, kerangka umum definisi keamanan adalah sebagai berikut:

Suatu skema aman jika untuk setiap musuh dengan waktu polinomial probabilistik $\mathcal{A}$ yang melakukan serangan dengan jenis tertentu yang ditentukan secara formal, probabilitas $\mathcal{A}$ keberhasilan serangan (yang keberhasilannya juga ditentukan secara formal) dapat diabaikan. Definisi seperti itu tidak menunjukkan gejala karena ada kemungkinan bahwa untuk nilai n yang kecil, musuh dapat berhasil dengan probabilitas yang tinggi. Untuk melihat hal ini secara lebih rinci, kami memperluas istilah "dapat diabaikan" dalam pernyataan di atas:

Suatu skema dikatakan aman jika untuk setiap musuh PPT $\mathcal{A}$ yang melakukan serangan dengan tipe tertentu yang ditentukan secara formal, dan untuk setiap polinomial positif p, terdapat bilangan bulat N sehingga ketika $n > N$ probabilitas $\mathcal{A}$ berhasil dalam serangan tersebut lebih kecil dari $\frac{1}{p(n)}$.
Perhatikan bahwa tidak ada yang dijamin untuk nilai $n \leq N$.

Tentang Pilihan yang Dibuat dalam Mendefinisikan Keamanan Asimptotik

Dalam mendefinisikan gagasan umum tentang keamanan asimtotik, kami telah membuat dua pilihan: kami telah mengidentifikasi strategi permusuhan yang efisien dengan kelas algoritma probabilistik, waktu polinomial, dan telah menyamakan peluang keberhasilan yang kecil dengan probabilitas yang dapat diabaikan. Kedua pilihan ini sampai batas tertentu bersifat sewenang-wenang, dan kita dapat membangun teori yang masuk akal dengan mendefinisikan, katakanlah, strategi efisien sebagai strategi yang berjalan dalam waktu kuadrat, atau probabilitas keberhasilan kecil sebagai strategi yang dibatasi oleh 2^{-n}. Namun demikian, kami secara singkat membenarkan pilihan yang telah kami buat (yang merupakan pilihan standar).

Mereka yang akrab dengan teori atau algoritma kompleksitas akan menyadari bahwa gagasan untuk menyamakan komputasi yang efisien dengan algoritma waktu polinomial (probabilistik) tidak hanya terjadi pada kriptografi. Salah satu keuntungan menggunakan waktu polinomial (probabilistik) sebagai ukuran efisiensi adalah bahwa hal ini membebaskan kita dari keharusan menentukan secara tepat model komputasi kita, karena tesis Church-Turing yang diperluas menyatakan bahwa semua model komputasi yang “masuk akal” adalah ekuivalen secara polinomial. Jadi, kita tidak perlu menentukan apakah kita menggunakan mesin Turing, sirkuit boolean, atau mesin akses acak; kita dapat menyajikan algoritma dalam pseudocode tingkat tinggi dan yakin bahwa jika analisis kita menunjukkan bahwa algoritma ini berjalan dalam waktu polinomial, maka implementasi yang masuk akal juga akan berjalan.

Keuntungan lain dari algoritme waktu polinomial (probabilistik) adalah bahwa algoritme tersebut memenuhi sifat penutupan yang diinginkan: khususnya, algoritme yang membuat banyak panggilan polinomial ke subrutin waktu polinomial (dan hanya melakukan komputasi polinomial sebagai tambahan) akan berjalan dengan sendirinya dalam waktu polinomial.

Ciri terpenting dari probabilitas yang dapat diabaikan adalah sifat penutupan yang telah kita lihat pada Proposisi 3.6(2): setiap kali polinomial, suatu fungsi yang dapat diabaikan tetap dapat diabaikan. Hal ini berarti, khususnya, jika suatu algoritma membuat banyak panggilan secara polinomial ke beberapa subrutin yang “gagal” dengan probabilitas yang dapat diabaikan setiap kali subrutin tersebut dipanggil, maka probabilitas bahwa salah satu panggilan ke subrutin tersebut gagal masih dapat diabaikan.

Perlunya Relaksasi

Kerahasiaan komputasi memperkenalkan dua relaksasi dari kerahasiaan sempurna: pertama, keamanan dijamin hanya terhadap musuh yang efisien; kedua, kemungkinan sukses yang kecil diperbolehkan. Kedua relaksasi ini penting untuk mencapai skema enkripsi yang

praktis, dan khususnya untuk menghindari hasil negatif dari enkripsi yang sangat rahasia. Kami secara informal mendiskusikan mengapa hal ini terjadi. Asumsikan kita mempunyai skema enkripsi dimana ukuran ruang kunci $\mathcal{K}$ jauh lebih kecil daripada ukuran ruang pesan $\mathcal{M}$. (Seperti yang ditunjukkan dalam bab sebelumnya, ini berarti skema tersebut tidak dapat sepenuhnya dirahasiakan.) Ada dua serangan yang berlaku terlepas dari bagaimana skema enkripsi dibuat:

- Dengan adanya ciphertext c, musuh dapat mendekripsi c menggunakan semua kunci $k \in \mathcal{K}$ · Ini memberikan daftar semua pesan yang mungkin berhubungan dengan c. Karena daftar ini tidak dapat memuat semuanya $\mathcal{M}$ (karena $\mathcal{K} < \mathcal{M}$), serangan ini membocorkan beberapa informasi tentang pesan yang dienkripsi.

  Terlebih lagi, katakanlah musuh melakukan serangan teks biasa dan mengetahui bahwa teks tersandi $c_1, \cdots c_\ell$sesuai dengan pesan $m_1, \cdots, m_\ell$ masing-masing. Musuh dapat kembali mencoba mendekripsi masing-masing ciphertext ini dengan semua kunci yang ada sampai ia menemukan kunci k yang mana Deck(ci) = mi untuk semua i. Kemudian, dengan adanya ciphertext c yang merupakan enkripsi dari pesan yang tidak diketahui m, maka hampir pasti Deck(c) = m.

  Serangan penelusuran mendalam seperti di atas memungkinkan musuh berhasil dengan probabilitas dasarnya 1 dalam waktu linier dalam |K|.
- Pertimbangkan kembali kasus dimana musuh mengetahui bahwa ciphertext $c_1, \cdots c_\ell$ sesuai dengan pesan $m_1, \cdots, m_\ell$. Musuh dapat menebak kunci seragam $k \in \mathcal{K}$ · dan memeriksa apakah $Dec_k(c_i) = m_i$ untuk semua i. Jika demikian, maka seperti di atas, penyerang dapat menggunakan k untuk mendekripsi apa pun yang kemudian dienkripsi oleh pihak yang jujur.

  Di sini musuh berjalan dalam waktu yang pada dasarnya konstan dan berhasil dengan probabilitas bukan nol (meskipun sangat kecil) $1/|\mathcal{K}|$.

Oleh karena itu, jika kita ingin mengenkripsi banyak pesan menggunakan satu kunci pendek, keamanan hanya dapat dicapai jika kita membatasi waktu berjalannya musuh (sehingga musuh tidak memiliki cukup waktu untuk melakukan pencarian brute-force) dan bersedia memberikan kemungkinan keberhasilan yang sangat kecil (sehingga "serangan" kedua dikesampingkan).

3.2 MENDEFINISIKAN ENKRIPSI YANG AMAN SECARA KOMPUTASI

Mengingat latar belakang bagian sebelumnya, kami siap menyajikan definisi keamanan komputasi untuk enkripsi kunci pribadi. Pertama, kami mendefinisikan ulang sintaks enkripsi kunci pribadi; ini pada dasarnya akan sama dengan sintaks yang diperkenalkan pada Bab 2 kecuali bahwa kita sekarang secara eksplisit memperhitungkan parameter keamanan n. Kami juga mengizinkan algoritme dekripsi mengeluarkan pesan kesalahan jika pesan tersebut disajikan dengan teks sandi yang tidak valid. Terakhir, secara default, kita membiarkan ruang pesan menjadi himpunan 0, 1 $*$ dari semua string biner (panjang terbatas).

DEFINISI 3.7 Skema enkripsi kunci pribadi adalah rangkaian algoritma waktu polinomial probabilistik (Gen, Enc, Dec) sedemikian rupa sehingga:

1. Algoritme pembangkitan kunci Gen mengambil masukan 1^n (yaitu, parameter keamanan yang ditulis dalam unary) dan mengeluarkan kunci k; kita menulis $k \leftarrow \text{Gen}(1^n)$ (menekankan bahwa Gen adalah algoritma acak). Kami berasumsi tanpa kehilangan keumuman bahwa kunci apa pun yang dihasilkan k oleh $\text{Gen}(1^n)$ memenuhi $|k| \geq n$.
2. Algoritma enkripsi Enc mengambil masukan kunci k dan pesan teks biasa $m \in \{0,1\}^*$, dan mengeluarkan teks sandi c. Karena Enc dapat diacak, kita menuliskannya sebagai $c \leftarrow \text{Enc}_k(m)$
3. Algoritma dekripsi Dec mengambil kunci k dan teks sandi c sebagai masukan, dan mengeluarkan pesan m atau kesalahan. Kita berasumsi bahwa Dec bersifat deterministik, maka tulislah $m := \text{Dec}_k(c)$ (dengan asumsi di sini bahwa Dec tidak menghasilkan kesalahan). Kami menyatakan kesalahan umum dengan simbol $\perp$.

Diperlukan bahwa untuk setiap n, setiap kunci k yang dihasilkan oleh $\text{Gen}(1^n)$, dan setiap m $\in \{0,1\}^*$, maka $\text{Dec}_k(\text{Enc}_k(m)) = m$

Jika (Gen, Enc, Dec) sedemikian rupa sehingga untuk k output oleh $\text{Gen}(1^n)$, algoritma Enc_k hanya didefinisikan untuk pesan $m \in \{0,1\}^{\ell(n)}$, maka kita katakan bahwa (Gen, Enc, Dec) adalah tetap -skema enkripsi kunci pribadi panjang untuk pesan dengan panjang $\ell(n)$.

Hampir selalu, $\text{Gen}(1^n)$ hanya mengeluarkan string n-bit yang seragam sebagai kuncinya. Jika hal ini terjadi, kita akan menghilangkan Gen dan hanya mendefinisikan skema enkripsi kunci privat dengan sepasang algoritma (Enc, Des).

Definisi di atas mempertimbangkan skema tanpa kewarganegaraan, di mana setiap pemanggilan Enc (dan Dec) tidak bergantung pada semua pemanggilan sebelumnya. Nanti di bab ini, kita kadang-kadang akan membahas skema stateful di mana pengirim (dan mungkin penerima) diharuskan mempertahankan status di seluruh pemanggilan. Kecuali dinyatakan sebaliknya secara eksplisit, semua hasil kami mengasumsikan enkripsi/dekripsi tanpa kewarganegaraan.

Definisi Dasar Keamanan

Kita mulai dengan menyajikan gagasan paling dasar tentang keamanan enkripsi kunci pribadi: keamanan terhadap serangan hanya ciphertext di mana musuh hanya mengamati satu ciphertext atau, setara, keamanan ketika kunci tertentu digunakan untuk mengenkripsi satu pesan saja. Kita akan membahas definisi keamanan yang lebih kuat pada bab selanjutnya.

Memotivasi definisi. Seperti yang telah kita diskusikan, setiap definisi keamanan terdiri dari dua komponen berbeda: model ancaman (yaitu spesifikasi asumsi kekuatan musuh) dan tujuan keamanan (biasanya ditentukan dengan menjelaskan apa yang dimaksud dengan "putusnya" keamanan). skema). Kami memulai pembahasan definisi kami dengan mempertimbangkan model ancaman yang paling sederhana, di mana kami memiliki musuh yang menguping dan mengamati enkripsi sebuah pesan. Ini adalah model ancaman yang telah dibahas dalam bab sebelumnya dengan pengecualian bahwa, seperti yang dijelaskan di bagian

sebelumnya, kita sekarang hanya tertarik pada musuh yang dibatasi secara komputasi dan dibatasi untuk berjalan dalam waktu polinomial.

Meskipun kami telah membuat dua asumsi tentang kemampuan musuh (yaitu, bahwa ia hanya menguping, dan berjalan dalam waktu polinomial), kami tidak membuat asumsi apa pun tentang strategi musuh dalam mencoba menguraikan teks sandi yang diamatinya. Hal ini penting untuk memperoleh pemahaman yang bermakna tentang keamanan; definisi tersebut memastikan perlindungan terhadap musuh yang dibatasi secara komputasi, apa pun algoritma yang digunakannya.

Mendefinisikan tujuan keamanan enkripsi dengan benar bukanlah hal yang sepele, namun kita telah membahas masalah ini secara panjang lebar di Bagian 1.4 dan bab sebelumnya. Oleh karena itu, kami hanya mengingatkan bahwa gagasan di balik definisi tersebut adalah bahwa musuh tidak boleh mempelajari sebagian informasi tentang teks biasa dari teks tersandi. Definisi keamanan semantik (lihat. Bagian 3.2) secara tepat memformalkan gagasan ini, dan merupakan definisi pertama enkripsi aman secara komputasi yang diusulkan. Keamanan semantik rumit dan sulit untuk dikerjakan. Untungnya, ada definisi setara yang disebut indistinguishability yang jauh lebih sederhana.

Definisi indistinguishability berpola pada definisi alternatif kerahasiaan sempurna yang diberikan pada Definisi 2.5. (Ini berfungsi sebagai motivasi lebih lanjut bahwa definisi indistinguishability adalah definisi yang baik.) Ingatlah bahwa Definisi 2.5 mempertimbangkan eksperimen $PrivK^{eav}_{\mathcal{A},\prod}$ di mana musuh A mengeluarkan dua pesan m_0 dan m_1, dan kemudian diberi enkripsi satu pesan pesan-pesan itu menggunakan kunci seragam. Definisi tersebut menyatakan bahwa skema $\prod$ aman jika tidak ada musuh yang dapat menentukan pesan mana m0, m1 yang dienkripsi dengan probabilitas berbeda dari 1/2, yang merupakan probabilitas yang benar jika ia hanya membuat tebakan acak. Di sini, kami menjaga eksperimen $PrivK^{eav}_{\mathcal{A},\prod}$ hampir sama persis (kecuali untuk beberapa perbedaan teknis yang dibahas di bawah), namun memperkenalkan dua modifikasi utama dalam definisi itu sendiri:

1. Sekarang kita hanya mempertimbangkan musuh yang berlari dalam waktu polinomial, sedangkan Definisi 2.5 menganggap musuh genap dengan waktu berjalan tak terbatas.
2. Kami sekarang mengakui bahwa musuh mungkin menentukan pesan terenkripsi dengan probabilitas lebih baik dari 1/2.

Seperti yang dibahas secara luas di bagian sebelumnya, pelonggaran di atas merupakan elemen inti keamanan komputasi.

Adapun perbedaan lainnya, yang paling menonjol adalah sekarang kita memparameterkan eksperimen dengan parameter keamanan n. Kami kemudian mengukur waktu berjalannya musuh serta probabilitas keberhasilannya sebagai fungsi dari n. Kami menulis $PrivK^{eav}_{\mathcal{A},\prod}(n)$ untuk menunjukkan eksperimen yang dijalankan dengan keamanan A, Π parameter n, dan menulis

$$\Pr\left[PrivK^{eav}_{\mathcal{A},\prod}(n) = 1\right] \qquad (3.1)$$

untuk menunjukkan probabilitas bahwa keluaran percobaan $PrivK_{\mathcal{A},v}^{eav}(n)$ adalah 1. Perhatikan bahwa dengan A, Π tetap, Persamaan (3.1) adalah fungsi dari n. Perbedaan kedua dalam eksperimen $PrivK_{\mathcal{A},\Pi}^{eav}$ adalah kita sekarang secara eksplisit meminta musuh untuk mengeluarkan dua pesan m_0, m_1 dengan panjang yang sama. (Dalam Definisi 2.5 persyaratan ini implisit jika ruang pesan hanya berisi pesan dengan panjang tetap, seperti halnya skema enkripsi one-time pad.) Ini berarti, secara default, kami tidak memerlukan skema enkripsi aman untuk menyembunyikan panjang teks biasa. Kami meninjau kembali poin ini di akhir bagian ini; lihat juga Latihan 3.2 dan 3.3.

Tidak dapat dibedakan dengan adanya penyadap. Kami sekarang memberikan definisi formal, dimulai dengan eksperimen yang diuraikan di atas. Percobaan ini didefinisikan untuk setiap skema enkripsi kunci pribadi Π = (Gen, Enc, Dec), setiap musuh $\mathcal{A}$, dan setiap nilai n untuk parameter keamanan:

Eksperimen ketidakmungkinan membedakan $PrivK_{\mathcal{A},\Pi}^{eav}(n)$:

1. Musuh $\mathcal{A}$ diberi masukan 1^n, dan mengeluarkan sepasang pesan $m_0 m_1$ dengan $|m_0| = |m_1|$.
2. Kunci k dihasilkan dengan menjalankan $Gen(1^n)$, dan bit seragam $b \in \{0,1\}$ dipilih. $Ciphertext\ c \rightarrow Enc_k(m_b)$ dihitung dan diberikan kepada $\mathcal{A}$. Kita menyebut *c* sebagai tantangan ciphertext.
3. $\mathcal{A}$ mengeluarkan sedikit *b'*.
4. Keluaran percobaan ditetapkan $1\ jika\ b' = b, dan\ 0$ jika tidak. Jika $PrivK_{\mathcal{A},\Pi}^{eav}(n)$, kita katakan $\mathcal{A}$ berhasil.

Tidak ada batasan panjang m_0 dan m_1, asalkan sama. (Tentu saja, jika berjalan dalam waktu polinomial, maka m_0 dan m_1 memiliki panjang polinomial dalam n.) Jika Π adalah skema dengan panjang tetap untuk pesan dengan panjang $\ell(n)$, percobaan di atas dimodifikasi dengan memerlukan $m_0 m_1 \in \{0,1\}^{\ell(n)}$.

Fakta bahwa musuh hanya dapat menguping tersirat dalam kenyataan bahwa masukannya terbatas pada (tunggal) ciphertext, dan musuh tidak memiliki interaksi lebih lanjut dengan pengirim atau penerima. (Seperti yang akan kita lihat nanti, mengizinkan interaksi tambahan akan membuat musuh menjadi lebih kuat secara signifikan.)

Definisi indistinguishability menyatakan bahwa skema enkripsi aman jika tidak ada musuh PPT yang berhasil menebak pesan mana yang dienkripsi dalam percobaan di atas dengan probabilitas yang jauh lebih baik daripada tebakan acak (yang benar dengan probabilitas 1/2):

DEFINISI 3.8 *Skema enkripsi kunci pribadi Π = (Gen, Enc, Dec) memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap, atau aman untuk EAV, jika untuk semua musuh waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan, abaikan hal tersebut itu, untuk semua n,*

$$\Pr\left[PrivK_{\mathcal{A},\Pi}^{eav}(n) = 1\right] \leq \frac{1}{2} + negl(n),$$

dimana probabilitas diambil alih keacakan yang digunakan oleh dan keacakan yang digunakan dalam percobaan (untuk memilih kunci dan bit b, serta setiap keacakan yang digunakan oleh Enc).

Catatan: kecuali memenuhi syarat lain, ketika kita menulis "$f(n) \leq g(n)$" yang kita maksud adalah pertidaksamaan berlaku untuk semua n.

Harus jelas bahwa Definisi 3.8 lebih lemah dibandingkan Definisi 2.5, yang setara dengan kerahasiaan sempurna. Jadi, setiap skema enkripsi yang sangat rahasia memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap. Oleh karena itu, tujuan kami adalah untuk menunjukkan bahwa terdapat skema enkripsi yang memenuhi kriteria di atas, dimana kuncinya lebih pendek daripada pesannya. Artinya, kami akan menunjukkan skema yang memenuhi Definisi 3.8 tetapi tidak dapat memenuhi Definisi 2.5.

Formulasi yang setara. Definisi 3.8 mensyaratkan bahwa tidak ada musuh PPT yang dapat menentukan mana dari dua pesan yang dienkripsi, dengan probabilitas yang jauh lebih baik daripada 1/2. Formulasi yang setara adalah bahwa setiap musuh PPT berperilaku sama apakah ia melihat enkripsi m_0 atau m_1. Karena menghasilkan satu bit, "berperilaku sama" berarti menghasilkan 1 dengan probabilitas yang hampir sama di setiap kasus. Untuk memformalkannya, definisikan $PrivK^{eav}_{\mathcal{A},\Pi}(n,b)$ seperti di atas kecuali bahwa bit tetap b digunakan (bukan dipilih secara acak). Biarkan keluar $\mathcal{A}^{PrivK^{eav}_{\mathcal{A},\Pi}(n,b)}$ menunjukkan bit keluaran b' dalam percobaan. Pernyataan berikut ini pada dasarnya menyatakan bahwa tidak ada yang dapat menentukan apakah eksperimen tersebut berjalan dalam eksperimen $PrivK^{eav}_{\mathcal{A},\Pi}(n,0)$ atau eksperimen $PrivK^{eav}_{\mathcal{A},\Pi}(n,1)$.

DEFINISI 3.9 Skema enkripsi kunci pribadi Π = (Gen, Enc, Des) memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap jika untuk semua musuh PPT $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sedemikian rupa sehingga

$$\left|\Pr\left[\text{out}_{\mathcal{A}}\left(Priv^{eav}_{\mathcal{A},\Pi}(n,0)\right)=1\right]-\Pr\left[\text{out}_{\mathcal{A}}\left(PrivK^{eav}_{\mathcal{A},\Pi}(n,1)\right)=1\right]\right| \leq \text{negl}(n) \cdot$$

Fakta bahwa ini setara dengan Definisi 3.8 dibiarkan sebagai latihan.

Enkripsi dan Panjang Plaintext

Gagasan default enkripsi aman tidak memerlukan skema enkripsi untuk menyembunyikan panjang teks biasa dan, pada kenyataannya, semua skema enkripsi yang umum digunakan mengungkapkan panjang teks biasa (atau perkiraan yang mendekatinya). Alasan utamanya adalah tidak mungkin mendukung pesan dengan panjang sembarang sambil menyembunyikan semua informasi tentang panjang teks biasa (lihat Latihan 3.2). Dalam banyak kasus, hal ini tidak penting karena panjang teks biasa sudah bersifat publik atau tidak

sensitif. Namun hal ini tidak selalu terjadi, dan terkadang membocorkan panjang teks biasa merupakan masalah. Sebagai contoh:

- **Data numerik/teks sederhana**: Katakanlah skema enkripsi yang digunakan mengungkapkan panjang teks biasa dengan tepat. Kemudian informasi gaji terenkripsi akan mengungkapkan apakah seseorang mendapat gaji 5 digit atau 6 digit. Demikian pula, enkripsi jawaban "ya"/"tidak" akan membocorkan jawabannya dengan tepat.
- **Saran otomatis**: Situs web sering kali menyertakan fungsi "pelengkapan otomatis" atau "saran otomatis" yang dengannya server web menyarankan daftar kata atau frasa potensial berdasarkan sebagian informasi yang sudah diketik pengguna. Besar kecilnya daftar ini dapat mengungkap informasi mengenai huruf-huruf yang diketik pengguna selama ini. (Misalnya, jumlah pelengkapan otomatis yang dihasilkan untuk "th" jauh lebih besar daripada jumlah untuk "zo.")
- **Pencarian basis data**: Pertimbangkan pengguna yang menanyakan database untuk semua catatan yang cocok dengan beberapa istilah pencarian. Jumlah catatan yang dikembalikan dapat mengungkapkan banyak informasi tentang apa yang dicari pengguna. Hal ini bisa sangat merugikan jika pengguna mencari informasi medis dan kueri tersebut mengungkapkan informasi tentang penyakit yang diderita pengguna.
- **Data terkompresi**: Jika teks biasa dikompresi sebelum dienkripsi, maka informasi tentang teks biasa mungkin terungkap meskipun hanya data dengan panjang tetap yang dienkripsi. (Oleh karena itu, skema enkripsi seperti itu tidak memenuhi Definisi 3.8.) Misalnya, teks biasa terkompresi pendek akan menunjukkan bahwa teks biasa asli (tidak terkompresi) memiliki banyak redundansi. Jika musuh dapat mengontrol sebagian dari apa yang dienkripsi, kerentanan ini dapat memungkinkan musuh mempelajari informasi tambahan tentang teks biasa; telah terbukti kemungkinan untuk menggunakan serangan semacam ini (serangan CRIME) terhadap lalu lintas HTTP terenkripsi untuk mengungkap cookie sesi rahasia.

Saat menggunakan enkripsi, seseorang harus menentukan apakah kebocoran panjang teks biasa merupakan suatu kekhawatiran dan, jika demikian, mengambil langkah-langkah untuk mengurangi atau mencegah kebocoran tersebut dengan memasukkan semua pesan ke panjang tertentu yang telah ditentukan sebelum mengenkripsinya.

Keamanan Semantik

Kami memotivasi definisi enkripsi aman dengan mengatakan bahwa musuh tidak mungkin mempelajari sebagian informasi tentang teks biasa dari teks tersandi. Namun, definisi indistinguishability terlihat sangat berbeda. Seperti yang telah kami sebutkan, Definisi 3.8 setara dengan definisi yang disebut keamanan semantik yang memformalkan gagasan bahwa sebagian informasi tidak dapat dipelajari. Kami membangun definisi tersebut dengan membahas dua gagasan yang lebih lemah dan menunjukkan bahwa kedua gagasan tersebut tersirat dalam ketidakmungkinan membedakan.

Kita mulai dengan menunjukkan bahwa indistinguishability berarti bahwa ciphertext tidak membocorkan informasi tentang bit-bit individual dari plaintext. Secara formal, katakanlah skema enkripsi (Enc, Des) aman untuk EAV (ingat ketika Gen dihilangkan, kuncinya

adalah string n-bit yang seragam), dan $m \in \{0,1\}^{\ell}$ seragam. Kemudian kami menunjukkan bahwa untuk indeks i apa pun, tidak mungkin menebak m^i dari $\mathrm{Enc}_k(m)$ (di mana, di bagian ini, mi menunjukkan bit ke-i dari m) dengan probabilitas yang jauh lebih baik daripada 1/2.

TEOREMA 3.10 Misalkan $\prod = (\mathrm{Enc}, \mathrm{Des})$ adalah skema enkripsi kunci pribadi dengan panjang tetap untuk pesan dengan panjang ℓ yang memiliki enkripsi yang tidak dapat dibedakan dengan adanya penyadap. Kemudian untuk semua musuh PPT $\mathcal{A}$ dan semua i ∈ {1, . . . , ℓ }, ada fungsi yang dapat diabaikan sehingga

$$\Pr\left[\mathcal{A}\left(1^n, \mathrm{Enc}_k(\mathrm{m})\right) = \mathrm{m}^i\right] \leq \frac{1}{2} + negl(n),$$

dimana probabilitas diambil alih seragam $m \in \{0,1\}^{\ell}\ dan\ k \in \{0,1\}^n$, keacakan $\mathcal{A}$, dan keacakan Enc.

BUKTI Ide di balik pembuktian teorema ini adalah jika bit ke-i dari m dapat ditebak dari Enck(m), maka enkripsi pesan m_0 dan m_1 yang bit-bit ke-i-nya berbeda juga dapat dibedakan. Kami meresmikannya melalui pembuktian dengan reduksi, di mana kami menunjukkan cara menggunakan musuh efisien $\mathcal{A}$ untuk membangun musuh efisien $\mathcal{A}'$ sehingga jika $\mathcal{A}$ melanggar gagasan keamanan teorema untuk Π, maka $\mathcal{A}'$ melanggar definisi ketidakmampuan untuk dibedakan untuk Π. (Lihat Bagian 3.3) Karena Π memiliki enkripsi yang tidak dapat dibedakan, maka Π juga harus aman sesuai dengan teorema.

Perbaiki musuh PPT yang sewenang-wenang $\mathcal{A}$ dan $i \in \{1, \ldots, \ell\}$. Misalkan $I_0 \subset \{0,1\}^{\ell}$ adalah himpunan semua string yang bit ke-i-nya adalah 0, dan misalkan $I_1 \subset \{0,1\}^{\ell}$ adalah himpunan semua string yang bit ke-i-nya adalah 1. Kita peroleh

$$\begin{aligned}
&\Pr\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m)) = m^i\right] \\
&= \frac{1}{2} \cdot \Pr_{m_0 \leftarrow I_0}\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m_0)) = 0\right] + \frac{1}{2} \cdot \Pr_{m_1 \leftarrow I_1}\left[\mathcal{A}(1^n, \mathsf{Enc}_k(m_1)) = 1\right].
\end{aligned}$$

Buatlah musuh penyadap berikut A’:
Musuh $\mathcal{A}$’:

1. Pilih seragam $m_0 \in I_0 dan\ m_1 \in I_1$. Keluaran $m_0 m_1$.
2. Setelah mengamati ciphertext c, aktifkan $\mathcal{A}(1^n, c)$. Jika A menghasilkan 0, keluaran b’ = 0; jika tidak, keluaran b’ = 1.

A’ berjalan dalam waktu polinomial sejak $\mathcal{A}$ berjalan.

Berdasarkan definisi eksperimen $PrivK^{eav}_{\mathcal{A}',\Pi}(n)$kita mendapatkan bahwa $\mathcal{A}'$ berhasil jika dan hanya jika $\mathcal{A}$ mengeluarkan b setelah menerima $\mathrm{Enc}_k(m_b)$. Jadi

$$\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)=1\right]\\
&=\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_b))=b\right]\\
&=\frac{1}{2}\cdot\Pr_{m_0\leftarrow I_0}\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_0))=0\right]+\frac{1}{2}\cdot\Pr_{m_1\leftarrow I_1}\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m_1))=1\right]\\
&=\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=m^i\right].
\end{aligned}$$

Dengan asumsi bahwa (Enc, Des) memiliki enkripsi yang tidak dapat dibedakan dengan adanya penyadap, terdapat fungsi yang dapat diabaikan sehingga $\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi}(n)=1\right]\leq\frac{1}{2}+\mathsf{negl}(n)$. Kami menyimpulkan bahwa

$$\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=m^i\right]\leq\frac{1}{2}+\mathsf{negl}(n),$$

melengkapi buktinya.

Kami selanjutnya mengklaim, secara kasar, bahwa indistinguishability berarti bahwa tidak ada musuh PPT yang dapat mempelajari fungsi apa pun dari plaintext yang diberi ciphertext, terlepas dari distribusi pesan yang dikirim. Hal ini dimaksudkan untuk menangkap gagasan bahwa tidak ada informasi tentang teks biasa yang dibocorkan oleh teks tersandi yang dihasilkan. Namun persyaratan ini tidak mudah untuk didefinisikan secara formal. Untuk mengetahui alasannya, perhatikan bahwa bahkan untuk kasus yang dipertimbangkan di atas, mudah untuk menghitung bit ke-i dari m jika m dipilih, katakanlah, secara seragam dari himpunan semua string yang bit ke-i-nya adalah 0 (daripada secara seragam dari $\{0,1\}^\ell$). Jadi, yang sebenarnya ingin kita katakan adalah jika terdapat musuh yang dapat menghitung $f(m)$ dengan benar dengan probabilitas tertentu ketika diberikan Enck(m), maka terdapat musuh yang dapat menghitung $f(m)$ dengan benar dengan probabilitas yang sama tanpa diberi ciphertext sama sekali (dan hanya mengetahui distribusi m). Berikut ini kita fokus pada kasus ketika m dipilih secara seragam dari suatu himpunan $S \subseteq \{0,1\}^\ell$.

TEOREMA 3.11 *Misalkan (Enc, Des) adalah skema enkripsi kunci pribadi dengan panjang tetap untuk pesan dengan panjang l yang memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap. Maka untuk setiap algoritma PPT terdapat algoritma PPT $\mathcal{A}'$ sehingga untuk setiap $S \subseteq \{0,1\}^\ell$.dan setiap fungsi $f:\{0,1\}^\ell \to \{0,1\}$ terdapat fungsi yang dapat diabaikan sehingga*:

$$\left|\Pr\left[\mathcal{A}(1^n,\mathsf{Enc}_k(m))=f(m)\right]-\Pr\left[\mathcal{A}'(1^n)=f(m)\right]\right|\leq\mathsf{negl}(n),$$

dimana probabilitas pertama diambil alih pilihan seragam dari $k \in \{0,1\}^n$ dan $m \in S$, keacakan $\mathcal{A}$, dan keacakan Enc, dan probabilitas kedua diambil alih pilihan seragam $m \in S$ dan keacakan dari $\mathcal{A}'$.

BUKTI (Sketsa) Fakta bahwa (Enc, Des) aman untuk EAV berarti, untuk setiap $S \subseteq \{0,1\}^{\ell}$, tidak ada musuh PPT yang dapat membedakan antara $Enc_k(m)$ (untuk seragam m ∈ S) dan $Enc_k(1^{\ell})$. Sekarang pertimbangkan probabilitas bahwa A berhasil menghitung f (m) jika diberi $Enc_k(m)$. Kami mengklaim bahwa f (m) harus berhasil dihitung dengan $Enc_k(1^{\ell})$ dengan probabilitas yang hampir sama; jika tidak, dapat digunakan untuk membedakan antara $Enc_k(m)$ dan $Enc_k(1^{\ell})$. Pembedanya mudah dibuat: pilih seragam $m \in S$, dan keluaran $m_0 = m, m_1 = 1^{\ell}$. Ketika diberikan ciphertext c yang merupakan enkripsi $m_0 \; atau \; m_1$, aktifkan $\mathcal{A}(1^n, c)$ dan keluaran 0 jika $\mathcal{A}$ dan hanya jika $\mathcal{A}$ keluaran $f(m)$. Jika keluaran $f(m)$ bila diberi enkripsi m dengan probabilitas berbeda nyata dengan probabilitas keluaran $f(m)$ bila diberi enkripsi1^{ℓ}, maka pembeda yang dijelaskan melanggar Definisi 3.9.

Persamaan di atas menyarankan algoritma berikut $\mathcal{A}'$ yang tidak menerima $c = Enc_k(m)$, namun menghitung $f\ (m)$ hampir sama baiknya $\mathcal{A}$ dengan $\mathcal{A}'(1^n)$ memilih kunci yang seragam $k \in \{0,1\}^n$, memanggil $\mathcal{A}\ pada\ c \rightarrow\ Enc_k(1^{\ell})$, dan mengeluarkan apa pun yang dilakukan $\mathcal{A}$. Berdasarkan persamaan di atas, kita mendapatkan bahwa $\mathcal{A}$ menghasilkan $f(m)$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}'$ dengan probabilitas yang hampir sama dengan ketika ia menerima$Enc_k(m)$. Jadi, $\mathcal{A}'$memenuhi sifat yang disyaratkan oleh klaim.

Keamanan Semantik

Definisi lengkap jaminan keamanan semantik jauh lebih banyak daripada properti yang dipertimbangkan dalam Teorema 3.11. Definisi ini memungkinkan panjang teks biasa bergantung pada parameter keamanan, dan pada dasarnya memungkinkan distribusi sewenang-wenang pada teks biasa. (Sebenarnya, kami hanya memperbolehkan distribusi sampel yang efisien. Ini berarti bahwa ada beberapa algoritma waktu polinomial probabilistik $\boldsymbol{Samp}\ sehingga\ \boldsymbol{Samp}(1^n)$ mengeluarkan pesan sesuai dengan distribusinya.) Definisi ini juga memperhitungkan informasi "*eksternal*" yang sewenang-wenang $h\ (m)$ tentang teks biasa yang mungkin dibocorkan ke musuh melalui cara lain (misalnya, karena pesan yang sama m juga digunakan untuk tujuan lain).

DEFINISI 3.12 Skema enkripsi kunci pribadi (Enc, Des) aman secara semantik dengan adanya penyadap jika untuk setiap algoritma PPT $\mathcal{A}$ terdapat algoritma PPT $\mathcal{A}'$ sehingga untuk algoritma PPT apa pun $\boldsymbol{Samp}$ dan fungsi komputasi waktu polinomial f dan h, hal berikut dapat diabaikan:

$$\left| \Pr[\mathcal{A}(1^n, \mathrm{Enc}_k(m), h(m)) = f(m)] - \Pr[\mathcal{A}'(1^n, |m|, h(m)) = f(m)] \right|,$$

dimana probabilitas pertama diambil alih seragam $k \in \{0,1\}^n, m$ keluaran oleh $\boldsymbol{Samp}(1^n)$, keacakan $\mathcal{A}$, dan keacakan Enc, dan probabilitas kedua diambil alih m keluaran oleh $\boldsymbol{Samp}(1^n)$ dan keacakan $\mathcal{A}'$.

Musuh $\mathcal{A}$ diberikan ciphertext $Enc_k(m)$ serta informasi eksternal $h(m)$, dan mencoba menebak nilai $f(m)$. Algoritma $\mathcal{A}'$ juga mencoba menebak nilai $f(m)$, namun yang diberikan hanya $h(m)$ dan panjang m. Persyaratan keamanan menyatakan bahwa probabilitas $\mathcal{A}$ untuk

menebak dengan benar $f(m)$ hampir sama dengan probabilitas $\mathcal{A}'$. Maka secara intuitif, ciphertext $Enc_k(m)$) tidak mengungkapkan informasi tambahan apa pun tentang nilai $f(m)$. Definisi 3.12 merupakan rumusan yang sangat kuat dan meyakinkan mengenai jaminan keamanan yang harus diberikan oleh skema enkripsi. Namun, lebih mudah untuk bekerja dengan definisi tidak dapat dibedakan (Definisi 3.8). Untungnya, definisinya setara:

TEOREMA 3.13 *Suatu skema enkripsi kunci pribadi mempunyai enkripsi yang tidak dapat dibedakan di hadapan penyadap jika dan hanya jika skema tersebut aman secara semantik di hadapan penyadap.*

Ke depan, kesetaraan serupa antara keamanan semantik dan indistinguishability diketahui untuk semua definisi yang kami sajikan dalam bab ini dan juga dalam Bab 11. Oleh karena itu, kami dapat menggunakan indistinguishability sebagai definisi kerja kami, sambil yakin bahwa jaminan tersebut tercapai. adalah keamanan semantik.

3.3 MEMBANGUN SKEMA ENKRIPSI YANG AMAN

Setelah mendefinisikan apa arti keamanan skema enkripsi, pembaca mungkin berharap kita segera beralih ke konstruksi skema enkripsi aman. Namun sebelum melakukan hal tersebut, kita perlu mengenalkan pengertian *generator pseudorandom (PRG$_S$)* dan *stream cipher,* yang merupakan elemen penting dalam enkripsi kunci pribadi. Hal ini, pada gilirannya, akan mengarah pada diskusi tentang keacakan semu, yang memainkan peran mendasar dalam kriptografi pada umumnya dan enkripsi kunci privat pada khususnya.

Generator Pseudorandom dan Stream Cipher

Generator pseudorandom G adalah algoritma yang efisien dan deterministik untuk mengubah string pendek dan seragam yang disebut seed menjadi string keluaran yang lebih panjang dan "tampak seragam" (atau "acak semu"). Dengan kata lain, generator pseudorandom menggunakan sejumlah kecil keacakan sebenarnya untuk menghasilkan sejumlah besar keacakan semu. Hal ini berguna bila diperlukan bit acak (tampak) dalam jumlah besar, karena menghasilkan bit acak yang sebenarnya sulit dan lambat. (Lihat pembahasan di awal Bab 2.) Memang benar, generator pseudorandom telah dipelajari setidaknya sejak tahun 1940an ketika mereka diusulkan untuk menjalankan simulasi statistik. Dalam konteks tersebut, peneliti mengusulkan berbagai uji statistik yang harus dilalui oleh generator pseudorandom agar dianggap "baik". Sebagai contoh sederhana, bit pertama dari output generator pseudorandom harus sama dengan 1 dengan probabilitas yang sangat dekat dengan 1/2 (di mana probabilitas diambil alih pemilihan benih yang seragam), karena bit pertama dari sebuah generator pseudorandom harus sama dengan 1. string seragam sama dengan 1 dengan probabilitas tepat 1/2. Faktanya, paritas dari setiap subset tetap dari bit keluaran juga harus 1 dengan probabilitas sangat dekat dengan 1/2. Uji statistik yang lebih kompleks juga dapat dipertimbangkan.

Pendekatan historis untuk menentukan kualitas beberapa calon generator acak semu ini bersifat ad hoc, dan tidak jelas apakah melewati serangkaian uji statistik sudah cukup untuk menjamin kelayakan penggunaan calon generator acak semu untuk beberapa aplikasi.

(Khususnya, mungkin ada uji statistik lain yang berhasil membedakan keluaran generator dari bit acak sebenarnya.) Pendekatan historis bahkan lebih bermasalah ketika menggunakan generator pseudorandom untuk aplikasi kriptografi; dalam situasi tersebut, keamanan dapat dikompromikan jika penyerang mampu membedakan keluaran generator dari seragam, dan kita tidak mengetahui sebelumnya strategi apa yang mungkin digunakan penyerang.

Pertimbangan di atas memotivasi pendekatan kriptografi untuk mendefinisikan generator pseudorandom pada tahun 1980an. Realisasi dasarnya adalah bahwa generator pseudorandom yang baik harus lulus semua uji statistik (efisien). Artinya, untuk uji statistik (atau pembeda) D yang efisien, probabilitas bahwa D mengembalikan 1 ketika diberikan keluaran generator pseudorandom harus mendekati probabilitas bahwa D mengembalikan 1 ketika diberikan string seragam dengan panjang yang sama. Maka secara informal, output dari generator pseudorandom harus "terlihat seperti" string yang seragam bagi pengamat yang efisien.

(Kami menekankan bahwa, secara formal, tidak masuk akal untuk mengatakan bahwa string tetap apa pun adalah "pseudorandom," sama seperti tidak masuk akal untuk menyebut string tetap sebagai "acak." Sebaliknya, pseudorandomness adalah properti dari Meskipun demikian, terkadang kita secara informal menyebut string yang diambil sampelnya berdasarkan distribusi seragam sebagai "string seragam", dan string yang dihasilkan oleh generator pseudorandom disebut "string acak semu.")

Perspektif lain diperoleh dengan mendefinisikan apa yang dimaksud dengan distribusi menjadi pseudorandom. Biarkan Dist menjadi distribusi pada string l-bit. (Ini berarti bahwa Dist memberikan beberapa probabilitas untuk setiap string dalam $\{0,1\}^{\ell}$; pengambilan sampel dari Dist berarti kita memilih string l-bit berdasarkan distribusi probabilitas ini.) Secara informal, Dist adalah *pseudorandom* jika eksperimen di mana sebuah string diambil sampelnya. from Dist tidak dapat dibedakan dari eksperimen yang mengambil sampel string seragam dengan panjang ℓ. (Sebenarnya, karena kita berada dalam keadaan asimtotik, kita perlu berbicara tentang keacakan semu dari rangkaian distribusi $Dist = Dist_n$, di mana distribusi $Dist_n$ digunakan untuk parameter keamanan n. Kita mengabaikan poin ini dalam diskusi kita saat ini.) Lebih tepatnya, algoritma waktu polinomial mana pun tidak dapat mengetahui (lebih baik daripada menebak) apakah algoritma tersebut diberikan sampel string berdasarkan Dist, atau apakah ia diberikan string $\ell - bit$ yang seragam. Ini berarti bahwa string pseudorandom sama bagusnya dengan string seragam, selama kita hanya mempertimbangkan pengamat waktu polinomial. Sama seperti indistinguishability yang merupakan relaksasi komputasi dari kerahasiaan yang sempurna, pseudorandomness adalah relaksasi komputasi dari keacakan yang sebenarnya. (Kami akan menggeneralisasi perspektif ini ketika kita membahas gagasan tidak dapat dibedakan di Bab 7.)

Sekarang misalkan $G : \{0,1\}^n \rightarrow \{01\}^{\ell}$ adalah sebuah fungsi, dan definisikan Dist sebagai distribusi pada string $\ell - bit$ yang diperoleh dengan memilih seragam $s \in \{0,1\}^n$ dan mengeluarkan $G(s)$. Maka G adalah generator pseudorandom jika dan hanya jika distribusi Dist adalah pseudorandom.

Definisi formal. Seperti dibahas di atas, G adalah generator pseudorandom jika tidak ada pembeda yang efisien yang dapat mendeteksi apakah ia diberi keluaran string oleh G atau string yang dipilih secara seragam dan acak. Seperti dalam Definisi 3.9, hal ini diformalkan dengan mensyaratkan bahwa setiap algoritma yang efisien menghasilkan 1 dengan probabilitas yang hampir sama ketika diberikan $G(s)$ (untuk seed s yang seragam) atau string yang seragam. (Untuk definisi yang setara dengan Definisi 3.8) Kita memperoleh definisi dalam pengaturan asimtotik dengan membiarkan parameter keamanan n menentukan panjang benih. Kami kemudian mendesak agar G dapat dihitung dengan algoritma yang efisien. Secara teknis, kami juga mengharuskan keluaran G lebih panjang dari masukannya; jika tidak, G tidak terlalu berguna atau menarik.

DEFINISI 3.14 Misalkan ℓ adalah suatu polinomial dan misalkan G adalah algoritma waktu polinomial deterministik sehingga untuk sembarang n dan sembarang masukan $s \in \{0,1\}^n$, hasilnya $G(s)$ adalah string dengan panjang $\ell(n)$. Kita mengatakan bahwa G adalah generator pseudorandom jika kondisi berikut dipenuhi:

1) **(Ekspansi)** Untuk setiap n dinyatakan bahwa $\ell(n) > n$.
2) **(Pseudorandomness)** Untuk setiap algoritma PPT D, ada fungsi yang dapat diabaikan sehingga

$$\left| \Pr[D(G(s)) = 1] - \Pr[D(r) = 1] \right| \leq \mathrm{negl}(n),$$

dimana probabilitas pertama diambil alih pilihan seragam $s \in \{0,1\}^n$ dan keacakan D, dan probabilitas kedua diambil alih pilihan seragam $r \in \{0,1\}^{\ell(n)}$dan keacakan D.
Kita menyebut ℓ sebagai faktor ekspansi G. Kami memberikan contoh generator pseudorandom yang tidak aman untuk memahami definisinya.

Contoh 3.15

Definisikan $G(s)$ pada keluaran s diikuti dengan, $\bigoplus_{i=1}^{n} s_i$ sehingga faktor muai G adalah $\ell(n) = n + 1$. Keluaran G dapat dengan mudah dibedakan dari seragam. Pertimbangkan pembeda efisien D berikut ini: pada masukan string w, keluaran 1 jika dan hanya jika bit akhir w sama dengan XOR semua bit sebelumnya dari w. Karena properti ini berlaku untuk semua string yang dihasilkan oleh G, kita mempunyai $Pr[D(G(s)) = 1] = 1$. Di sisi lain, jika w seragam, bit terakhir dari w juga seragam sehingga $Pr[D(w) = 1] = \frac{1}{2}$. Kuantitas $|\frac{1}{2} - 1|$ adalah konstan, tidak dapat diabaikan, sehingga G ini bukanlah generator pseudorandom. (Perhatikan bahwa D tidak selalu "benar", karena terkadang menghasilkan 1 meskipun diberi string yang seragam. Hal ini tidak mengubah fakta bahwa D adalah pembeda yang baik.)

Diskusi. Distribusi keluaran generator pseudorandom G jauh dari seragam. Untuk melihatnya, perhatikan kasus $\ell(n) = 2n$ sehingga G menggandakan panjang masukannya. Di bawah distribusi seragam pada $\{0,1\}^{2n}$, masing-masing dari 2^{2n} kemungkinan string dipilih dengan

probabilitas tepat 2^{-2n} . Sebaliknya, pertimbangkan distribusi keluaran G (ketika G dijalankan pada benih yang seragam). Ketika G menerima masukan dengan panjang n, jumlah string berbeda dalam rentang G paling banyak adalah 2^n. Fraksi string dengan panjang 2^n yang berada dalam rentang G paling banyak adalah $2^n/2^{2n} = 2^{-n}$, dan kita melihat bahwa sebagian besar string dengan panjang 2^n tidak muncul sebagai keluaran dari G.

Hal ini secara khusus berarti bahwa membedakan antara string acak dan string pseudorandom sangatlah mudah jika diberikan jangka waktu yang tidak terbatas. Misalkan G seperti di atas dan pertimbangkan pembeda waktu eksponensial D yang berfungsi sebagai berikut: $D(w)$ menghasilkan 1 jika dan hanya jika terdapat $s \epsilon \{0,1\}^n$ sehingga $G(s) = w$. (Perhitungan ini dilakukan dalam waktu eksponensial dengan menghitung $G(s)$ secara mendalam untuk setiap $s \epsilon \{0,1\}^n$. Ingat bahwa berdasarkan prinsip Kerckhoffs, spesifikasi G diketahui D.) Sekarang, jika w dikeluarkan oleh G, maka D menghasilkan 1 dengan probabilitas 1. Sebaliknya, jika w terdistribusi secara merata pada $\{0,1\}^{2n}$, maka probabilitas terdapatnya s dengan $G(s) = w$ paling banyak adalah 2^{-n}, sehingga D menghasilkan 1 dalam kasus ini dengan probabilitas paling banyak 2^{-n} Jadi

$$\left|\Pr[D(r)=1] - \Pr[D(G(s))=1]\right| \geq 1 - 2^{-n},$$

yang besar. Ini hanyalah contoh lain dari serangan brute force, dan tidak bertentangan dengan keacakan semu G karena serangan tersebut tidak efisien.

Benih dan panjangnya

Benih untuk generator pseudorandom dianalogikan dengan kunci kriptografi yang digunakan oleh skema enkripsi, dan benih tersebut harus dipilih secara seragam dan dirahasiakan dari musuh mana pun. Hal penting lainnya, yang terlihat dari pembahasan serangan brute force di atas, adalah bahwa s harus cukup panjang sehingga tidak mungkin untuk menghitung semua kemungkinan benih. Dalam pengertian asimtotik, hal ini diatasi dengan mengatur panjang benih sama dengan parameter keamanan, sehingga pencarian menyeluruh atas semua benih yang mungkin memerlukan waktu eksponensial. Dalam prakteknya, benih harus cukup panjang sehingga tidak mungkin untuk mencoba semua kemungkinan benih dalam jangka waktu tertentu.

Tentang keberadaan generator pseudorandom. Apakah generator pseudorandom ada? Tampaknya sulit untuk membangunnya, dan orang mungkin bertanya apakah ada algoritma yang memenuhi Definisi 3.14. Meskipun kita tidak tahu bagaimana membuktikan keberadaan generator pseudorandom tanpa syarat, kita punya alasan kuat untuk meyakini keberadaannya. Pertama, mereka dapat dibangun dengan asumsi yang agak lemah bahwa fungsi satu arah memang ada (hal ini benar jika permasalahan tertentu seperti memfaktorkan bilangan besar sulit dilakukan); hal ini akan dibahas secara rinci di Bab 7. Kita juga mempunyai beberapa konstruksi praktis dari calon generator pseudorandom yang disebut stream cipher yang tidak diketahui pembeda efisiennya. (Kemudian, kami akan memperkenalkan primitif yang lebih kuat yang disebut cipher blok.) Selanjutnya kami memberikan gambaran umum tingkat tinggi tentang cipher aliran, dan membahas cipher aliran konkret di Bab 6.

Sandi Aliran

Definisi kami tentang generator pseudorandom dibatasi dalam dua hal: faktor ekspansi tetap, dan generator menghasilkan seluruh keluarannya dalam "satu tembakan". Stream cipher, yang digunakan dalam praktik untuk membuat instance generator pseudorandom, bekerja dengan cara yang agak berbeda. Bit keluaran pseudorandom dari stream cipher diproduksi secara bertahap dan sesuai permintaan, sehingga aplikasi dapat meminta bit pseudorandom sebanyak yang diperlukan. Hal ini meningkatkan efisiensi (karena aplikasi dapat meminta lebih sedikit bit, jika mencukupi) dan fleksibilitas (karena tidak ada batasan atas jumlah bit yang dapat diminta).

Secara formal, kami melihat stream cipher2 sebagai sepasang algoritma deterministik (Init, GetBits) di mana:

- Init mengambil input berupa seed s dan *vektor inisialisasi* opsional IV, dan menghasilkan status awal st_0.
- GetBits mengambil informasi status masukan st_i, dan mengeluarkan sedikit y dan status terkini st_{i+1}. (Dalam praktiknya, y adalah blok yang terdiri dari beberapa bit; kami memperlakukan y sebagai satu bit di sini untuk tujuan umum dan kesederhanaan.)

Dengan adanya stream cipher dan faktor ekspansi apa pun yang diinginkan ℓ, kita dapat mendefinisikan algoritma G_ℓ yang memetakan input dengan panjang n ke output dengan panjang $\ell(n)$. Algoritme hanya menjalankan Init, dan kemudian berulang kali menjalankan GetBits sebanyak ℓ kali.

ALGORITHM 3.16
Constructing G_ℓ from (Init, GetBits)

Input: Seed s and optional initialization vector IV
Output: $y_1, \ldots, y_\ell$

$\mathsf{st}_0 := \mathsf{Init}(s, IV)$
for $i = 1$ to ℓ:
  $(y_i, \mathsf{st}_i) := \mathsf{GetBits}(\mathsf{st}_{i-1})$
return $y_1, \ldots, y_\ell$

Sebuah stream cipher aman dalam arti dasar jika tidak memerlukan IV dan untuk polinomial apa pun ℓ dengan $\ell(n) > n$, fungsi G_ℓ yang dibangun di atas adalah generator acak semu dengan faktor ekspansi ℓ. Kami membahas secara singkat satu kemungkinan gagasan keamanan untuk stream cipher yang menggunakan IV di Bagian 3.6

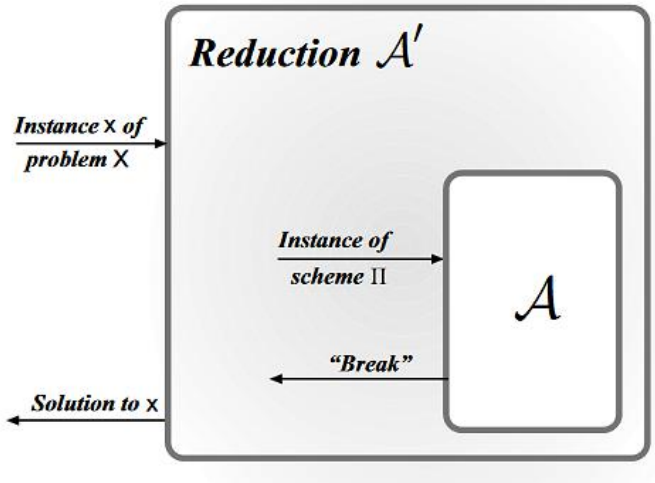
Pembuktian dengan Reduksi

Jika kita ingin membuktikan bahwa konstruksi tertentu aman secara komputasi, maka kita harus mengandalkan asumsi yang belum terbukti (kecuali skema tersebut aman secara informasi dan teoritis). Strategi kami adalah berasumsi bahwa beberapa masalah matematika itu sulit, atau bahwa beberapa kriptografi primitif tingkat rendah aman, dan kemudian membuktikan bahwa konstruksi tertentu berdasarkan masalah/primitif ini aman berdasarkan

asumsi ini. Pada Bagian 1.4 kami telah menjelaskan secara rinci mengapa pendekatan ini lebih disukai sehingga kami tidak mengulangi argumen tersebut di sini.

Pembuktian bahwa konstruksi kriptografi aman selama beberapa masalah yang mendasarinya sulit umumnya dilanjutkan dengan menyajikan pengurangan eksplisit yang menunjukkan bagaimana mengubah musuh $\mathcal{A}$ efisien yang berhasil "memecah" konstruksi tersebut menjadi algoritma $\mathcal{A}$ yang efisien $\mathcal{A}$ ' yang memecahkan masalah tersebut. masalah yang dianggap sulit. Karena ini sangat penting, kami akan menelusuri garis besar langkah-langkah pembuktian tersebut secara mendetail. (Kita akan melihat banyak contoh nyata dalam buku ini, dimulai dengan pembuktian Teorema 3.18.) Kita mulai dengan asumsi bahwa beberapa masalah X tidak dapat diselesaikan (dalam pengertian tertentu) dengan algoritma waktu polinomial apa pun, kecuali dengan probabilitas yang dapat diabaikan. Kami ingin membuktikan bahwa beberapa konstruksi kriptografi Π aman (sekali lagi, dalam beberapa hal hal itu didefinisikan secara tepat). Pembuktian dilakukan melalui langkah-langkah berikut (lihat juga Gambar 3.1):

1. Perbaiki beberapa serangan musuh $\mathcal{A}$ yang efisien (yaitu, waktu polinomial probabilistik) Π. Nyatakan probabilitas keberhasilan musuh ini dengan $\varepsilon(n)$.
2. Buatlah algoritma yang efisien $\mathcal{A}'$, yang disebut "reduksi," yang mencoba memecahkan masalah X menggunakan musuh $\mathcal{A}$ sebagai subrutin. Hal penting di sini adalah bahwa $\mathcal{A}'$ tidak tahu $\mathcal{A}$ apa-apa tentang cara kerjanya; satu-satunya hal yang diketahui ' adalah mengharapkan serangan Π. Jadi, dengan beberapa contoh masukan x dari masalah X, algoritma kita $\mathcal{A}'$ akan melakukan simulasi untuk $\mathcal{A}$ contoh Π sehingga:
   (a) Sejauh yang diketahui $\mathcal{A}$, ia berinteraksi dengan Π. Artinya, tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}'$ harus didistribusikan secara identik (atau setidaknya mendekati) tampilan $\mathcal{A}$ ketika berinteraksi dengan Π itu sendiri.
   (b) Jika berhasil "mematahkan" contoh Π yang disimulasikan oleh $\mathcal{A}'$, hal ini akan memungkinkan $\mathcal{A}$' menyelesaikan contoh x yang diberikan, setidaknya dengan probabilitas polinomial terbalik $1/p(n)$.



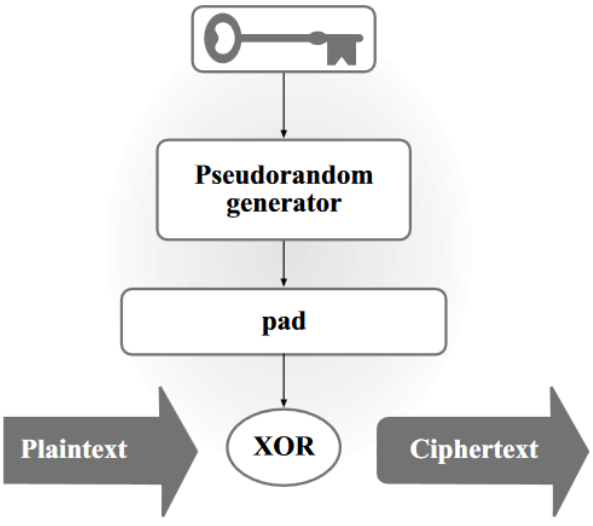
Gambar 3.1: A Tinjauan Tingkat Tinggi Dari Bukti Keamanan Dengan Pengurangan.

3. Secara keseluruhan, $2(a)\ dan\ 2(b)$ menyiratkan bahwa $\mathcal{A}'$ menyelesaikan X dengan probabilitas $\varepsilon(n)/p(n)$. Jika $\varepsilon(n)$ tidak dapat diabaikan, maka $\varepsilon(n)/p(n)$ juga tidak dapat diabaikan. Terlebih lagi, jika $\mathcal{A}$ efisien maka kita memperoleh algoritma $\mathcal{A}'$ yang efisien menyelesaikan X dengan probabilitas yang tidak dapat diabaikan, bertentangan dengan asumsi awal.
4. Mengingat asumsi kita mengenai X, kita menyimpulkan bahwa tidak ada musuh efisien yang berhasil memecahkan Π dengan probabilitas yang tidak dapat diabaikan. Dengan kata lain, Π aman secara komputasi.

Pada bagian berikut ini kami akan mengilustrasikan gagasan di atas dengan tepat: kami akan menunjukkan cara menggunakan generator pseudorandom G untuk membuat skema enkripsi; kami membuktikan skema enkripsi aman dengan menunjukkan bahwa penyerang mana pun yang dapat "merusak" skema enkripsi dapat digunakan untuk membedakan keluaran G dari string seragam. Dengan asumsi bahwa G adalah generator pseudorandom, maka skema enkripsi aman.

Skema Enkripsi Panjang Tetap yang Aman

Generator pseudorandom menyediakan cara alami untuk membangun skema enkripsi yang aman dengan panjang tetap dengan kunci yang lebih pendek dari pesan. Ingatlah bahwa dalam one-time pad (lihat Bagian 2.2), enkripsi dilakukan dengan meng-XOR pad acak dengan pesan tersebut. Kesimpulannya adalah kita bisa menggunakan pad pseudorandom sebagai gantinya. Namun, alih-alih berbagi pad pseudorandom yang panjang ini, pengirim dan penerima dapat berbagi seed yang digunakan untuk menghasilkan pad tersebut bila diperlukan (lihat Gambar 3.2); benih ini akan lebih pendek dari pad dan karenanya lebih pendek dari pesannya. Mengenai keamanan, intuisinya adalah bahwa string pseudorandom "terlihat acak" bagi musuh dengan waktu polinomial mana pun sehingga penyadap yang dibatasi secara komputasi tidak dapat membedakan antara pesan yang dienkripsi menggunakan one-time pad atau pesan yang dienkripsi menggunakan "pseudo-" ini. -skema enkripsi pad waktu.



Gambar 3.2: Enkripsi Dengan Generator Pseudorandom.

Skema enkripsi, perbaiki beberapa panjang pesan ℓ dan biarkan G menjadi generator acak semu dengan faktor ekspansi ℓ (yaitu, $|G(s)| = \ell|s|$). Ingatlah bahwa skema enkripsi didefinisikan oleh tiga algoritma: algoritma pembangkitan kunci Gen, algoritma enkripsi Enc, dan algoritma dekripsi Des. Algoritma pembangkitan kunci adalah algoritma yang sepele: $Gen(1^n)$ hanya mengeluarkan kunci seragam k dari panjang n. Enkripsi bekerja dengan menerapkan G pada kunci (yang berfungsi sebagai seed) untuk mendapatkan pad yang kemudian di-XOR dengan teks biasa. Dekripsi menerapkan G ke kunci dan meng-XOR pad yang dihasilkan dengan ciphertext untuk memulihkan pesan. Skema ini dijelaskan secara formal dalam Konstruksi 3.17. Pada Bagian 3.6, kami menjelaskan bagaimana stream cipher digunakan untuk mengimplementasikan varian skema ini dalam praktiknya.

CONSTRUCTION 3.17

Let G be a pseudorandom generator with expansion factor ℓ. Define a private-key encryption scheme for messages of length ℓ as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it as the key.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^{\ell(n)}$, output the ciphertext
$$c := G(k) \oplus m.$$
- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c \in \{0,1\}^{\ell(n)}$, output the message
$$m := G(k) \oplus c.$$

Skema enkripsi kunci pribadi berdasarkan generator pseudorandom apa pun.

TEOREMA 3.18 Jika G adalah generator pseudorandom, maka Konstruksi 3.17 adalah skema enkripsi kunci pribadi dengan panjang tetap yang memiliki enkripsi yang tidak dapat dibedakan dengan adanya penyadap.

BUKTI Misalkan Π menyatakan Konstruksi 3.17. Kami menunjukkan bahwa Π memenuhi Definisi 3.8. Yakni, kami menunjukkan bahwa untuk musuh waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi yang dapat diabaikan sehingga

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n). \qquad (3.2)$$

Intuisinya adalah jika Π menggunakan pad seragam sebagai pengganti pad pseudorandom $G(k)$, maka skema yang dihasilkan akan identik dengan skema enkripsi pad satu kali dan $\mathcal{A}$ tidak akan dapat menebak dengan benar pesan mana yang dienkripsi dengan probabilitas apapun. lebih baik dari 1/2. Jadi, jika Persamaan (3.2) tidak berlaku maka $\mathcal{A}$ secara implisit harus membedakan keluaran G dari string acak. Kami memperjelas hal ini dengan menunjukkan pengurangan; yaitu, dengan menunjukkan cara menggunakan $\mathcal{A}$ untuk membuat pembeda yang efisien, dengan properti bahwa kemampuan D untuk membedakan

keluaran G dari string seragam berhubungan langsung dengan $\mathcal{A}$ kemampuan untuk menentukan pesan mana yang dienkripsi oleh $\prod$. Keamanan G kemudian menyiratkan keamanan $\prod$.

Biarlah menjadi musuh PPT yang sewenang-wenang. Kita membangun sebuah pembeda D yang mengambil sebuah string w sebagai masukan, dan yang tujuannya adalah untuk menentukan apakah w dipilih secara seragam (yaitu, w adalah "string acak") atau apakah w dihasilkan dengan memilih k yang seragam dan menghitung $w := G(k)$ (yaitu, w adalah "string acak semu"). Kami membangun D sehingga meniru eksperimen penyadapan untuk , seperti dijelaskan di bawah, dan mengamati apakah berhasil atau tidak. Jika $\mathcal{A}$ berhasil maka D menebak w pasti merupakan string pseudorandom, sedangkan jika $\mathcal{A}$ tidak berhasil maka D menebak w adalah string acak. Secara terperinci:

Pembeda D:
D diberikan sebagai input string $w \in \{0,1\}^{\ell(n)}$. (Kami berasumsi bahwa n dapat ditentukan dari $\ell(n)$.)

1. Jalankan $A(1^n)$ untuk mendapatkan sepasang pesan $m0, m1 \in \{0,1\}^{\ell(n)}$
2. Pilih bit yang seragam $b \in \{0,1\}$. Tetapkan $c := w \oplus m_b$.
3. Berikan c ke $\mathcal{A}$ dan dapatkan keluaran b'. Keluaran 1 jika $b' = b$, dan keluaran 0 sebaliknya.

D jelas berjalan dalam waktu polinomial (dengan asumsi A berjalan).

Sebelum menganalisis perilaku D, kita mendefinisikan skema enkripsi yang dimodifikasi $\widetilde{\prod} = (\widetilde{Gen}, \widetilde{Enc}, \widetilde{Dec})$ yang merupakan skema enkripsi one-time pad, kecuali sekarang kita memasukkan parameter keamanan yang menentukan panjang pesan untuk dienkripsi. Artinya, $Gen(1^n)$ mengeluarkan kunci seragam k dengan panjang $\ell(n)$, dan enkripsi pesan $m \in 2^{\ell(n)}$) menggunakan kunci $k \in \{0,1\}^{\ell(n)}$adalah teks tersandi $c := k \oplus m$. (Dekripsi dapat dilakukan seperti biasa, namun tidak penting untuk hal berikut.) Kerahasiaan sempurna dari one-time pad menyiratkan

$$\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] = \frac{1}{2}. \qquad (3.3)$$

Untuk menganalisis perilaku D, pengamatan utamanya adalah:

1. Jika w dipilih secara seragam dari $\{0,1\}^{\ell(n)}$, maka tampilan A ketika dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan A dalam eksperimen $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\widetilde{\Pi}}$. Hal ini karena ketika $\mathcal{A}$ dijalankan sebagai subrutin oleh $D(w)$ dalam hal ini, $\mathcal{A}$ diberikan ciphertext $c = w \oplus m_b$ dimana $w \epsilon \{0,1\}^{\ell(n)}$ seragam. Karena D menghasilkan 1 tepat ketika percobaan penyadapannya berhasil, maka kita mempunyai

$$\Pr_{w \leftarrow \{0,1\}^{\ell(n)}}[D(w) = 1] = \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right] = \frac{1}{2}. \qquad (3.4)$$

(Subskrip pada probabilitas pertama memperjelas bahwa w dipilih secara seragam dari $\{0,1\}^{\ell(n)}$ di sana.)

2. Jika w dihasilkan dengan memilih $k \in \{0,1\}^{(n)}$ yang seragam dan kemudian mengatur $w := G(k)$, tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan A dalam eksperimen $\mathrm{PrivK}^{\mathrm{eav}}_{\mathcal{A},\widetilde{\Pi}}(n)$. Hal ini karena A, ketika dijalankan sebagai subrutin oleh D, sekarang diberikan ciphertext $c = w \oplus m_b$ dimana $w = G(k)$ untuk seragam $k \in \{0,1\}^{(n)}$. Dengan demikian,

$$\Pr_{k\leftarrow\{0,1\}^n}[D(G(k))=1] = \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)=1\right]. \qquad (3.5)$$

Karena G adalah generator pseudorandom (dan karena D berjalan dalam waktu polinomial), kita tahu bahwa ada fungsi yang dapat diabaikan sehingga

$$\left|\frac{1}{2} - \Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)=1\right]\right| \leq \mathsf{negl}(n),$$

yang menyiratkan$\Pr\left[PrivK^{eav}_{\mathcal{A},\Pi}(n)=1\right] \leq \frac{1}{2} + negl(n)$. Karena $\mathcal{A}$ bersifat arbitrer Musuh PPT, ini melengkapi bukti bahwa Π memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap.

Sangat mudah untuk tersesat dalam rincian bukti dan bertanya-tanya apakah ada sesuatu yang diperoleh dibandingkan dengan catatan satu kali saja; lagi pula, one-time pad juga mengenkripsi pesan $\ell - bit$ dengan meng-XOR-nya dengan string $\ell - bit$! Inti dari konstruksinya, tentu saja, adalah bahwa string $\ell - bit$ G(k) bisa lebih panjang daripada kunci bersama k. Secara khusus, dengan menggunakan skema di atas, dimungkinkan untuk mengenkripsi file 1 Mb dengan aman hanya dengan menggunakan kunci 128-bit. Dengan mengandalkan kerahasiaan komputasi, kita telah menghindari hasil ketidakmungkinan dari Teorema 2.10, yang menyatakan bahwa setiap skema enkripsi yang sangat rahasia harus menggunakan kunci setidaknya sepanjang pesannya.

Pengurangan diskusi. Kami tidak membuktikan tanpa syarat bahwa Konstruksi 3.17 aman. Sebaliknya, kami membuktikan bahwa ini aman dengan asumsi bahwa G adalah generator pseudorandom. Pendekatan yang mengurangi keamanan konstruksi tingkat tinggi ke primitif tingkat rendah memiliki sejumlah keuntungan (seperti yang dibahas dalam Bagian 1.4). Salah satu keuntungannya adalah, secara umum, lebih mudah merancang primitif tingkat rendah daripada primitif tingkat tinggi; secara umum juga lebih mudah untuk menganalisis secara langsung algoritma G sehubungan dengan definisi tingkat yang lebih rendah daripada menganalisis skema yang lebih kompleks Π sehubungan dengan definisi tingkat yang lebih tinggi. Ini tidak berarti bahwa membuat generator pseudorandom itu “mudah”, hanya saja

lebih mudah daripada membuat skema enkripsi dari awal. (Dalam kasus ini skema enkripsi tidak melakukan apa pun kecuali meng-XOR keluaran generator pseudorandom dengan pesan sehingga hal ini tidak benar. Namun, kita akan melihat konstruksi yang lebih kompleks dan dalam kasus tersebut kemampuan untuk mengurangi tugas ke skema yang lebih sederhana sangatlah penting.) Keuntungan lainnya adalah ketika G yang sesuai telah dibangun, maka G dapat digunakan sebagai komponen dari berbagai skema lainnya.

Keamanan konkrit. Meskipun Teorema 3.18 dan pembuktiannya berada dalam keadaan asimtotik, kita dapat dengan mudah mengadaptasi pembuktiannya untuk membatasi keamanan konkrit skema enkripsi dalam kerangka keamanan konkrit G. Tetapkan beberapa nilai n untuk sisa pembahasan ini, dan misalkan Π sekarang menunjukkan Konstruksi 3.17 menggunakan nilai n ini. Asumsikan G adalah (t, ε)-pseudorandom (untuk nilai n yang diberikan), dalam arti bahwa untuk semua pembeda D yang berjalan dalam waktu paling banyak t kita punya

$$\left|\Pr[D(r)=1]-\Pr\left[D\left(G(s)\right)=1\right]\right| \leq \varepsilon. \qquad (3.6)$$

(Pikirkan $t \approx 2^{80}\ dan\ \varepsilon \approx 2^{-60}$, meskipun nilai presisinya tidak relevan untuk diskusi kita.) Kita mengklaim bahwa $\prod$ adalah $(t-c,\varepsilon)$-aman untuk beberapa konstanta (kecil) c, dalam arti bahwa untuk semua A berjalan dalam waktu paling banyak $t-c$ yang kita miliki

$$\Pr\left[PrivK^{eav}_{\mathcal{A},\prod}=1\right] \leq \frac{1}{2}+\varepsilon \qquad (3.7)$$

(Perhatikan bahwa bilangan di atas sekarang merupakan bilangan tetap, bukan fungsi dari n, karena kita tidak berada dalam keadaan asimtotik di sini.) Untuk melihat hal ini, misalkan suatu musuh sembarang berjalan dalam waktu paling lama $t-c$. Pembeda D, seperti yang dibangun dalam pembuktian Teorema 3.18, memiliki overhead yang sangat kecil selain running; pengaturan c dengan tepat memastikan bahwa D berjalan dalam waktu paling banyak t. Asumsi kita mengenai keamanan konkrit G kemudian mengimplikasikan Persamaan (3.6); melanjutkan persis seperti pembuktian Teorema 3.18, kita memperoleh Persamaan (3.7).

3.4 GAGASAN KEAMANAN YANG LEBIH KUAT

Sampai saat ini kita telah mempertimbangkan definisi keamanan yang relatif lemah di mana musuh hanya secara pasif menguping satu teks sandi yang dikirim antara pihak-pihak yang jujur. Pada bagian ini, kami membahas dua gagasan keamanan yang lebih kuat. Ingatlah bahwa definisi keamanan menentukan tujuan keamanan dan model serangan. Dalam mendefinisikan gagasan keamanan baru yang pertama, kami memodifikasi tujuan keamanan; untuk yang kedua kami memperkuat model serangan.

Keamanan untuk Banyak Enkripsi

Definisi 3.8 berkaitan dengan kasus di mana pihak-pihak yang berkomunikasi mengirimkan satu teks tersandi yang diamati oleh penyadap. Akan tetapi, akan lebih mudah

jika pihak-pihak yang berkomunikasi dapat mengirimkan beberapa ciphertext satu sama lain semua dihasilkan menggunakan kunci yang sama bahkan jika penyadap mungkin mengamati semuanya. Untuk aplikasi seperti itu kita memerlukan skema enkripsi yang aman untuk enkripsi banyak pesan.

Kita mulai dengan definisi keamanan yang tepat untuk pengaturan ini. Seperti pada Definisi 3.8, pertama-tama kami memperkenalkan eksperimen yang sesuai yang ditentukan untuk skema enkripsi apa pun $\prod$, musuh $\mathcal{A}$, dan parameter keamanan n:

Eksperimen penyadapan beberapa pesan $\Pr PrivK^{mult}_{\mathcal{A},\prod}(n)$

1. Musuh $\mathcal{A}$ diberi masukan 1^n, dan mengeluarkan sepasang daftar pesan yang panjangnya sama $M_0 = (m_{0,1} \cdot \cdots \cdot m_{0,t}) \, dan \, M_1 = (m_{1,1} \cdot \cdots \cdot m_{1,t})$, dengan $|m_{0,i}| = |m_{1,i}|$ untuk semua i.
2. Kunci k dihasilkan dengan menjalankan $Gen(1^n)$, dan bit seragam $b \in \{0,1\}$ dipilih. Untuk semua i, $ciphertext\ c_i \rightarrow Enc_k(m_{b,i})$ dihitung dan daftar $\vec{C} = (c_1 \cdot \cdots \cdot c_t)$ diberikan kepada $\mathcal{A}$.
3. $\mathcal{A}$ mengeluarkan sedikit b'.
4. Keluaran percobaan ditetapkan 1 jika $b' = b$, dan 0 jika tidak.

Pengertian keamanan sama dengan sebelumnya, hanya saja sekarang mengacu pada eksperimen di atas.

DEFINISI 3.19 Skema A enkripsi kunci pribadi $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Des})$ mempunyai beberapa enkripsi yang tidak dapat dibedakan dengan adanya penyadap jika untuk semua musuh $\mathcal{A}$ waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan sehingga

$$\Pr\left[\mathsf{PrivK}^{\mathsf{mult}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n),$$

dimana probabilitas diambil alih keacakan yang digunakan dan keacakan yang digunakan dalam percobaan.

Skema apa pun yang memiliki beberapa enkripsi yang tidak dapat dibedakan dengan adanya penyadap jelas juga memenuhi Definisi 3.8, karena eksperimen $PrivK^{eav}$ sesuai dengan kasus khusus $PrivK^{mult}$ di mana musuh menghasilkan dua daftar yang masing-masing hanya berisi satu pesan. Faktanya, definisi baru kami lebih kuat daripada Definisi 3.8, seperti yang ditunjukkan berikut ini.

PROPOSISI 3.20 Terdapat skema enkripsi kunci pribadi yang memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap, namun tidak memiliki enkripsi ganda yang tidak dapat dibedakan jika ada penyadap.

BUKTI Kita tidak perlu mencari jauh-jauh untuk menemukan contoh skema enkripsi yang memenuhi proposisi tersebut. One-time pad benar-benar rahasia, dan juga memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap. Kami menunjukkan bahwa ini tidak aman

dalam pengertian Definisi 3.19. (Kami telah membahas serangan ini di Bab 2; di sini, kami hanya menganalisis serangan tersebut sehubungan dengan Definisi 3.19.)

Secara konkret, pertimbangkan musuh berikut $\mathcal{A}$ yang menyerang skema (dalam pengertian yang ditentukan oleh eksperimen $PrivK^{mult}$): $\mathcal{A}$ mengeluarkan $\overrightarrow{M_0} = (0^\ell 0^\ell)$ dan $\overrightarrow{M_1} = (0^\ell 1^\ell)$. (Yang pertama berisi teks biasa yang sama dua kali, sedangkan yang kedua berisi dua pesan berbeda.) Misalkan $\overrightarrow{C_0} = (c_1 c_2)$ adalah daftar ciphertext yang diterima A. Jika If$= c_1 = c_2$, maka $\mathcal{A}$ menghasilkan $b' = 0$; jika tidak, $\mathcal{A}$ menghasilkan $b' = 1$.

Kami sekarang menganalisis probabilitas bahwa $b' = b$. Poin krusialnya adalah bahwa one-time pad bersifat deterministik, sehingga mengenkripsi pesan yang sama dua kali (menggunakan kunci yang sama) akan menghasilkan ciphertext yang sama. Jadi, jika $b = 0$ maka kita harus memiliki $c_1 = c_2$ dan $\mathcal{A}$ menghasilkan 0 dalam kasus ini. Sebaliknya, jika $b = 1$ maka pesan yang berbeda akan dienkripsi setiap kali; maka $c_1 \neq c_2$dan $\mathcal{A}$ keluaran 1. Kita menyimpulkan bahwa $\mathcal{A}$ keluaran $b' = b$ benar dengan probabilitas 1, sehingga skema enkripsi tidak aman sehubungan dengan Definisi 3.19.

Perlunya enkripsi probabilistik. Hal di atas mungkin tampak menunjukkan bahwa Definisi 3.19 tidak mungkin dicapai dengan menggunakan skema enkripsi apa pun. Namun pada kenyataannya hal ini hanya berlaku jika skema enkripsi bersifat deterministik sehingga mengenkripsi pesan yang sama beberapa kali (menggunakan kunci yang sama) selalu menghasilkan hasil yang sama. Ini cukup penting untuk dinyatakan sebagai teorema.

TEOREMA 3.21 Jika Π adalah skema enkripsi (*stateless*[4]) di mana Enc adalah fungsi deterministik dari kunci dan pesan, maka Π tidak dapat memiliki banyak enkripsi yang tidak dapat dibedakan dengan adanya penyadap. Hal ini tidak boleh diartikan bahwa Definisi 3.19 terlalu kuat. Memang benar, membocorkan kepada penyadap fakta bahwa dua pesan terenkripsi itu sama bisa menjadi pelanggaran keamanan yang signifikan. (Pertimbangkan, misalnya, skenario di mana siswa mengenkripsi serangkaian jawaban benar/salah!)

Untuk membangun skema yang aman untuk mengenkripsi banyak pesan, kita harus merancang skema di mana enkripsi diacak sehingga ketika pesan yang sama dienkripsi beberapa kali, teks tersandi yang berbeda dapat dihasilkan. Hal ini mungkin tampak mustahil karena dekripsi harus selalu dapat memulihkan pesan. Namun, kita akan segera melihat bagaimana cara mencapainya.

Serangan Teks Biasa Terpilih dan Keamanan CPA

Serangan teks biasa yang dipilih menangkap kemampuan musuh untuk melakukan kontrol (sebagian) atas apa yang dienkripsi oleh pihak yang jujur. Kita bayangkan sebuah skenario di mana dua pihak yang jujur berbagi kunci k, dan penyerang dapat mempengaruhi pihak-pihak tersebut untuk mengenkripsi pesan $m_1, m_2, \ldots$ (menggunakan k) dan mengirimkan ciphertext yang dihasilkan melalui saluran yang dapat diamati oleh penyerang. Di kemudian hari, penyerang mengamati teks tersandi yang berhubungan dengan pesan tak dikenal m yang dienkripsi menggunakan kunci k yang sama; mari kita asumsikan bahwa penyerang mengetahui bahwa m adalah salah satu dari dua kemungkinan m_0, m_1. Keamanan terhadap serangan teks biasa yang dipilih berarti bahwa bahkan dalam kasus ini penyerang

tidak dapat membedakan pesan mana yang dienkripsi dengan probabilitas yang jauh lebih baik daripada tebakan acak. (Untuk saat ini kita kembali ke kasus di mana penyadap hanya diberikan satu enkripsi dari pesan yang tidak diketahui. Tak lama lagi, kita akan kembali ke pertimbangan kasus banyak pesan.)

Serangan teks biasa terpilih di dunia nyata. Apakah serangan teks biasa yang dipilih merupakan masalah yang realistis? Sebagai permulaan, perhatikan bahwa serangan teks biasa yang dipilih juga mencakup serangan teks biasa yang diketahui di mana penyerang mengetahui pesan apa yang dienkripsi, meskipun ia tidak dapat memilihnya sebagai kasus khusus. Selain itu, ada beberapa skenario dunia nyata di mana musuh mungkin memiliki pengaruh signifikan terhadap pesan apa yang dienkripsi. Contoh sederhana diberikan oleh penyerang yang mengetik di terminal, yang pada gilirannya mengenkripsi dan mengirimkan semua yang diketik musuh menggunakan kunci yang dibagikan dengan server jarak jauh (dan tidak diketahui oleh penyerang). Di sini penyerang mengontrol dengan tepat apa yang dienkripsi, namun skema enkripsi harus tetap aman ketika digunakan dengan kunci yang sama untuk mengenkripsi data bagi pengguna lain.

Menariknya, serangan teks biasa yang dipilih juga telah berhasil digunakan sebagai bagian dari upaya bersejarah untuk menghancurkan skema enkripsi militer. Misalnya, selama Perang Dunia II, Inggris menempatkan ranjau di lokasi tertentu, mengetahui bahwa Jerman ketika menemukan ranjau tersebut akan mengenkripsi lokasi tersebut dan mengirimkannya kembali ke markas besar. Pesan terenkripsi ini digunakan oleh cryptanalyst di Bletchley Park untuk memecahkan skema enkripsi Jerman. Contoh lain diberikan oleh kisah terkenal yang melibatkan Pertempuran di Tengah Jalan. Pada bulan Mei 1942, cryptanalyst Angkatan Laut AS mencegat pesan terenkripsi dari Jepang yang dapat mereka pecahkan sebagian. Hasilnya menunjukkan bahwa Jepang merencanakan serangan terhadap AF, di mana AF merupakan fragmen ciphertext yang tidak dapat didekode oleh AS.

Karena alasan lain, AS yakin Pulau Midway adalah sasarannya. Sayangnya, upaya mereka untuk meyakinkan para perencana Washington bahwa hal ini memang terjadi sia-sia; kepercayaan umum adalah bahwa Midway tidak mungkin menjadi targetnya. Para cryptanalyst Angkatan Laut menyusun rencana berikut: Mereka menginstruksikan pasukan AS di Midway untuk mengirimkan pesan palsu bahwa persediaan air tawar mereka hampir habis. Pihak Jepang menyadap pesan ini dan segera melaporkan kepada atasan mereka bahwa “AF kekurangan air.” Para cryptanalyst Angkatan Laut sekarang memiliki bukti bahwa AF berhubungan dengan Midway, dan AS mengirimkan tiga kapal induk ke lokasi tersebut. Hasilnya, Midway terselamatkan dan Jepang mengalami kerugian yang cukup besar. Pertempuran ini menjadi titik balik perang antara AS dan Jepang di Pasifik.

Kriptanalis Angkatan Laut di sini melakukan serangan teks biasa terpilih, karena mereka mampu mempengaruhi Jepang (walaupun secara tidak langsung) untuk mengenkripsi kata “Midway.” Jika skema enkripsi Jepang aman terhadap serangan teks biasa yang dipilih, strategi yang dilakukan oleh para cryptanalyst AS ini tidak akan berhasil (dan sejarah mungkin akan berubah menjadi sangat berbeda)!

Keamanan CPA.

Dalam definisi formal, kami memodelkan serangan teks biasa yang dipilih dengan memberikan akses kepada musuh $\mathcal{A}$ ke oracle enkripsi, yang dipandang sebagai "kotak hitam" yang mengenkripsi pesan pilihan $\mathcal{A}'s$ menggunakan kunci k yang tidak diketahui oleh $\mathcal{A}$. Artinya, kita bayangkan $\mathcal{A}$ telah akses ke "oracle" $Enc_k(\cdot)$; ketika $\mathcal{A}$ menanyakan oracle ini dengan memberikan pesan m sebagai masukan, oracle mengembalikan ciphertext $c \leftarrow Enc_k(m)$ sebagai balasannya. (Ketika Enc diacak, oracle menggunakan keacakan baru setiap kali menjawab pertanyaan.) Musuh diperbolehkan berinteraksi dengan oracle enkripsi secara adaptif, sebanyak yang diinginkannya.

Pertimbangkan eksperimen berikut yang ditentukan untuk skema enkripsi apa pun $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Des})$, musuh $\mathcal{A}$, dan nilai n untuk parameter keamanan:
Eksperimen ketidakmampuan CPA untuk membedakan $PrivK^{cpa}_{\mathcal{A}\prod}(n)$

1. Kunci k dihasilkan dengan menjalankan $\mathrm{Gen}(1^n)$.
2. Musuh $\mathcal{A}$ diberi masukan 1^n dan akses oracle ke $Enc_k(\cdot)$, dan mengeluarkan sepasang pesan m_0, m_1 dengan panjang yang sama.
3. Bit seragam $b \in \{0,1\}$ dipilih, dan kemudian sebuah ciphertext $c \leftarrow Enc_k(m_b)$, dihitung dan diberikan kepada $\mathcal{A}$.
4. Musuh $\mathcal{A}$ terus memiliki akses Oracle ke $Enc_k(\cdot)$, dan mengeluarkan sedikit b'.
5. Keluaran percobaan ditetapkan 1 jika $b' = b$, dan 0 jika tidak. Dalam kasus pertama, kita katakan $\mathcal{A}$ berhasil.

DEFINISI 3.22 Skema enkripsi kunci pribadi $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Des})$ memiliki enkripsi yang tidak dapat dibedakan pada serangan teks biasa yang dipilih, atau aman terhadap CPA, jika untuk semua musuh waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan, abaikan hal tersebut itu

$$\Pr\left[PivK^{cpa}_{\mathcal{A},\prod}(n) = 1\right] \leq \frac{1}{2} + negl(n),$$

dimana probabilitas diambil alih keacakan yang digunakan oleh A, serta keacakan yang digunakan dalam percobaan.

Keamanan CPA untuk Banyak Enkripsi

Definisi 3.22 dapat diperluas untuk kasus enkripsi ganda dengan cara yang sama seperti Definisi 3.8 diperluas untuk memberikan Definisi 3.19, yaitu dengan menggunakan daftar teks biasa. Di sini, kami mengambil pendekatan berbeda yang lebih sederhana dan memiliki keuntungan dalam memodelkan penyerang yang dapat secara adaptif memilih teks biasa untuk dienkripsi, bahkan setelah mengamati teks tersandi sebelumnya. Dalam definisi ini, kita memberikan penyerang akses ke oracle "kiri atau kanan" LRk,b yang, pada input sepasang pesan dengan panjang yang sama m_0, m_1, menghitung ciphertext $c \leftarrow Enc_k(m_b)$, dan mengembalikan C. Artinya, jika b = 0 maka musuh menerima enkripsi dari plaintext "kiri", dan jika b = 1 maka musuh menerima enkripsi dari plaintext "kanan". Di sini, b adalah bit acak yang dipilih pada awal percobaan, dan seperti pada definisi sebelumnya, tujuan penyerang

adalah menebak b. Hal ini menggeneralisasikan definisi keamanan multi-pesan sebelumnya (Definisi 3.19) karena alih-alih mengeluarkan daftar $(\mathrm{m}_{0,1}, \mathrm{m}_{0,\mathrm{t}})$ dan $(\mathrm{m}_{1,1}, \cdots, \mathrm{m}_{1,\mathrm{t}})$, salah satu pesannya akan dienkripsi, penyerang sekarang dapat menanyakan $LR_{k,b}(m_{0,1}, m_{1,1}), \cdots, LR_{k,b}(m_{0,t}m_{1,t})$.. Ini juga mencakup akses penyerang ke oracle enkripsi, karena penyerang cukup menanyakan $LR_{k,b}(m, m)$ untuk mendapatkan $Enc_k(\mathrm{m})$.

Kami sekarang secara formal mendefinisikan eksperimen ini, yang disebut eksperimen LR-oracle. Misalkan Π adalah skema enkripsi, A adalah musuh, dan n adalah parameter keamanan:

Eksperimen LR-Oracle $\mathrm{PrivK}_{\mathcal{A}\prod}^{\mathrm{LR-cpa}}(\mathrm{n})$:

1. Kunci k dihasilkan dengan menjalankan $\mathrm{Gen}(1^n)$.
2. Bit seragam $b \in \{0,1\}$ dipilih.
3. Musuh $\mathcal{A}$ diberi masukan 1^n dan akses Oracle ke $LR_{k,b}(\cdot,\cdot)$,seperti yang didefinisikan di atas.
4. Musuh $\mathcal{A}$ mengeluarkan sedikit b'.
5. Keluaran percobaan ditetapkan 1 jika $\mathrm{b}' = \mathrm{b}$, dan 0 jika tidak. Dalam kasus pertama, kita katakan $\mathcal{A}$ berhasil.

DEFINISI 3.23 Skema enkripsi kunci pribadi Π memiliki beberapa enkripsi yang tidak dapat dibedakan pada serangan teks biasa yang dipilih, atau aman terhadap CPA untuk beberapa enkripsi, jika untuk semua musuh waktu polinomial probabilistik terdapat $\mathcal{A}$ fungsi yang dapat diabaikan sehingga

$$\Pr\left[PrivK_{\mathcal{A}\prod}^{LR-cpa}(n) = 1\right] \leq \frac{1}{2} + neg|(n),$$

dimana probabilitas diambil alih keacakan yang digunakan dan keacakan yang digunakan dalam percobaan.

Diskusi kami sebelumnya menunjukkan bahwa keamanan CPA untuk beberapa enkripsi setidaknya sama kuatnya dengan semua definisi kami sebelumnya. Secara khusus, jika skema enkripsi kunci pribadi aman terhadap CPA untuk beberapa enkripsi, maka skema tersebut jelas aman terhadap CPA juga. Yang penting, hal sebaliknya juga berlaku; artinya, keamanan CPA berarti keamanan CPA untuk beberapa enkripsi. (Hal ini berbeda dengan kasus musuh yang menguping; lihat Proposisi 3.20.).

TEOREMA 3.24 Setiap skema enkripsi kunci pribadi yang aman terhadap CPA juga aman terhadap CPA untuk beberapa enkripsi. Ini adalah keuntungan teknis yang signifikan dari keamanan CPA: Cukup dengan membuktikan bahwa suatu skema aman terhadap CPA (untuk enkripsi tunggal), dan kami kemudian memperoleh "gratis" bahwa skema tersebut juga aman terhadap CPA untuk beberapa enkripsi. Keamanan terhadap serangan teks biasa yang dipilih saat ini merupakan gagasan keamanan minimal yang harus dipenuhi oleh skema enkripsi, meskipun semakin umum diperlukan properti keamanan yang lebih kuat yang dibahas di Bagian 4.5.

Pesan dengan panjang tetap vs. pesan dengan panjang sewenang-wenang. Keuntungan lain menggunakan definisi keamanan CPA adalah memungkinkan kita menangani skema enkripsi dengan panjang tetap tanpa kehilangan keumumannya. Khususnya, mengingat skema enkripsi dengan panjang tetap aman CPA $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Des})$, maka dimungkinkan untuk membuat skema enkripsi *aman CPA* $\Pi' = (\mathrm{Gen}', \mathrm{Enc}', \mathrm{Des}')$ untuk pesan dengan panjang sewenang-wenang dengan cukup mudah. Untuk mempermudah, katakanlah Π mengenkripsi pesan yang panjangnya 1-bit (walaupun semua yang kita katakan diperluas secara alami tanpa memperhatikan panjang pesan yang didukung oleh Π). Biarkan Gen' sama dengan Gen. Define Enc'k untuk pesan apa pun m (memiliki panjang sembarang ℓ) sebagai $Enc'_k(m) = Enc_k(m_1), \ldots, Enc_k(m_\ell)$, di mana mi menunjukkan bit ke-i dari m. Dekripsi dilakukan dengan cara alami. Π' aman untuk BPA jika Π'; bukti berikut dari Teorema 3.24.

Ada cara yang lebih efisien untuk mengenkripsi pesan dengan panjang sembarang dibandingkan dengan mengadaptasi skema enkripsi dengan panjang tetap seperti cara di atas. Kami mengeksplorasi hal ini lebih lanjut di Bagian 3.6.

3.5 MEMBANGUN SKEMA ENKRIPSI AMAN CPA

Sebelum membangun skema enkripsi yang aman terhadap serangan teks biasa, pertama-tama kami memperkenalkan gagasan penting tentang fungsi pseudorandom.

Fungsi Pseudorandom dan Block Cipher

Fungsi Pseudorandom (PRF) menggeneralisasi pengertian generator pseudorandom. Sekarang, daripada mempertimbangkan string yang "tampak acak", kami mempertimbangkan fungsi yang "tampak acak". Seperti dalam pembahasan kita sebelumnya tentang keacakan semu, tidak masuk akal untuk mengatakan bahwa setiap fungsi tetap $f : \{0,1\}^* \rightarrow \{0,1\}^*$ adalah acak semu (seperti halnya tidak masuk akal untuk mengatakan bahwa setiap fungsi tetap acak). Jadi, kita harus mengacu pada keacakan semu dari suatu distribusi fungsi. Distribusi seperti itu terjadi secara alami dengan mempertimbangkan fungsi-fungsi utama, yang didefinisikan selanjutnya.

Fungsi berkunci $f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^*$ merupakan fungsi dengan dua masukan, dimana masukan pertama disebut kunci dan dilambangkan dengan k. Kita mengatakan F efisien jika terdapat algoritma waktu polinomial yang menghitung $\mathrm{F}\ (\mathrm{k}, \mathrm{x})$ jika diberikan k dan x. (Kami hanya akan tertarik pada fungsi-fungsi berkunci yang efisien.) Dalam penggunaan umum, kunci k dipilih dan ditetapkan, dan kami kemudian tertarik pada fungsi masukan tunggal $F_k: \{0,1\}^* \rightarrow \{0,1\}^*$ yang didefinisikan oleh $F_k(x) = F(k, x)$. Parameter keamanan n menentukan panjang kunci, panjang masukan, dan panjang keluaran. Artinya, kita mengasosiasikan dengan F tiga fungsi $\ell_{key}, \ell_{in}, dan\ \ell_{out}$; untuk kunci apa pun $k \in \{0,1\}^{\ell key(n)}$, fungsi F_k hanya didefinisikan untuk masukan $x \in \{0,1\}^{\ell\, in(n)}$, dalam hal ini $F_k(x) \in \{0,1\}^{\ell out(n)}$. Kecuali jika dinyatakan sebaliknya, kita asumsikan untuk kesederhanaan bahwa F adalah mempertahankan panjang, yang berarti $\ell_{key}(n) =, \ell_{in}(n) =, dan\ \ell_{out}(n) = n$. Artinya, dengan menetapkan kunci $k \in \{0,1\}^n$ kita memperoleh fungsi Fk yang memetakan string masukan n-bit ke string keluaran n-bit.

Fungsi berkunci F menginduksi distribusi natural pada fungsi-fungsi yang diberikan dengan memilih kunci seragam $k \in \{0,1\}^n$ dan kemudian mempertimbangkan fungsi masukan tunggal yang dihasilkan F_k. Kita menyebut F pseudorandom jika fungsi F_k (untuk kunci seragam k) tidak dapat dibedakan dari fungsi yang dipilih secara acak secara seragam dari himpunan semua fungsi yang memiliki domain dan jangkauan yang sama; yaitu, jika tidak ada musuh efisien yang dapat membedakan dalam artian kita akan mendefinisikannya dengan lebih hati-hati di bawah apakah ia berinteraksi dengan F_k (untuk k seragam) atau f (di mana f dipilih secara seragam dari himpunan semua fungsi yang memetakan input n-bit ke keluaran n-bit).

Fungsi berkunci F menginduksi distribusi natural pada fungsi-fungsi yang diberikan dengan memilih kunci seragam $k \in \{0,1\}^n$ dan kemudian mempertimbangkan fungsi masukan tunggal yang dihasilkan Fk. Kita menyebut F pseudorandom jika fungsi F_k (untuk kunci seragam k) tidak dapat dibedakan dari fungsi yang dipilih secara acak secara seragam dari himpunan semua fungsi yang memiliki domain dan jangkauan yang sama; yaitu, jika tidak ada musuh efisien yang dapat membedakan dalam artian kita akan mendefinisikannya dengan lebih hati-hati di bawah apakah ia berinteraksi dengan F_k (untuk k seragam) atau f (di mana f dipilih secara seragam dari himpunan semua fungsi yang memetakan input n-bit ke keluaran n-bit).

Karena memilih fungsi secara acak kurang intuitif dibandingkan memilih string secara acak, ada baiknya meluangkan lebih banyak waktu untuk ide ini. Pertimbangkan himpunan Fungsi semua fungsi yang memetakan string n-bit ke string n-bit. Himpunan ini terbatas, dan memilih fungsi seragam yang memetakan string n-bit ke string n-bit berarti memilih elemen secara seragam dari himpunan ini. Seberapa besarkah $Func_n$? Suatu fungsi f ditentukan dengan memberikan nilainya pada setiap titik dalam domainnya. Kita dapat melihat fungsi apa pun (pada domain berhingga) sebagai tabel pencarian besar yang menyimpan $f(x)$ pada baris tabel yang diberi label x. Untuk $f \in Func_n$, tabel pencarian untuk f mempunyai 2^n baris (satu untuk setiap titik domain $\{0,1\}^n$), dengan setiap baris berisi string n-bit (karena rentang f adalah $\{0,1\}^n$). Menggabungkan semua entri tabel, kita melihat bahwa fungsi apa pun di Funcn dapat diwakili oleh string dengan panjang $2^n \cdot n \cdot$. Selain itu, korespondensi ini adalah satu-ke-satu, karena setiap string dengan panjang $2^n \cdot n$ (yaitu, setiap tabel berisi 2n entri dengan panjang n) mendefinisikan fungsi unik di Funcn. Jadi, ukuran Funcn sama persis dengan n jumlah string yang panjangnya $n \cdot 2^n$, atau $|Func_n| = 2^n \cdot n$

Melihat fungsi sebagai tabel pencarian memberikan cara lain yang berguna untuk memikirkan pemilihan fungsi seragam f Fungsi: Ini sama persis dengan memilih setiap baris dalam tabel pencarian f secara seragam. Hal ini berarti, khususnya, bahwa nilai $f(x)\ dan\ f(y)$ (untuk dua input $x = y$) adalah seragam dan independen. Kita dapat melihat tabel pencarian ini diisi dengan entri acak terlebih dahulu, sebelum f dievaluasi pada input apa pun, atau kita dapat melihat entri tabel dipilih secara seragam "on-the-fly," sesuai kebutuhan, setiap kali f dievaluasi pada masukan baru yang f belum pernah dievaluasi sebelumnya.

Kembali ke pembahasan kita mengenai fungsi pseudorandom, ingatlah bahwa fungsi pseudorandom adalah fungsi berkunci F sedemikian rupa sehingga F_k (untuk $k \in \{0,1\}^n$).dipilih secara seragam secara acak) tidak dapat dibedakan dari f (untuk f Fungsi yang dipilih secara acak secara seragam) . Yang pertama dipilih dari distribusi (paling banyak) 2n fungsi berbeda, sedangkan yang kedua dipilih dari semua $2^n \cdot 2^n$ fungsi di Funcn. Meskipun demikian, "perilaku" fungsi-fungsi ini harus terlihat sama bagi pembeda waktu polinomial mana pun.

Upaya pertama untuk memformalkan gagasan fungsi pseudorandom adalah dengan melanjutkan dengan cara yang sama seperti pada Definisi 3.14. Artinya, kita dapat mensyaratkan bahwa setiap pembeda waktu polinomial D yang menerima deskripsi fungsi pseudorandom Fk menghasilkan 1 dengan probabilitas "hampir" sama seperti ketika ia menerima deskripsi fungsi acak f . Namun, definisi ini tidak tepat karena deskripsi fungsi acak memiliki panjang eksponensial (diberikan oleh tabel pencarian dengan panjang $n \cdot 2^n$), sedangkan D dibatasi untuk berjalan dalam waktu polinomial. Jadi, D bahkan tidak punya cukup waktu untuk memeriksa seluruh masukannya.

Oleh karena itu, definisi tersebut memberikan D akses ke oracle O yang sama dengan F_k (untuk seragam k) atau f (untuk fungsi seragam f). Pembeda D dapat menanyakan oraclenya kapan saja x, sebagai respons terhadap oracle yang mengembalikan (x). Kami memperlakukan oracle sebagai kotak hitam dengan cara yang sama seperti saat kami memberikan akses oracle ke algoritma enkripsi kepada musuh dalam definisi serangan teks biasa yang dipilih. Namun di sini, Oracle menghitung fungsi deterministik dan mengembalikan hasil yang sama jika ditanyakan dua kali pada input yang sama. (Untuk alasan ini, kita dapat berasumsi tanpa kehilangan keumuman bahwa D tidak pernah menanyakan oracle dua kali pada masukan yang sama.) D dapat berinteraksi secara bebas dengan oraclenya, memilih kuerinya secara adaptif berdasarkan semua keluaran sebelumnya. Namun, karena D berjalan dalam waktu polinomial, D hanya dapat menanyakan banyak pertanyaan secara polinomial. Kami sekarang menyajikan definisi formal. (Definisi ini mengasumsikan F mempertahankan panjang untuk kesederhanaan saja.)

DEFINISI 3.25 Misalkan $f : \{0,1\} \times \{0,1\}^* \rightarrow \{0,1\}^*$adalah fungsi berkunci yang efisien, mempertahankan panjang. F adalah fungsi pseudorandom jika untuk semua pembeda waktu polinomial probabilistik D, terdapat fungsi yang dapat diabaikan sehingga:

$$\left|\Pr\left[\mathrm{D}^{Fk(\cdot)}(1^n) = 1\right] - \Pr\left[\mathrm{D}^{f(\cdot)}(1^n) = 1\right]\right| \leq \mathrm{negl}\,(n),$$

dimana probabilitas pertama diambil alih pilihan seragam $k \in \{0,1\}^n$dan keacakan D, dan probabilitas kedua diambil alih pilihan seragam $f \in Func_n$ dan keacakan D.

Hal yang penting adalah D tidak diberikan kunci k. Tidak ada gunanya mengharuskan F_k menjadi pseudorandom jika k diketahui, karena mengingat k maka mudah untuk membedakan sebuah oracle untuk F_k dari sebuah oracle untuk f . (Yang harus dilakukan oleh

pembeda adalah menanyakan oracle pada titik mana pun x untuk mendapatkan jawaban y, dan membandingkannya dengan hasil $y' := F^k(x)$ yang dihitung sendiri menggunakan nilai k yang diketahui. Oracle untuk Fk akan kembalikan y = y', sedangkan ramalan untuk fungsi acak akan memiliki y = y' dengan probabilitas hanya 2^{-2}.) Ini berarti bahwa jika k terungkap, klaim apa pun tentang keacakan semu Fk tidak lagi berlaku. Untuk mengambil contoh konkret, jika F adalah fungsi pseudorandom, maka jika diberi akses oracle ke F_k (untuk k seragam), pasti sulit menemukan input x yang $F_k(x) = 0^n$ (karena akan sulit menemukan input seperti itu masukan untuk fungsi yang benar-benar acak f). Namun jika k diketahui, mencari masukan seperti itu mungkin mudah.

Contoh 3.26

Seperti biasa, kita dapat memahami definisi tersebut dengan melihat contoh yang tidak aman. Definisikan fungsi kunci yang mempertahankan panjang F dengan $\mathrm{F}(\mathrm{k},\mathrm{x}) = \mathrm{k} \oplus \mathrm{x}$. Untuk setiap masukan x, nilai $\mathrm{F_k(x)}$ terdistribusi secara merata (bila k seragam). Meskipun demikian, F bukanlah pseudorandom karena nilainya pada dua titik mana pun berkorelasi. Secara konkret, pertimbangkan pembeda D yang menanyakan oracle $\mathcal{O}$ nya pada titik-titik $x_1x_2$yang berbeda dan berubah-ubah untuk mendapatkan nilai $y_1 = \mathcal{O}(x_1)\ dan\ y_2 = \mathcal{O}(x_2)$, dan menghasilkan 1 jika dan hanya jika $y_1 \oplus y_2 = x_1 \oplus x_2$. Jika O = Fk, untuk sembarang k, maka D menghasilkan 1. Sebaliknya, jika O = f untuk f dipilih secara seragam dari Funcn, maka peluang $f(x_1) \oplus f(x_2) = x_1 \oplus x_2$ tepat adalah probabilitas bahwa $f(x_2) = x_1 \oplus x_2$ ($f(x_1)$ atau 2−n, dan D menghasilkan 1 dengan probabilitas ini. Perbedaannya adalah $|1 - 2^{-n}|$, yang tidak dapat diabaikan.

Permutasi Pseudorandom/Block Cipher

Misalkan Permn adalah himpunan semua permutasi (yaitu bijeksi) pada $\{0,1\}^n$. Melihat f Permn apa pun sebagai tabel pencarian seperti sebelumnya, kita sekarang mempunyai batasan tambahan bahwa entri dalam dua baris berbeda harus berbeda. Kami memiliki 2^n pilihan berbeda untuk entri di baris pertama tabel; setelah kita memperbaiki entri ini, kita hanya mempunyai $2^n - 1$ pilihan untuk baris kedua, dan seterusnya. Jadi kita melihat bahwa ukuran Permn adalah $(2^n)!$.

Biarkan F menjadi fungsi berkunci. Kita menyebut F sebagai permutasi kunci $jika\ lin = lout$, dan selanjutnya untuk semua $k \in \ell_{key}(n)$ fungsi $F_k : \{0,1\}^{\ell in(n)} \to \{0,1\}^{\ell in(n)}$ adalah fungsi satu-ke-satu (yaitu, F_k adalah permutasi). Kami menyebut ℓ_{key} panjang blok F . Seperti sebelumnya, kecuali dinyatakan lain, kita asumsikan F adalah pertahankan panjang sehingga $\ell_{key}(n) = \ell_{key}(n) = n$. Permutasi berkunci efisien jika terdapat algoritma waktu polinomial untuk menghitung $F_k(x)$ jika diberikan k dan x, serta algoritma waktu polinomial untuk menghitung $F_k^{-1}(y)$ jika diberikan k dan y. Artinya, Fk harus dapat dihitung secara efisien dan dapat dibalik secara efisien jika diberikan k.

Definisi yang dimaksud dengan permutasi berkunci F yang efisien menjadi permutasi pseudorandom sama persis dengan Definisi 3.25, dengan satu-satunya perbedaan adalah bahwa sekarang kita mengharuskan F_k tidak dapat dibedakan dari permutasi seragam, bukan

fungsi seragam. Artinya, kita memerlukan bahwa tidak ada algoritma efisien yang dapat membedakan antara akses ke F_k (untuk kunci seragam k) dan akses ke ƒ (untuk seragam ƒ Permn). Ternyata ini hanyalah sebuah pilihan estetis karena, kapan pun panjang balok cukup panjang, permutasi acak itu sendiri tidak dapat dibedakan dari fungsi acak. Secara intuitif hal ini disebabkan oleh fakta bahwa fungsi seragam ƒ terlihat identik dengan permutasi seragam kecuali jika ditemukan nilai x dan y yang berbeda dimana $f(x) = f(y)$, karena dalam kasus seperti ini fungsi tersebut tidak dapat berupa permutasi. Namun, kemungkinan menemukan nilai x, y tersebut menggunakan sejumlah kueri polinomial dapat diabaikan. (Ini mengikuti hasil Lampiran A.4.)

PROPOSISI 3.27 Jika F merupakan permutasi pseudorandom dan tambahan lin(n) ≥ n, maka F juga merupakan fungsi pseudorandom. Jika F adalah permutasi berkunci maka skema kriptografi berdasarkan F mungkin memerlukan pihak yang jujur untuk menghitung invers F −1 selain menghitung Fk itu sendiri. Hal ini berpotensi menimbulkan masalah keamanan baru. Secara khusus, sekarang mungkin perlu untuk menerapkan persyaratan yang lebih kuat bahwa Fk tidak dapat dibedakan dari permutasi yang seragam bahkan jika pembeda tersebut juga diberikan akses oracle ke invers dari permutasi tersebut. Jika F mempunyai sifat ini, kita menyebutnya permutasi pseudorandom kuat.

DEFINISI 3.28 Misalkan $\mathrm{f} : \{0,1\} \times \{0,1\}^* \to \{0,1\}^*$adalah permutasi berkunci yang efisien, mempertahankan panjang. F adalah permutasi pseudorandom kuat jika untuk semua pembeda waktu polinomial probabilistik D, terdapat fungsi yang dapat diabaikan sehingga:

$$\left|\Pr\left[\mathrm{D}^{Fk(\cdot),F_k^{-1}(\cdot)}(1^{\mathrm{n}}) = 1\right] - \Pr\left[\mathrm{D}^{f(\cdot),f^{-1}(\cdot)}(1^{\mathrm{n}}) = 1\right]\right| \leq negl(n),$$

dimana probabilitas pertama diambil alih pilihan seragam $k \in \{0,1\}^n$ dan keacakan D, dan probabilitas kedua diambil alih pilihan seragam $f \in Perm_n$ dan keacakan D.

Tentu saja, setiap permutasi pseudorandom kuat juga merupakan permutasi pseudorandom. Blokir sandi. Dalam praktiknya, cipher blok dirancang untuk menjadi contoh aman dari permutasi pseudorandom (kuat) dengan panjang kunci dan panjang blok yang tetap. Kita membahas pendekatan untuk membangun cipher blok, dan beberapa kandidat cipher blok yang populer, di Bab 6. Untuk keperluan bab ini, rincian konstruksi ini tidak penting, dan untuk saat ini kita hanya berasumsi bahwa ada permutasi pseudorandom (kuat).

Fungsi pseudorandom dan generator pseudorandom. Seperti yang diharapkan, ada hubungan erat antara fungsi pseudorandom dan generator pseudorandom. Cukup mudah untuk membuat generator pseudorandom G dari fungsi pseudorandom F hanya dengan mengevaluasi F pada serangkaian input berbeda; misalnya, kita dapat mendefinisikan $\mathrm{G}(\mathrm{s}) \stackrel{\text{def}}{=} F_s(1)||F_s(2)||\cdots||F_s(\ell)$untuk setiap yang diinginkan ℓ. Jika F_s diganti dengan fungsi seragam ƒ , keluaran dari G akan seragam; jadi, saat menggunakan F, outputnya adalah pseudorandom.

Secara umum, kita dapat menggunakan ide di atas untuk membuat stream cipher (Init, GetBits) yang menerima vektor inisialisasi IV . Satu-satunya perbedaan adalah bahwa alih-alih

mengevaluasi F_s pada rangkaian masukan tetap 1, 2, 3, . . ., kita evaluasi F pada masukan IV + 1, IV + 2, ...

CONSTRUCTION 3.29

Let F be a pseudorandom function. Define a stream cipher (Init, GetBits), where each call to GetBits outputs n bits, as follows:

- Init: on input $s \in \{0,1\}^n$ and $IV \in \{0,1\}^n$, set $\mathsf{st}_0 := (s, IV)$.
- GetBits: on input $\mathsf{st}_i = (s, IV)$, compute $IV' := IV + 1$ and set $y := F_s(IV')$ and $\mathsf{st}_{i+1} := (s, IV')$. Output (y, st_{i+1}).

Sandi aliran dari fungsi pseudorandom/sandi blok apa pun.

Meskipun stream cipher dapat dibuat dari block cipher, stream cipher khusus yang digunakan dalam praktik biasanya memiliki kinerja yang lebih baik, terutama di lingkungan dengan sumber daya terbatas. Di sisi lain, stream cipher tampaknya kurang dipahami (dalam praktiknya) dibandingkan block cipher, dan kepercayaan terhadap keamanannya lebih rendah. Oleh karena itu disarankan untuk menggunakan cipher blok (mungkin dengan mengonversinya menjadi stream cipher terlebih dahulu) bila memungkinkan.

Mengingat arah yang lain, generator pseudorandom G segera memberikan fungsi pseudorandom F dengan panjang blok kecil. Secara khusus, misalkan G mempunyai faktor muai $n \cdot 2^{t(n)}$. Kita dapat mendefinisikan fungsi berkunci $\mathrm{f} : \{0,1\}^n \times \{0,1\}^{t(n)} \rightarrow \{0,1\}^{\mathrm{n}}$ sebagai berikut: untuk menghitung $F_k(i)$, pertama-tama hitung G(k) dan interpretasikan hasilnya sebagai pencarian tabel dengan $2^{t(n)}$baris masing-masing berisi n bit; keluaran baris ke-i. Ini berjalan dalam waktu polinomial hanya jika $t(n) = \mathcal{O}(log\ n)$. Meskipun lebih sulit, dimungkinkan untuk membangun fungsi pseudorandom dengan panjang blok yang besar dari generator pseudorandom; ini ditunjukkan pada Bagian 7.5. Generator pseudorandom, pada gilirannya, dapat dibangun berdasarkan masalah matematika tertentu yang dianggap sulit. Keberadaan fungsi pseudorandom berdasarkan permasalahan matematika yang sulit ini mewakili salah satu kontribusi luar biasa dari kriptografi modern.

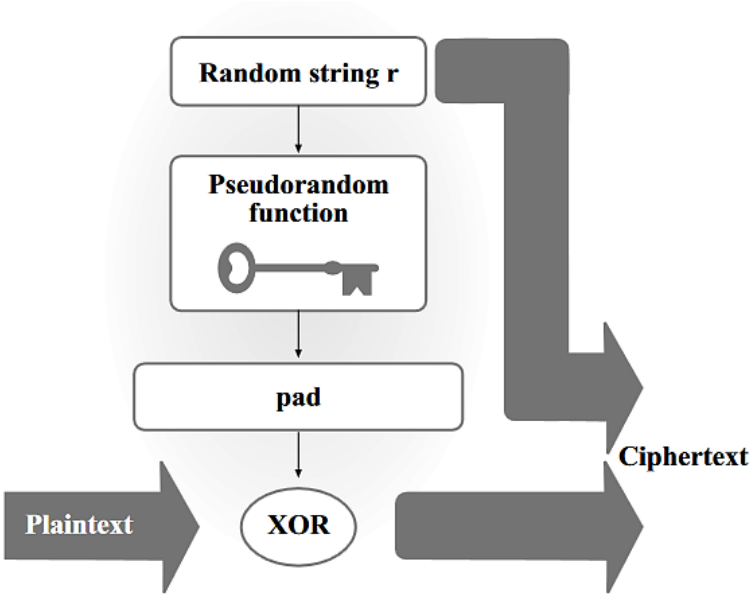
Enkripsi Aman CPA dari Fungsi Pseudorandom

Di sini kami fokus pada pembuatan skema enkripsi dengan panjang tetap dan aman terhadap CPA. Berdasarkan apa yang telah kami katakan di akhir Bagian 3.4, hal ini menyiratkan adanya skema enkripsi aman CPA untuk pesan dengan panjang sembarang. Pada Bagian 3.6 kita akan membahas cara yang lebih efisien untuk mengenkripsi pesan dengan panjang yang berubah-ubah.

Upaya naif dalam membangun skema enkripsi aman dari permutasi pseudorandom adalah dengan mendefinisikan $Enc_k(m) = F_k(m)$. Meskipun kita berharap bahwa ini "tidak mengungkapkan informasi tentang m" (karena, jika ƒ adalah fungsi yang seragam, maka $f(m)$ hanyalah sebuah string n-bit yang seragam), metode enkripsi ini bersifat deterministik dan

karenanya tidak mungkin Aman terhadap BPA. Secara khusus, mengenkripsi teks biasa yang sama dua kali akan menghasilkan teks tersandi yang sama.

Konstruksi aman kami diacak. Secara khusus, kami mengenkripsi dengan menerapkan fungsi pseudorandom ke nilai acak r (bukan pesan) dan meng-XOR hasilnya dengan teks biasa. (Lihat Gambar 3.3 dan Konstruksi 3.30.) Sekali lagi ini dapat dilihat sebagai contoh XOR pad pseudorandom dengan teks biasa, dengan perbedaan besar adalah fakta bahwa pad pseudorandom baru digunakan setiap saat. (Faktanya, pad pseudorandom hanya "baru" jika fungsi pseudorandom diterapkan pada nilai "baru" yang belum pernah diterapkan sebelumnya. Meskipun ada kemungkinan bahwa r acak akan sama dengan beberapa nilai r yang dipilih sebelumnya, hal ini terjadi dengan probabilitas yang dapat diabaikan.)



GAMBAR 3.3: Enkripsi dengan fungsi pseudorandom.

Bukti keamanan berdasarkan fungsi pseudorandom. Sebelum beralih ke bukti bahwa konstruksi di atas aman terhadap CPA, kami menyoroti templat umum yang digunakan oleh sebagian besar bukti keamanan (bahkan di luar konteks enkripsi) untuk konstruksi berdasarkan fungsi pseudorandom. Langkah pertama dari pembuktian tersebut adalah dengan mempertimbangkan versi hipotetis dari konstruksi di mana fungsi pseudorandom diganti dengan fungsi acak. Kemudian dikemukakan dengan menggunakan pembuktian melalui reduksi bahwa modifikasi ini tidak berdampak signifikan terhadap kemungkinan keberhasilan penyerang. Kita kemudian tinggal menganalisis skema yang menggunakan fungsi acak sepenuhnya. Pada titik ini, pembuktian lainnya biasanya bergantung pada analisis probabilistik dan tidak bergantung pada asumsi komputasi apa pun. Kami akan menggunakan templat bukti ini beberapa kali dalam bab ini dan bab berikutnya.

CONSTRUCTION 3.30

Let F be a pseudorandom function. Define a private-key encryption scheme for messages of length n as follows:

- Gen: on input 1^n, choose uniform $k \in \{0,1\}^n$ and output it.
- Enc: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, choose uniform $r \in \{0,1\}^n$ and output the ciphertext

$$c := \langle r, F_k(r) \oplus m \rangle.$$

- Dec: on input a key $k \in \{0,1\}^n$ and a ciphertext $c = \langle r, s \rangle$, output the plaintext message

$$m := F_k(r) \oplus s.$$

Skema enkripsi aman CPA dari fungsi pseudorandom apa pun.

TEOREMA 3.31 Jika F adalah fungsi pseudorandom, maka Konstruksi 3.30 adalah skema enkripsi kunci pribadi aman-CPA untuk pesan dengan panjang n.

BUKTI Misalkan $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ merupakan skema enkripsi yang persis sama dengan $\Pi = (\text{Gen}, \text{Enc}, \text{Dec})$ dari Konstruksi 3.30, kecuali fungsi acak f digunakan sebagai pengganti F_k. Artinya, $Gen(1^n)$ memilih fungsi seragam $f \in Func_n$, dan Enc mengenkripsi seperti Enc kecuali bahwa f digunakan sebagai pengganti F_k. (Skema enkripsi yang dimodifikasi ini tidak efisien. Namun kita masih dapat mendefinisikannya sebagai skema enkripsi hipotetis demi pembuktian.)

Perbaiki musuh PPT yang sewenang-wenang, dan biarkan $q(n)$ menjadi batas atas jumlah kueri yang $\mathcal{A}(1^n)$ buat pada oracle enkripsinya. (Perhatikan bahwa q harus dibatasi atas oleh beberapa polinomial.) Sebagai langkah pertama pembuktian, kita tunjukkan bahwa ada fungsi yang dapat diabaikan sehingga

$$\left|\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1\right] - \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right]\right| \leq \mathsf{negl}(n). \qquad (3.8)$$

Kami membuktikannya dengan reduksi. Kami menggunakan untuk membuat pembeda D untuk fungsi pseudorandom F . Pembeda D diberikan akses oracle ke beberapa fungsi, dan tujuannya adalah untuk menentukan apakah fungsi ini adalah "pseudorandom" (yaitu, sama dengan F_k untuk seragam $k \in \{0,1\}^n$ atau "acak" (yaitu, sama dengan f untuk seragam $f \in Func_n$). Untuk melakukannya, D mengemulasi eksperimen $PrivK^{cpa}$ dengan $\mathcal{A}$ cara yang dijelaskan di bawah, dan mengamati apakah $\mathcal{A}$ berhasil atau tidak. Jika $\mathcal{A}$ berhasil maka D menebak bahwa oraclenya pasti merupakan fungsi pseudorandom, sedangkan jika $\mathcal{A}$ tidak berhasil maka D menebak bahwa oraclenya pasti merupakan fungsi acak. Secara terperinci:

Pembeda D:

D diberikan masukan 1^n dan akses ke oracle $\mathcal{O}: \{0,1\}^n \rightarrow \{0,1\}^n$.

1. Jalankan $\mathcal{A}(1^n)$. Setiap kali A menanyakan oracle enkripsinya pada pesan m ∈ $\{0,1\}^n$, jawab pertanyaan ini dengan cara berikut:
   (a) Pilih seragam r ∈ $\{0,1\}^n$.
   (b) Kueri $\mathcal{O}(\mathrm{r})$ dan dapatkan respons y.
   (c) Kembalikan ciphertext $\langle \mathrm{r}, \mathrm{y} \oplus \mathrm{m} \rangle$ ke $\mathcal{A}$.
2. Ketika $\mathcal{A}$ mengeluarkan pesan $\mathrm{m}_0, \mathrm{m}_1 \in \{0,1\}^n$, pilih bit yang seragam b ∈ $\{0,1\}$ dan kemudian:
   (a) Pilih seragam r ∈ $\{0,1\}^n$.
   (b) Kueri $\mathcal{O}(\mathrm{r})$ dan dapatkan respons y.
   (c) Kembalikan tantangan ciphertext $\langle \mathrm{r}, \mathrm{y} \oplus \mathrm{m}_b \rangle$ ke $\mathcal{A}$.
3. Lanjutkan menjawab pertanyaan enkripsi-oracle dari $\mathcal{A}$ seperti sebelumnya sampai A mengeluarkan sedikit b′. Keluaran 1 jika b′ = b, dan 0 sebaliknya.

D berjalan dalam waktu polinomial sejak $\mathcal{A}$ melakukannya. Poin-poin utamanya adalah sebagai berikut:

- Jika oracle D adalah fungsi pseudorandom, maka tampilan saat $\mathcal{A}$ dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan dalam $\mathcal{A}$ eksperimen $PrivK^{cpa}_{\mathcal{A}\prod}(n)$. Hal ini karena, dalam hal ini, kunci k dipilih secara acak secara seragam dan kemudian setiap enkripsi dilakukan dengan memilih r yang seragam, menghitung $y := F_k(r)$, dan menetapkan ciphertext sama dengan $\langle r, y \oplus m \rangle$, persis seperti di Konstruksi 3.30. Dengan demikian,

$$\Pr_{k\leftarrow\{0,1\}^n}\left[D^{F_k(\cdot)}(1^n)=1\right]=\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)=1\right], \qquad (3.9)$$

  dimana kami menekankan bahwa k dipilih secara seragam di sisi kiri.

- Jika oracle $D's$ adalah fungsi acak, maka tampilan saat dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan dalam eksperimen $PrivK^{cpa}_{\mathcal{A}\tilde{\prod}}(n)$. Hal ini dapat dilihat persis seperti di atas, dengan satu-satunya perbedaan adalah bahwa fungsi seragam f ∈ Func_n digunakan sebagai pengganti F_k. Dengan demikian,

$$\Pr_{f\leftarrow\mathsf{Func}_n}\left[D^{f(\cdot)}(1^n)=1\right]=\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\tilde{\Pi}}(n)=1\right], \qquad (3.10)$$

  dimana f dipilih secara seragam dari Func_n di sisi kiri.

Dengan asumsi bahwa F adalah fungsi pseudorandom (dan karena D efisien), maka terdapat fungsi yang dapat diabaikan yang mana

$$\left|\Pr\left[\mathrm{D}^{Fk(\cdot)}(1^n)=1\right]-\Pr\left[\mathrm{D}^{f(\cdot)}(1^n)=1\right]\right|\leq \mathrm{negl}(\mathrm{n}).$$

Menggabungkan persamaan di atas dengan Persamaan (3.9) dan (3.10) menghasilkan Persamaan (3.8).

Untuk bukti bagian kedua, kami tunjukkan itu

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\right] \leq \frac{1}{2}+\frac{q(n)}{2^n}. \qquad (3.11)$$

(Ingat bahwa $q(n)$ adalah batas jumlah permintaan enkripsi yang dibuat oleh $\mathcal{A}$. Persamaan di atas berlaku bahkan jika kita tidak memberikan batasan komputasi pada $\mathcal{A}$.) Untuk melihat bahwa Persamaan (3.11) berlaku, amati bahwa setiap kali pesan m dienkripsi di $PrivK^{cpa}_{\mathcal{A}\widetilde{\Pi}}(n)$ (baik melalui oracle enkripsi atau ketika teks sandi tantangan dihitung), $\mathrm{r} \in \{0,1\}^n$yang seragam dipilih dan teks sandi diatur sama dengan $\langle r, f(r) \oplus m\rangle$. Misal r^* menyatakan string acak yang digunakan saat membuat teks sandi tantangan $\langle r^*, f(r^*) \oplus m_b\rangle$. Ada dua kemungkinan:

1. Nilai r^* tidak pernah digunakan ketika menjawab pertanyaan $\mathcal{A}'s$ oracle enkripsi apa pun: Dalam hal ini, $\mathcal{A}$ tidak mempelajari apa pun tentang f (r$*$) dari interaksinya dengan oracle enkripsi (karena f adalah fungsi yang benar-benar acak). Artinya, sejauh menyangkut $\mathcal{A}$, nilai $f(r^*)$ yang $\mathrm{di}-\mathrm{XORed}$ dengan mb terdistribusi merata dan tidak bergantung pada sisa eksperimen, sehingga probabilitas $\mathcal{A}$ keluaran $b' = b$ dalam kasus ini adalah tepat 1/2 (seperti halnya pad sekali pakai).
2. Nilai r^* digunakan ketika menjawab setidaknya satu $\mathcal{A}$ pertanyaan enkripsi-oracle: Dalam hal ini, $\mathcal{A}$ dapat dengan mudah menentukan apakah $m_0\, atau\, m_1$ dienkripsi. Hal ini terjadi karena jika oracle enkripsi mengembalikan ciphertext $\langle r^*\, s\rangle$ sebagai respons terhadap permintaan untuk mengenkripsi pesan m, musuh mengetahui bahwa $f(r^*) = s \oplus m$.

Namun, sejak $\mathcal{A}$ membuat paling banyak $q(n)$ kueri ke oracle enkripsinya (dan dengan demikian paling banyak nilai $q(n)$ dari r digunakan saat menjawab $\mathcal{A}'s$ kueri oracle enkripsi), dan karena $r*$ dipilih secara seragam dari $\{0,1\}^n$ peluang kejadian ini paling banyak adalah $q(n)/2^n$.

Misalkan Repeat menunjukkan kejadian dimana $r*$ digunakan oleh oracle enkripsi ketika menjawab setidaknya salah satu $\mathcal{A}'s$ pertanyaan . Seperti yang baru saja dibahas, peluang Pengulangan paling banyak adalah $q(n)/2^n$, dan peluang $\mathcal{A}$ berhasil di $PrivK^{cpa}_{\mathcal{A}\widetilde{\Pi}}(n)$ jika Pengulangan tidak terjadi tepat 1/2. Karena itu:

$$\begin{aligned}
&\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1]\\
&\quad=\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1 \wedge \mathsf{Repeat}]+\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1 \wedge \overline{\mathsf{Repeat}}]\\
&\quad\leq \Pr[\mathsf{Repeat}]+\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1 \mid \overline{\mathsf{Repeat}}] \;\leq\; \frac{q(n)}{2^n}+\frac{1}{2}.
\end{aligned}$$

Menggabungkan persamaan di atas dengan Persamaan (3.8), kita melihat bahwa terdapat fungsi negl yang dapat diabaikan sehingga

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \frac{q(n)}{2^n} + \mathsf{negl}(n).$$

Sejak q polinomial, q(n) dapat diabaikan. Selain itu, jumlah dua fungsi yang dapat diabaikan juga dapat diabaikan, sehingga terdapat fungsi yang dapat diabaikan negl' sehingga , melengkapi pembuktiannya.

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}'(n),$$

Keamanan konkrit. Bukti di atas menunjukkan hal itu

$$\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \frac{q(n)}{2^n} + \mathsf{negl}(n)$$

untuk beberapa fungsi yang dapat diabaikan diabaikan. Suku terakhir bergantung pada keamanan F sebagai fungsi pseudorandom; itu adalah batasan pada probabilitas pembedaan algoritma D (yang memiliki waktu berjalan kira-kira sama dengan musuh A). Istilah q(n) mewakili batas probabilitas bahwa nilai $r*$ yang digunakan untuk mengenkripsi tantangan ciphertext digunakan untuk mengenkripsi beberapa pesan lainnya.

3.6 MODE OPERASI

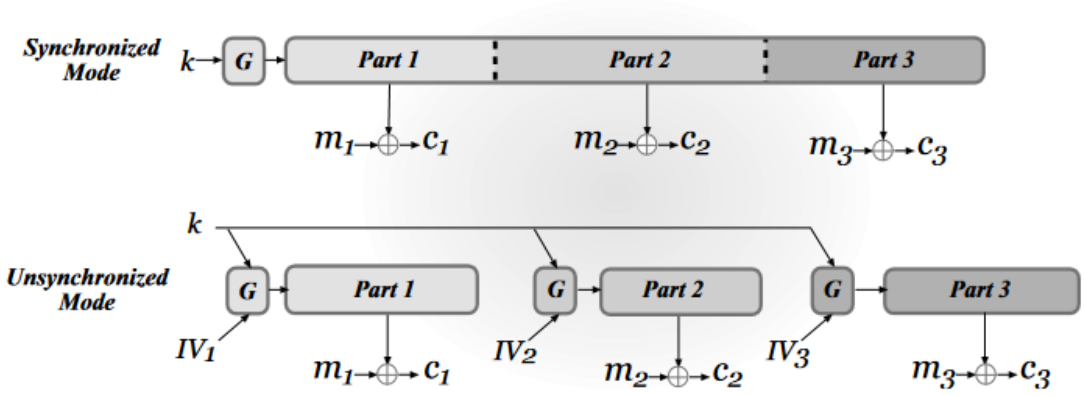
Mode operasi menyediakan cara untuk mengenkripsi pesan panjang dengan aman (dan efisien) menggunakan stream atau block cipher.

Mode Operasi Stream-Cipher

Konstruksi 3.17 menyediakan cara untuk membangun skema enkripsi menggunakan generator pseudorandom. Skema tersebut memiliki dua kelemahan utama. Pertama, seperti yang disajikan, panjang pesan yang akan dienkripsi harus ditetapkan dan diketahui sebelumnya. Kedua, skema ini hanya aman untuk EAV, bukan aman untuk CPA.

Stream cipher, yang dapat dipandang sebagai generator pseudorandom yang fleksibel, dapat digunakan untuk mengatasi kelemahan ini. Dalam praktiknya, stream cipher digunakan untuk enkripsi dalam dua cara: mode tersinkronisasi dan mode tidak tersinkronisasi. Mode tersinkronisasi. Dalam skema enkripsi stateful ini, pengirim dan penerima harus disinkronkan dalam artian mereka mengetahui berapa banyak teks biasa yang telah dienkripsi (resp., didekripsi) sejauh ini. Mode tersinkronisasi biasanya digunakan dalam satu sesi komunikasi antar pihak, di mana keadaan dapat diterima dan pesan diterima secara berurutan tanpa hilang. Intuisinya di sini adalah bahwa aliran pseudorandom yang panjang dihasilkan, dan bagian yang berbeda digunakan untuk mengenkripsi setiap pesan. Sinkronisasi diperlukan untuk memastikan dekripsi yang benar (yaitu, sehingga penerima mengetahui bagian mana dari aliran yang digunakan untuk mengenkripsi pesan berikutnya), dan untuk memastikan

bahwa tidak ada bagian dari aliran yang digunakan kembali. Kami menjelaskan mode ini secara rinci selanjutnya.



GAMBAR 3.4: Mode tersinkronisasi dan mode tidak tersinkronisasi.

Kita telah melihat dalam Algoritma 3.16 bahwa stream cipher dapat digunakan untuk membangun generator pseudorandom G_ℓ dengan faktor ekspansi apa pun yang diinginkanℓ. Kita dapat dengan mudah memodifikasi algoritma tersebut untuk mendapatkan generator pseudorandom G_∞ dengan panjang keluaran variabel. G_∞ mengambil dua input: sebuah seed s dan panjang output yang diinginkan 1^ℓ (kita tentukan ini dalam unary karena G_∞ akan berjalan dalam polinomial waktu dalam ℓ). Seperti pada Algoritma 3.16, $\mathrm{G}_\infty(\mathrm{s}, 1^\ell)$ menjalankan $\mathrm{Init}(\mathrm{s})$ dan kemudian berulang kali menjalankan GetBits sebanyak l kali.

Kita dapat menggunakan G_∞ pada Konstruksi 3.17 untuk menangani enkripsi pesan dengan panjang sembarang: enkripsi pesan m menggunakan kunci k dilakukan dengan menghitung ciphertext $\mathrm{c} := \mathrm{G}_\infty(\mathrm{k}, 1^{|\mathrm{m}|}) \oplus \mathrm{m}$; dekripsi ciphertext c menggunakan kunci k dilakukan dengan menghitung pesan $\mathrm{m} := \mathrm{G}_\infty(\mathrm{k}, 1^{|\mathrm{c}|}) \mathrm{c}$. Modifikasi kecil pada pembuktian Teorema 3.18 menunjukkan bahwa jika stream cipher aman maka skema enkripsi ini mempunyai enkripsi yang tidak dapat dibedakan dengan adanya penyadap.

Sedikit pemikiran menunjukkan bahwa jika pihak yang berkomunikasi ingin mempertahankan keadaan, maka mereka dapat menggunakan kunci yang sama untuk mengenkripsi banyak pesan. (Lihat Gambar 3.4.) Wawasan konseptualnya adalah para pihak dapat menangani banyak pesan m_1, m_2, . . . sebagai satu pesan yang panjang; lebih jauh lagi, Konstruksi 3.17 (serta versi modifikasi pada paragraf sebelumnya) memiliki properti bahwa bagian awal dari sebuah pesan dapat dienkripsi dan dikirimkan bahkan jika pesan lainnya belum diketahui. Konkritnya, kedua pihak berbagi kunci k dan keduanya memulai dengan menghitung $\mathrm{st}_0 := \mathrm{Init}(\mathrm{k})$. Untuk mengenkripsi pesan pertama m_1 dengan panjang ℓ_1, pengirim berulang kali menjalankan GetBits sebanyak ℓ_1 kali, dimulai dari st_0 , untuk mendapatkan aliran bit $pad_1 \stackrel{\text{def}}{=} y_1, \ldots, y_{\ell_1}$ bersama dengan status yang diperbarui $st_{\ell 1}; it$ kemudian mengirimkan $c_1 := pad_1 \oplus m_1$. Setelah menerima c_1, pihak lain berulang kali menjalankan GetBits sebanyak ℓ_1 kali untuk mendapatkan nilai yang sama pad_1 dan $st_{\ell 1}; it$ ia menggunakan pad_1 untuk memulihkan $m_1 := pad_1 \oplus c_1$. Kemudian, untuk mengenkripsi pesan kedua m_2 dengan panjang ℓ_2, pengirim akan berulang kali menjalankan GetBits

sebanyak ℓ_2 kali, dimulai dari $st_{\ell 1}$, untuk mendapatkan $pad_1 \stackrel{\text{def}}{=} y_1 + 1, \ldots, y_{\ell_1} + 2\ dan\ memperbarui\ status\ st_{\ell 1+\ell 1}$, lalu menghitung ciphertext $c_2 := pad_2 \oplus m_2$, dan seterusnya. Hal ini dapat berlanjut tanpa batas waktu, memungkinkan para pihak untuk mengirim pesan dalam jumlah tidak terbatas dengan panjang yang berubah-ubah. Kami mencatat bahwa dalam mode ini, stream cipher tidak perlu menggunakan IV.

Metode mengenkripsi banyak pesan ini mengharuskan pihak yang berkomunikasi untuk mempertahankan keadaan tersinkronisasi, yang menjelaskan terminologi "mode tersinkronisasi." Oleh karena itu, metode ini cocok digunakan jika dua pihak berkomunikasi dalam satu "sesi", namun metode ini tidak cocok untuk komunikasi sporadis atau jika salah satu pihak, seiring berjalannya waktu, dapat berkomunikasi dari perangkat yang berbeda. (Relatif mudah untuk menyimpan salinan kunci tetap di lokasi yang berbeda; lebih sulit untuk mempertahankan keadaan tersinkronisasi di beberapa lokasi.) Selain itu, jika pihak-pihak tersebut tidak sinkron (misalnya, karena salah satu transmisi antara pihak dibatalkan), dekripsi akan memberikan hasil yang salah. Sinkronisasi ulang dapat dilakukan, namun menambah overhead tambahan.

Mode tidak tersinkronisasi. Untuk stream cipher yang fungsi Initnya menerima vektor inisialisasi sebagai masukan, kita dapat mencapai enkripsi aman CPA tanpa status untuk pesan dengan panjang sembarang. Di sini kita memodifikasi G_∞ untuk menerima tiga masukan: benih s, vektor inisialisasi IV, dan panjang keluaran yang diinginkan 1A. Sekarang algoritme ini terlebih dahulu menghitung $st_0 := \mathrm{Init}(\mathrm{s}, \mathrm{IV})$ sebelum menjalankan GetBits berulang kali sebanyak ℓ kali. Enkripsi kemudian dapat dilakukan dengan menggunakan varian Konstruksi 3.30: enkripsi pesan m menggunakan kunci k dilakukan dengan memilih vektor inisialisasi seragam $IV \in \{0,1\}^n$ dan menghitung ciphertext $\mathrm{IV}, \mathrm{G}_\infty(\mathrm{s}, \mathrm{IV}, 1^{|\mathrm{m}|}) \oplus \mathrm{m}$; dekripsi dilakukan dengan cara alami. (Lihat Gambar 3.4.) Skema ini aman untuk CPA jika stream cipher sekarang memiliki properti yang lebih kuat sehingga, untuk polinomial apa pun l, fungsi F yang didefinisikan oleh $F_k(IV) \stackrel{\text{def}}{=} G_\infty(k, IV, 1^\ell)$ adalah a fungsi pseudoacak. (Faktanya, F memerlukan hanya berupa pseudorandom ketika dievaluasi pada input yang seragam. Fungsi berkunci dengan properti yang lebih lemah ini disebut fungsi pseudorandom lemah.)

Mode Operasi Block-Cipher

Kita telah melihat konstruksi skema enkripsi aman CPA berdasarkan fungsi pseudorandom/cipher blok. Namun Konstruksi 3.30 (dan perluasannya pada pesan dengan panjang sembarang seperti yang dibahas di akhir Bagian 3.4) memiliki kelemahan yaitu panjang ciphertext adalah dua kali lipat panjang plaintext. Mode operasi block-cipher menyediakan cara mengenkripsi pesan dengan panjang sembarang menggunakan teks sandi yang lebih pendek.

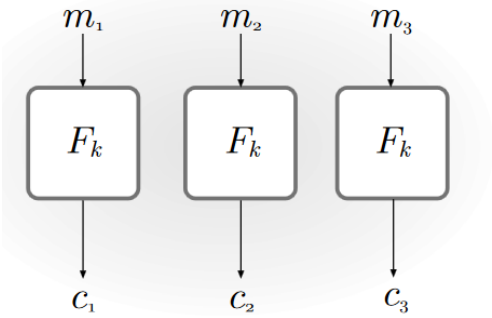
Pada bagian ini, misalkan F adalah sebuah cipher blok dengan panjang blok n. Kita asumsikan di sini bahwa semua pesan m yang dienkripsi mempunyai panjang kelipatan n, dan tulis $\mathrm{m} = \mathrm{m}_1, \mathrm{m}_2, \ldots, \mathrm{m}_\ell$ dimana setiap $\mathrm{m}_1 \{0,1\}^n$ mewakili satu blok teks biasa. Pesan yang bukan merupakan kelipatan n selalu dapat diisi dengan jelas agar memiliki panjang kelipatan n dengan menambahkan angka 1 diikuti dengan angka 0 secukupnya, sehingga asumsi ini tidak

kehilangan keumumannya. Beberapa mode operasi block-cipher diketahui; kami menyajikan empat yang paling umum dan mendiskusikan keamanannya.

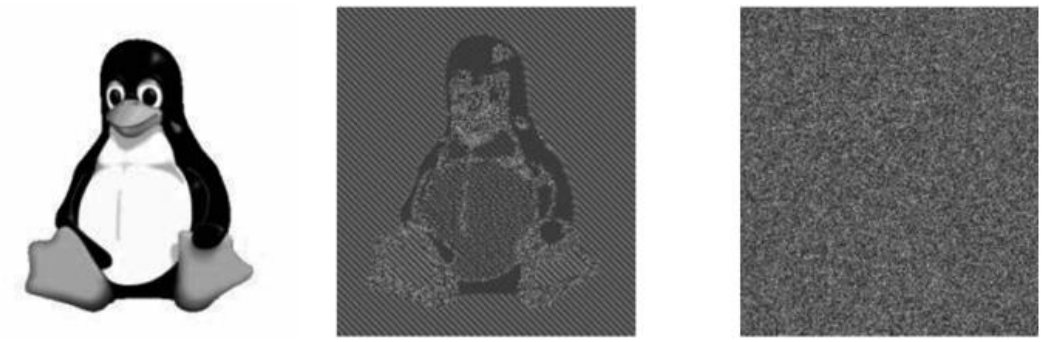
Mode Buku Kode Elektronik (ECB).

Ini adalah mode operasi naif di mana teks sandi diperoleh dengan penerapan langsung sandi blok ke setiap blok teks biasa. Artinya, $c := < F_k(m_1), F_k(m_2), \ldots, F_k(m_\ell) >$; lihat Gambar 3.5. Dekripsi dilakukan dengan cara yang jelas, menggunakan fakta bahwa F_k^{-1} dapat dihitung secara efisien. Mode ECB bersifat deterministik dan oleh karena itu tidak aman bagi CPA. Lebih buruk lagi, enkripsi mode ECB bahkan tidak memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap. Hal ini dikarenakan jika suatu blok diulang pada plaintext, maka akan mengakibatkan blok berulang pada ciphertext. Dengan demikian, mudah untuk membedakan enkripsi teks biasa yang terdiri dari dua blok identik dari enkripsi teks biasa yang terdiri dari dua blok berbeda. Ini bukan sekedar masalah teoretis. Pertimbangkan untuk mengenkripsi gambar dengan sekelompok kecil piksel yang sesuai dengan blok teks biasa. Enkripsi menggunakan mode ECB dapat mengungkap sejumlah besar informasi tentang pola pada gambar, sesuatu yang tidak boleh terjadi saat menggunakan skema enkripsi aman. Gambar 3.6 menunjukkan hal ini.

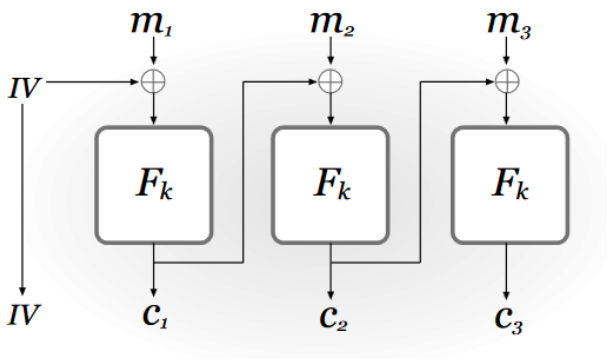
Oleh karena itu, mode ECB tidak boleh digunakan. (Kami memasukkannya hanya karena signifikansi historisnya.)



GAMBAR 3.5: Mode Buku Kode Elektronik (ECB).



GAMBAR 3.6: Ilustrasi bahaya penggunaan mode ECB. Gambar tengah adalah enkripsi gambar di sebelah kiri menggunakan mode ECB; gambar di sebelah kanan adalah enkripsi gambar yang sama menggunakan mode aman. (Diambil dari http://en.wikipedia.org dan berasal dari gambar yang dibuat oleh Larry Ewing (lewing@isc.tamu.edu) menggunakan The GIMP.)



GAMBAR 3.7: Mode Cipher Block Chaining (CBC).

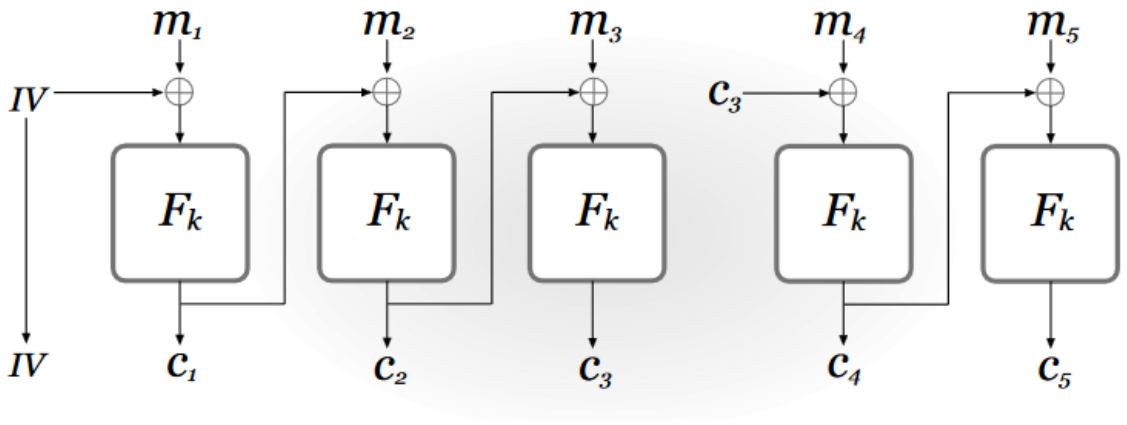
Mode Rantai Blok Sandi (CBC). Untuk mengenkripsi menggunakan mode ini, vektor inisialisasi seragam (IV) dengan panjang n dipilih terlebih dahulu. Kemudian, blok ciphertext dihasilkan dengan menerapkan blok cipher ke XOR dari blok plaintext saat ini dan blok ciphertext sebelumnya. Artinya, himpunan $c_0 := IV$ lalu, untuk $i = 1$ ke ℓ, himpunan $c_i := F_k(c_{i-1} \oplus m_i)$. Teks tersandi terakhir adalah $\langle c_0, c_1, \ldots, c_\ell \rangle$. (Lihat Gambar 3.7.) Dekripsi ciphertext $c_0, \ldots, c_\ell$ dilakukan dengan menghitung $m_i := F_k^{-1}(c_i) \oplus c_{i-1}$ untuk $i = 1, \ldots, \ell$. Kami menekankan bahwa IV termasuk dalam ciphertext; ini penting agar dekripsi dapat dilakukan.

Yang penting, enkripsi dalam mode CBC bersifat probabilistik dan telah terbukti bahwa jika F adalah permutasi pseudorandom maka enkripsi mode CBC aman terhadap CPA. Kelemahan utama dari mode ini adalah enkripsi harus dilakukan secara berurutan karena blok ciphertext c_{i-1} diperlukan untuk mengenkripsi blok plaintext mi. Jadi, jika pemrosesan paralel tersedia, enkripsi mode CBC mungkin bukan pilihan yang paling efisien.

Seseorang mungkin tergoda untuk berpikir bahwa cukup menggunakan IV yang berbeda (daripada IV acak) untuk setiap enkripsi, misalnya, menggunakan IV = 1 terlebih dahulu dan kemudian menambah IV sebanyak satu setiap kali pesan dienkripsi. Dalam Latihan 3.20, kami meminta Anda untuk menunjukkan bahwa varian enkripsi mode CBC ini tidak aman. Kita juga dapat mempertimbangkan varian stateful dari enkripsi mode CBC disebut mode CBC berantai di mana blok terakhir dari teks sandi sebelumnya digunakan sebagai IV saat mengenkripsi pesan berikutnya. Hal ini mengurangi bandwidth, karena IV tidak perlu dikirim setiap saat. Lihat Gambar 3.8, di mana pesan awal $m_1 m_2 m_3$, dienkripsi menggunakan IV acak, dan kemudian pesan kedua $m_4 m_5$ dienkripsi menggunakan c_1 sebagai IV. (Sebaliknya, enkripsi menggunakan mode CBC stateless akan menghasilkan IV baru saat mengenkripsi pesan kedua.) Mode CBC berantai digunakan di SSL 3.0 dan TLS 1.0.

Tampaknya mode CBC berantai sama amannya dengan mode CBC, karena enkripsi CBC berantai m_1, m_2, m_3 diikuti dengan enkripsi m_4, m_5 menghasilkan blok ciphertext yang sama dengan enkripsi mode CBC pada pesan (tunggal). bijak m_1, m_2, m_3, m_4, m_5. Namun demikian, mode CBC berantai rentan terhadap serangan teks biasa yang dipilih. Dasar dari serangan ini adalah musuh mengetahui terlebih dahulu "vektor inisialisasi" yang akan digunakan untuk pesan terenkripsi kedua. Kami menggambarkan serangan tersebut secara informal, berdasarkan Gambar 3.8. Asumsikan penyerang mengetahui bahwa $m_1 \in \{m_1^0 m_1^1\}$, dan

mengamati ciphertext pertama IV, c_1, c_2, c_3. Penyerang kemudian meminta enkripsi pesan kedua m_4, m_5 dengan $m_4 = IV \oplus m_1^0 \oplus c_3$, dan mengamati ciphertext kedua c_4, c_5. Seseorang dapat memverifikasi bahwa $m_1 = m_1^0$ jika dan hanya jika $c_4 = c_1$. Contoh ini harus menjadi peringatan keras untuk tidak melakukan modifikasi apa pun terhadap skema kriptografi, meskipun modifikasi tersebut tampaknya tidak berbahaya.



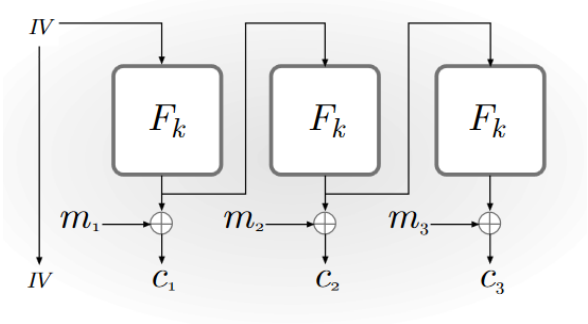
GAMBAR 3.8: CBC yang dirantai.

Mode Umpan Balik Keluaran (OFB).

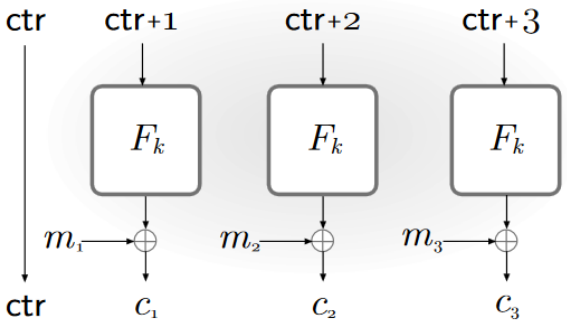
Mode ketiga yang kami hadirkan dapat dilihat sebagai mode stream-cipher yang tidak tersinkronisasi, dimana stream cipher dibangun dengan cara tertentu dari block cipher yang mendasarinya. Kami menjelaskan modenya secara langsung. Pertama, seragam $\text{IV} \in \{0,1\}^n$ dipilih. Kemudian, aliran pseudoran-dom dihasilkan dari IV dengan cara berikut: Definisikan $y_0 := IV$, dan atur blok ke y_i dari aliran tersebut menjadi $y_i := F_k(y_{i-1})$. Setiap blok teks biasa dienkripsi dengan melakukan XOR dengan blok aliran yang sesuai; yaitu $c_i := y_i \oplus m_i$. (Lihat Gambar 3.9.) Seperti dalam mode CBC, IV disertakan sebagai bagian dari ciphertext untuk memungkinkan dekripsi. Namun, berbeda dengan mode CBC, di sini F tidak diharuskan dapat dibalik. (Bahkan, hal ini tidak perlu berupa permutasi.) Lebih jauh lagi, seperti pada mode operasi stream-cipher, di sini panjang teks biasa tidak perlu merupakan kelipatan panjang blok. Sebaliknya, aliran yang dihasilkan dapat dipotong sesuai panjang teks biasa. Keuntungan lain dari mode OFB adalah varian statefulnya (di mana nilai akhir y_ℓ yang digunakan untuk mengenkripsi beberapa pesan digunakan sebagai IV untuk mengenkripsi pesan berikutnya) aman. Varian stateful ini setara dengan mode stream-cipher tersinkronisasi, dengan stream cipher yang dibuat dari block cipher dengan cara spesifik yang baru saja dijelaskan.

Mode OFB dapat terbukti aman terhadap CPA jika F adalah fungsi pseudorandom. Meskipun enkripsi dan dekripsi harus dilakukan secara berurutan, mode ini memiliki keunggulan dibandingkan mode CBC yaitu sebagian besar komputasi (yaitu, komputasi aliran pseudorandom) dapat dilakukan secara independen dari pesan sebenarnya yang akan dienkripsi. Jadi, dimungkinkan untuk menghasilkan aliran pseudorandom terlebih dahulu

menggunakan prapemrosesan, setelah itu enkripsi titik teks biasa (setelah diketahui) menjadi sangat cepat.



GAMBAR 3.9: Mode Umpan Balik Keluaran (OFB).



GAMBAR 3.10: Mode Penghitung (RKT).

Mode Penghitung (RKT).

Mode counter juga dapat dilihat sebagai mode stream-cipher yang tidak tersinkronisasi, dimana stream cipher dibangun dari block cipher seperti pada Konstruksi 3.29. Kami memberikan deskripsi lengkapnya di sini. Untuk mengenkripsi menggunakan mode CTR, nilai seragam $\text{ctr} \in \{0,1\}^n$ dipilih terlebih dahulu. Kemudian, aliran pseudorandom dihasilkan dengan menghitung $y_i := F_k(\text{ctr} + i)$, di mana ctr dan i dipandang sebagai bilangan bulat dan penjumlahan dilakukan modulo 2^n. Blok ciphertext ke-i adalah $c_i := y_i \oplus m_i$, dan IV dikirim lagi sebagai bagian dari ciphertext; lihat Gambar 3.10. Perhatikan lagi bahwa dekripsi tidak memerlukan F dapat dibalik, atau bahkan permutasi. Seperti halnya mode OFB, mode "stream-cipher" lainnya, aliran yang dihasilkan dapat dipotong sesuai dengan panjang teks biasa, pemrosesan awal dapat digunakan untuk menghasilkan aliran pseudorandom sebelum pesan diketahui, dan varian stateful dari mode CTR aman.

Berbeda dengan semua mode aman yang dibahas sebelumnya, mode CTR memiliki keunggulan yaitu enkripsi dan dekripsi dapat diparalelkan sepenuhnya, karena semua blok aliran pseudorandom dapat dihitung secara independen satu sama lain. Berbeda dengan OFB, blok ke-i dari ciphertext juga dapat didekripsi hanya dengan menggunakan evaluasi tunggal F. Fitur-fitur ini menjadikan mode CTR sebagai pilihan yang menarik.

Mode CTR juga cukup mudah untuk dianalisis:

TEOREMA 3.32 Jika F adalah fungsi pseudorandom, maka mode CTR aman untuk CPA.

BUKTI Kita mengikuti pola yang sama seperti pada pembuktian Teorema 3.31: pertama-tama kita mengganti F dengan fungsi acak dan kemudian menganalisis skema yang dihasilkan. Misalkan $\Pi = (\mathrm{Gen}, \mathrm{Enc}, \mathrm{Dec})$ adalah skema enkripsi mode CTR (stateless), dan misalkan $\widetilde{\Pi} = (\widetilde{\mathrm{Gen}}, \widetilde{\mathrm{Enc}}, \widetilde{\mathrm{Dec}})$ adalah skema enkripsi yang identik dengan Π kecuali bahwa fungsi yang benar-benar acak digunakan dalam tempat F_k. Artinya, $\widetilde{\mathrm{Gen}}(1^n)$ memilih fungsi seragam $f \in \mathrm{Func}_n$, dan $\widetilde{Enc}$ mengenkripsi seperti Enc kecuali bahwa f digunakan sebagai pengganti F_k. (Sekali lagi, baik Gen maupun Enc tidak efisien tetapi hal ini tidak menjadi masalah untuk tujuan mendefinisikan eksperimen yang melibatkan $\widetilde{\Pi}$.) Perbaiki musuh PPT sembarang $\mathcal{A}$, dan misalkan $\mathrm{q}(\mathrm{n})$ menjadi batas atas polinomial pada jumlah enkripsi- kueri Oracle dibuat oleh $\mathcal{A}(1^n)$ serta jumlah maksimum blok dalam kueri tersebut dan jumlah maksimum blok dalam m_0 dan m_1. Sebagai pembuktian langkah pertama, kita menyatakan bahwa terdapat fungsi yang dapat diabaikan sehingga dapat diabaikan

$$\left|\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)=1\right] - \Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\right]\right| \leq \mathsf{negl}(n). \qquad (3.12)$$

Hal ini dibuktikan dengan reduksi dengan cara serupa dengan langkah analogi dalam pembuktian Teorema 3.31, dan dibiarkan sebagai latihan bagi pembaca.

Kami selanjutnya mengklaim itu

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1\right] < \frac{1}{2} + \frac{2q(n)^2}{2^n}. \qquad (3.13)$$

Dikombinasikan dengan Persamaan (3.12) artinya

$$\Pr\left[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n)=1\right] < \frac{1}{2} + \frac{2q(n)^2}{2^n} + \mathsf{negl}(n).$$

Karena q adalah polinomial, $\frac{2_{q(n)^2}}{2^n}$dapat diabaikan dan ini melengkapi pembuktian.

Sekarang kita buktikan Persamaan (3.13). Perbaiki beberapa nilai n untuk parameter keamanan. Misalkan $\ell^* \mathrm{q}(\mathrm{n})$ menunjukkan panjang (dalam blok) pesan $\mathrm{m}_0\ \mathrm{m}_1$yang dihasilkan dalam percobaan $PrivK^{cpa}_{\mathcal{A}\widetilde{\Pi}}(n)$, dan biarkan ctr^* menunjukkan nilai awal yang digunakan ketika A,Π˜ menghasilkan teks sandi tantangan. Demikian pula, misalkan $\mathrm{l}_i \leq \mathrm{q}(\mathrm{n})$ adalah panjang (dalam blok) kueri oracle enkripsi ke-i yang dibuat oleh $\mathcal{A}$, dan misalkan ctr_i menunjukkan nilai awal yang digunakan saat menjawab kueri ini. Ketika kueri enkripsi-Oracle ke-i dijawab, f diterapkan ke nilai $\mathrm{ctr}_i + 1, \ldots, \mathrm{ctr}_i + \ell_i$. Ketika tantangan ciphertext

dienkripsi, ƒ diterapkan ke $ctr^* + 1, \ldots, ctr^* + \ell^i$, dan blok ciphertext ke-i dihitung dengan melakukan XORing ƒ $(ctr^* + i)$ dengan blok pesan ke-i. Ada dua kasus:

- Tidak ada $i, j, j^* \geq 1$ (dengan $j \leq \ell_i \; dan \; j^* \; \ell^i$) yang mana $\text{ctr}_i + j = ctr^* + j^*$: Dalam hal ini, nilai ƒ $(ctr^* + 1), \ldots,$ ƒ $(ctr^* + \ell^*)$ yang digunakan saat mengenkripsi teks sandi tantangan terdistribusi secara merata dan tidak bergantung pada eksperimen lainnya karena ƒ tidak diterapkan pada input mana pun saat mengenkripsi kueri oracle musuh. Ini berarti bahwa tantangan ciphertext dihitung dengan melakukan XOR pada aliran bit yang seragam dengan pesan mb, sehingga probabilitas $\mathcal{A}$ keluaran $b' = b$ adalah tepat 1/2 (seperti pada kasus one-time pad).
- Terdapat $i, j, j^* \geq 1$ (dengan $j \leq li$ dan $j^* \leq \ell^*$) yang mana $\text{ctr}_i + j = ctr^* + j^*$: Kita menyatakan kejadian ini dengan Tumpang Tindih. Dalam hal ini berpotensi menentukan pesan mana yang dienkripsi untuk memberikan tantangan ciphertext, karena dapat menyimpulkan nilai ƒ $(\text{ctr}_i + j) =$ ƒ $(ctr^* + j^*)$ dari jawaban oracle ke-i pertanyaan.

Mari kita menganalisis kemungkinan terjadinya Tumpang Tindih. Probabilitasnya maksimal jika ℓ^* dan masing-masing ℓ_i sebesar mungkin, jadi asumsikan $\ell^* = \ell_i = q(n)$ untuk semua i. Misal Tumpang tindih menyatakan kejadian barisan $ctr_i + 1, \ldots, ctr_i + q(n)$ tumpang tindih dengan barisan ctr∗ + 1, . . . , ctr∗ + q(n); maka Overlap adalah kejadian terjadinya Tumpang Tindih untuk beberapa i. Karena terdapat paling banyak q(n) kueri oracle, ikatan gabungan (lih. Proposisi A.7) memberikan

$$\Pr[\text{Overlap}] \leq \sum_{i=1}^{q(n)} \Pr[\text{Overlap}_i]. \qquad (3.14)$$

Memperbaiki ctr^*, kejadian Tumpang tindih terjadi tepat ketika ctr_i memenuhi

$$\text{ctr}^* - \text{q(n)} + 1 \leq \text{ctr}_\text{i} \leq \text{ctr}^* + \text{q(n)} - 1.$$

Karena terdapat $2q(n) - 1$ nilai ctri yang mana terjadi Tumpang Tindih, dan ctri dipilih secara seragam dari {0, 1}n, kita melihat bahwa

$$\Pr[\text{Overlap}_i] = \frac{2q(n) - 1}{2^n} < \frac{2q(n)}{2^n}.$$

Dikombinasikan dengan Persamaan (3.14), diperoleh $\Pr[\text{Overlap}] < 2q(n)^2/2^n$. Mengingat hal di atas, kita dapat dengan mudah membatasi probabilitas keberhasilan A:

$$\begin{aligned}\Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1] &= \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1 \wedge \overline{\mathsf{Overlap}}] \\ &\quad + \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1 \wedge \mathsf{Overlap}] \\ &\leq \Pr[\mathsf{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\widetilde{\Pi}}(n)=1 \mid \overline{\mathsf{Overlap}}] + \Pr[\mathsf{Overlap}] \\ &< \frac{1}{2} + \frac{2q(n)^2}{2^n}.\end{aligned}$$

Ini membuktikan Persamaan (3.13) dan melengkapi pembuktiannya.

Mode operasi dan gangguan pesan. Dalam banyak teks, mode operasi juga dibandingkan berdasarkan seberapa baik mode tersebut melindungi terhadap modifikasi ciphertext yang bersifat permusuhan. Kami tidak menyertakan perbandingan seperti itu di sini karena masalah integritas pesan atau otentikasi pesan harus ditangani secara terpisah dari enkripsi, dan kami akan membahasnya di bab berikutnya. Tak satu pun dari mode di atas mencapai integritas pesan seperti yang akan kita definisikan di sana. Kami menunda diskusi lebih lanjut ke bab berikutnya.

Berkenaan dengan perilaku mode yang berbeda ketika terdapat kesalahan transmisi yang "jinak" (yakni, non-adversarial). Namun kami mencatat bahwa secara umum kesalahan tersebut dapat diatasi dengan menggunakan teknik standar (misalnya, koreksi kesalahan atau transmisi ulang).

Panjang blok dan keamanan beton. Mode CBC, OFB, dan CTR semuanya menggunakan IV acak. Hal ini berdampak pada pengacakan proses enkripsi, dan memastikan bahwa (dengan probabilitas tinggi) cipher blok yang mendasarinya selalu dievaluasi berdasarkan masukan baru (yaitu, baru). Hal ini penting karena, seperti yang telah kita lihat pada pembuktian Teorema 3.31 dan Teorema 3.32, jika input ke block cipher digunakan lebih dari satu kali maka keamanan dapat dilanggar.

Panjang blok dari sebuah cipher blok memiliki dampak yang signifikan terhadap keamanan konkret skema enkripsi berdasarkan cipher tersebut. Pertimbangkan, misalnya, mode CTR menggunakan cipher blok F dengan panjang blok ℓ. IV kemudian menjadi string ℓ -bit yang seragam, dan kita mengharapkan IV berulang setelah mengenkripsi sekitar pesan 2A/2. Jika ℓ terlalu pendek, bahkan jika F aman sebagai permutasi pseudorandom—batas keamanan konkrit yang dihasilkan akan terlalu lemah untuk penerapan praktis. Konkretnya, jika $\ell = 64$ seperti kasus DES (sebuah cipher blok yang akan kita pelajari di Bab 6), maka setelah $2^{32} \approx 4.300.000.000$ enkripsi atau kira-kira 34 gigabyte plaintext—diperkirakan akan terjadi IV berulang. Meskipun ini mungkin tampak seperti data yang banyak, namun lebih kecil dari kapasitas hard drive modern.

penyalahgunaan IV. Dalam uraian dan pembahasan kami tentang berbagai mode (aman), kami berasumsi IV acak dipilih setiap kali pesan dienkripsi. Bagaimana jika asumsi ini salah, misalnya karena keacakan yang buruk atau penerapan yang salah? Tentu saja kami tidak dapat lagi menjamin keamanan sebagaimana dimaksud dalam Definisi 3.22. Namun, dari

sudut pandang praktis, “mode stream-cipher” (OFB dan CTR) jauh lebih buruk daripada mode CBC. Jika IV berulang saat menggunakan IV, penyerang dapat melakukan XOR pada dua ciphertext yang dihasilkan dan mempelajari banyak informasi tentang keseluruhan isi dari kedua pesan terenkripsi (seperti yang telah kita lihat sebelumnya dalam konteks one-time pad jika kuncinya adalah digunakan kembali). Namun, dengan mode CBC, kemungkinan besar setelah hanya beberapa blok, masukan ke cipher blok akan “menyimpang” dan penyerang tidak akan dapat mempelajari informasi apa pun di luar beberapa blok pesan pertama. Salah satu cara untuk mengatasi potensi penyalahgunaan IV adalah dengan menggunakan enkripsi stateful, seperti yang dibahas dalam konteks mode OFB dan CTR. Jika enkripsi stateful tidak memungkinkan, dan terdapat kekhawatiran mengenai potensi penyalahgunaan IV, maka mode CBC direkomendasikan karena alasan yang dijelaskan di atas.

3.7 SERANGAN CIPHERTEXT TERPILIH

Mendefinisikan Keamanan CCA

Hingga saat ini kami telah mendefinisikan keamanan terhadap dua jenis serangan: penyadapan pasif dan serangan teks biasa yang dipilih. Serangan teks sandi terpilih bahkan lebih kuat. Dalam serangan teks sandi yang dipilih, musuh mempunyai kemampuan tidak hanya untuk mendapatkan enkripsi pesan pilihannya (seperti dalam serangan teks biasa yang dipilih), namun juga untuk mendapatkan dekripsi teks sandi pilihannya (dengan satu pengecualian yang akan dibahas nanti). Secara formal, kami memberi musuh akses ke oracle dekripsi selain oracle enkripsi. Kami menyajikan definisi formal dan menunda diskusi lebih lanjut.

Pertimbangkan eksperimen berikut untuk skema enkripsi kunci pribadi apa pun Π = (Gen, Enc, Des), musuh A, dan nilai n untuk parameter keamanan.

Eksperimen CCA yang tidak dapat dibedakan $\mathrm{PrivK}_{\mathcal{A}\Pi}^{cca}(\mathrm{n})$:

1. Kunci k dihasilkan dengan menjalankan $Gen(1^n)$.
2. Musuh $\mathcal{A}$ diberi masukan 1^n dan akses oracle ke $\mathrm{Enc}_k(\cdot)$ dan $\mathrm{Dec}_k(\cdot)$. Ini menghasilkan sepasang pesan $\mathrm{m}_0, \mathrm{m}_1$ dengan panjang yang sama.
3. Bit seragam $\mathrm{b} \in \{0,1\}$ dipilih, dan kemudian teks tersandi $\mathrm{c} \leftarrow \mathrm{Enc}_k(\mathrm{m}_b)$ dihitung dan diberikan kepada $\mathcal{A}$. Kami menyebut c sebagai tantangan ciphertext.
4. Musuh $\mathcal{A}$ terus memiliki akses oracle ke $\mathrm{Enc}_k(\cdot)$dan $\mathrm{Dec}_k(\cdot)$, namun tidak diperbolehkan untuk menanyakan yang terakhir pada tantangan ciphertext itu sendiri. Akhirnya, $\mathcal{A}$ mengeluarkan sedikit b′.
5. Keluaran percobaan ditetapkan 1 jika $\mathrm{b}' = \mathrm{b}$, dan 0 jika tidak. Jika keluaran percobaannya adalah 1, kita katakan $\mathcal{A}$ berhasil.

DEFINISI 3.33 Skema enkripsi kunci pribadi Π memiliki enkripsi yang tidak dapat dibedakan pada serangan teks sandi yang dipilih, atau aman terhadap CCA, jika untuk semua musuh waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan sehingga:

$$\Pr[\mathsf{PrivK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n),$$

di mana probabilitas diambil alih semua keacakan yang digunakan dalam percobaan.

Untuk kelengkapannya, kami mencatat bahwa analogi alami dari Teorema 3.24 juga berlaku untuk keamanan CCA. (Yaitu, jika suatu skema memiliki enkripsi yang tidak dapat dibedakan dalam serangan teks sandi yang dipilih, maka skema tersebut memiliki beberapa enkripsi yang tidak dapat dibedakan dalam serangan teks sandi yang dipilih, yang didefinisikan dengan tepat.)

Dalam percobaan di atas, akses musuh ke oracle dekripsi tidak terbatas kecuali ada batasan bahwa musuh tidak boleh meminta dekripsi teks sandi tantangan itu sendiri. Pembatasan ini diperlukan atau jika tidak maka jelas tidak ada harapan bagi skema enkripsi apa pun untuk memenuhi definisi tersebut. Pada titik ini Anda mungkin bertanya-tanya apakah serangan teks sandi yang dipilih secara realistis memodelkan serangan di dunia nyata. Seperti dalam kasus serangan teks biasa yang dipilih, kami tidak mengharapkan pihak yang jujur mendekripsi teks tersandi pilihan musuh secara sewenang-wenang. Namun demikian, mungkin ada skenario di mana musuh mungkin dapat memengaruhi apa yang didekripsi dan mempelajari sebagian informasi tentangnya.

Misalnya:

1. Dalam contoh Midway dari Bagian 3.4 dapat dibayangkan bahwa cryptanalyst AS juga mencoba mengirim pesan terenkripsi ke Jepang dan kemudian memantau perilaku mereka. Perilaku seperti itu (misalnya, pergerakan kekuatan dan sejenisnya) dapat memberikan informasi penting tentang teks biasa yang mendasarinya.
2. Bayangkan seorang pengguna mengirimkan pesan terenkripsi ke banknya. Musuh mungkin dapat mengirimkan teks sandi atas nama pengguna tersebut; bank akan mendekripsi ciphertext tersebut dan musuh dapat mengetahui sesuatu tentang hasilnya. Misalnya, jika ciphertext berhubungan dengan plaintext yang bentuknya buruk (misalnya, pesan yang tidak dapat dipahami, atau pesan yang tidak diformat dengan benar), musuh mungkin dapat menyimpulkan hal ini dari reaksi bank (yaitu, pola komunikasi selanjutnya).). Serangan praktis jenis ini disajikan pada Bagian 3.7 di bawah.
3. Enkripsi sering digunakan pada protokol tingkat yang lebih tinggi; misalnya, skema enkripsi dapat digunakan sebagai bagian dari protokol otentikasi di mana satu pihak mengirimkan teks sandi ke pihak lain, yang mendekripsinya dan mengembalikan hasilnya. Dalam hal ini, salah satu pihak yang jujur berperilaku persis seperti oracle dekripsi.

Ketidakamanan skema yang telah kami pelajari. Tidak ada skema enkripsi yang kita lihat sejauh ini yang aman terhadap CCA. Kami mendemonstrasikan hal ini untuk Konstruksi 3.30, di mana enkripsi dihitung sebagai $Enc_k(m) = r,\ F_k(r)m.$ Pertimbangkan musuh $\mathcal{A}$yang menjalankan eksperimen ketidakmampuan membedakan CCA yang memilih $\mathrm{m}_0 = 0^{\mathrm{n}} \,\mathrm{dan}\, \mathrm{m}_1 = 1^{\mathrm{n}}$. Kemudian, setelah menerima ciphertext $c = r, s$, musuh dapat membalik bit pertama s dan meminta dekripsi ciphertext c′ yang dihasilkan. Karena $\mathrm{c}' \neq \mathrm{c}$, kueri ini diperbolehkan dan

oracle dekripsi menjawab dengan $10n - 1$ (dalam hal ini jelas bahwa $b = 0$) atau $01n - 1$ (dalam hal ini $b = 1$). Contoh ini menunjukkan bahwa keamanan CCA cukup ketat.

Skema enkripsi apa pun yang memungkinkan teks tersandi "dimanipulasi" dengan cara apa pun yang terkontrol tidak dapat aman dengan CCA. Dengan demikian, keamanan CCA menyiratkan properti yang sangat penting yang disebut non-malleability. Secara sederhana, skema enkripsi yang tidak dapat ditempa memiliki sifat bahwa jika musuh mencoba memodifikasi ciphertext tertentu, hasilnya adalah ciphertext yang tidak valid atau yang mendekripsi menjadi plaintext yang tidak ada hubungannya dengan ciphertext aslinya. Ini adalah properti yang sangat berguna untuk skema yang digunakan dalam protokol kriptografi yang kompleks.

Membangun skema enkripsi yang aman CCA. Kami menunjukkan cara membangun skema enkripsi yang aman dengan CCA di Bagian 4.5. Konstruksinya disajikan di sana karena menggunakan alat yang kami kembangkan di Bab 4.

Serangan Padding-Oracle

Serangan teks sandi terpilih pada Konstruksi 3.30 yang dijelaskan pada bagian sebelumnya agak dibuat-buat, karena mengasumsikan penyerang dapat memperoleh dekripsi lengkap teks sandi yang dimodifikasi. Meskipun serangan semacam ini diperbolehkan berdasarkan Definisi 3.33, tidak jelas apakah serangan ini menimbulkan kekhawatiran serius dalam praktiknya. Di sini kami menunjukkan serangan teks sandi terpilih pada skema enkripsi yang alami dan banyak digunakan; Selain itu, serangan tersebut hanya memerlukan kemampuan penyerang untuk menentukan apakah teks sandi yang dimodifikasi dapat didekripsi dengan benar atau tidak. Informasi tersebut sering kali sangat mudah diperoleh karena, misalnya, server mungkin meminta transmisi ulang atau mengakhiri sesi jika menerima ciphertext yang tidak didekripsi dengan benar, dan salah satu dari peristiwa ini akan menghasilkan perubahan nyata dalam lalu lintas yang diamati. Serangan tersebut telah terbukti berhasil dalam praktiknya pada berbagai protokol yang digunakan; kami memberikan satu contoh konkrit di akhir bagian ini.

Seperti disebutkan sebelumnya, saat menggunakan mode CBC, panjang teks biasa harus kelipatan panjang blok; jika teks biasa tidak memenuhi properti ini, teks tersebut harus diisi sebelum dienkripsi. Kami menyebut teks biasa asli sebagai pesan, dan hasil setelah padding sebagai data yang dikodekan. Skema padding yang kita gunakan harus memungkinkan penerima untuk menentukan dengan jelas dari data yang dikodekan di mana pesan berakhir. Salah satu pendekatan yang populer dan terstandarisasi adalah dengan menggunakan padding PKCS #5. Asumsikan pesan asli memiliki jumlah byte yang tidak terpisahkan, dan misalkan L menunjukkan panjang blok (dalam byte) dari cipher blok yang digunakan. Misal b menunjukkan jumlah byte yang perlu ditambahkan ke pesan agar panjang total data yang disandikan menjadi kelipatan panjang blok. Di sini, b adalah bilangan bulat antara 1 dan L, inklusif. (Kita tidak dapat memiliki b = 0 karena ini akan menyebabkan padding yang ambigu. Jadi, jika panjang pesan sudah merupakan kelipatan dari panjang blok, L byte padding akan ditambahkan.) Kemudian kita menambahkan string yang berisi bilangan bulat b ke dalam pesan. (diwakili dalam 1 byte, atau 2 digit heksadesimal) diulang sebanyak b kali.

Artinya, jika diperlukan padding 1 byte maka string 0x01 (ditulis dalam heksadesimal) ditambahkan; jika diperlukan padding 4 byte maka string heksadesimal 0x04040404 ditambahkan; dll. Data yang dikodekan kemudian dienkripsi menggunakan enkripsi mode CBC biasa.

Saat mendekripsi, penerima pertama-tama menerapkan dekripsi mode CBC seperti biasa untuk memulihkan data yang dikodekan, dan kemudian memeriksa apakah data yang dikodekan sudah diisi dengan benar. (Ini mudah dilakukan: cukup baca nilai b dari byte terakhir dan kemudian verifikasi bahwa b byte terakhir dari hasil semuanya memiliki nilai b.) Jika ya, maka padding akan dilepas dan pesan asli dikembalikan. Jika tidak, prosedur standarnya adalah mengembalikan kesalahan "*padding buruk*" (misalnya, di Java, pengecualian standar disebut javax.crypto.BadPaddingException). Kehadiran pesan kesalahan seperti itu memberikan oracle dekripsi parsial kepada musuh. Artinya, musuh dapat mengirim ciphertext apa pun ke server dan mempelajari (berdasarkan apakah kesalahan "*padding buruk*" dikembalikan atau tidak) apakah data yang disandikan mendasarinya diisi dalam format yang benar. Meskipun ini mungkin tampak seperti informasi yang tidak berarti, kami menunjukkan bahwa hal ini memungkinkan musuh untuk sepenuhnya memulihkan pesan asli untuk teks tersandi apa pun yang dipilihnya.

Kami menggambarkan serangan pada ciphertext 3 blok untuk kesederhanaan. Misal IV, c_1, c_2 adalah ciphertext yang diamati oleh penyerang, dan m_1, m_2adalah data dasar yang disandikan (tidak diketahui oleh penyerang) yang berhubungan dengan pesan berlapis, seperti dibahas di atas. (Setiap blok panjangnya L byte.) Perhatikan itu

$$m_2 = F_k^{-1}(c_2) \oplus c_1, \qquad (3.15)$$

dimana k adalah kuncinya (yang tentunya tidak diketahui oleh penyerang) yang digunakan oleh pihak yang jujur. Blok kedua m_2 diakhiri dengan

$$\underbrace{0\text{x}b \cdots 0\text{x}b}_{b \text{ times}},$$

dimana kita biarkan 0xb menunjukkan representasi 1-byte dari bilangan bulat b. Properti utama yang digunakan dalam serangan ini adalah bahwa perubahan tertentu pada ciphertext menghasilkan perubahan yang dapat diprediksi pada data yang disandikan setelah dekripsi mode CBC. Secara khusus, misalkan $\mathrm{c'}_1$ identik dengan c_1 kecuali untuk modifikasi pada byte terakhir. Pertimbangkan dekripsi ciphertext IV, c'_1, c_2 yang dimodifikasi. Ini akan menghasilkan data yang dikodekan $\mathrm{m'}_1, \mathrm{m'}_2$ dengan $\mathrm{m'}_2 = \mathrm{F}_K^{-1}(\mathrm{c}_2) \oplus \mathrm{c'}_1$. Dibandingkan dengan Persamaan (3.15) kita melihat bahwa $\mathrm{m'}_2$ akan identik dengan m_2 kecuali modifikasi pada byte terakhir. (Nilai $\mathrm{m'}_1$ tidak dapat diprediksi, tetapi hal ini tidak akan berdampak buruk pada serangan.) Demikian pula, jika $\mathrm{c'}_1$sama dengan c_1 ecuali perubahan byte ke-i, maka dekripsi $\mathrm{IV}, \mathrm{c'}_1, c_2$ akan menghasilkan $\mathrm{m'}_1\mathrm{m'}_2$ dimana $\mathrm{m'}_2$ sama dengan m2 kecuali ada perubahan pada byte ke-i. Secara umum, jika $\mathrm{c'}_1 = \mathrm{c}_1 \oplus \Delta$ untuk string apa pun Δ, maka

dekripsi $\mathrm{IV}, \mathrm{c'}_1, c_2$menghasilkan $\mathrm{m'}_1\mathrm{m'}_2$ di mana $\mathrm{m'}_2 = \mathrm{m}_2 \oplus \Delta$. Hasilnya adalah penyerang dapat melakukan kontrol signifikan terhadap blok terakhir data yang dikodekan.

Sebagai pemanasan, mari kita lihat bagaimana musuh dapat memanfaatkan ini untuk mempelajari b, jumlah padding. (Ini mengungkapkan panjang pesan asli.) Ingatlah bahwa setelah dekripsi, penerima melihat nilai b dari byte terakhir dari blok kedua dari data yang dikodekan, dan kemudian memverifikasi bahwa b byte terakhir semuanya memiliki nilai yang sama. Penyerang memulai dengan memodifikasi byte pertama c_1 dan mengirimkan ciphertext IV, c'_1, c_2 yang dihasilkan ke penerima. Jika dekripsi gagal (yaitu, penerima mengembalikan kesalahan) maka penerima harus memeriksa semua L byte dari m'_2, dan oleh karena itu $b = L$! Jika tidak, penyerang mengetahui bahwa $b < L$, dan kemudian dapat mengulangi proses tersebut dengan byte kedua, dan seterusnya. Byte termodifikasi paling kiri yang gagal dekripsi menunjukkan dengan tepat byte paling kiri yang diperiksa oleh penerima, dan dengan demikian mengungkapkan dengan tepat b.

Dengan diketahuinya b, penyerang dapat melanjutkan mempelajari byte pesan satu per satu. Kita mengilustrasikan ide untuk byte terakhir dari pesan, yang kita nyatakan dengan B. Penyerang mengetahui bahwa m_2 berakhiran $0xB0xb \cdots 0xb$ (dengan $0xb$ diulang sebanyak b kali) dan ingin mempelajari B. Definisikan

$$\Delta_i \stackrel{\text{def}}{=} 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}i\ \overbrace{0\text{x}(b+1) \cdots 0\text{x}(b+1)}^{b \text{ times}}$$
$$\oplus\ 0\text{x}00 \cdots 0\text{x}00\ 0\text{x}00\ \overbrace{0\text{x}b \cdots 0\text{x}b}^{b \text{ times}}$$

untuk $0 \leq \mathrm{i} < 2^8$; yaitu, $\mathrm{b} + 1$ bye terakhir dari Δ_i berisi bilangan bulat i (dalam heksadesimal) diikuti dengan nilai $(\mathrm{b} + 1) \oplus \mathrm{b}$ (dalam heksadesimal) yang diulang sebanyak b kali. Jika penyerang mengirimkan ciphertext $\mathrm{IV}, c_1 \oplus \Delta_i, c_2$ ke penerima, kemudian, setelah dekripsi mode CBC, data yang dikodekan akan sama dengan $0\text{x}(\mathrm{B} \oplus i)0\text{x}(\mathrm{b} + 1) \cdots 0\text{x}(\mathrm{b} + 1)$ (dengan $0\text{x}\,(\mathrm{b} + 1)$ diulang b kali). Dekripsi akan gagal kecuali $0\text{x}(\mathrm{B} \oplus \mathrm{i}) = 0\text{x}(\mathrm{b} + 1)$. Penyerang hanya perlu mencoba paling banyak 2^8 nilai $\Delta_0, \ldots, \Delta_{2^8-1}$ hingga dekripsi berhasil untuk beberapa Δi, dan pada titik ini dapat disimpulkan bahwa $\mathrm{B} = 0\text{x}(\mathrm{b} + 1) \oplus i$. Kami memberikan penjelasan lengkap tentang cara memperluas serangan ini sehingga dapat mempelajari keseluruhan teks biasa dan bukan hanya blok terakhir sebagai latihan.

Serangan padding-oracle pada CAPTCHA.

CAPTCHA adalah gambar terdistorsi, katakanlah, sebuah kata bahasa Inggris yang mudah dibaca manusia, tetapi sulit diproses oleh komputer. CAPTCHA digunakan untuk memastikan bahwa pengguna manusia dan bukan perangkat lunak otomatis berinteraksi dengan halaman web. CAPTCHA digunakan, misalnya, oleh layanan email online untuk memastikan bahwa manusia membuka akun; ini penting untuk mencegah pelaku spam membuka ribuan akun secara otomatis dan menggunakannya untuk mengirim spam.

Salah satu cara CAPTCHA dapat dikonfigurasi adalah sebagai layanan terpisah yang dijalankan pada server independen. Untuk melihat cara kerjanya, kami menandai server web dengan S_W, server CAPTCHA dengan S_C, dan pengguna $\mathcal{U}$ dengan $\mathcal{U}$. Ketika pengguna memuat halaman web yang dilayani oleh S_W, peristiwa berikut terjadi: S_W mengenkripsi kata bahasa Inggris acak w menggunakan kunci k yang awalnya dibagikan antara S_W dan S_C, dan mengirimkan ciphertext yang dihasilkan ke pengguna (bersama dengan halaman web). $\mathcal{U}$ meneruskan ciphertext ke S_C, yang mendekripsinya, memperoleh $\mathcal{w}$, dan membuat gambar $\mathcal{w}$ yang terdistorsi ke $\mathcal{U}$. Akhirnya, $\mathcal{U}$ mengirimkan kata $\mathcal{w}$ kembali ke S_W untuk verifikasi. Pengamatan yang menarik di sini adalah bahwa S_C mendekripsi ciphertext apa pun yang diterimanya $\mathcal{U}$ dan akan mengeluarkan pesan kesalahan "bad padding" jika dekripsi gagal seperti yang dijelaskan sebelumnya. Hal ini memberikan $\mathcal{U}$ peluang untuk melakukan serangan padding-oracle seperti dijelaskan di atas, dan dengan demikian menyelesaikan CAPTCHA (yaitu, untuk menentukan $\mathcal{w}$) secara otomatis tanpa keterlibatan manusia, sehingga menjadikan CAPTCHA tidak efektif. Meskipun tindakan ad hoc dapat digunakan (misalnya, meminta C mengembalikan gambar acak dan bukan kesalahan dekripsi), yang sebenarnya diperlukan adalah menggunakan skema enkripsi yang aman dengan CCA.

BAB 4
KODE OTENTIKASI PESAN

4.1 INTEGRITAS PESAN

Kerahasiaan vs. Integritas

Salah satu tujuan paling mendasar dari kriptografi adalah untuk memungkinkan pihak-pihak berkomunikasi melalui saluran komunikasi terbuka dengan cara yang aman. Tapi apa yang dimaksud dengan "komunikasi aman"? Pada Bab 3 kita telah menunjukkan bahwa komunikasi rahasia dapat diperoleh melalui saluran terbuka. Artinya, kami menunjukkan bagaimana enkripsi dapat digunakan untuk mencegah penyadap (atau mungkin musuh yang lebih aktif) mengetahui apa pun tentang isi pesan yang dikirim melalui saluran komunikasi yang tidak terlindungi. Namun, tidak semua masalah keamanan terkait dengan kerahasiaan. Dalam banyak kasus, jaminan integritas pesan (atau otentikasi pesan) sama pentingnya atau lebih penting dalam arti bahwa masing-masing pihak harus dapat mengidentifikasi kapan pesan yang diterimanya dikirim oleh pihak yang mengaku mengirimkannya, dan tidak diubah. sedang transit. Kita melihat dua contoh kanonik.

Misalnya saja kasus seorang pengguna yang berkomunikasi dengan banknya melalui Internet. Ketika bank menerima permintaan untuk mentransfer Rp. 10.000.000 dari rekening pengguna ke rekening beberapa pengguna X, lainnya bank harus mempertimbangkan hal berikut:

1. Apakah permintaan tersebut asli? Artinya, apakah pengguna yang dimaksud benar-benar mengeluarkan permintaan ini, atau apakah permintaan tersebut dikeluarkan oleh musuh (mungkin X sendiri) yang meniru identitas pengguna yang sah?
2. Dengan asumsi permintaan transfer dikeluarkan oleh pengguna yang sah, apakah rincian permintaan yang diterima persis seperti yang dimaksudkan oleh pengguna yang sah? Atau apakah, misalnya, jumlah transfer diubah?

Perhatikan bahwa teknik koreksi kesalahan standar tidak cukup untuk permasalahan kedua. Kode koreksi kesalahan hanya dimaksudkan untuk mendeteksi dan memulihkan kesalahan "acak" yang hanya memengaruhi sebagian kecil transmisi, namun tidak melakukan apa pun untuk melindungi dari musuh jahat yang dapat memilih dengan tepat di mana akan menyebabkan sejumlah kesalahan.

Skenario kedua di mana kebutuhan akan integritas pesan muncul dalam praktiknya adalah terkait dengan cookie web. Protokol HTTP yang digunakan untuk lalu lintas web tidak memiliki kewarganegaraan, sehingga ketika klien dan server berkomunikasi dalam beberapa sesi (misalnya, ketika pengguna [klien] berbelanja di situs web [server] pedagang), keadaan apa pun yang dihasilkan sebagai bagian dari sesi tersebut (misalnya, isi keranjang belanja pengguna) sering ditempatkan dalam "cookie" yang disimpan oleh klien dan dikirim dari klien ke server sebagai bagian dari setiap pesan yang dikirimkan klien. Asumsikan bahwa cookie yang disimpan oleh beberapa pengguna mencakup item dalam keranjang belanja pengguna bersama dengan harga untuk setiap item, seperti yang mungkin dilakukan jika pedagang menawarkan harga yang berbeda kepada pengguna yang berbeda (mencerminkan, misalnya,

diskon dan promosi, atau harga khusus pengguna). penetapan harga). Di sini seharusnya tidak mungkin bagi pengguna untuk memodifikasi cookie yang disimpannya untuk mengubah harga barang di keranjangnya. Oleh karena itu, pedagang memerlukan teknik untuk memastikan integritas cookie yang disimpannya di pengguna. Perlu diperhatikan bahwa konten cookie (yaitu item dan harganya) bukanlah rahasia dan, pada kenyataannya, harus diketahui oleh pengguna. Masalahnya di sini adalah murni integritas.

Secara umum, seseorang tidak dapat mengasumsikan integritas komunikasi tanpa mengambil tindakan khusus untuk menjaminnya. Memang benar, setiap pesanan pembelian online, operasi perbankan online, email, atau pesan SMS yang tidak dilindungi, secara umum, tidak dapat dipercaya berasal dari sumber yang diklaim dan tidak diubah dalam perjalanan. Sayangnya, masyarakat pada umumnya percaya sehingga informasi seperti ID penelepon atau alamat pengirim email dianggap sebagai "bukti asal" dalam banyak kasus, meskipun informasi tersebut relatif mudah dipalsukan. Hal ini membuka pintu bagi serangan yang berpotensi merusak.

Dalam bab ini kami akan menunjukkan bagaimana mencapai integritas pesan dengan menggunakan teknik kriptografi untuk mencegah gangguan yang tidak terdeteksi pada pesan yang dikirim melalui saluran komunikasi terbuka. Perhatikan bahwa kita tidak dapat berharap untuk mencegah gangguan pesan secara keseluruhan, karena hal ini hanya dapat dipertahankan pada tingkat fisik. Sebaliknya, yang kami jamin adalah bahwa gangguan apa pun akan terdeteksi oleh pihak yang jujur.

Enkripsi vs. Otentikasi Pesan

Sebagaimana tujuan kerahasiaan dan integritas pesan berbeda, demikian pula teknik dan alat untuk mencapainya. Sayangnya, kerahasiaan dan integritas sering kali membingungkan dan saling terkait, jadi mari kita perjelas: enkripsi tidak (secara umum) memberikan integritas apa pun, dan enkripsi tidak boleh digunakan dengan tujuan mencapai otentikasi pesan kecuali jika dirancang secara khusus dengan tujuan itu dalam pikiran (sesuatu yang akan kita bahas kembali di Bagian 4.5).

Orang mungkin salah mengira bahwa enkripsi memecahkan masalah otentikasi pesan. (Faktanya, ini adalah kesalahan umum.) Hal ini disebabkan oleh alasan yang tidak jelas dan salah bahwa karena ciphertext sepenuhnya menyembunyikan isi pesan, musuh tidak mungkin mengubah pesan terenkripsi dengan cara apa pun yang berarti. Meskipun memiliki daya tarik intuitif, alasan ini sepenuhnya salah. Kami mengilustrasikan hal ini dengan menunjukkan bahwa semua skema enkripsi yang kami lihat sejauh ini tidak memberikan integritas pesan.

Enkripsi menggunakan stream cipher. Pertimbangkan skema enkripsi sederhana di mana Enck(m) menghitung ciphertext $c := G(k) \oplus m$, di mana G adalah generator pseudorandom. Teks tersandi dalam hal ini sangat mudah untuk dimanipulasi: membalik bit apa pun dalam teks tersandi c menghasilkan bit yang sama dibalik dalam pesan yang dipulihkan setelah dekripsi. Jadi, dengan ciphertext c yang mengenkripsi pesan (mungkin tidak diketahui) m, dimungkinkan untuk menghasilkan ciphertext c′ yang dimodifikasi sedemikian rupa sehingga $m' := Dec_k(c')$ sama dengan m tetapi dengan satu (atau lebih) dari potongannya terbalik. Serangan sederhana ini dapat menimbulkan konsekuensi yang parah.

Sebagai contoh, pertimbangkan kasus pengguna yang mengenkripsi sejumlah dolar yang ingin dia transfer dari rekening banknya, yang jumlah tersebut direpresentasikan dalam biner. Membalikkan bit paling tidak signifikan mempunyai efek mengubah jumlah ini hanya sebesar Rp. 1, namun membalik bit paling tidak signifikan ke-11 akan mengubah jumlahnya lebih dari $Rp.1.000$! (Menariknya, musuh dalam contoh ini tidak serta merta mengetahui apakah ia menaikkan atau menurunkan jumlah awal, yaitu apakah ia membalik angka 0 menjadi 1 atau sebaliknya. Namun jika musuh memiliki sebagian pengetahuan tentang jumlah tersebut—katakanlah, yang awalnya kurang dari Rp. 1.000— maka modifikasi yang dilakukan dapat mempunyai dampak yang dapat diprediksi.) Kami menekankan bahwa serangan ini tidak bertentangan dengan kerahasiaan skema enkripsi (dalam pengertian Definisi 3.8). Faktanya, serangan yang sama juga terjadi pada skema enkripsi one-time pad, yang menunjukkan bahwa kerahasiaan sempurna saja tidak cukup untuk menjamin tingkat integritas pesan yang paling dasar.

Enkripsi menggunakan cipher blok. Serangan yang dijelaskan di atas memanfaatkan fakta bahwa membalik satu bit dalam teks tersandi membuat teks biasa yang mendasarinya tidak berubah kecuali untuk bit terkait (yang juga dibalik). Serangan yang sama berlaku untuk skema enkripsi mode OFB dan CTR, yang juga mengenkripsi pesan dengan meng-XOR-nya dengan aliran pseudorandom (walaupun aliran tersebut berubah setiap kali pesan dienkripsi). Oleh karena itu, kami melihat bahwa penggunaan enkripsi aman CPA saja tidak cukup untuk mencegah gangguan pesan.

Kita mungkin berharap bahwa menyerang enkripsi mode ECB atau CBC akan lebih sulit karena dekripsi dalam kasus ini melibatkan pembalikan permutasi pseudorandom (kuat) F, dan kita berharap bahwa $f_k^{-1}(x)$ dan $f_k^{-1}(x')$ akan sama sekali tidak berkorelasi meskipun x dan x' hanya berbeda satu bit. (Tentu saja, mode ECB bahkan tidak menjamin gagasan paling mendasar tentang kerahasiaan, tapi hal itu tidak menjadi masalah dalam diskusi ini.) Namun demikian, modifikasi bit tunggal pada ciphertext masih menyebabkan sebagian perubahan yang dapat diprediksi pada plaintext. Misalnya, ketika menggunakan mode ECB, membalik sedikit blok ke-i dari ciphertext hanya mempengaruhi blok ke-i dari plaintext—semua blok lainnya tetap tidak berubah. Meskipun efek pada blok ke-i dari teks biasa mungkin sulit untuk diprediksi, mengubah satu blok tersebut (sambil membiarkan blok lainnya tidak berubah) mungkin merupakan serangan berbahaya. Selain itu, urutan blok teks biasa dapat diubah (tanpa mengacaukan blok mana pun) hanya dengan mengubah urutan blok teks tersandi yang sesuai, dan pesan dapat dipotong hanya dengan menghilangkan blok teks tersandi.

Demikian pula, ketika menggunakan mode CBC, membalik bit ke-j dari IV hanya akan mengubah bit ke-j dari blok pesan pertama m_1 (karena $m_i := F_k^{-1}(c_1) \oplus IV'$, di mana IV' adalah IV yang dimodifikasi); semua blok teks biasa selain yang pertama tetap tidak berubah (karena blok teks biasa ke-i dihitung sebagai $m_i := f_k^{-1}(c_i) \oplus c_i - 1$, dan blok ci dan ci−1 belum dimodifikasi). Oleh karena itu, blok pertama dari pesan terenkripsi CBC dapat diubah secara sewenang-wenang. Hal ini menjadi perhatian serius dalam praktiknya karena blok pertama sering kali berisi informasi header yang penting. Terakhir, perhatikan bahwa semua skema enkripsi yang telah kita lihat sejauh ini memiliki properti bahwa setiap ciphertext

(mungkin memenuhi batasan panjang tertentu) berhubungan dengan beberapa pesan yang valid. Jadi sangatlah mudah bagi musuh untuk "memalsukan" sebuah pesan atas nama salah satu pihak yang berkomunikasi—dengan mengirimkan ciphertext sembarangan—bahkan jika musuh tidak tahu apa pesan yang mendasarinya. Seperti yang akan kita lihat ketika kita secara resmi mendefinisikan enkripsi yang diautentikasi di Bagian 4.5, serangan semacam ini pun harus dikesampingkan.

4.2 KODE OTENTIKASI PESAN

Kita telah melihat bahwa, secara umum, enkripsi tidak menyelesaikan masalah integritas pesan. Sebaliknya, diperlukan mekanisme tambahan yang memungkinkan pihak-pihak yang berkomunikasi mengetahui apakah suatu pesan telah dirusak atau tidak. Alat yang tepat untuk tugas ini adalah kode otentikasi pesan (MAC).

Tujuan dari kode otentikasi pesan adalah untuk mencegah musuh memodifikasi pesan yang dikirim oleh satu pihak ke pihak lain, atau memasukkan pesan baru, tanpa penerima mendeteksi bahwa pesan tersebut tidak berasal dari pihak yang dituju. Seperti dalam kasus enkripsi, hal ini hanya mungkin terjadi jika pihak-pihak yang berkomunikasi berbagi suatu rahasia yang tidak diketahui oleh musuh (jika tidak, tidak ada yang dapat mencegah musuh untuk menyamar sebagai pihak yang mengirim pesan). Di sini, kami akan terus mempertimbangkan pengaturan kunci privat di mana para pihak berbagi kunci rahasia yang sama.

Sintaks Kode Otentikasi Pesan

Sebelum secara formal mendefinisikan keamanan kode otentikasi pesan, pertama-tama kita mendefinisikan apa itu MAC dan bagaimana penggunaannya. Dua pengguna yang ingin berkomunikasi dengan cara yang diautentikasi memulai dengan membuat dan berbagi kunci rahasia k sebelum komunikasi mereka. Ketika satu pihak ingin mengirim pesan m ke pihak lain, ia menghitung tag MAC (atau hanya sebuah tag) t berdasarkan pesan dan kunci bersama, dan mengirimkan pesan m dan tag t ke pihak lain. Tag dihitung menggunakan algoritma pembuatan tag yang dilambangkan dengan Mac; jadi, mengulangi apa yang telah kita katakan, pengirim pesan m menghitung t → Mack(m) dan mengirimkan (m, t) ke penerima. Setelah menerima (m, t), pihak kedua memverifikasi apakah t adalah tag yang valid pada pesan m (sehubungan dengan kunci bersama) atau tidak. Hal ini dilakukan dengan menjalankan algoritma verifikasi Vrfy yang mengambil input kunci bersama serta pesan m dan tag t, dan menunjukkan apakah tag yang diberikan valid. Secara formal:

DEFINISI 4.1 Kode otentikasi pesan (atau MAC) terdiri dari tiga algoritma waktu polinomial probabilistik (Gen, Mac, Vrfy) sedemikian rupa sehingga:

1. Algoritme pembangkitan kunci yang diambil Gen sebagai masukan parameter keamanan
1. 1n dan mengeluarkan kunci k dengan $|k| \geq n$.

2. Algoritme pembuatan tag yang Mac ambil sebagai masukan sebuah kunci k dan sebuah pesan $m \in \{0,1\}$ dan mengeluarkan sebuah tag t. Karena algoritma ini mungkin diacak, kita menuliskannya sebagai $t \leftarrow Mac_k(m)$.
3. Algoritma verifikasi deterministik Vrfy mengambil masukan berupa kunci k, pesan m, dan tag t. Outputnya sedikit b, dengan $b = 1$ berarti valid dan $b = 0$ berarti tidak valid. Kami menulis ini sebagai $b := Vrfy_k(m,t)$.

Diperlukan bahwa untuk setiap n, setiap kunci k dikeluarkan oleh Gen(1^n), dan setiap $m \in \{0,1\}^*$, menyatakan bahwa $Vrfy_k(m, Mac_k(m)) = 1$.
Jika terdapat fungsi l sehingga untuk setiap k keluaran oleh Gen(1^n), algoritma Mac_k hanya didefinisikan untuk pesan $m \in \{0,1\}^{\ell(n)}$, maka kita menyebut skema tersebut sebagai MAC dengan panjang tetap untuk pesan dengan panjang $\ell(n)$.

Seperti halnya enkripsi kunci pribadi, Gen(1^n) hampir selalu memilih kunci seragam $k \in \{0,1\}^n$, dan kami menghilangkan Gen dalam kasus tersebut.

Verifikasi kanonik. Untuk kode autentikasi pesan deterministik (yaitu, jika Mac merupakan algoritme deterministik), cara kanonik untuk melakukan verifikasi adalah dengan menghitung ulang tag dan memeriksa kesetaraan. Dengan kata lain, $Vrfy_k(m,t)$ terlebih dahulu menghitung $\tilde{t} = Mac_k(m)$ dan kemudian mengeluarkan 1 jika dan hanya jika $\tilde{t} = t$. Bahkan untuk MAC deterministik, akan berguna untuk mendefinisikan algoritma Vrfy terpisah untuk secara eksplisit membedakan semantik dari mengautentikasi pesan vs. memverifikasi keasliannya.

Keamanan Kode Otentikasi Pesan

Kami sekarang mendefinisikan gagasan default keamanan untuk kode otentikasi pesan. Gagasan intuitif di balik definisi ini adalah bahwa tidak ada musuh yang efisien yang mampu menghasilkan tag yang valid pada pesan "baru" apa pun yang sebelumnya tidak dikirim (dan diautentikasi) oleh salah satu pihak yang berkomunikasi.

Seperti halnya definisi keamanan lainnya, untuk memformalkan gagasan ini kita harus mendefinisikan kekuatan musuh serta apa yang dianggap sebagai "terobosan". Seperti biasa, kami hanya mempertimbangkan musuh dengan waktu polinomial yang probabilistik2 sehingga pertanyaan sebenarnya adalah bagaimana kami memodelkan interaksi musuh dengan pihak yang berkomunikasi. Dalam pengaturan otentikasi pesan, musuh yang mengamati komunikasi antara pihak-pihak yang jujur mungkin dapat melihat semua pesan yang dikirim oleh pihak-pihak tersebut beserta tag MAC-nya yang sesuai. Musuh juga mungkin dapat mempengaruhi isi pesan-pesan ini, baik secara langsung maupun tidak langsung (jika, misalnya, tindakan eksternal dari musuh mempengaruhi pesan-pesan yang dikirimkan oleh para pihak). Hal ini berlaku, misalnya, pada contoh cookie web sebelumnya, di mana tindakan pengguna memengaruhi konten cookie yang disimpan di komputernya.

Untuk memodelkan hal di atas secara formal, kami mengizinkan musuh meminta tag MAC untuk pesan apa pun pilihannya. Secara formal, kami memberikan musuh akses ke oracle $Mac_k(\cdot)$; musuh dapat berulang kali mengirimkan pesan apa pun m pilihannya ke oracle ini, dan sebagai balasannya diberikan tag $t \leftarrow Mac_k(m)$. (Untuk MAC dengan panjang tetap, hanya pesan dengan panjang yang benar yang dapat dikirimkan.)

Kami akan menganggapnya sebagai "perusakan" skema jika musuh mampu mengeluarkan pesan m apa pun bersama dengan tag t sedemikian rupa sehingga: (1) t adalah tag yang valid pada pesan m (i.e. . , $Vrfy_k(m,t) = 1$), dan (2) musuh sebelumnya belum pernah meminta tag MAC pada pesan m (yaitu, dari oraclenya). Syarat pertama berarti jika pihak lawan mengirimkan (m,t) kepada salah satu pihak yang jujur, maka pihak tersebut akan salah mengira bahwa m berasal dari pihak yang sah karena $Vrfy_k(m,t) = 1$. kondisi kedua diperlukan karena selalu ada kemungkinan bagi musuh untuk hanya menyalin pesan dan tag MAC yang sebelumnya dikirim oleh salah satu pihak yang sah (dan, tentu saja, ini akan dianggap sah). Serangan replay seperti itu tidak dianggap sebagai "perusakan" kode otentikasi pesan. Hal ini tidak berarti bahwa serangan ulangan bukan merupakan masalah keamanan; memang benar, dan kami akan membahas lebih banyak tentangnya di bawah.

MAC yang memenuhi tingkat keamanan yang ditentukan di atas dikatakan tidak dapat dipalsukan secara eksistensial dalam serangan pesan pilihan adaptif. "Ketidakpastian eksistensial" mengacu pada fakta bahwa musuh tidak boleh memalsukan tag yang valid pada pesan apa pun, dan "serangan pesan pilihan adaptif" mengacu pada fakta bahwa musuh dapat memperoleh tag MAC pada pesan sewenang-wenang yang dipilih secara adaptif. selama serangannya.

Menuju definisi formal, pertimbangkan eksperimen berikut untuk kode otentikasi pesan Π = (Gen, Mac, Vrfy), musuh $\mathcal{A}$ dan nilai n untuk parameter keamanan:
Eksperimen otentikasi pesan $Mac - forge_{\mathcal{A}\Pi}(n)$:

1. Kunci k dihasilkan dengan menjalankan Gen (1^n).
2. Musuh $\mathcal{A}$ diberikan input 1^n dan akses oracle ke $Mac_k(\cdot)$. Musuh akhirnya mengeluarkan (m,t). Biarkan Q menunjukkan himpunan semua pertanyaan yang ditanyakan $\mathcal{A}$ kepada oraclenya.
3. berhasil jika dan hanya jika (1) $Vrfy_k(m,t) = 1$ dan (2) $m \notin Q$ Type equation here. . Dalam hal ini keluaran percobaan didefinisikan sebagai 1.

MAC aman jika tidak ada musuh yang efisien yang dapat berhasil dalam eksperimen di atas dengan probabilitas yang tidak dapat diabaikan:

DEFINISI 4.2 A Kode otentikasi pesan Π = (Gen, Mac, Vrfy) secara eksistensial tidak dapat dipalsukan di bawah serangan pesan pilihan adaptif, atau hanya aman, jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi yang dapat diabaikan sehingga :

$$\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

Apakah definisinya terlalu kuat? Definisi di atas cukup kuat dalam dua hal. Pertama, musuh diperbolehkan meminta tag MAC untuk pesan apa pun pilihannya. Kedua, musuh dianggap telah "merusak" skema jika ia dapat mengeluarkan tag yang valid pada pesan yang sebelumnya tidak diautentikasi. Ada yang mungkin keberatan karena kedua komponen definisi ini tidak realistis dan terlalu kuat: dalam penggunaan MAC di "dunia nyata", pihak yang jujur hanya akan mengautentikasi pesan yang "bermakna" (di mana musuh mungkin hanya

memiliki kendali terbatas), dan demikian pula, hal ini hanya dapat dianggap sebagai pelanggaran keamanan jika musuh dapat memalsukan tag yang valid pada pesan yang "bermakna". Mengapa tidak menyesuaikan definisi untuk menangkap hal ini?

Hal krusialnya adalah bahwa pesan yang bermakna sepenuhnya bergantung pada penerapannya. Meskipun beberapa aplikasi MAC hanya dapat mengautentikasi pesan teks berbahasa Inggris, aplikasi lain dapat mengautentikasi file spreadsheet, entri database lain, dan data mentah lainnya. Protokol juga dapat dirancang di mana segala sesuatu akan diautentikasi—bahkan, protokol tertentu untuk otentikasi entitas melakukan hal ini. Dengan membuat definisi keamanan untuk MAC sekuat mungkin, kami memastikan bahwa MAC yang aman dapat diterapkan secara luas untuk berbagai tujuan, tanpa harus mengkhawatirkan kompatibilitas MAC dengan semantik aplikasi.

Putar ulang serangan. Kami menekankan bahwa definisi di atas, dan kode otentikasi pesan itu sendiri, tidak memberikan perlindungan terhadap serangan replay dimana pesan yang dikirim sebelumnya (dan tag MAC-nya) diputar ulang ke pihak yang jujur. Namun demikian, serangan ulangan merupakan masalah serius! Pertimbangkan lagi skenario di mana pengguna (misalnya, Alice) mengirimkan permintaan ke banknya untuk mentransfer $1.000 dari rekeningnya ke pengguna lain (misalnya, Bob). Dengan melakukan hal ini, Alice dapat menghitung tag MAC dan menambahkannya ke permintaan sehingga bank mengetahui bahwa permintaan tersebut asli. Jika MAC aman, Bob tidak akan dapat mencegat permintaan tersebut dan mengubah jumlahnya menjadi $10.000 karena hal ini akan melibatkan pemalsuan tag yang valid pada pesan yang sebelumnya tidak diautentikasi. Namun, tidak ada yang menghalangi Bob untuk menyadap pesan Alice dan memutarnya ulang sepuluh kali ke bank. Jika bank menerima setiap pesan ini, dampak akhirnya adalah $10.000 akan ditransfer ke rekening Bob, bukan $1.000 yang diinginkan. Terlepas dari ancaman nyata yang ditimbulkan oleh serangan replay, MAC dengan sendirinya tidak dapat melindungi terhadap serangan tersebut karena definisi MAC (Definisi 4.1) tidak memasukkan gagasan status apa pun ke dalam algoritma verifikasi (dan setiap kali pasangan yang valid (m, t) disajikan ke algoritma verifikasi, maka akan selalu menghasilkan 1). Sebaliknya, perlindungan terhadap serangan replay—jika perlindungan tersebut diperlukan—harus ditangani oleh aplikasi tingkat tinggi. Alasan mengapa definisi MAC disusun seperti ini adalah, sekali lagi, karena kami tidak mau mengasumsikan semantik apa pun mengenai aplikasi yang menggunakan MAC; khususnya, keputusan apakah pesan yang diputar ulang harus dianggap "sah" atau tidak mungkin bergantung pada aplikasi.

Dua teknik umum untuk mencegah serangan replay adalah dengan menggunakan nomor urut (juga dikenal sebagai counter) atau stempel waktu. Pendekatan pertama, yang dijelaskan (dalam konteks yang lebih umum) di Bagian 4.5.3, mengharuskan pengguna yang berkomunikasi untuk mempertahankan keadaan (yang disinkronkan), dan dapat menjadi masalah ketika pengguna berkomunikasi melalui saluran lossy di mana pesan kadang-kadang hilang (meskipun hal ini masalah dapat dikurangi). Dalam pendekatan kedua, dengan menggunakan stempel waktu, pengirim menambahkan waktu saat ini T (katakanlah, ke milidetik terdekat) ke pesan sebelum mengautentikasi, dan mengirimkan T bersama dengan

pesan dan tag yang dihasilkan t. Ketika penerima memperoleh T, m, t, ia memverifikasi bahwa t adalah tag yang valid pada $T \parallel m$ dan bahwa T berada dalam kemiringan jam yang dapat diterima dari waktu saat ini T' di penerima. Metode ini juga memiliki kelemahan tertentu, termasuk kebutuhan pengirim dan penerima untuk menjaga sinkronisasi jam yang erat, dan kemungkinan bahwa serangan replay masih dapat terjadi jika dilakukan dengan cukup cepat (khususnya, dalam jangka waktu yang dapat diterima).

MAC yang kuat. Sebagaimana didefinisikan, MAC yang aman memastikan bahwa musuh tidak dapat menghasilkan tag yang valid pada pesan baru yang belum pernah diautentikasi sebelumnya. Namun tidak menutup kemungkinan bahwa penyerang dapat membuat tag baru pada pesan yang telah diautentikasi sebelumnya. Artinya, MAC menjamin bahwa jika penyerang mempelajari tag $t_1, \ldots$ pada pesan $m_1, \ldots$, maka ia tidak akan dapat memalsukan tag t yang valid pada pesan apa pun $m \notin \{m_1, \ldots\}$. Namun, ada kemungkinan bagi musuh untuk "memalsukan" tag valid yang berbeda $t'_1 \neq t_1$ pada pesan m1. Secara umum, perilaku permusuhan seperti ini tidak perlu dikhawatirkan. Namun demikian, dalam beberapa situasi, ada baiknya untuk mempertimbangkan definisi keamanan yang lebih kuat untuk MAC di mana perilaku seperti itu dikesampingkan.

Secara formal, kami mempertimbangkan eksperimen Mac-sforge yang dimodifikasi, yang didefinisikan dengan cara yang persis sama seperti Mac-forge, kecuali sekarang himpunan Q berisi pasangan kueri Oracle dan respons terkaitnya. (Artinya, $(m, t) \in Q$ jika $\mathcal{A}$menanyakan Mack(m) dan menerima tag t sebagai jawabannya.) Musuh $\mathcal{A}$ berhasil (dan eksperimen Mac-sforge bernilai 1) jika dan hanya jika $\mathcal{A}$menghasilkan (m, t) sehingga $Vrfy_k(m, t) = 1 \; dan \; (m, t) \notin Q$

DEFINISI 4.3 A Kode otentikasi pesan Π = (Gen, Mac, Vrfy) sangat aman, atau MAC yang kuat, jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi yang dapat diabaikan sehingga:

$$\Pr[\text{Mac-sforge}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

Tidak sulit untuk melihat bahwa jika MAC yang aman menggunakan verifikasi kanonik maka MAC tersebut juga sangat aman. Ini penting karena semua MAC di dunia nyata menggunakan verifikasi kanonik. Bukti berikut ini kami tinggalkan sebagai latihan.

PROPOSISI 4.4 Misalkan Π = (Gen, Mac, Vrfy) adalah MAC aman yang menggunakan verifikasi kanonik. Maka Π adalah MAC yang kuat.

Pertanyaan Verifikasi

Definisi 4.2 dan 4.3 memberi musuh akses ke oracle MAC, yang sesuai dengan musuh dunia nyata yang dapat mempengaruhi pengirim yang jujur untuk menghasilkan tag untuk beberapa pesan m. Kita juga dapat mempertimbangkan musuh yang berinteraksi dengan penerima yang jujur, mengirimkan m', t' ke penerima untuk mengetahui apakah $Vrfy_k(m', t') = 1$. Musuh

tersebut dapat ditangkap secara formal dengan cara alami dengan memberikan musuh dalam definisi di atas juga memiliki akses ke oracle verifikasi.

Definisi yang menggabungkan oracle verifikasi dengan cara ini, mungkin, adalah cara yang "benar" untuk mendefinisikan keamanan kode otentikasi pesan. Namun ternyata, untuk MAC yang menggunakan verifikasi kanonik tidak ada bedanya: MAC apa pun yang memenuhi Definisi 4.2 juga memenuhi varian definisi di mana permintaan verifikasi diperbolehkan. Demikian pula, MAC kuat apa pun secara otomatis juga tetap aman bahkan dalam pengaturan yang memungkinkan permintaan verifikasi. (Hal ini, pada kenyataannya, berfungsi sebagai salah satu motivasi untuk mendefinisikan keamanan yang kuat untuk MAC.) Namun, untuk MAC yang tidak menggunakan verifikasi kanonik, mengizinkan kueri verifikasi dapat membuat perbedaan.

Potensi serangan waktu. Salah satu masalah yang tidak diatasi di atas adalah kemungkinan melakukan serangan waktu pada verifikasi MAC. Di sini, kami mempertimbangkan musuh yang dapat mengirimkan pasangan pesan/tag ke penerima—sehingga menggunakan penerima sebagai oracle verifikasi—dan mempelajari tidak hanya apakah penerima menerima atau menolak, tetapi juga waktu yang diperlukan penerima untuk membuat keputusan ini. . Kami menunjukkan bahwa jika serangan seperti itu mungkin terjadi maka penerapan verifikasi MAC secara alami akan menyebabkan kerentanan yang mudah dieksploitasi. (Perhatikan bahwa dalam definisi keamanan kriptografi yang biasa kita gunakan, penyerang hanya mempelajari output dari oracle yang dapat diaksesnya, tetapi tidak mempelajari yang lain. Serangan yang kami uraikan di sini, yang merupakan contoh serangan saluran samping, menunjukkan bahwa serangan nyata tertentu -serangan dunia tidak ditangkap oleh definisi biasa.)

Konkritnya, asumsikan MAC menggunakan verifikasi kanonik. Untuk memverifikasi tag t pada pesan m, penerima menghitung $t' := Mac_k(m)$ dan kemudian membandingkan t' dengan t, menghasilkan 1 jika dan hanya jika t' dan t sama. Asumsikan perbandingan ini diimplementasikan menggunakan rutinitas standar (seperti strcmp di C) yang membandingkan t dan t' satu byte pada satu waktu, dan menolak segera setelah byte pertama yang tidak sama ditemukan. Pengamatannya adalah, ketika diimplementasikan dengan cara ini, waktu untuk menolak berbeda-beda bergantung pada posisi byte pertama yang tidak sama.

Informasi yang tampaknya tidak penting ini dapat digunakan untuk memalsukan tag pada pesan apa pun yang diinginkan m. Ide dasarnya adalah ini: katakanlah penyerang mengetahui i byte pertama dari tag yang benar untuk m. (Pada awalnya, $i = 0$.) Penyerang akan mempelajari byte berikutnya dari tag yang benar dengan mengirimkan $(m, t_0), \ldots, (m, t_{255})$ ke penerima, dengan tj adalah string dengan i byte pertama yang disetel dengan benar, byte $(i + 1)$st sama dengan j (dalam heksadesimal), dan byte sisanya disetel ke 0x00. Semua tag ini kemungkinan besar akan ditolak (jika tidak, penyerang tetap berhasil); namun, untuk salah satu tag ini, byte pertama $(i + 1)$ akan cocok dengan tag yang benar dan penolakan akan memakan waktu sedikit lebih lama dibandingkan byte lainnya. Jika t_j adalah tag yang menyebabkan penolakan paling lama, penyerang mengetahui bahwa $(i + 1)$ byte pertama dari tag yang benar adalah j. Dengan cara ini, penyerang mempelajari setiap byte dari

tag yang benar menggunakan paling banyak 256 kueri ke oracle verifikasi. Untuk tag 16 byte, serangan ini hanya memerlukan 4096 kueri dalam kasus terburuk.

Orang mungkin bertanya-tanya apakah serangan ini realistis, karena memerlukan akses ke oracle verifikasi serta kemampuan untuk mengukur perbedaan waktu yang dibutuhkan untuk membandingkan i vs. $i + 1$ byte. Faktanya, serangan seperti itu telah dilakukan terhadap sistem nyata! Sebagai salah satu contoh saja, MAC digunakan untuk memverifikasi pembaruan kode di Xbox 360, dan penerapan verifikasi MAC yang digunakan memiliki perbedaan 2,2 milidetik antara waktu penolakan. Penyerang dapat mengeksploitasi ini dan memuat game bajakan ke perangkat keras.

Berdasarkan penjelasan di atas, kami menyimpulkan bahwa verifikasi MAC harus menggunakan perbandingan string yang tidak tergantung waktu yang selalu membandingkan semua byte.

4.3 MEMBUAT KODE OTENTIKASI PESAN AMAN

MAC dengan Panjang Tetap

Fungsi pseudorandom adalah alat alami untuk membuat kode otentikasi pesan yang aman. Secara intuitif, jika tag MAC t diperoleh dengan menerapkan fungsi pseudorandom pada pesan m, maka memalsukan tag pada pesan yang sebelumnya tidak diautentikasi mengharuskan musuh menebak dengan benar nilai fungsi pseudorandom pada titik input "baru". Peluang menebak nilai fungsi acak pada titik baru adalah 2^{-n} (jika panjang keluaran fungsi tersebut adalah n). Kemungkinan menebak nilai seperti itu untuk fungsi pseudorandom bisa sangat besar.

Ide di atas, yang ditunjukkan pada Konstruksi 4.5, berfungsi untuk membuat MAC dengan panjang tetap yang aman untuk pesan dengan panjang n (karena fungsi pseudorandom kami secara default memiliki panjang blok n-bit). Ini berguna, namun tidak mencapai tujuan kami. Di Bagian 4.3, kami menunjukkan cara memperluas ini untuk menangani pesan dengan panjang yang berubah-ubah. Kami mengeksplorasi konstruksi MAC_s yang lebih efisien untuk pesan dengan panjang sembarang di Bagian 4.4 dan 4.5.

CONSTRUCTION 4.5

Let F be a pseudorandom function. Define a fixed-length MAC for messages of length n as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^n$, output the tag $t := F_k(m)$. (If $|m| \neq |k|$ then output nothing.)
- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^n$, and a tag $t \in \{0,1\}^n$, output 1 if and only if $t \stackrel{?}{=} F_k(m)$. (If $|m| \neq |k|$, then output 0.)

MAC dengan panjang tetap dari fungsi pseudorandom apa pun.

TEOREMA 4.6 Jika F adalah fungsi pseudorandom, maka Konstruksi 4.5 adalah MAC dengan panjang tetap yang aman untuk pesan dengan panjang n.

BUKTI Seperti penggunaan fungsi pseudorandom sebelumnya, pembuktian ini mengikuti paradigma pertama menganalisis keamanan skema menggunakan fungsi yang benar-benar acak, dan kemudian mempertimbangkan hasil penggantian fungsi yang benar-benar acak dengan fungsi pseudorandom.

Misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik. Pertimbangkan kode otentikasi pesan $\widetilde{\Pi} = (\widetilde{\text{Gen}}, \widetilde{\text{Mac}}, \widetilde{\text{Vrfy}})$ yang sama dengan Π = (Mac, Vrfy) dalam Konstruksi 4.5 kecuali bahwa fungsi yang benar-benar acak ƒ digunakan sebagai pengganti fungsi pseudorandom F_k. Artinya, $\widetilde{\text{Gen}}(1^n)$ bekerja dengan memilih fungsi seragam $f \in Func_n$, dan Mac menghitung tag seperti yang dilakukan Mac, kecuali bahwa ƒ digunakan sebagai pengganti F_k. Hal itu segera terjadi

$$\Pr[\text{Mac-forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] \leq 2^{-n} \qquad (4.1)$$

karena untuk pesan apa pun $m \notin Q$, nilai $t = f(m)$ terdistribusi secara merata di $\{0,1\}^n$ dari sudut pandang musuh $\mathcal{A}$.

Selanjutnya kita tunjukkan bahwa ada fungsi yang dapat diabaikan sehingga

$$\left|\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] - \Pr[\text{Mac-forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1]\right| \leq \text{negl}(n); \qquad (4.2)$$

dikombinasikan dengan Persamaan (4.1), hal ini menunjukkan bahwa

$$\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] \leq 2^{-n} + \text{negl}(n),$$

membuktikan teorema tersebut.

Untuk membuktikan Persamaan (4.2), kita membuat pembeda waktu polinomial D yang memberikan akses oracle ke beberapa fungsi, dan yang tujuannya adalah untuk menentukan apakah fungsi ini pseudorandom (yaitu, sama dengan F_k untuk seragam $k \in \{0,1\}^n$) atau acak (i.e., yaitu, sama dengan ƒ untuk seragam $f \in Func_n$). Untuk melakukan hal ini, D mengemulasi eksperimen autentikasi pesan dan mengamati apakah berhasil mengeluarkan tag yang valid pada pesan “baru”. Jika demikian, D menebak bahwa oraclenya adalah fungsi pseudorandom; jika tidak, D menebak bahwa ramalannya adalah fungsi acak. Secara terperinci:

Pembeda D:
D diberikan input 1n dan akses ke Oracle $\mathcal{O} : \{0,1\}^n \longrightarrow \{0,1\}^n$, dan berfungsi sebagai berikut:

1. Jalankan $\mathcal{A}$ (1^n). Setiap kali $\mathcal{A}$ menanyakan oracle MAC-nya pada pesan m (i.e., yaitu, setiap kali $\mathcal{A}$ meminta tag pada pesan m), jawab pertanyaan ini dengan cara berikut: Queri $\mathcal{O}$ dengan m dan dapatkan respons t; kembalikan t ke $\mathcal{A}$.
2. Ketika A mengeluarkan (m, t) di akhir eksekusinya, lakukan:
   (a) Queri $\mathcal{O}$ dengan m dan dapatkan respons $\hat{t}$.
   (b) Jika (1) $\hat{t}$ = t dan (2) $\mathcal{A}$ tidak pernah menanyakan oracle MAC-nya pada m, maka keluaran 1; jika tidak, keluaran 0.
   Jelas bahwa D berjalan dalam waktu polinomial.

Perhatikan bahwa jika oracle D adalah fungsi pseudorandom, maka tampilan $\mathcal{A}$ ketika dijalankan sebagai sub-rutin oleh D didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen $Mac - forge_{\mathcal{A},\Pi}(n)$. Selanjutnya, D menghasilkan 1 tepat ketika Mac-$forge_{\mathcal{A},\prod}(n) = 1$. Oleh karena itu

$$\Pr\left[D^{F_k(\cdot)}(1^n) = 1\right] = \Pr\left[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1\right],$$

dimana $k \in \{0,1\}^n$ dipilih secara seragam di atas. Jika oracle D adalah fungsi acak, maka tampilan $\mathcal{A}$ ketika dijalankan sebagai sub-rutin oleh D didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam percobaan $Mac - forge_{\mathcal{A},\widetilde{\prod}}(n)$, dan sekali lagi D menghasilkan 1 persis ketika $Mac - forge_{\mathcal{A},\widetilde{\prod}}(n) = 1$. Jadi,

$$\Pr\left[D^{f(\cdot)}(1^n) = 1\right] = \Pr\left[\text{Mac-forge}_{\mathcal{A},\widetilde{\Pi}}(n) = 1\right],$$

dimana $f \in Func_n$ dipilih secara seragam.
Karena F adalah fungsi pseudorandom dan D berjalan dalam waktu polinomial, terdapat fungsi yang dapat diabaikan sehingga

$$\left|\Pr\left[D^{F_k(\cdot)}(1^n) = 1\right] - \Pr\left[D^{f(\cdot)}(1^n) = 1\right]\right| \leq \text{negl}(n).$$

Ini menyiratkan Persamaan (4.2), yang melengkapi pembuktian teorema.

Ekstensi Domain untuk MAC

Konstruksi 4.5 penting karena menunjukkan paradigma umum untuk membangun kode otentikasi pesan aman dari fungsi pseudorandom. Sayangnya, konstruksi ini hanya mampu menangani pesan dengan panjang tetap dan juga agak pendek. Keterbatasan ini tidak dapat diterima di sebagian besar aplikasi. Di sini kita tunjukkan bagaimana MAC umum, yang menangani pesan dengan panjang sembarang, dapat dibuat dari MAC dengan panjang tetap untuk pesan dengan panjang n. Konstruksi yang kami tunjukkan tidak terlalu efisien dan kecil kemungkinannya untuk digunakan dalam praktik. Memang benar, konstruksi MAC aman yang jauh lebih efisien telah diketahui, seperti yang kita bahas di Bagian 4.4 dan 4.5 Kami menyertakan konstruksi ini karena kesederhanaan dan keumumannya.

Misalkan $\prod' = (Mac', Vrfy')$ adalah MAC dengan panjang tetap yang aman untuk pesan dengan panjang n. Sebelum menyajikan konstruksi MAC untuk pesan dengan panjang sembarang berdasarkan $\prod'$ kami mengesampingkan beberapa ide sederhana dan menjelaskan beberapa serangan kanonik yang harus dicegah. Di bawah ini, kami mengurai pesan m untuk diautentikasi sebagai rangkaian blok $m_1, \ldots, m_d$; perhatikan bahwa, karena tujuan kita adalah menangani pesan dengan panjang yang berubah-ubah, d dapat bervariasi dari satu pesan ke pesan lainnya.

1. Ide pertama yang wajar adalah dengan mengautentikasi setiap blok secara terpisah, yaitu menghitung $t_i := Mac'_k(m_i)$ untuk semua i, dan menghasilkan $\langle t_1, \ldots, t_d \rangle$ sebagai tagnya. Hal ini mencegah musuh mengirimkan blok yang sebelumnya tidak diautentikasi tanpa terdeteksi. Namun, hal ini tidak mencegah serangan pengurutan ulang blok di mana penyerang mengacak urutan blok dalam pesan yang diautentikasi. Khususnya, jika $\langle t_1, t_2 \rangle$ merupakan tag valid pada pesan m_1, m_2 (dengan $m_1 \neq m_2$), maka $\langle t_2, t_1 \rangle$ merupakan tag valid pada pesan (berbeda) m_2, m_1 (sesuatu yang tidak diperbolehkan menurut Definisi 4.2).
2. Kita dapat mencegah serangan sebelumnya dengan mengautentikasi indeks blok beserta setiap blok. Artinya, kita sekarang menghitung $t_i = Mac'_k(i \parallel m_i)$ untuk semua i, dan menghasilkan $\langle t_1, \ldots, t_d \rangle$ sebagai tagnya. (Perhatikan bahwa panjang blok $|m_i|$ harus diubah.) Hal ini tidak mencegah serangan pemotongan dimana penyerang hanya menjatuhkan blok dari akhir pesan (dan juga menjatuhkan blok tag yang sesuai).
3. Serangan pemotongan dapat digagalkan dengan mengautentikasi tambahan panjang pesan beserta setiap bloknya. (Mengautentikasi panjang pesan sebagai blok terpisah tidak berfungsi. Tahukah Anda alasannya?) Artinya, hitung $ti = Mac'_k(\ell \parallel i \parallel m_i)$ untuk semua i, di mana ℓ menunjukkan panjang pesan dalam bit. (Sekali lagi, panjang blok $|m_i|$ perlu dikurangi.) Skema ini rentan terhadap serangan "campur-dan-cocok" di mana musuh menggabungkan blok-blok dari pesan yang berbeda. Misalnya, jika musuh mendapatkan tag $\langle t_1, \ldots, t_d \rangle$ dan $\langle t'_1, \ldots, t'_d \rangle$ pada pesan $m = m_1, \ldots, m_d$ dan $m' = m'_1, \ldots, m'_d$, masing-masing, dapat menampilkan tag yang valid $\langle t_1, t'_2, t_3, t'_4, \ldots \rangle$ pada pesan $m_1, m'_2, m_3, m'_4, \ldots$.

Kita dapat mencegah serangan terakhir ini dengan juga menyertakan "pengidentifikasi pesan" acak bersama dengan setiap blok yang mencegah penggabungan blok dari pesan yang berbeda. Hal ini membawa kita ke Konstruksi 4.7.

CONSTRUCTION 4.7

Let $\Pi' = (\mathsf{Mac}', \mathsf{Vrfy}')$ be a fixed-length MAC for messages of length n. Define a MAC as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message $m \in \{0,1\}^*$ of (nonzero) length $\ell < 2^{n/4}$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Choose a uniform identifier $r \in \{0,1\}^{n/4}$.

  For $i = 1, \ldots, d$, compute $t_i \leftarrow \mathsf{Mac}'_k(r \| \ell \| i \| m_i)$, where i, ℓ are encoded as strings of length $n/4$.† Output the tag $t := \langle r, t_1, \ldots, t_d \rangle$.

- Vrfy: on input a key $k \in \{0,1\}^n$, a message $m \in \{0,1\}^*$ of length $\ell < 2^{n/4}$, and a tag $t = \langle r, t_1, \ldots, t_{d'} \rangle$, parse m into d blocks $m_1, \ldots, m_d$, each of length $n/4$. (The final block is padded with 0s if necessary.) Output 1 if and only if $d' = d$ and $\mathsf{Vrfy}'_k(r \| \ell \| i \| m_i, t_i) = 1$ for $1 \le i \le d$.

† Note that i and ℓ can be encoded using $n/4$ bits because $i, \ell < 2^{n/4}$.

MAC untuk pesan dengan panjang sembarang dari MAC dengan panjang tetap. (Secara teknis, skema ini hanya menangani pesan yang panjangnya kurang dari $2^{n/4}$.

Secara asimtotik, karena ini adalah batasan eksponensial, pihak yang jujur tidak akan mengautentikasi pesan yang panjangnya dan musuh dengan waktu polinomial mana pun tidak dapat mengirimkan pesan sepanjang itu ke oracle MAC-nya. Dalam praktiknya, ketika nilai konkrit n ditetapkan, kita harus memastikan bahwa batasan ini dapat diterima.)

TEOREMA 4.8 Jika Π' adalah MAC dengan panjang tetap yang aman untuk pesan dengan panjang n, maka Konstruksi 4.7 adalah MAC aman (untuk pesan dengan panjang sembarang).

BUKTI Intuisinya adalah selama $\prod$' aman, musuh tidak dapat memasukkan blok baru dengan tag yang valid. Selain itu, informasi tambahan yang disertakan dalam setiap blok mencegah berbagai serangan (menjatuhkan blok, menyusun ulang blok, dll.) yang digambarkan sebelumnya. Kami akan membuktikan keamanan dengan menunjukkan bahwa serangan ini adalah satu-satunya serangan yang mungkin terjadi.

Misalkan $\prod$ adalah MAC yang diberikan oleh Konstruksi 4.7, dan misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik. Kami menunjukkan bahwa $\Pr[Mac - forge_{\mathcal{A},\prod}(n) = 1]$ dapat diabaikan. Pertama-tama kita perkenalkan beberapa notasi yang akan digunakan dalam pembuktian. Biarkan Ulang menunjukkan kejadian dimana pengidentifikasi acak yang sama muncul di dua tag yang dikembalikan oleh oracle MAC dalam percobaan $Mac - forge_{\mathcal{A},\prod}(n)$. Membiarkan $(m, t = \langle r, t_1, \ldots \rangle)$ menyatakan keluaran akhir dari $\mathcal{A}$, dimana $m = m_1, \ldots$ memiliki panjang ℓ, kita membiarkan NewBlock menjadi peristiwa dimana setidaknya salah satu blok $r \parallel \ell \parallel i \parallel m_i$ belum pernah diautentikasi sebelumnya oleh Mac' dalam rangka menjawab pertanyaan Mac. (Perhatikan bahwa, dengan konstruksi $\prod$, mudah untuk mengetahui dengan tepat blok mana yang diautentikasi oleh Mac'_k saat menghitung

$Mac_k(m).$) Secara informal, NewBlock adalah kejadian di mana $\mathcal{A}$ mencoba mengeluarkan tag yang valid pada blok yang tadinya tidak pernah diautentikasi oleh MAC $\prod'$ dengan panjang tetap yang mendasarinya.
Kita punya

$$\begin{aligned}\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1] &= \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1 \wedge \text{Repeat}] \\ &\quad + \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\text{Repeat}} \wedge \text{NewBlock}] \\ &\quad + \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\text{Repeat}} \wedge \overline{\text{NewBlock}}] \\ &\leq \Pr[\text{Repeat}] \qquad (4.3) \\ &\quad + \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1 \wedge \text{NewBlock}] \\ &\quad + \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\text{Repeat}} \wedge \overline{\text{NewBlock}}].\end{aligned}$$

Kita tunjukkan bahwa dua suku pertama pada Persamaan (4.3) dapat diabaikan, dan suku terakhirnya adalah 0. Hal ini berarti Pr$[Mac - forge_{\mathcal{A},\prod}(n) = 1]$ dapat diabaikan, sesuai keinginan.

KLAIM 4.9 Pr[Repeat] dapat diabaikan.

BUKTI Misalkan $q(n)$ adalah banyaknya query oracle MAC yang dibuat oleh $\mathcal{A}$. Untuk menjawab query oracle ke-i, oracle memilih ri secara seragam dari himpunan berukuran $2^{n/4}$. Peluang kejadian Ulang sama persis dengan peluang $r_i = r_j$ untuk beberapa $i \neq j$. Menerapkan "batas ulang tahun" (Lemma A.15), kita mendapatkan Pr$[Repeat] \leq \frac{q(n)^2}{2^{n/4}}$. Karena $\mathcal{A}$ hanya membuat banyak kueri secara polinomial, nilai ini dapat diabaikan.

Selanjutnya kita perhatikan suku terakhir di sisi kanan Persamaan (4.3). Kami berpendapat bahwa jika $Mac - forge_{\mathcal{A},\prod}(n) = 1$, tetapi Pengulangan tidak terjadi, maka NewBlock pasti terjadi. Artinya, $Mac - forge_{\mathcal{A},\prod}(n) = 1 \wedge$ Repeat menyiratkan NewBlock, dan seterusnya

$$\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n)=1 \wedge \overline{\text{Repeat}} \wedge \overline{\text{NewBlock}}] = 0.$$

Dalam beberapa hal, inilah inti pembuktiannya.

Sekali lagi misalkan $q = q(n)$ menunjukkan jumlah kueri oracle MAC yang dibuat oleh $\mathcal{A}$, dan misalkan r_i menunjukkan pengidentifikasi acak yang digunakan untuk menjawab kueri oracle ke-i dari $\mathcal{A}$. Jika Pengulangan tidak terjadi maka nilainya $r_1, \ldots, r_q$ berbeda. Misalkan $(m, t = \langle r, t_1, \ldots \rangle)$ adalah output dari $\mathcal{A}$, dengan $m = m_1, \ldots$. Jika $r \notin \{r_1, \ldots, r_q\}$, maka NewBlock jelas muncul. Jika tidak, maka $r = r_j$ untuk beberapa *j* unik, dan balok $r \| \ell \| 1 \| m_1, \ldots$ maka tidak mungkin diautentikasi selama menjawab pertanyaan Mac apa pun selain dari pertanyaan tersebut. Misalkan $m^{(j)}$ adalah pesan yang digunakan oleh $\mathcal{A}$ untuk kueri oracle ke-j, dan misalkan ℓ_j adalah panjangnya. Ada dua kasus yang perlu dipertimbangkan:

Kasus 1: $\ell \neq \ell_j$. Blok yang diautentikasi saat menjawab kueri Mac ke-j semuanya memiliki $\ell_j \neq \ell$ di posisi kedua. Jadi $r \parallel \ell \parallel 1 \parallel m_1$, khususnya, tidak pernah diautentikasi saat menjawab kueri Mac ke-j, dan NewBlock muncul.
Kasus 2: $\ell = \ell_j$. Jika $Mac - forge_{\mathcal{A},\prod}(n) = 1$, maka kita harus mempunyai $m \neq m^{(j)}$. Misalkan $m^{(j)} = m_1^{(j)}, \ldots$. Karena m dan $m^{(j)}$ memiliki panjang yang sama, maka harus ada paling sedikit satu indeks i yang $m_i \neq m_i^{(j)}$. Blok $r \parallel \ell \parallel i \parallel m_i$ kemudian tidak pernah diautentikasi saat menjawab permintaan Mac ke-j. (Karena i termasuk dalam posisi ketiga blok, maka blok $r \parallel \ell \parallel i \parallel m_i$ hanya mungkin terautentikasi jika $r \parallel \ell \parallel i \parallel m_i = r_j \parallel \ell_j \parallel i \parallel m_i^{(j)}$, namun hal ini tidak benar karena $m_i \neq m_i^{(j)}$.)
Untuk melengkapi pembuktian teorema tersebut, kita ikat suku kedua di ruas kanan Persamaan (4.3):4

KLAIM 4.10 Pr[Mac-forge$_{\mathcal{A},\prod}(n) = 1 \wedge$ NewBlock] dapat diabaikan.
Klaim tersebut bergantung pada keamanan $\prod'$. Kami membuat musuh PPT $\mathcal{A}'$ yang menyerang MAC dengan panjang tetap $\prod'$ dan berhasil menghasilkan pemalsuan yang valid pada pesan yang sebelumnya tidak diautentikasi dengan probabilitas Pr[Mac-forge$_{\mathcal{A}',\prod'}(n) = 1] \geq$ Pr[Mac-forge$_{\mathcal{A},\prod}(n) = 1 \wedge$ Blok Baru]. (4.4)

Keamanan $\prod'$ berarti ruas kiri dapat diabaikan, sehingga membuktikan klaim tersebut. Konstruksi $\mathcal{A}'$ sudah jelas dan kami akan menjelaskannya secara singkat. $\mathcal{A}'$ menjalankan $\mathcal{A}$ sebagai sub-rutin, dan menjawab permintaan $\mathcal{A}$ untuk tag pada m dengan memilih $r \leftarrow \{0,1\}^{n/4}$ itu sendiri, menguraikan m dengan tepat, dan membuat kueri yang diperlukan ke MAC oracle Mac-nya sendiri Mac'$_k(\cdot)$. Ketika $\mathcal{A}$ menghasilkan keluaran $(m, t = \langle r, t_1, \ldots \rangle)$, maka $\mathcal{A}'$ memeriksa apakah NewBlock muncul (ini mudah dilakukan karena $\mathcal{A}'$ dapat melacak semua kueri yang dibuatnya ke oraclenya sendiri). Jika demikian, maka $\mathcal{A}'$ menemukan blok pertama $r \parallel \ell \parallel i \parallel m_i$ yang sebelumnya tidak pernah diautentikasi oleh Mac' dan menghasilkan keluaran $(r \parallel \ell \parallel i \parallel m_i, t_i)$. (Jika tidak, $\mathcal{A}'$ tidak menghasilkan apa pun.)

Tampilan $\mathcal{A}$ ketika dijalankan sebagai sub-rutin oleh $\mathcal{A}'$ didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen Mac-forge$_{\mathcal{A},\prod}(n)$, sehingga peluang kejadian Mac-forge$_{\mathcal{A},\prod}(n) = 1$ dan Blok Baru tidak berubah. Jika NewBlock terjadi maka $\mathcal{A}'$ mengeluarkan blok $r \parallel \ell \parallel i \parallel m_i$ yang sebelumnya tidak pernah diautentikasi oleh oracle MAC-nya sendiri; jika Mac-forge$_{\mathcal{A},\prod}(n) = 1$ maka tag pada setiap blok adalah valid (sehubungan dengan $\prod'$), dan khususnya hal ini berlaku untuk keluaran blok oleh $\mathcal{A}'$. Ini berarti bahwa setiap kali Mac-forge$_{\mathcal{A},\prod}(n) = 1$ dan NewBlock muncul, kita memiliki Mac-forge$A_{\mathcal{A}',\prod'}(n) = 1$, yang membuktikan Persamaan (4.4).

4.4 CBC-MAC

Teorema 4.6 dan 4.8 menunjukkan bahwa adalah mungkin untuk membuat kode otentikasi pesan aman untuk pesan dengan panjang sembarang dari fungsi pseudorandom yang mengambil input dengan panjang tetap n. Hal ini menunjukkan, pada prinsipnya, bahwa

MAC yang aman dapat dibangun dari cipher blok. Sayangnya, konstruksi yang dihasilkan sangat tidak efisien: untuk menghitung tag pada pesan dengan panjang dn, cipher blok dievaluasi sebanyak 4d kali; panjang tag lebih dari 4dn bit. Untungnya, tersedia konstruksi yang jauh lebih efisien. Kami mengeksplorasi salah satu konstruksi di sini yang hanya mengandalkan cipher blok, dan konstruksi lainnya di Bagian 5.3. yang menggunakan primitif kriptografi tambahan.

Konstruksi Dasar

CBC-MAC adalah kode otentikasi pesan standar yang digunakan secara luas dalam praktik. Versi A dasar CBC-MAC, aman saat mengautentikasi pesan dengan panjang tetap, diberikan sebagai Konstruksi 4.11. (Lihat juga Gambar 4.1.) Kami mengingatkan bahwa skema dasar ini tidak aman dalam kasus umum ketika pesan dengan panjang berbeda dapat diautentikasi; simak pembahasan lebih lanjut dibawah ini.

CONSTRUCTION 4.11

Let F be a pseudorandom function, and fix a length function $\ell > 0$. The basic CBC-MAC construction is as follows:

- Mac: on input a key $k \in \{0,1\}^n$ and a message m of length $\ell(n) \cdot n$, do the following (we set $\ell = \ell(n)$ in what follows):
  1. Parse m as $m = m_1, \ldots, m_\ell$ where each m_i is of length n.
  2. Set $t_0 := 0^n$. Then, for $i = 1$ to ℓ:
     Set $t_i := F_k(t_{i-1} \oplus m_i)$.

  Output t_ℓ as the tag.
- Vrfy: on input a key $k \in \{0,1\}^n$, a message m, and a tag t, do: If m is not of length $\ell(n) \cdot n$ then output 0. Otherwise, output 1 if and only if $t \stackrel{?}{=} \mathsf{Mac}_k(m)$.

CBC-MAC dasar (untuk pesan dengan panjang tetap).

TEOREMA 4.12 Misalkan ℓ adalah polinomial. Jika F adalah fungsi pseudorandom, maka Konstruksi 4.11 adalah MAC aman untuk pesan dengan panjang $\ell(n) \cdot n$.

BUKTI Teorema 4.12 agak rumit. Pada bagian berikut ini kita akan membuktikan hasil yang lebih umum dari teorema di atas.

Meskipun Konstruksi 4.11 dapat diperluas dengan cara yang jelas untuk menangani pesan yang panjangnya merupakan kelipatan n, konstruksi tersebut hanya aman jika panjang pesan yang diautentikasi ditetapkan dan disepakati sebelumnya oleh pengirim dan penerima.

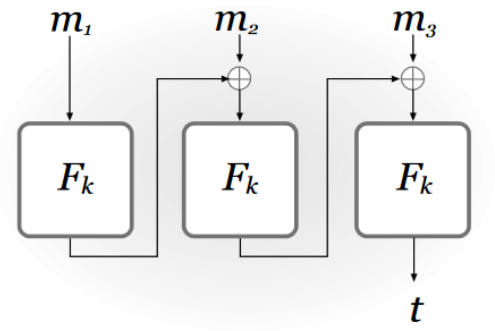
Keuntungan konstruksi ini dibandingkan Konstruksi 4.5, yang juga memberikan MAC dengan panjang tetap, adalah konstruksi ini dapat mengautentikasi pesan yang lebih panjang. Dibandingkan dengan Konstruksi 4.7, CBC-MAC jauh lebih efisien, hanya memerlukan evaluasi blok-cipher d untuk pesan dengan panjang dn, dan dengan tag dengan panjang n saja.

Enkripsi mode CBC-MAC vs. CBC. CBC-MAC mirip dengan mode operasi CBC. Namun ada beberapa perbedaan penting:

1. Enkripsi mode CBC menggunakan IV acak dan ini penting untuk keamanan. Sebaliknya, CBC-MAC tidak menggunakan IV (sebagai alternatif, dapat dilihat menggunakan nilai tetap $IV = 0^n$) dan ini juga penting untuk keamanan. Secara khusus, CBC-MAC yang menggunakan IV acak tidak aman.
2. Dalam enkripsi mode CBC, semua nilai antara t_i (disebut ci dalam kasus enkripsi mode CBC) dikeluarkan oleh algoritma enkripsi sebagai bagian dari ciphertext, sedangkan dalam CBC-MAC hanya blok terakhir yang dikeluarkan sebagai tag. Jika CBC-MAC dimodifikasi untuk menampilkan semua $\{t_i\}$ yang diperoleh selama komputasi, maka CBC-MAC tidak lagi aman.

Contoh-contoh ini menggambarkan fakta bahwa modifikasi konstruksi kriptografi yang tampak tidak berbahaya dapat membuat konstruksi tersebut tidak aman. Seseorang harus selalu menerapkan konstruksi kriptografi persis seperti yang ditentukan dan tidak memperkenalkan variasi apa pun (kecuali variasi itu sendiri dapat dibuktikan aman). Selain itu, penting untuk memahami konstruksi yang digunakan. Dalam banyak kasus, perpustakaan kriptografi menyediakan “fungsi CBC” kepada pemrogram, namun tidak membedakan antara penggunaan fungsi ini untuk enkripsi atau otentikasi pesan.

Amankan CBC-MAC untuk pesan dengan panjang sewenang-wenang. Kami menjelaskan secara singkat dua cara Konstruksi 4.11 dapat dimodifikasi, dengan cara yang terbukti aman, untuk menangani pesan dengan panjang yang berubah-ubah. (Di sini untuk mempermudah kita berasumsi bahwa semua pesan yang diautentikasi memiliki panjang kelipatan n, dan bahwa Vrfy menolak pesan apa pun yang panjangnya bukan kelipatan n. Pada bagian berikut ini kita membahas kasus yang lebih umum di mana pesan dapat memiliki panjang sembarang.)



GAMBAR 4.1: CBC-MAC Dasar (untuk pesan dengan panjang tetap).

1. Tambahkan pesan m dengan panjangnya $|m|$ (dikodekan sebagai string n-bit), lalu hitung CBC-MAC dasar pada hasilnya; lihat Gambar 4.2. Keamanan varian ini mengikuti hasil yang dibuktikan di bagian selanjutnya.
   Perhatikan bahwa menambahkan $|m|$ ke akhir pesan dan kemudian menghitung CBC-MAC dasar tidaklah aman.

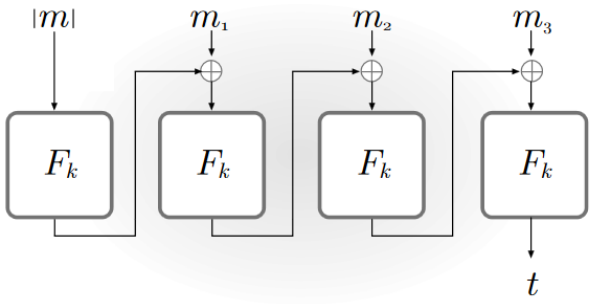
2. Ubah skema sehingga pembangkitan kunci memilih dua kunci independen dan seragam $k_1 \in \{0,1\}^n$ dan $k_2 \in \{0,1\}^n$. Kemudian untuk mengautentikasi pesan m, pertama-tama hitung CBC-MAC dasar m menggunakan k_1 dan biarkan t menjadi hasilnya; keluaran tag $\hat{t} := F_{k2}(t)$.

Opsi kedua memiliki keuntungan karena tidak perlu mengetahui panjang pesan terlebih dahulu (yaitu saat mulai menghitung tag). Namun, ada kelemahannya yaitu menggunakan dua tombol untuk F. Perhatikan bahwa, dengan mengorbankan dua aplikasi tambahan fungsi pseudorandom, dimungkinkan untuk menyimpan satu kunci k dan kemudian memperoleh kunci $k_1 := F_k(1)$ dan $k_2 := F_k(2)$ di awal komputasi. Meskipun demikian, dalam praktiknya, operasi inisialisasi kunci untuk cipher blok dianggap relatif mahal. Oleh karena itu, memerlukan dua kunci yang berbeda—walaupun kunci tersebut diturunkan dengan cepat—kurang diinginkan.

***Bukti Keamanan**

Pada bagian ini kami membuktikan keamanan berbagai varian CBC-MAC. Kita mulai dengan merangkum hasilnya, dan kemudian memberikan rincian buktinya. Sebelum memulai, kami mencatat bahwa pembuktian di bagian ini cukup rumit, dan ditujukan untuk pembaca tingkat lanjut.

Sepanjang bagian ini, perbaiki fungsi berkunci F yang, untuk parameter keamanan n, memetakan kunci n-bit dan masukan n-bit ke keluaran n-bit. Kita mendefinisikan fungsi berkunci CBC yang, untuk parameter keamanan n, memetakan kunci n-bit dan masukan dalam $(\{0,1\}^n)^*$ (yaitu, string yang panjangnya merupakan kelipatan n) ke keluaran n-bit. Fungsi ini didefinisikan sebagai



Gambar 4.2: Varian CBC-MAC aman untuk mengautentikasi pesan dengan panjang sembarang.

$$\mathrm{CBC}_k(x_1,\ldots,x_\ell) \stackrel{\text{def}}{=} F_k\left(F_k\left(\cdots F_k(F_k(x_1)\oplus x_2)\oplus\cdots\right)\oplus x_\ell\right),$$

dimana $|x_1| = \cdots |x_\ell| = |k| = n$. (Kami membiarkan CBC_k tidak terdefinisi pada string kosong.) Perhatikan bahwa CBC adalah CBC-MAC dasar, meskipun di sini kami mempertimbangkan input dengan panjang yang berbeda.

Satu set string $P \subset (\{0,1\}^n)^*$ bebas awalan jika tidak berisi string kosong, dan tidak ada string $X \in P$ yang merupakan awalan dari string lain $X' \in P$. Kami menunjukkan:

TEOREMA 4.13 Jika F adalah fungsi pseudorandom, maka CBC adalah fungsi pseudorandom selama himpunan input yang ditanyakan bebas prefiks. Secara formal, untuk semua pembeda waktu polinomial probabilistik D yang menanyakan oracle mereka pada kumpulan input bebas awalan, terdapat pengabaian fungsi yang dapat diabaikan sehingga

$$\left| \Pr[D^{\mathrm{CBC}_k(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1] \right| \leq \mathrm{negl}(n),$$

di mana k dipilih secara seragam dari $\{0,1\}^n$ dan ƒ dipilih secara seragam dari himpunan pemetaan fungsi $(\{0,1\}^n)^*$ hingga $\{0,1\}^n$ (yaitu, nilai ƒ pada setiap masukan adalah seragam dan tidak tergantung pada nilainya dari ƒ pada semua input lainnya).

Dengan demikian, kita dapat mengkonversi fungsi pseudorandom F untuk input dengan panjang tetap menjadi fungsi pseudorandom CBC untuk input dengan panjang sembarang (tergantung pada batasan input mana yang dapat ditanyakan)! Untuk menggunakan ini untuk otentikasi pesan, kami mengadaptasi gagasan Konstruksi 4.5 sebagai berikut: untuk mengautentikasi pesan m, pertama-tama terapkan beberapa fungsi pengkodean encode untuk mendapatkan string (tidak kosong) encode $(m) \in (\{0,1\}^n)^*$; lalu keluarkan tag CBC_k(encode(m)). Agar hal ini aman (lih. pembuktian Teorema 4.6), pengkodean harus bebas awalan, yaitu, memiliki properti bahwa untuk pesan (legal) yang berbeda m_1, m_2, string encode(m_1) tidak awalan encode(m_2). Ini menyiratkan bahwa untuk setiap kumpulan pesan (legal) $\{m_1, \ldots\}$, kumpulan pesan yang disandikan $\{\text{encode}(m_1), \ldots\}$ bebas awalan.

Kami sekarang memeriksa dua penerapan nyata dari gagasan ini:

Perbaiki ℓ, dan biarkan kumpulan pesan legal menjadi $\{0,1\}^{\ell(n).n}$. Kemudian kita dapat mengambil pengkodean sepele encode$(m) = m$, yang bebas awalan karena satu string tidak dapat menjadi awalan dari string berbeda dengan panjang yang sama. Ini adalah CBC-MAC dasar, dan apa yang telah kami katakan di atas menyiratkan bahwa CBC-MAC dasar aman untuk pesan dengan panjang tetap (lih. Teorema 4.12).

Salah satu cara menangani pesan dengan panjang sembarang (tidak kosong) (secara teknis, pesan dengan panjang kurang dari 2^n) adalah dengan menyandikan string $m \in \{0,1\}^*$ dengan mengawali panjangnya $|m|$ (dikodekan sebagai string n-bit), dan kemudian menambahkan sebagai sebanyak 0 yang diperlukan untuk membuat panjang string yang dihasilkan menjadi kelipatan n. (Ini pada dasarnya adalah apa yang ditunjukkan pada Gambar 4.2.) Pengkodean ini bebas awalan, dan oleh karena itu kami memperoleh MAC yang aman untuk pesan dengan panjang sembarang.

Sisa bagian ini dikhususkan untuk pembuktian Teorema 4.13. Untuk membuktikan teorema tersebut, kami menganalisis CBC ketika CBC “dikunci” dengan fungsi acak g dan

bukan dengan kunci acak k untuk beberapa fungsi pseudorandom yang mendasari F . Artinya, kami menganggap fungsi kunci CBC_g didefinisikan sebagai

$$\mathrm{CBC}_g(x_1, \ldots, x_\ell) \stackrel{\text{def}}{=} g\left(g\left(\cdots g(g(x_1) \oplus x_2) \oplus \cdots\right) \oplus x_\ell\right)$$

dimana, untuk parameter keamanan n, fungsi g memetakan masukan n-bit ke keluaran n-bit, dan $|x_1| = \cdots = |x_\ell| = n$. Perhatikan bahwa CB$C_g$ sebagaimana didefinisikan di sini tidak efisien (karena representasi g memerlukan ruang eksponensial dalam n); namun demikian, ini masih merupakan fungsi penting yang terdefinisi dengan baik.

Kami menunjukkan bahwa jika g dipilih secara seragam dari Func$_n$, maka CBC_g tidak dapat dibedakan dari pemetaan fungsi acak ($\{0,1\}^n$)* ke string n -bit, selama rangkaian input bebas prefiks ditanyakan. Lebih tepatnya:

KLAIM 4.14 Perbaiki $n \geq 1$. Untuk semua pembeda D yang menanyakan oracle mereka pada kumpulan input q yang bebas awalan, dengan input terpanjang berisi ℓ blok, maka berlaku bahwa:

$$\left|\Pr[D^{\mathrm{CBC}_g(\cdot)}(1^n) = 1] - \Pr[D^{f(\cdot)}(1^n) = 1]\right| \leq \frac{q^2\ell^2}{2^n},$$

dimana g dipilih secara seragam dari Funcn, dan f dipilih secara seragam dari himpunan pemetaan fungsi $(\{0,1\}^n)^*$ hingga $\{0,1\}^n$.

(Klaim ini tidak bersyarat, dan tidak memaksakan batasan apa pun pada waktu berjalan D. Jadi kita dapat menganggap D bersifat deterministik.) Hal di atas menyiratkan Teorema 4.13 dengan menggunakan teknik standar yang telah kita lihat. Secara khusus, untuk setiap D yang berjalan dalam waktu polinomial kita harus memiliki $q(n), \ell(n) = \text{poli}(n)$ sehingga $q(n)^2\ell(n)^2 \cdot 2^{-n}$ dapat diabaikan.

BUKTI (Klaim 4.14) Perbaiki beberapa $n \geq 1$. Pembuktian dilakukan dalam dua langkah: Pertama-tama kita mendefinisikan gagasan kelancaran dan membuktikan bahwa CBC mulus; kami kemudian menunjukkan bahwa kehalusan menyiratkan klaim tersebut.

Misalkan $P = \{X_1, \ldots, X_q\}$ merupakan himpunan input q yang bebas awalan, dengan masing-masing X_i berada di $(\{0,1\}^n)^*$ dan string terpanjang di P berisi ℓ blok (yaitu, setiap $X_i \in P$ berisi paling banyak ℓ blok dengan panjang n). Perhatikan bahwa untuk $t_1, \ldots, t_q \in \{0,1\}^n$ menyatakan bahwa Pr $[\forall i : f(X_i) = t_i] = 2^{-nq}$, dengan probabilitas pemilihan fungsi f yang seragam dari pemetaan himpunan fungsi $(\{0,1\}^n)^*$ hingga $\{0,1\}^n$. Kita katakan bahwa CBC adalah (q, ℓ, δ)-halus jika untuk setiap himpunan bebas awalan $P = \{X_1, \ldots, X_q\}$ seperti di atas dan setiap $t_1, \ldots, t_q \in \{0,1\}^n$, menyatakan demikian

$$\Pr\left[\forall i : \mathrm{CBC}_g(X_i) = t_i\right] \geq (1-\delta)\cdot 2^{-nq},$$

dimana probabilitasnya melebihi pilihan seragam $g \in \mathrm{Func}_n$.

Dengan kata lain, CBC dikatakan mulus jika untuk setiap himpunan pasangan input/output tetap $\{(X_i, t_i)\}$, dimana $\{X_i\}$ membentuk himpunan bebas prefiks, peluang $\mathrm{CB}C_g(X_i) = t_i$ untuk semua i adalah δ -mendekati probabilitas bahwa $f(X_i) = t_i$ untuk semua i (dengan g adalah fungsi acak dari $\{0,1\}^n$ hingga $\{0,1\}^n$, dan f adalah fungsi acak dari $(\{0,1\}^n)^*$ hingga $\{0,1\}^n$).

KLAIM 4.15 $\mathrm{CB}C_g$ adalah (q, ℓ, δ)-halus, untuk $\delta = q^2\ell^2 \cdot 2^{-n}$.

BUKTI Untuk sembarang $X \in (\{0,1\}^n)^*$, dengan $X = x_1, \ldots$ dan $x_i \in \{0,1\}^n$, misalkan $C_g(X)$ menyatakan himpunan masukan yang g dievaluasi selama penghitungan $\mathrm{CB}C_g(X)$; yaitu, jika $X \in (\{0,1\}^n)^m$ maka

$$\mathcal{C}_g(X) \stackrel{\text{def}}{=} (x_1,\ \mathrm{CBC}_g(x_1)\oplus x_2,\ \ldots,\ \mathrm{CBC}_g(x_1,\ldots,x_{m-1})\oplus x_m)\,.$$

Untuk $X \in (\{0,1\}^n)^m$ dan $X' \in (\{0,1\}^n)^{m'}$, dengan $C_g(X) = (I_1, \ldots, I_m)$ dan $C_g(X') = (I'_1, \ldots, I'_{m'})$, katakanlah ada tumbukan non-sepele di X jika $I_i = I_j$ untuk beberapa $i \neq j$, dan katakan ada tumbukan non-sepele antara X dan X' jika $I_i = I'_j$ tetapi $(x_1, \ldots, x_i) \neq (x'_1, \ldots, x'_j)$ (dalam kasus terakhir ini i mungkin sama dengan j). Kita katakan ada tumbukan non-trivial di P jika ada tumbukan non-trivial di beberapa $X \in P$ atau antara sepasang string $X, X' \in P$. Misalkan Coll adalah kejadian terjadinya tumbukan nontrivial di P .

Kami membuktikan klaim tersebut dalam dua langkah. Pertama, kita tunjukkan bahwa jika tidak ada tumbukan non-sepele di P , peluang $\mathrm{CB}C_g(X_i) = t_i$ untuk semua i adalah tepat 2^{-nq}. Selanjutnya, kita tunjukkan bahwa peluang terjadinya tumbukan non-trivial di P lebih kecil dari $\delta = q^2\ell^2 \cdot 2^{-n}$.

Pertimbangkan untuk memilih g yang seragam dengan memilih, satu per satu, nilai seragam untuk keluaran g pada masukan yang berbeda. Menentukan ada tidaknya tumbukan non-trivial antara dua string $X, X' \in P$ dapat dilakukan dengan terlebih dahulu memilih nilai $g(I_1)$ dan $g(I'_1)$ (jika $I'_1 = I_1$, nilai-nilai tersebut sama) , lalu pilih nilai untuk $g(I_2)$ dan $g(I'_2)$ (perhatikan bahwa $I_2 = g(I_1) \oplus x_2$ dan $I'_2 = g(I'_1) \oplus x'_2$ didefinisikan setelah $g(I_1), g(I'_1)$ telah diperbaiki), dan terus demikian hingga kita memilih nilai untuk $g(I_{m-1})$ dan $g(I'_{m'-1})$. Secara khusus, nilai $g(I_m), g(I'_{m'})$ tidak perlu dipilih untuk menentukan apakah terdapat tumbukan non-trivial antara X dan X'. Melanjutkan alur pemikiran ini, adalah mungkin untuk menentukan apakah Coll terjadi dengan memilih nilai g pada semua kecuali entri akhir dari masing-masing $C_g(X_1), \ldots, C_g(X_q)$.

Asumsikan Coll belum terjadi setelah menetapkan nilai g pada berbagai input seperti dijelaskan di atas. Pertimbangkan entri terakhir di masing-masing $C_g(X_1), \ldots, C_g(X_q)$. Entri-entri ini semuanya berbeda (ini langsung dari fakta bahwa Coll belum terjadi), dan kami mengklaim bahwa nilai g pada masing-masing titik tersebut belum ditetapkan. Memang benar, satu-satunya cara agar nilai g dapat ditetapkan pada salah satu titik tersebut adalah jika entri akhir I_m dari beberapa $C_g(X)$ sama dengan entri non-final I_j dari beberapa $C_g(X')$. Namun karena Coll belum terjadi, hal ini hanya dapat terjadi jika $X \neq X'$ dan $(x'_1, \ldots, x'_j) = (x_1, \ldots, x_m)$. Namun X akan menjadi awalan dari X', melanggar asumsi bahwa P bebas awalan.

Karena g adalah fungsi acak, arti di atas adalah CB$C_g(X_1)$,..., CB$C_g(X_q)$ seragam dan independen satu sama lain serta semua nilai g lainnya yang telah ditetapkan. (Ini karena CB$C_g(X_i)$ adalah nilai g ketika dievaluasi pada entri akhir $C_g(X_i)$, suatu nilai masukan yang berbeda dari semua masukan lainnya yang g telah ditetapkan.) Jadi, untuk setiap $t_1, \ldots, t_q \in \{0,1\}^n$ kita punya:

$$\Pr\left[\forall i : \mathrm{CBC}_g(X_i) = t_i \mid \overline{\mathrm{Coll}}\right] = 2^{-nq}. \qquad (4.5)$$

Kami selanjutnya menunjukkan bahwa $\overline{Coll}$ terjadi dengan probabilitas tinggi pada batas atas Pr[Coll]. Untuk $X_i, X_j \in P$ yang berbeda, misalkan $Coll_{i,j}$ menyatakan kejadian tumbukan non-trivial di X atau X', atau tumbukan non-trivial antara X dan X'. Kita mempunyai Coll $= \bigvee_{i,j} Coll_{i,j}$ dan ikatan gabungan menghasilkan

$$\Pr[\mathrm{Coll}] \leq \sum_{i,j:\, i<j} \Pr[\mathrm{Coll}_{i,j}] = \binom{q}{2} \cdot \Pr[\mathrm{Coll}_{i,j}] \leq \frac{q^2}{2} \cdot \Pr[\mathrm{Coll}_{i,j}]. \qquad (4.6)$$

Memperbaiki $X = X_i$ dan $X' = X_j$ yang berbeda di P, sekarang kita mengikat $Coll_{i,j}$. Seperti yang akan terlihat jelas dari analisis, probabilitasnya akan maksimal ketika X dan X' sama-sama sepanjang mungkin, dan dengan demikian kita asumsikan panjangnya masing-masing ℓ blok. Misalkan $X = (x_1, .., x_\ell)$ dan $X' = (x'_1, \ldots, x'_\ell)$, dan misalkan t adalah bilangan bulat terbesar sehingga $(x_1, \ldots, x_t) = (x'_1, \ldots, x'_t)$,. (Perhatikan bahwa $t < \ell$ atau $X = X'$.) Kita berasumsi $t > 0$, namun analisis di bawah ini dapat dengan mudah dimodifikasi, memberikan hasil yang sama, jika $t = 0$. Kita lanjutkan dengan membiarkan $I_1, I_2, \ldots$ (resp., $I'_1, I'_2, \ldots$) menunjukkan input ke g selama komputasi CB$C_g(X)$ (resp., CB$C_g(X')$); perhatikan bahwa $(I'_1, \ldots, I'_t) = (I_1, \ldots, I_t)$. Pertimbangkan untuk memilih g dengan memilih nilai seragam untuk keluaran g, satu per satu. Kami melakukan ini dalam $2\ell - 2$ langkah sebagai berikut:

Langkah 1 hingga $t - 1$ (jika $t > 1$): Pada setiap langkah i, pilih nilai seragam untuk $g(I_i)$, sehingga mendefinisikan $I_i + 1$ dan $I'_i + 1$ (yang setara).

Langkah t: Pilih nilai seragam untuk $g(I_t)$, sehingga mendefinisikan $I_t + 1$ dan $I'_t + 1$.

Langkah $t + 1$ ke $\ell - 1$ (jika $t < \ell - 1$): Pilih, secara bergantian, nilai seragam untuk masing-masing $g(I_{t+1}), g(I_{t+2}), \ldots, g(I_{\ell-1})$, sehingga mendefinisikan $I_{t+2}, I_{t+3}, \ldots, I_{\ell}$.
Langkah ℓ ke $2\ell - 2$ (jika $t < \ell - 1$): Pilih, secara bergantian, nilai seragam untuk masing-masing
$g(I'_{t+1}), g(I'_{t+2}), \ldots, g(I'_{\ell-1})$, sehingga mendefinisikan $I'_{t+2}, I'_{t+3}, \ldots, I'_{\ell}$.
Misalkan Coll(k) adalah kejadian terjadinya tumbukan nontrivial pada langkah k. Kemudian

$$\Pr[\text{Coll}_{i,j}] = \Pr\left[\bigvee_k \text{Coll}(k)\right] \leq \Pr[\text{Coll}(1)] + \sum_{k=2}^{2\ell-2} \Pr[\text{Coll}(k) \mid \overline{\text{Coll}(k-1)}], \quad (4.7)$$

menggunakan Proposisi A.9. Untuk $k < t$, kita mengklaim Pr[Coll(k) | $\overline{Coll(k-1)}] = k/2^n$: tentu saja, jika tidak ada tumbukan non-trivial yang terjadi pada langkah $k - 1$, nilai $g(I_k)$ dipilih secara seragam pada langkah k; tumbukan non-sepele hanya terjadi jika $I_{k+1} = g(I_k) \oplus x_{k+1}$ sama dengan salah satu dari $\{I_1, \ldots, I_k\}$ (yang semuanya berbeda, karena Coll$(k-1)$ belum muncul). Dengan alasan serupa, kita mempunyai Pr[Coll(t) $\overline{Coll(t-1)}] \leq 2t/2^n$ (di sini ada dua nilai I_{t+1}, I'_{t+1} yang perlu dipertimbangkan; perhatikan bahwa keduanya tidak bisa sama satu sama lain). Terakhir, berargumentasi seperti sebelumnya, untuk $k > t$ kita mempunyai Pr[Coll(k) $\overline{Coll(k-1)}] = (k+1)/2^n$. Dengan menggunakan Persamaan (4.7), kita mendapatkan

$$\begin{aligned}\Pr[\text{Coll}_{i,j}] &\leq 2^{-n} \cdot \left(\sum_{k=1}^{t-1} k + 2t + \sum_{k=t+1}^{2\ell-2} (k+1)\right) \\ &= 2^{-n} \cdot \sum_{k=2}^{2\ell-1} k \;=\; 2^{-n} \cdot (2\ell+1) \cdot (\ell-1) \;<\; 2\ell^2 \cdot 2^{-n}.\end{aligned}$$

Dari Persamaan (4.6) kita peroleh Pr[Coll] $< q^2\ell^2 \cdot 2^{-n} = \delta$. Akhirnya, dengan menggunakan Persamaan (4.5) kita melihatnya

$$\begin{aligned}\Pr\left[\forall i : \text{CBC}_g(X_i) = t_i\right] &\geq \Pr\left[\forall i : \text{CBC}_g(X_i) = t_i \mid \overline{\text{Coll}}\right] \cdot \Pr[\overline{\text{Coll}}] \\ &= 2^{-nq} \cdot \Pr[\overline{\text{Coll}}] \;\geq (1-\delta) \cdot 2^{-nq},\end{aligned}$$

seperti yang diklaim.

Kami sekarang menunjukkan bahwa kehalusan menyiratkan teorema. Asumsikan tanpa kehilangan keumuman bahwa D selalu membuat q kueri (berbeda), masing-masing berisi paling banyak ℓ blok. D dapat memilih kuerinya secara adaptif (yaitu, bergantung pada jawaban kueri sebelumnya), namun kumpulan kueri $D's$ harus bebas awalan.

Untuk $X_1, \ldots, X_q \in (\{0,1\}^n)^*$ dan $t_1, \ldots, t_q \in \{0,1\}^n$,
baik $\alpha(X_1, \ldots, X_q; t_1, \ldots, t_q)$ menjadi 1 jika dan hanya jika D mengeluarkan 1 saat membuat kueri $X_1, \ldots, X_q$ dan mendapatkan tanggapan $t_1, \ldots, t_q$. (Jika, katakanlah, D tidak

menjadikan kueri $X1$ sebagai kueri pertamanya, maka $\alpha(X_1,\ldots;\ldots) = 0.$) Membiarkan $\vec{X} = (X_1,\ldots,X_q)$ dan $\vec{t} = (t_1,\ldots,t_q)$, kita punya

$$\begin{aligned}\Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n)=1] &= \sum_{\vec{X}\text{ prefix-free; }\vec{t}} \alpha(\vec{X},\vec{t})\cdot\Pr[\forall i:\mathsf{CBC}_g(X_i)=t_i] \\ &\geq \sum_{\vec{X}\text{ prefix-free; }\vec{t}} \alpha(\vec{X},\vec{t})\cdot(1-\delta)\cdot\Pr[\forall i:f(X_i)=t_i] \\ &= (1-\delta)\cdot\Pr[D^{f(\cdot)}(1^n)=1],\end{aligned}$$

dimana, di atas, g dipilih secara seragam dari Func$_n$, dan ƒ dipilih secara seragam dari himpunan pemetaan fungsi $(\{0,1\}^n)^*$ hingga $\{0,1\}^n$. Ini menyiratkan

$$\Pr[D^{f(\cdot)}(1^n)=1]-\Pr[D^{\mathsf{CBC}_g(\cdot)}(1^n)=1]\leq\delta\cdot\Pr[D^{f(\cdot)}(1^n)=1]\leq\delta.$$

Argumen simetris ketika D menghasilkan 0 melengkapi pembuktiannya.

4.5 ENKRIPSI YANG DIAUTENTIKASI

Pada Bab 3, kita mempelajari bagaimana cara memperoleh kerahasiaan dalam pengaturan kunci privat dengan menggunakan enkripsi. Dalam bab ini, kami telah menunjukkan cara memastikan integritas menggunakan kode otentikasi pesan. Tentu saja seseorang ingin mencapai kedua tujuan tersebut secara bersamaan, dan inilah masalah yang kita hadapi sekarang.

Praktik terbaiknya adalah selalu memastikan kerahasiaan dan integritas secara default dalam pengaturan kunci privat. Memang benar, dalam banyak aplikasi yang memerlukan kerahasiaan, ternyata integritas juga penting. Selain itu, kurangnya integritas terkadang dapat menyebabkan pelanggaran kerahasiaan.

Definisi

Kita mulai, seperti biasa, dengan mendefinisikan dengan tepat apa yang ingin kita capai. Pada tingkat abstrak, tujuan kami adalah mewujudkan saluran komunikasi yang “aman secara ideal” yang memberikan kerahasiaan dan integritas. Mencari definisi semacam ini berada di luar cakupan buku ini. Sebaliknya, kami memberikan serangkaian definisi sederhana yang memperlakukan kerahasiaan dan integritas secara terpisah. Definisi-definisi ini dan analisis kami selanjutnya cukup untuk memahami isu-isu utama yang ada. (Namun, kami memperingatkan pembaca bahwa—berbeda dengan enkripsi dan kode autentikasi pesan—bidang ini belum menetapkan terminologi dan definisi standar untuk enkripsi yang diautentikasi.)

Misalkan $\prod$ = (Gen, Enc, Dec) adalah skema enkripsi kunci pribadi. Seperti yang telah disebutkan, kami mendefinisikan keamanan dengan mendefinisikan secara terpisah kerahasiaan dan integritas. Gagasan tentang kerahasiaan yang kami pertimbangkan adalah sesuatu yang telah kami lihat sebelumnya: kami mengharuskan $\prod$ aman terhadap serangan

ciphertext yang dipilih, yaitu aman CCA. (Lihat Bagian 3.7 untuk diskusi dan definisi keamanan CCA.) Kami prihatin dengan serangan teks sandi yang dipilih di sini karena kami secara eksplisit mempertimbangkan musuh aktif yang dapat mengubah data yang dikirim dari satu pihak jujur ke pihak lain. Gagasan kita tentang integritas pada dasarnya adalah gagasan yang tidak dapat diubah secara eksistensial di bawah serangan pesan pilihan yang adaptif. Karena $\prod$ tidak memenuhi sintaks kode otentikasi pesan, kami memperkenalkan definisi khusus untuk kasus ini. Pertimbangkan eksperimen berikut yang ditentukan untuk skema enkripsi kunci pribadi $\prod$ = (Gen, Enc, Des), adversary $\mathcal{A}$, dan nilai n untuk parameter keamanan:
Eksperimen enkripsi yang tidak dapat dipalsukan Enc-Forge$_{\mathcal{A},\prod}(n)$:

1. Jalankan Gen(1^n) untuk mendapatkan kunci k.
2. Musuh $\mathcal{A}$ diberi masukan 1^n dan akses ke oracle enkripsi Enc$_k(\cdot)$. Musuh mengeluarkan ciphertext c.
3. Misalkan $m :=$ Dec$_k(c)$, dan misalkan Q menyatakan himpunan semua pertanyaan yang ditanyakan $\mathcal{A}$ pada oracle enkripsinya. Keluaran percobaan adalah 1 jika dan hanya jika $(1)\ m \neq \perp$ dan $(2)\ m \notin Q$.

DEFINISI 4.16 Skema enkripsi kunci pribadi $\prod$ tidak dapat dipalsukan jika untuk semua musuh waktu polinomial probabilistik , terdapat pengabaian fungsi yang dapat diabaikan sehingga:

$$\Pr[\text{Enc-Forge}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

Sejalan dengan diskusi kita tentang permintaan verifikasi setelah Definisi 4.2, di sini kita juga dapat mempertimbangkan definisi yang lebih kuat $\mathcal{A}$ yang juga diberikan akses ke oracle dekripsi. Kita dapat memverifikasi bahwa konstruksi aman yang kami sajikan di bawah ini juga memenuhi definisi yang lebih kuat tersebut. Kami sekarang mendefinisikan skema enkripsi terotentikasi (aman).

DEFINISI 4.17 A Skema enkripsi kunci pribadi adalah skema enkripsi yang diautentikasi jika skema tersebut aman terhadap CCA dan tidak dapat dipalsukan.

Konstruksi Generik

Mungkin tergoda untuk berpikir bahwa kombinasi yang masuk akal antara skema enkripsi aman dan kode otentikasi pesan aman akan menghasilkan skema enkripsi yang diautentikasi. Pada bagian ini kami menunjukkan bahwa hal ini tidak terjadi. Hal ini menunjukkan bahwa bahkan alat kriptografi yang aman pun dapat digabungkan sedemikian rupa sehingga hasilnya tidak aman, dan sekali lagi menyoroti pentingnya definisi dan bukti keamanan. Sisi positifnya, kami menunjukkan bagaimana enkripsi dan otentikasi pesan dapat digabungkan dengan baik untuk mencapai kerahasiaan dan integritas bersama.

Secara keseluruhan, misalkan $\prod_E$ = (Enc, Des) merupakan skema enkripsi yang aman terhadap CPA dan misalkan $\prod_M$ = (Mac, Vrfy) menunjukkan kode autentikasi pesan, dimana pembangkitan kunci pada kedua skema hanya melibatkan pemilihan kunci n-bit yang

seragam. Ada tiga pendekatan alami untuk menggabungkan enkripsi dan otentikasi pesan menggunakan kunci $independen^4$ k_E dan k_M untuk $\prod_E$ dan $\prod_M$, masing-masing:

1. Enkripsi dan autentikasi: Dalam metode ini, enkripsi dan autentikasi pesan dihitung secara independen secara paralel. Artinya, jika diberi pesan teks biasa m, pengirim mengirimkan teks tersandi $\langle c, t \rangle$ di mana:

   $$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(m).$$

   Penerima mendekripsi c untuk memulihkan m; dengan asumsi tidak ada kesalahan yang terjadi, maka tag t akan diverifikasi. Jika $\mathrm{Vrfy}_{k\,M}\ (m, t) = 1$, penerima mengeluarkan m; jika tidak, ini akan menghasilkan kesalahan.
2. Otentikasi-lalu-enkripsi: Di sini tag MAC t dihitung terlebih dahulu, lalu pesan dan tag dienkripsi bersama. Artinya, jika diberi pesan m, pengirim mengirimkan ciphertext c yang dihitung sebagai:

   $$t \leftarrow \mathsf{Mac}_{k_M}(m) \quad \text{and} \quad c \leftarrow \mathsf{Enc}_{k_E}(m \| t).$$

   Penerima mendekripsi c untuk mendapatkan $m \parallel t$; dengan asumsi tidak ada kesalahan yang terjadi, maka tag t akan diverifikasi. Seperti sebelumnya, jika $\mathrm{Vrfy}_{k\,M}$ $(m, t) = 1$, penerima mengeluarkan m; jika tidak, ini akan menghasilkan kesalahan.
3. Enkripsi-lalu-otentikasi: Dalam hal ini, pesan m dienkripsi terlebih dahulu dan kemudian tag MAC dihitung berdasarkan hasilnya. Artinya, cipherteksnya adalah pasangan $\langle c, t \rangle$ dimana:

   $$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(c).$$

   (Lihat juga Konstruksi 4.18.) Jika $\mathrm{Vrfy}_{k\,M}\ (c, t) = 1$, maka penerima mendekripsi c dan mengeluarkan hasilnya; jika tidak, ini akan menghasilkan kesalahan.

Kami menganalisis masing-masing pendekatan di atas ketika pendekatan tersebut dibuat dengan komponen aman "generik", yaitu skema enkripsi aman CPA arbitrer dan MAC aman arbitrer (sangat). Kami menginginkan pendekatan yang memberikan kerahasiaan dan integritas bersama ketika menggunakan komponen apa pun (aman), dan oleh karena itu kami akan menolak pendekatan apa pun yang dianggap "tidak aman" bahkan jika terdapat satu contoh tandingan dari skema enkripsi aman/MAC yang kombinasinya tidak aman. Pendekatan "semua atau tidak sama sekali" ini mengurangi kemungkinan terjadinya kelemahan dalam implementasi. Secara khusus, skema enkripsi yang diautentikasi dapat diterapkan dengan melakukan panggilan ke "subrutin enkripsi" dan "subrutin autentikasi pesan", dan penerapan subrutin tersebut dapat diubah di kemudian hari. (Hal ini biasanya terjadi ketika perpustakaan kriptografi diperbarui, atau ketika standar diubah.) Oleh karena itu, menerapkan pendekatan yang keamanannya bergantung pada bagaimana komponen-komponennya diimplementasikan (bukan pada keamanan yang diberikannya) adalah berbahaya. Kami menekankan bahwa jika suatu pendekatan ditolak, hal ini tidak berarti bahwa pendekatan tersebut tidak aman untuk semua contoh komponen yang mungkin; Namun hal ini berarti bahwa setiap contoh pendekatan harus dianalisis dan dibuktikan aman sebelum digunakan.

Enkripsi-dan-otentikasi. Ingatlah bahwa dalam pendekatan ini enkripsi dan otentikasi pesan dilakukan secara independen. Diberikan pesan m, nilai yang dikirimkan adalah ⟨c, t⟩ di mana

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(m).$$

Pendekatan ini mungkin tidak mencapai tingkat kerahasiaan paling dasar sekalipun. Untuk melihat hal ini, perhatikan bahwa MAC yang aman tidak menjamin kerahasiaan apa pun sehingga tag $\mathrm{Mac}_{k\,M}\,(m)$ mungkin membocorkan informasi tentang m kepada penyadap.

(Sebagai contoh sepele, pertimbangkan MAC yang aman dimana bit pertama dari tag selalu sama dengan bit pertama dari pesan.) Jadi pendekatan enkripsi dan autentikasi dapat menghasilkan skema yang bahkan tidak memiliki enkripsi yang tidak dapat dibedakan. di hadapan penyadap.

Faktanya, pendekatan enkripsi dan autentikasi kemungkinan besar tidak aman terhadap serangan teks biasa yang dipilih bahkan ketika dipakai dengan komponen standar (tidak seperti contoh tandingan yang dibuat di paragraf sebelumnya). Secara khusus, jika MAC deterministik seperti CBC-MAC digunakan, maka tag yang dihitung pada pesan (untuk beberapa kunci tetap k_M) adalah sama setiap saat. Hal ini memungkinkan penyadap untuk mengidentifikasi kapan pesan yang sama dikirim dua kali, sehingga skema tersebut tidak aman terhadap CPA. Kebanyakan MAC yang digunakan dalam praktik bersifat deterministik, sehingga hal ini menimbulkan kekhawatiran yang nyata.

Otentikasi-lalu-enkripsi. Di sini, tag MAC $t \leftarrow \mathrm{Mac}_{k\,M}\,(m)$ pertama kali dihitung; kemudian $m \parallel t$ dienkripsi dan nilai yang dihasilkan $\mathrm{Enc}_{k\,M}\,(m \parallel t)$ dikirimkan. Kami menunjukkan bahwa kombinasi ini juga tidak selalu menghasilkan skema enkripsi yang diautentikasi.

Sebenarnya, kita telah menemukan skema enkripsi aman-CPA yang mana pendekatan ini tidak aman: skema mode-CBC-dengan-padding dibahas di Bagian 3.7. (Kami berasumsi berikut ini bahwa pembaca sudah familiar dengan bagian tersebut.) Ingatlah bahwa skema ini bekerja dengan terlebih dahulu mengisi teks biasa (yang dalam kasus kami adalah $m \parallel t$) dengan cara tertentu sehingga hasilnya adalah kelipatan dari blok tersebut. panjangnya, dan kemudian mengenkripsi hasilnya menggunakan mode CBC. Selama dekripsi, jika kesalahan pada padding terdeteksi setelah melakukan dekripsi mode CBC, maka kesalahan "padding buruk" akan muncul. Sehubungan dengan autentikasi-lalu-enkripsi, ini berarti sekarang ada dua sumber potensi kegagalan dekripsi: padding mungkin salah, atau tag MAC mungkin tidak terverifikasi. Secara skematis, algoritma dekripsi Dec' dalam skema gabungan bekerja sebagai berikut:

$\mathsf{Dec}'_{k_E,k_M}(c)$:

1. Hitung $\widetilde{m} := \mathrm{Dec}_{k\,E}\,(c)$. Jika kesalahan pada padding terdeteksi, kembalikan "padding buruk" dan hentikan.
2. Parsing $\widetilde{m}$ sebagai $m \parallel t$. Jika $\mathrm{Vrfy}_{k\,M}\,(m,t) = 1$ kembalikan m; jika tidak, keluaran "kegagalan otentikasi."

Dengan asumsi penyerang dapat membedakan dua pesan kesalahan tersebut, penyerang dapat menerapkan serangan teks sandi terpilih yang sama seperti yang dijelaskan dalam Bagian 3.7 pada skema di atas untuk memulihkan seluruh teks biasa asli dari teks sandi tertentu. (Hal ini disebabkan oleh fakta bahwa serangan padding-Oracle yang ditunjukkan pada Bagian 3.7 hanya mengandalkan kemampuan untuk mempelajari apakah ada kesalahan padding atau tidak, sesuatu yang diungkapkan oleh skema ini.) Jenis serangan ini telah berhasil dilakukan di dunia nyata dalam berbagai pengaturan, misalnya dalam konfigurasi IPsec yang menggunakan autentikasi-lalu-enkripsi.

Salah satu cara untuk memperbaiki skema di atas adalah dengan memastikan bahwa hanya satu pesan kesalahan yang dikembalikan, terlepas dari sumber kegagalan dekripsi. Ini adalah solusi yang tidak memuaskan karena beberapa alasan: (1) mungkin ada alasan yang sah (misalnya, kegunaan, debugging) untuk memiliki banyak pesan kesalahan; (2) memaksa pesan kesalahan menjadi sama berarti bahwa kombinasi tersebut tidak lagi benar-benar umum, yaitu mengharuskan pelaksana pendekatan autentikasi-lalu-enkripsi untuk menyadari pesan kesalahan apa yang dikembalikan oleh CPA yang mendasarinya. -skema enkripsi yang aman; (3) yang terpenting, sangat sulit untuk memastikan bahwa kesalahan-kesalahan yang berbeda tidak dapat dibedakan karena, misalnya, bahkan perbedaan waktu untuk mengembalikan kesalahan-kesalahan ini dapat digunakan oleh musuh untuk membedakan kesalahan-kesalahan tersebut (lih. .diskusi kita sebelumnya tentang pengaturan waktu serangan di akhir Bagian 4.2). Beberapa versi SSL mencoba hanya menggunakan satu pesan kesalahan bersamaan dengan pendekatan autentikasi-lalu-enkripsi, namun serangan padding-Oracle masih berhasil dilakukan dengan menggunakan informasi waktu semacam ini. Kami menyimpulkan bahwa autentikasi-lalu-enkripsi tidak menyediakan enkripsi yang diautentikasi secara umum, dan tidak boleh digunakan.

Enkripsi-lalu-otentikasi. Dalam pendekatan ini, pesan dienkripsi terlebih dahulu dan kemudian MAC dihitung berdasarkan hasilnya. Artinya, pesannya adalah pasangan $\langle c, t \rangle$ di mana

$$c \leftarrow \mathsf{Enc}_{k_E}(m) \quad \text{and} \quad t \leftarrow \mathsf{Mac}_{k_M}(c).$$

Dekripsi $\langle c, t \rangle$ dilakukan dengan mengeluarkan $\perp$ jika $\mathrm{Vrfy}_{k\,M}\ (c, t) \neq 1$, dan sebaliknya mengeluarkan $\mathrm{Dec}_{k\,E}\ (c)$. Lihat Konstruksi 4.18 untuk penjelasan formal.

CONSTRUCTION 4.18

Let $\Pi_E = (\mathsf{Enc}, \mathsf{Dec})$ be a private-key encryption scheme and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code, where in each case key generation is done by simply choosing a uniform n-bit key. Define a private-key encryption scheme $(\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ as follows:

- Gen': on input 1^n, choose independent, uniform $k_E, k_M \in \{0,1\}^n$ and output the key (k_E, k_M).
- Enc': on input a key (k_E, k_M) and a plaintext message m, compute $c \leftarrow \mathsf{Enc}_{k_E}(m)$ and $t \leftarrow \mathsf{Mac}_{k_M}(c)$. Output the ciphertext $\langle c, t \rangle$.
- Dec': on input a key (k_E, k_M) and a ciphertext $\langle c, t \rangle$, first check whether $\mathsf{Vrfy}_{k_M}(c, t) \stackrel{?}{=} 1$. If yes, then output $\mathsf{Dec}_{k_E}(c)$; if no, then output $\perp$.

Pendekatan ini masuk akal, selama MAC sangat aman, seperti pada Definisi 4.3. Sebagai intuisi untuk keamanan pendekatan ini, katakanlah ciphertext $\langle c, t\rangle$ valid jika t adalah tag MAC yang valid pada c. Keamanan MAC yang kuat memastikan bahwa musuh tidak akan dapat menghasilkan ciphertext valid apa pun yang tidak diterimanya dari oracle enkripsinya. Hal ini secara langsung menyiratkan bahwa Konstruksi 4.18 tidak dapat dipalsukan. Sedangkan untuk keamanan CCA, MAC yang dihitung melalui ciphertext memiliki efek menjadikan oracle dekripsi tidak berguna karena untuk setiap ciphertext c, sehingga musuh tunduk pada oracle dekripsinya, musuh sudah mengetahui dekripsinya (jika menerima $\langle c, t\rangle$ dari oracle enkripsinya sendiri) atau jika tidak, hasilnya akan berupa kesalahan (karena musuh tidak dapat menghasilkan teks sandi baru yang valid). Ini berarti bahwa keamanan CCA dari skema gabungan berkurang menjadi keamanan CPA dari $\prod_E$. Perhatikan juga bahwa MAC diverifikasi sebelum dekripsi dilakukan; dengan demikian, verifikasi MAC tidak dapat membocorkan apa pun tentang teks biasa (berbeda dengan serangan padding-Oracle yang kita lihat pada pendekatan autentikasi-lalu-enkripsi). Kami sekarang meresmikan argumen di atas.

TEOREMA 4.19 Misalkan $\prod_M$ adalah skema enkripsi kunci pribadi yang aman dengan CPA, dan ΠM adalah kode autentikasi pesan yang sangat aman. Kemudian Konstruksi 4.18 adalah skema enkripsi yang diautentikasi.

BUKTI Misalkan $\prod'$ menunjukkan skema yang dihasilkan dari Konstruksi 4.18. Kita perlu menunjukkan bahwa $\prod'$ tidak dapat dipalsukan, dan aman untuk CCA. Mengikuti intuisi yang diberikan di atas, katakanlah teks sandi $\langle c, t\rangle$ valid (sehubungan dengan beberapa kunci rahasia tetap (k_E, k_M)) jika $\text{Vrf}_{k_M}(c, t) = 1$. Kami menunjukkan bahwa keamanan yang kuat dari $\prod_M$ menyiratkan bahwa (kecuali dengan probabilitas yang dapat diabaikan) teks sandi "baru" apa pun yang dikirimkan musuh ke oracle dekripsi akan menjadi tidak valid. Seperti telah dibahas, hal ini secara langsung menyiratkan ketidakmungkinan untuk ditempa. (Faktanya, ini lebih kuat daripada unforgeability.) Fakta ini juga membuat oracle dekripsi tidak berguna, dan berarti bahwa keamanan CCA dari $\prod'$ = (Gen', Enc', Dec') berkurang menjadi keamanan CPA dari $\prod_E$.

Secara lebih rinci, misalkan $\mathcal{A}$ menjadi musuh waktu polinomial probabilistik yang menyerang Konstruksi 4.18 dalam serangan teks sandi yang dipilih (lih. Definisi 3.33). Katakanlah teks sandi $\langle c, t\rangle$ baru jika $\mathcal{A}$ tidak menerima $\langle c, t\rangle$ dari oracle enkripsinya atau sebagai teks sandi tantangan. Misalkan ValidQuery adalah kejadian $\mathcal{A}$ mengirimkan ciphertext baru $\langle c, t\rangle$ ke oracle dekripsinya yang valid, yaitu $\text{Vrf}_{k_M}(c, t)$ = 1. Kita buktikan:

KLAIM 4.20 Pr[ValidQuery] dapat diabaikan.

BUKTI Secara intuitif, hal ini disebabkan oleh fakta bahwa jika ValidQuery terjadi maka musuh telah membuat pasangan baru yang valid (c, t) dalam percobaan Mac-sforge. Secara formal, misalkan $q(\cdot)$ adalah batas atas polinomial pada jumlah kueri oracle dekripsi yang dibuat oleh

$\mathcal{A}$, dan pertimbangkan musuh berikut $\mathcal{A}_M$ yang menyerang kode autentikasi pesan $\prod_M$ (yaitu, berjalan dalam eksperimen Mac-sforg$e_{\mathcal{A}M,\prod M}(n)$):

Adversarial $\mathcal{A}_M$:
$\mathcal{A}_M$ diberikan input 1^n dan mempunyai akses ke oracle MAC Ma$c_{kM}(\cdot)$.

1. Pilih seragam $k_E \in \{0,1\}^n$ dan $i \in \{1,\ldots,q(n)\}$.
2. Jalankan $\mathcal{A}$ pada masukan 1^n. Saat $\mathcal{A}$ membuat kueri enkripsi-Oracle untuk pesan m, jawablah sebagai berikut:
   (i) Hitung $c \longleftarrow$ En$c_{kE}(m)$.
   (ii) Kueri c ke oracle MAC dan terima t sebagai tanggapan. Kembalikan $\langle c,t\rangle$ ke $\mathcal{A}$.

Tantangan ciphertext disiapkan dengan cara yang persis sama (dengan bit seragam $b \in \{0,1\}$ dipilih untuk memilih pesan mb yang dienkripsi).
Saat $\mathcal{A}$ membuat kueri oracle dekripsi untuk ciphertext c, t , jawablah sebagai berikut: Jika ini adalah kueri oracle dekripsi keluaran (c,t). Jika tidak:

(i) Jika c,t adalah respons terhadap kueri oracle enkripsi sebelumnya untuk pesan m, kembalikan m.
(ii) Jika tidak, kembalikan $\perp$.

Intinya, $\mathcal{A}_M$ "menebak" bahwa kueri oracle dekripsi ke-i dari $\mathcal{A}$ akan menjadi kueri baru dan valid pertama yang $\mathcal{A}$ dibuat. Dalam hal ini, $\mathcal{A}_M$ mengeluarkan pemalsuan yang valid pada pesan c yang sebelumnya tidak pernah dikirimkan ke oracle MAC-nya sendiri. Jelas $\mathcal{A}_M$ berjalan dalam waktu polinomial probabilistik. Kami sekarang menganalisis kemungkinan bahwa $\mathcal{A}_M$ menghasilkan pemalsuan yang baik. Poin kuncinya adalah tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}_M$ didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen Priv$K^{cca}_{\mathcal{A},\prod}(n)$ hingga peristiwa ValidQuery terjadi. Untuk melihatnya, perhatikan bahwa kueri oracle enkripsi $\mathcal{A}$ (serta komputasi tantangan ciphertext) disimulasikan dengan sempurna oleh $\mathcal{A}_M$. Adapun permintaan dekripsi-Oracle, hingga ValidQuery terjadi $\mathcal{A}$, semuanya disimulasikan dengan benar. Dalam kasus (i) hal ini sudah jelas. Sedangkan untuk kasus (ii), jika ciphertext c,t yang dikirimkan ke oracle dekripsi adalah baru, maka selama ValidQuery belum muncul, jawaban yang benar untuk query oracle dekripsi adalah memang $\perp$. (Perhatikan bahwa kasus (i) adalah kasus dimana c,t bukanlah kasus baru, dan kasus (ii) adalah kasus yang tepat dimana c,t adalah kasus baru.) Penarikan kembali $\mathcal{A}$ hal ini tidak diperbolehkan dalam mengirimkan tantangan ciphertext ke oracle dekripsi.

Karena tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}_M$ didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen Priv$K^{cca}_{\mathcal{A},\prod}(n)$ hingga kejadian ValidQuery terjadi, probabilitas kejadian ValidQuery dalam eksperimen Mac-forg$e_{\mathcal{A}M,\prod M}(n)$ adalah sama sebagai peluang kejadian tersebut dalam percobaan Priv$K^{cca}_{\mathcal{A},\prod}(n)$.

Jika $\mathcal{A}_M$ menebak dengan benar indeks pertama i ketika ValidQuery muncul, maka $\mathcal{A}_M$ mengeluarkan (c,t) yang Vrf$y_{kM}(c,t) = 1$ (karena $\langle c,t\rangle$ valid) dan tidak pernah diberi tag t di respons terhadap kueri Ma$c_{kM}(c)$ (karena $\langle c,t\rangle$ baru). Dalam hal ini, $\mathcal{A}_M$ berhasil melakukan eksperimen Mac-sforg$e_{\mathcal{A}M,\prod M}(n)$.

Peluang $\mathcal{A}_M$ menebak i dengan benar adalah $1/q(n)$. Karena itu

$$\Pr[\text{Mac-sforge}_{\mathcal{A}_M,\Pi_M}(n) = 1] \geq \Pr[\text{ValidQuery}]/q(n).$$

Karena $\prod_M$ adalah MAC yang sangat aman dan q adalah polinomial, kami menyimpulkan bahwa Pr[ValidQuery] dapat diabaikan.

Kami menggunakan Klaim 4.20 untuk membuktikan keamanan $\prod'$. Kasus yang lebih mudah adalah membuktikan bahwa $\prod'$ tidak dapat dipalsukan. Hal ini langsung mengikuti klaim tersebut, sehingga kami hanya memberikan alasan informal dan bukan bukti formal. Perhatikan terlebih dahulu bahwa musuh $\mathcal{A}'$ dalam percobaan enkripsi yang tidak dapat dipalsukan adalah versi terbatas dari musuh dalam percobaan teks sandi yang dipilih (dalam percobaan pertama, musuh hanya memiliki akses ke oracle enkripsi). Ketika $\mathcal{A}'$ mengeluarkan ciphertext $\langle c, t\rangle$ pada akhir percobaannya, ia "berhasil" hanya jika $\langle c, t\rangle$ valid dan baru. Namun klaim sebelumnya menunjukkan dengan tepat bahwa kemungkinan terjadinya peristiwa semacam itu dapat diabaikan.

Sedikit lebih rumit untuk membuktikan bahwa $\prod'$ aman terhadap CCA. Sekali lagi $\mathcal{A}$ menjadi musuh waktu polinomial probabilistik yang menyerang $\prod'$ dalam serangan teks tersandi yang dipilih. Kita punya

$$\begin{aligned}&\Pr[\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi'}(n) = 1]\\ &\quad \leq \Pr[\text{ValidQuery}] + \Pr[\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\text{ValidQuery}}]. \qquad (4.8)\end{aligned}$$

Kami telah menunjukkan bahwa Pr[ValidQuery] dapat diabaikan. Klaim berikut melengkapi pembuktian teorema tersebut.

KLAIM 4.21 Terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\text{ValidQuery}}] \leq \frac{1}{2} + \text{negl}(n).$$

Untuk membuktikan klaim ini, kami mengandalkan keamanan CPA dari $\prod_E$. Pertimbangkan musuh $\mathcal{A}_E$ berikut yang menyerang $\prod_E$ dalam serangan teks biasa yang dipilih:

Musuh $\mathcal{A}_E$:

$\mathcal{A}_E$ diberikan input 1^n dan memiliki akses ke $\text{Enc}_{kE}(\cdot)$.

1. Pilih seragam $k_M \in \{0,1\}^n$.
2. Jalankan $\mathcal{A}$ pada masukan 1^n. Saat $\mathcal{A}$ membuat kueri enkripsi-Oracle untuk pesan m, jawablah sebagai berikut:
   (i) Kueri m ke $\text{Enc}_{kE}(\cdot)$ dan terima c sebagai tanggapan.
   (ii) Hitung $t \leftarrow \text{Mac}_{kM}(c)$ dan kembalikan $\langle c, t\rangle$ ke $\mathcal{A}$.

   Saat $\mathcal{A}$ membuat kueri oracle dekripsi untuk teks tersandi $\langle c, t\rangle$, jawablah sebagai berikut:

Jika ⟨c, t⟩ merupakan respons terhadap kueri oracle enkripsi sebelumnya untuk pesan m, kembalikan m. Jika tidak, kembalikan ⊥.

3. Saat $\mathcal{A}$ mengeluarkan pesan (m_0, m_1), keluarkan pesan yang sama dan terima tantangan ciphertext c sebagai tanggapannya. Hitung $t \leftarrow \text{Mac}_{kM}(c)$, dan kembalikan $\langle c, t \rangle$ sebagai teks sandi tantangan untuk $\mathcal{A}$. Lanjutkan menjawab pertanyaan oracle $\mathcal{A}$ seperti di atas.
4. Keluarkan bit b' yang sama dengan yang dikeluarkan oleh $\mathcal{A}$.
   Perhatikan bahwa $\mathcal{A}_E$ tidak memerlukan oracle dekripsi karena ia hanya berasumsi bahwa permintaan dekripsi apa pun yang bukan merupakan hasil dari permintaan oracle enkripsi sebelumnya adalah tidak valid.

Jelasnya, $\mathcal{A}_E$ berjalan dalam waktu polinomial probabilistik. Selanjutnya, tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}_E$ didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen $\text{Priv}K^{cca}_{\mathcal{A},\prod'}(n)$ selama event ValidQuery tidak pernah terjadi. Oleh karena itu, probabilitas $\mathcal{A}_E$ berhasil ketika ValidQuery tidak terjadi sama dengan probabilitas $\mathcal{A}$ berhasil ketika ValidQuery tidak terjadi; yaitu.,

$$\Pr[\text{PrivK}^{\text{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1 \wedge \overline{\text{ValidQuery}}] = \Pr[\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\text{ValidQuery}}],$$

menyiratkan hal itu

$$\begin{aligned}\Pr[\text{PrivK}^{\text{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1] &\geq \Pr[\text{PrivK}^{\text{cpa}}_{\mathcal{A}_E,\Pi_E}(n) = 1 \wedge \overline{\text{ValidQuery}}] \\ &= \Pr[\text{PrivK}^{\text{cca}}_{\mathcal{A},\Pi'}(n) = 1 \wedge \overline{\text{ValidQuery}}].\end{aligned}$$

Karena $\prod_E$ aman terhadap CPA, terdapat fungsi yang dapat diabaikan sehingga $\Pr[\text{Priv}K^{cpa}_{\mathcal{A}_E,\prod_E}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$. Ini membuktikan klaim tersebut.

Kebutuhan akan kunci independen. Kami menyimpulkan bagian ini dengan menekankan prinsip dasar kriptografi: contoh kriptografi primitif yang berbeda harus selalu menggunakan kunci independen. Untuk mengilustrasikannya di sini, pertimbangkan apa yang dapat terjadi pada metodologi enkripsi lalu autentikasi ketika kunci k yang sama digunakan untuk enkripsi dan autentikasi. Misalkan F adalah permutasi pseudorandom kuat. Oleh karena itu F^{-1} juga merupakan permutasi pseudorandom kuat. Definisikan $\text{Enc}_k(m) = F_k(m \parallel r)$ untuk $m \in \{0,1\}^{n/2}$ dan seragam $r \in \{0,1\}^{n/2}$, dan definisikan $\text{Mac}_k(c) = F_k^{-1}(c)$. Dapat ditunjukkan bahwa skema enkripsi ini aman terhadap CPA (bahkan, ia bahkan aman terhadap CCA; lihat Latihan 4.25), dan kita mengetahui bahwa kode autentikasi pesan yang diberikan adalah MAC yang aman. Namun, kombinasi enkripsi-lalu-otentikasi menggunakan kunci k yang sama seperti yang diterapkan pada pesan m menghasilkan:

$$\text{Enc}_k(m), \text{Mac}_k(\text{Enc}_k(m)) = F_k(m\|r), F_k^{-1}(F_k(m\|r)) = F_k(m\|r), m\|r,$$

dan pesan m terungkap dengan jelas! Hal ini sama sekali tidak bertentangan dengan Teorema 4.19, karena Konstruksi 4.18 secara tegas mensyaratkan bahwa k_M , k_E dipilih (secara seragam dan) independen. Kami mendorong pembaca untuk memeriksa di mana independensi ini digunakan dalam pembuktian Teorema 4.19.

Sesi Komunikasi yang Aman

Kami menjelaskan secara singkat penerapan enkripsi yang diautentikasi pada pengaturan dua pihak yang ingin berkomunikasi "secara aman"—yakni, dengan kerahasiaan dan integritas bersama—selama sesi komunikasi. (Untuk keperluan bagian ini, sesi komunikasi hanyalah suatu periode waktu di mana pihak-pihak yang berkomunikasi mempertahankan keadaan.) Dalam perlakuan kami di sini, kami sengaja bersikap informal; definisi formalnya cukup rumit, dan topik ini bisa dibilang lebih berkaitan dengan bidang keamanan jaringan daripada kriptografi.

Misalkan $\prod$ = (Enc, Des) adalah skema enkripsi yang diautentikasi. Pertimbangkan dua pihak A dan B yang berbagi kunci k dan ingin menggunakan kunci ini untuk mengamankan komunikasi mereka selama sesi berlangsung. Hal yang jelas untuk dilakukan di sini adalah menggunakan$\prod$: Setiap kali, katakanlah, A ingin mengirimkan pesan m ke B, ia menghitung $c \leftarrow \text{Enc}_k(m)$ dan mengirimkan c ke B; pada gilirannya, B mendekripsi c untuk memulihkan hasilnya (mengabaikan hasil jika dekripsi kembali $\perp$). Demikian pula, prosedur yang sama diikuti ketika B ingin mengirim pesan ke A. Namun pendekatan sederhana ini tidak cukup, karena terdapat berbagai potensi serangan:

Serangan pemesanan ulang Seorang penyerang dapat menukar urutan pesan. Contohnya, jika A mengirimkan c_1 (enkripsi m_1) dan selanjutnya mengirimkan c_2 (enkripsi m_2), penyerang yang memiliki kendali atas jaringan dapat mengirimkan c_2 sebelum c_1 dan dengan demikian menyebabkan B mengeluarkan pesan-pesan tersebut. dalam urutan yang salah. Hal ini menyebabkan ketidaksesuaian pandangan kedua belah pihak terhadap sesi komunikasinya.

Serangan memutar ulang Seorang penyerang dapat memutar ulang ciphertext c (valid) yang dikirim sebelumnya oleh salah satu pihak. Sekali lagi, hal ini menyebabkan ketidaksesuaian antara apa yang dikirim oleh satu pihak dan diterima oleh pihak lain.

Serangan refleksi Seorang penyerang dapat mengambil ciphertext c yang dikirim dari A ke B dan mengirimkannya kembali ke A. Hal ini sekali lagi dapat menyebabkan ketidakcocokan antara transkrip sesi komunikasi kedua pihak: A dapat mengeluarkan pesan m, meskipun B tidak pernah mengirim pesan seperti itu.

Untungnya, serangan di atas mudah dicegah dengan menggunakan counter untuk mengatasi dua serangan pertama dan bit arah untuk mencegah serangan ketiga.[5] Kami menjelaskan hal ini secara bersamaan. Masing-masing pihak memiliki dua penghitung ct$r_{A,B}$ dan ct$r_{B,A}$ yang mencatat jumlah pesan yang dikirim dari A ke B (resp., B ke A) selama sesi. Penghitung ini diinisialisasi ke 0 dan bertambah setiap kali salah satu pihak mengirim atau menerima pesan (yang valid). Para pihak juga menyepakati sedikit $b_{A,B,}$ dan mendefinisikan

$b_{B,A}$ sebagai pelengkapnya. (Salah satu cara untuk melakukannya adalah dengan menetapkan $b_{A,B} = 0$ jika identitas A secara leksikografis lebih kecil daripada identitas B.)

Ketika A ingin mengirimkan pesan m ke B, dia menghitung ciphertext $c \leftarrow \text{Enc}_k(b_{A,B} \| \text{ctr}_{A,B} \| m)$ dan mengirimkan c; dia kemudian menambah $\text{ctr}_{A,B}$. Setelah menerima c, pihak B mendekripsi; jika hasilnya $\perp$, dia langsung menolak. Jika tidak, dia akan mem-parsing pesan yang didekripsi sebagai $b \| ctr \| m$. Jika $b = b_{A,B}$ dan $\text{ctr} = \text{ctr}_{A,B}$, maka B mengeluarkan m dan menambah $\text{ctr}_{A,B}$; jika tidak, B menolak. Langkah-langkah di atas, mutatis mutandis, diterapkan ketika B mengirim pesan ke A. Kita perhatikan bahwa karena pihak-pihak tersebut tetap menjaga keadaan (yaitu, counter $\text{ctr}_{A,B}$ dan $\text{ctr}_{B,A}$), pihak-pihak tersebut dapat dengan mudah menggunakan stateful au-skema enkripsi tersirat $\prod$.

Enkripsi Aman CCA

Definisi ini mengikuti langsung bahwa setiap skema enkripsi yang diautentikasi juga aman terhadap serangan teks sandi yang dipilih. Apakah ada skema enkripsi kunci pribadi yang aman dengan CCA dan tidak dapat dipalsukan?

Kita dapat membayangkan aplikasi yang memerlukan keamanan CCA, namun enkripsi yang diautentikasi tidak diperlukan. Salah satu contohnya adalah ketika enkripsi kunci pribadi digunakan untuk pengangkutan kunci. Sebagai contoh nyata, katakanlah server memberikan token perangkat keras anti-rusak kepada pengguna, di mana tertanam dalam token tersebut adalah kunci jangka panjang k. Server dapat mengunggah kunci baru jangka pendek k' ke token ini dengan memberikan pengguna $\text{Enc}_k(k')$; pengguna seharusnya memberikan ciphertext ini ke token, yang akan mendekripsinya dan menggunakan k' untuk jangka waktu berikutnya. Serangan teks sandi yang dipilih dalam pengaturan ini dapat memungkinkan pengguna mempelajari k', sesuatu yang seharusnya tidak dapat dilakukan oleh pengguna. (Perhatikan bahwa di sini serangan padding-Oracle, yang hanya bergantung pada kemampuan pengguna untuk menentukan apakah terjadi kegagalan dekripsi, berpotensi dapat dilakukan dengan lebih mudah.) Di sisi lain, tidak banyak kerugian yang terjadi jika pengguna dapat melakukannya. menghasilkan ciphertext "valid" yang menyebabkan token menggunakan kunci k'' yang sewenang-wenang (tidak terkait) untuk periode waktu berikutnya. (Tentu saja, ini bergantung pada apa yang dilakukan token dengan kunci jangka pendek ini.)

Sekalipun demikian, untuk enkripsi kunci pribadi, sebagian besar konstruksi "alami" dari skema aman CCA yang kita ketahui memenuhi definisi yang lebih kuat dari enkripsi yang diautentikasi. Dengan kata lain, tidak ada alasan nyata untuk menggunakan skema aman CCA yang bukan merupakan skema enkripsi terotentikasi, hanya karena kita tidak benar-benar memiliki konstruksi yang lebih efisien daripada konstruksi yang memenuhi skema enkripsi yang terakhir.

Namun, dari sudut pandang konseptual, penting untuk menjaga pengertian keamanan CCA dan enkripsi yang diautentikasi tetap berbeda. Sehubungan dengan keamanan CCA, kami tidak tertarik pada integritas pesan itu sendiri; sebaliknya, kami ingin memastikan privasi bahkan terhadap musuh aktif yang dapat mengganggu komunikasi dari pengirim ke penerima. Sebaliknya, sehubungan dengan enkripsi yang diautentikasi, kami tertarik pada tujuan ganda yaitu kerahasiaan dan integritas. Kami menekankan hal ini di sini karena dalam pengaturan

kunci publik yang akan kita pelajari nanti di buku ini, perbedaan antara enkripsi yang diautentikasi dan keamanan CCA lebih jelas.

4.6 MAC TEORI

Pada bagian sebelumnya kita telah menjelajahi kode autentikasi pesan dengan keamanan komputasional, yakni asumsi batasan waktu berjalan penyerang. Mengingat hasil Bab 2, adalah wajar untuk bertanya apakah otentikasi pesan di hadapan musuh yang tidak terbatas mungkin dilakukan. Pada bagian ini, kami menunjukkan dalam kondisi apa keamanan teori informasi (dan bukan komputasi) dapat dicapai.

Pengamatan pertama adalah bahwa tidak mungkin untuk mencapai keamanan yang "sempurna" dalam konteks ini: yaitu, kita tidak dapat berharap untuk memiliki kode otentikasi pesan yang probabilitas bahwa musuh mengeluarkan tag yang valid pada pesan yang sebelumnya tidak diautentikasi adalah 0. Alasannya adalah bahwa musuh dapat dengan mudah menebak tag t yang valid pada pesan apa pun dan tebakannya akan benar dengan probabilitas (setidaknya) $1/2^{|t|}$, di mana $|t|$ menunjukkan panjang tag skema.

Contoh di atas memberi tahu kita apa yang ingin kita capai: MAC dengan tag dengan panjang $|t|$ di mana kemungkinan pemalsuan paling banyak $1/2^{|t|}$, bahkan untuk musuh yang tidak terikat. Kita akan melihat bahwa hal ini dapat dicapai, namun hanya dengan batasan berapa banyak pesan yang diautentikasi oleh pihak yang jujur.

Kami pertama-tama mendefinisikan keamanan teori informasi untuk kode otentikasi pesan. Titik awalnya adalah dengan melakukan eksperimen Mac-forge$_{\mathcal{A},\prod}(n)$ yang digunakan untuk mendefinisikan keamanan untuk MACs yang aman secara komputasi (lih. Definisi 4.2), tetapi untuk menghilangkan parameter keamanan n dan mengharuskan
Pr[Mac-forge$_{\mathcal{A},\prod} = 1$] harus "kecil" untuk semua musuh $\mathcal{A}$ (dan bukan hanya musuh yang berjalan dalam waktu polinomial). Sebagaimana disebutkan di atas (dan akan dibuktikan secara formal di Bagian 4.6), namun definisi seperti itu tidak mungkin dicapai. Sebaliknya, keamanan teori informasi hanya dapat dicapai jika kita membatasi jumlah pesan yang diautentikasi oleh pihak yang jujur. Di sini kita melihat pengaturan paling dasar, di mana para pihak hanya mengautentikasi satu pesan. Kami menyebutnya sebagai otentikasi pesan satu kali. Eksperimen berikut memodifikasi Mac-forge$_{\mathcal{A},\prod}(n)$ mengikuti pembahasan di atas:
Eksperimen autentikasi pesan satu kali pada Mac-forge$^{1-time}_{\mathcal{A},\prod}$:

1. Kunci k dihasilkan dengan menjalankan Gen.
2. Musuh $\mathcal{A}$ mengeluarkan pesan m', dan sebagai imbalannya diberi tag $t' \leftarrow$ Mac$_k(m')$.
3. Keluaran (m, t).
4. Keluaran percobaan ditetapkan 1 jika dan hanya jika
   (1) Vrfy$_k(m, t) = 1$ dan $(2)\ m \neq m'$.

DEFINISI 4.22 Kode otentikasi pesan $\prod$ = (Gen, Mac, Vrfy) adalah ε-aman satu kali, atau hanya ε-aman, jika untuk semua musuh (bahkan tanpa batas) $\mathcal{A}$:

$$\Pr[\text{Mac-forge}^{1\text{-time}}_{\mathcal{A},\Pi} = 1] \leq \varepsilon.$$

Membangun MAC Teori Informasi

Pada bagian ini kami menunjukkan bagaimana membangun MAC ε-secure berdasarkan pada fungsi yang sangat universal.[6] Kami kemudian menunjukkan konstruksi sederhana dari fungsi yang terakhir.

Misalkan $h : K \times M \rightarrow T$ adalah fungsi berkunci yang masukan pertamanya berupa kunci $k \in K$ dan masukan keduanya diambil dari suatu domain M. Seperti biasa, kita menulis $h_k(m)$ dan bukan $h(k, m)$. Maka h bersifat sangat universal (atau tidak bergantung berpasangan) jika untuk dua masukan berbeda m, m' nilai $h_k(m)$ dan $h_k(m')$ terdistribusi T secara seragam dan independen ketika k adalah kunci yang seragam. Hal ini sama dengan mengatakan bahwa probabilitas $h_k(m)$, $h_k(m')$ mengambil nilai tertentu t, t' adalah tepat $1/|T|^2$. Itu adalah:

DEFINISI 4.23 Suatu fungsi h : K×M → T bersifat sangat universal jika untuk semua m, m' ∈ M yang berbeda dan semua t, t' ∈ T dinyatakan bahwa

$$\Pr[h_k(m) = t \wedge h_k(m') = t'] = \frac{1}{|\mathcal{T}|^2},$$

dimana probabilitasnya diambil alih pilihan seragam k ∈ K.

Hal di atas harus memotivasi konstruksi kode otentikasi pesan satu kali dari fungsi yang sangat universal h. Tag t pada pesan m diperoleh dengan menghitung $h_k(m)$, dimana kunci k seragam; lihat Konstruksi 4.24. Secara intuitif, bahkan setelah musuh mengamati tag $t' := h_k(m')$ untuk pesan apa pun m', tag yang benar $h_k(m)$ untuk pesan lainnya m masih terdistribusi secara T merata dari sudut pandang musuh. Dengan demikian, musuh tidak dapat berbuat apa-apa selain menebak tag secara membabi buta, dan tebakan ini hanya akan benar jika probabilitasnya $1/|T|$.

CONSTRUCTION 4.24

Let $h : \mathcal{K} \times \mathcal{M} \rightarrow \mathcal{T}$ be a strongly universal function. Define a MAC for messages in $\mathcal{M}$ as follows:

- Gen: choose uniform $k \in \mathcal{K}$ and output it as the key.
- Mac: on input a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, output the tag $t := h_k(m)$.
- Vrfy: on input a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$, output 1 if and only if $t \stackrel{?}{=} h_k(m)$. (If $m \notin \mathcal{M}$, then output 0.)

MAC dari fungsi apa pun yang sangat universal.

Konstruksi di atas dapat dianalogikan dengan Konstruksi 4.5. Hal ini karena fungsi yang sangat universal h identik dengan fungsi acak, asalkan hanya dievaluasi dua kali.

TEOREMA 4.25 Misalkan $h : K \times M \rightarrow T$ merupakan fungsi universal kuat. Maka Konstruksi 4.24 adalah MAC $1/|T|$-aman untuk pesan dalam M.

BUKTI Biarkan $\mathcal{A}$ menjadi musuh. Seperti biasa dalam pengaturan teori informasi, kita dapat berasumsi bahwa $\mathcal{A}$ bersifat deterministik tanpa kehilangan keumumannya. Jadi pesan m' yang dikeluarkan $\mathcal{A}$ pada awal percobaan adalah tetap. Selanjutnya, pasangan (m, t) yang dihasilkan $\mathcal{A}$ pada akhir percobaan merupakan fungsi deterministik dari tag t' pada m' yang diterima $\mathcal{A}$. Oleh karena itu, kita punya

$$\begin{aligned}\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}^{1\text{-time}} = 1] &= \sum_{t' \in \mathcal{T}} \Pr[\text{Mac-forge}_{\mathcal{A},\Pi}^{1\text{-time}} = 1 \ \wedge \ h_k(m') = t'] \\ &= \sum_{\substack{t' \in \mathcal{T} \\ (m,t) := \mathcal{A}(t')}} \Pr[h_k(m) = t \ \wedge \ h_k(m') = t'] \\ &= \sum_{\substack{t' \in \mathcal{T} \\ (m,t) := \mathcal{A}(t')}} \frac{1}{|\mathcal{T}|^2} \ = \ \frac{1}{|\mathcal{T}|}.\end{aligned}$$

Ini membuktikan teorema tersebut.

Kita sekarang beralih ke konstruksi klasik dari fungsi yang sangat universal. Kami mengasumsikan beberapa pengetahuan dasar tentang modul aritmatika bilangan prima; pembaca dapat merujuk ke Bagian 8.1 dan 8.1 untuk latar belakang yang diperlukan.

Perbaiki beberapa p prima, dan misalkan $\mathbb{Z}_p \stackrel{\text{def}}{=} \{0, \ldots, p - 1\}$. Kami mengambil ruang pesan kami $M = \mathbb{Z}_p$; ruang tag yang mungkin juga akan menjadi $T = \mathbb{Z}_p$. Kunci (a, b) terdiri dari sepasang elemen dari $\mathbb{Z}_p$; jadi, $K = \mathbb{Z}_p \times \mathbb{Z}_p$. Definisikan h sebagai

$$h_{a,b}(m) \stackrel{\text{def}}{=} [a \cdot m + b \bmod p],$$

dimana notasi $[X \bmod p]$ mengacu pada pengurangan bilangan bulat X modulo p (dan $[X \bmod p] \in \mathbb{Z}_p$ selalu).

TEOREMA 4.26 Untuk sembarang bilangan prima p, fungsi h sangat universal.

BUKTI Perbaiki $m, m' \in \mathbb{Z}_p$ dan t apa pun, $t' \in \mathbb{Z}_p$. Untuk kunci (a, b) manakah yang dapat menampung $h_{a,b}(m) = t$ dan $h_{a,b}(m') = t'$? Ini hanya berlaku jika

$$a \cdot m + b = t \bmod p \quad \text{and} \quad a \cdot m' + b = t' \bmod p.$$

Jadi, kita mempunyai dua persamaan linier dalam dua persamaan a, b yang tidak diketahui. Kedua persamaan ini dipenuhi tepat ketika $a = [(t + t')\ (m + m')^{-1} \bmod p]$ dan $b = [t + a\ .\ m \bmod p]$; perhatikan bahwa $[(m + m'^{)^{-1}} \bmod p]$ ada karena $m \neq m'$ sehingga $m - m' \neq 0 \mod p$. Dinyatakan kembali, ini berarti bahwa untuk setiap m, m', t, t' seperti di atas terdapat kunci unik (a, b) dengan $h_{a,b}(m) = t$ dan $h_{a,b}(m') = t'$. Karena terdapat kunci $|T|^2$ kita, simpulkan bahwa peluang (terlebih pilihan kunci) bahwa $h_{a,b}(m) = t$ dan $h_{a,b}(m') = t'$ adalah tepat $1/|K| = 1/|T|^2$ sesuai kebutuhan.

Parameter Konstruksi 4.24. Kami membahas secara singkat parameter Konstruksi 4.24 ketika dipakai dengan fungsi sangat universal yang dijelaskan di atas, mengabaikan fakta bahwa p bukan pangkat 2. Konstruksinya adalah MAC $1/|T|$ -aman dengan tag panjang log panjang $|T|$; tag optimal untuk tingkat keamanan yang dicapai.

Misalkan M adalah ruang pesan tetap yang ingin kita buatkan MAC aman satu kali. Konstruksi di atas menghasilkan $1/|M|$ -amankan MAC dengan kunci yang panjangnya dua kali panjang pesan. Pembaca mungkin memperhatikan dua masalah di sini, pada ujung spektrum yang berlawanan: Pertama, jika $|M|$ kecil maka kemungkinan pemalsuan $1/|M|$ mungkin sangat besar. Sebaliknya, jika $|M|$ besar maka $1/|M|$ kemungkinan pemalsuan mungkin berlebihan; seseorang mungkin bersedia menerima kemungkinan pemalsuan (yang agak) lebih besar jika tingkat keamanan tersebut dapat dicapai dengan kunci yang lebih pendek. Masalah pertama (jika $|M|$ kecil) mudah diatasi hanya dengan menyematkan M ke ruang pesan yang lebih besar M' dengan, misalnya, mengisi pesan dengan 0s. Masalah kedua juga bisa diatasi; lihat referensi di akhir bab ini.

Keterbatasan pada MAC Teori Informasi

Pada bagian ini kita mengeksplorasi keterbatasan pada otentikasi pesan teori informasi. Kami menunjukkan bahwa setiap MAC 2^{-n}-aman harus memiliki panjang kunci minimal $2n$. Perpanjangan bukti menunjukkan bahwa MAC aman ℓ-waktu 2^{-n} (di mana keamanan didefinisikan melalui modifikasi alami Definisi 4.23) memerlukan kunci dengan panjang setidaknya $(\ell + 1)\ .\ n$. Konsekuensinya adalah tidak ada MAC dengan kunci dengan panjang terbatas yang dapat memberikan keamanan teori informasi ketika mengautentikasi pesan dalam jumlah tidak terbatas.

Berikut ini, kita asumsikan ruang pesan berisi setidaknya dua pesan; jika tidak, tidak ada gunanya berkomunikasi, apalagi mengautentikasi.

TEOREMA 4.27 Misalkan $\prod$ = (Gen, Mac, Vrfy) adalah MAC 2^{-n}-aman dimana semua kunci keluaran Gen memiliki panjang yang sama. Maka kunci keluaran Gen harus memiliki panjang minimal $2n$.

BUKTI Perbaiki dua pesan berbeda m_0, m_1 di ruang pesan. Intuisi pembuktiannya adalah harus ada setidaknya 2^n kemungkinan untuk tag m_0 (atau musuh dapat menebaknya dengan probabilitas lebih baik daripada 2^{-n}); lebih jauh lagi, bahkan dikondisikan pada nilai tag untuk m_0, harus ada 2^n kemungkinan untuk tag m_1 (atau musuh dapat memalsukan tag pada m_1 dengan probabilitas lebih baik dari 2^{-n}). Karena setiap kunci mendefinisikan tag untuk m_0

dan m_1, ini berarti harus ada setidaknya $2^n \times 2^n$ kunci. Kami menjadikannya formal di bawah ini. Misalkan K menyatakan ruang kunci (yaitu himpunan semua kemungkinan kunci yang dapat dihasilkan oleh Gen). Untuk setiap kemungkinan tag t_0, misalkan $K(t_0)$ menunjukkan himpunan kunci yang mana t_0 merupakan tag valid pada m_0; yaitu.,

$$\mathcal{K}(t_0) \stackrel{\text{def}}{=} \{k \mid \mathsf{Vrfy}_k(m_0, t_0) = 1\}.$$

Untuk $t_0$apa pun kita harus memiliki $|K(t_0)| \leq 2^{-n} \cdot |K|$. Kalau tidak, musuh bisa saja mengeluarkan $m_0, t_0)$ sebagai pemalsuannya; ini akan menjadi pemalsuan yang valid dengan probabilitas setidaknya $|K(t_0)|/|K| > 2^{-n}$, bertentangan dengan keamanan yang diklaim.

Misalkan sekarang musuh $\mathcal{A}$ yang meminta tag pada pesan m_0, menerima balasan tag t_0, memilih kunci seragam $k \in K(t_0)$, dan mengeluarkan output $(m_1, \mathrm{Mac}_k((m_1))$ sebagai pemalsuannya. Kemungkinan menghasilkan pemalsuan yang sah adalah $\mathcal{A}$ paling sedikit

$$\sum_{t_0} \Pr[\mathsf{Mac}_k(m_0) = t_0] \cdot \frac{1}{|\mathcal{K}(m_0, t_0)|} \geq \sum_{t_0} \Pr[\mathsf{Mac}_k(m_0) = t_0] \cdot \frac{2^n}{|\mathcal{K}|}$$
$$= \frac{2^n}{|\mathcal{K}|}.$$

Berdasarkan keamanan skema yang diklaim, probabilitas bahwa musuh dapat menghasilkan pemalsuan yang sah paling banyak adalah 2^{-n}. Jadi, kita harus memiliki $|K|2^{2n}$. Karena semua kunci memiliki panjang yang sama, setiap kunci harus memiliki panjang minimal $2n$.

BAB 5
FUNGSI DAN APLIKASI HASH

Dalam bab ini kami memperkenalkan fungsi hash kriptografi dan menjelajahi beberapa penerapannya. Pada tingkat paling dasar, fungsi hash menyediakan cara untuk memetakan string masukan yang panjang ke string keluaran yang lebih pendek yang terkadang disebut intisari. Persyaratan utamanya adalah menghindari tabrakan, atau dua masukan yang dipetakan ke intisari yang sama. Fungsi hash tahan benturan memiliki banyak kegunaan. Salah satu contoh yang akan kita lihat di sini adalah pendekatan lain—yang distandarisasi sebagai HMAC—untuk mencapai ekstensi domain untuk kode autentikasi pesan.

Selain itu, fungsi hash telah ada di mana-mana dalam kriptografi, dan sering kali digunakan dalam skenario yang memerlukan properti yang jauh lebih kuat daripada ketahanan terhadap tabrakan. Sudah menjadi hal yang umum untuk memodelkan fungsi hash kriptografi sebagai sesuatu yang "benar-benar tidak dapat diprediksi" (alias, ramalan acak), dan kita akan membahas kerangka kerja ini—dan kontroversi yang melingkupinya—secara rinci nanti di bab ini. Kita hanya membahas beberapa penerapan model random-oracle di sini, namun kita akan menemukannya lagi ketika kita beralih ke pengaturan kriptografi kunci publik.

Fungsi hash sangat menarik karena dapat dipandang berada di antara dunia kriptografi kunci privat dan publik. Di satu sisi, seperti yang akan kita lihat di Bab 6, fungsi tersebut (dalam praktiknya) dibuat menggunakan teknik kunci simetris, dan banyak aplikasi kanonik fungsi hash berada dalam pengaturan kunci simetris. Namun dari sudut pandang teoretis, keberadaan fungsi hash yang tahan benturan tampaknya mewakili asumsi yang secara kualitatif lebih kuat dibandingkan keberadaan fungsi pseudorandom (namun asumsi ini lebih lemah dibandingkan keberadaan enkripsi kunci publik).

5.1 DEFINISI

Fungsi hash hanyalah fungsi yang mengambil masukan dengan panjang tertentu dan memampatkannya menjadi keluaran pendek dengan panjang tetap. Penggunaan klasik fungsi hash adalah dalam struktur data, di mana fungsi tersebut dapat digunakan untuk membuat tabel hash yang mengaktifkan $\mathcal{O}(1)$ waktu pencarian saat menyimpan sekumpulan elemen. Khususnya, jika rentang fungsi hash H berukuran N , maka elemen x disimpan di baris $H(x)$ tabel berukuran N . Untuk mengambil x, cukup menghitung $H(x)$ dan menyelidiki baris tabel tersebut untuk elemen yang disimpan di sana. Fungsi hash yang "baik" untuk tujuan ini adalah fungsi yang menghasilkan sedikit tumbukan, dimana tumbukan adalah sepasang item berbeda x dan x' yang mana $H(x) = H(x')$; dalam hal ini kita juga mengatakan bahwa x dan x' bertumbukan. (Saat tabrakan terjadi, dua elemen akhirnya disimpan di sel yang sama, sehingga menambah waktu pencarian.)

Fungsi hash tahan benturan memiliki semangat yang serupa. Sekali lagi, tujuannya adalah untuk menghindari tabrakan. Namun terdapat perbedaan mendasar. Pertama,

keinginan untuk meminimalkan tabrakan dalam pengaturan struktur data menjadi persyaratan untuk menghindari tabrakan dalam pengaturan kriptografi. Lebih jauh lagi, dalam konteks struktur data kita dapat berasumsi bahwa kumpulan elemen data dipilih secara independen dari fungsi hash dan tanpa maksud untuk menyebabkan benturan. Sebaliknya, dalam konteks kriptografi, kita dihadapkan pada musuh yang mungkin memilih elemen dengan tujuan eksplisit untuk menyebabkan tabrakan. Ini berarti fungsi hash yang tahan benturan jauh lebih sulit untuk dirancang.

Ketahanan Tabrakan

Secara informal, suatu fungsi H tahan terhadap tumbukan jika algoritma waktu polinomial probabilistik mana pun tidak dapat menemukan tumbukan dalam H. Kita hanya akan tertarik pada fungsi hash yang domainnya lebih besar dari jangkauannya. Dalam hal ini tabrakan pasti ada, namun tabrakan seperti itu akan sulit ditemukan.

Secara formal, kami mempertimbangkan fungsi hash yang dikunci. Artinya, H adalah fungsi dua masukan yang mengambil kunci s dan string x sebagai masukan, dan menghasilkan string $H^s(x) \stackrel{\text{def}}{=} H(s,x)$. Persyaratannya adalah sulit menemukan tabrakan dalam H^s untuk kunci s yang dihasilkan secara acak. Setidaknya ada dua perbedaan antara kunci dalam konteks ini dan kunci yang kita gunakan hingga saat ini. Pertama, tidak semua string harus sesuai dengan kunci yang valid (yaitu, H^s mungkin tidak didefinisikan untuk s tertentu), dan oleh karena itu kunci s biasanya akan dihasilkan oleh algoritma Gen daripada dipilih secara seragam. Kedua, dan mungkin yang lebih penting, kunci s ini (umumnya) tidak dirahasiakan, dan ketahanan terhadap benturan diperlukan bahkan ketika lawan diberikan s. Untuk menekankan hal ini, kami membuat superskrip pada kunci dan menulis H^s bukan H_s.

DEFINISI 5.1 Fungsi hash (dengan panjang keluaran ℓ) adalah sepasang algoritma waktu polinomial probabilistik (Gen, H) yang memenuhi hal berikut:
Gen adalah algoritma probabilistik yang mengambil input parameter keamanan 1^n dan mengeluarkan kunci s. Kami berasumsi bahwa 1^n implisit dalam s.
H mengambil kunci s dan string $x \in \{0,1\}^*$ sebagai masukan dan mengeluarkan string $H^s(x) \in \{0,1\}^{\ell(n)}$ (di mana n adalah nilai parameter keamanan yang tersirat dalam s).

Jika H^s didefinisikan hanya untuk masukan $x \in \{0,1\}^{\ell'(n)}$ dan $\ell'(n) > \ell(n)$, maka kita katakan bahwa (Gen, H) adalah fungsi hash dengan panjang tetap untuk masukan dengan panjang ℓ' . Dalam hal ini, kami juga menyebut H sebagai fungsi kompresi.

Dalam kasus dengan panjang tetap kita mensyaratkan bahwa ℓ' lebih besar dari ℓ. Ini memastikan bahwa fungsi tersebut memampatkan masukannya. Dalam kasus umum, fungsi tersebut mengambil string masukan dengan panjang sembarang. Jadi, ia juga memampatkan (walaupun hanya string yang panjangnya lebih besar dari $\ell(n)$). Perhatikan bahwa tanpa kompresi, resistensi tumbukan tidak berarti apa-apa (karena kita dapat mengambil fungsi identitas $H^s(x) = x$). Kami sekarang melanjutkan untuk mendefinisikan keamanan. Seperti biasa, pertama-tama kita mendefinisikan eksperimen untuk fungsi hash $\prod$ = (Gen, H), musuh $\mathcal{A}$, dan parameter keamanan n:

Eksperimen penemuan tumbukan Hash-coll$_{\mathcal{A},\prod}(n)$:

1. Kunci dihasilkan dengan menjalankan Gen(1^n).
2. Musuh $\mathcal{A}$ diberikan s dan keluaran x, x'. (Jika $\prod$ adalah fungsi hash dengan panjang tetap untuk input dengan panjang $\ell'(n)$, maka kita memerlukan $x, x' \in \{0,1\}^{\ell'(n)}$.)
3. Keluaran percobaan didefinisikan 1 jika dan hanya jika $x \neq x'$ dan $H^s(x) = H^s(x')$. Dalam kasus $\mathcal{A}$ seperti ini kita katakan telah ditemukan tumbukan.

Definisi ketahanan tumbukan menyatakan bahwa tidak ada musuh yang efisien yang dapat menemukan tumbukan dalam percobaan di atas kecuali dengan probabilitas yang dapat diabaikan.

DEFINISI 5.2 Fungsi hash $\prod$ = (Gen, H) tahan benturan jika untuk semua musuh waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\text{Hash-coll}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

Untuk mempermudah, terkadang kami menyebut H atau H^s sebagai "fungsi hash yang tahan benturan", meskipun secara teknis kami hanya mengatakan bahwa (Gen, H) adalah fungsi hash tersebut. Hal ini seharusnya tidak menimbulkan kebingungan.

Fungsi hash kriptografi dirancang dengan tujuan eksplisit agar tahan terhadap benturan (antara lain). Kita akan membahas beberapa fungsi hash umum di dunia nyata pada Bab 6. Pada Bagian 8.4. kita akan melihat bagaimana mungkin untuk membangun fungsi hash dengan ketahanan tumbukan yang terbukti berdasarkan asumsi tentang kekerasan suatu masalah teori bilangan tertentu.

Fungsi hash yang tidak dikunci. Fungsi hash kriptografi yang digunakan dalam praktik umumnya memiliki panjang keluaran tetap (seperti halnya cipher blok yang memiliki panjang kunci tetap) dan biasanya tidak memiliki kunci, artinya fungsi hash hanyalah fungsi tetap $H : \{0,1\}^* \longrightarrow \{0,1\}^\ell$. Ini bermasalah dari sudut pandang teoritis karena untuk fungsi seperti itu selalu ada algoritma waktu-konstan yang menghasilkan tumbukan dalam H: algoritma ini hanya mengeluarkan pasangan tumbukan (x, x') yang dikodekan ke dalam algoritma itu sendiri. Menggunakan fungsi hash yang dikunci memecahkan masalah teknis ini karena tidak mungkin melakukan hardcode pada pasangan yang bertabrakan untuk setiap kunci yang mungkin menggunakan jumlah ruang yang wajar (dan dalam pengaturan asimtotik, tidak mungkin untuk melakukan hardcode pada pasangan yang bertabrakan untuk setiap nilai parameter keamanan).

Sekalipun demikian, fungsi hash kriptografi (yang tidak dikunci) yang digunakan di dunia nyata tahan terhadap benturan untuk semua tujuan praktis karena pasangan yang bertabrakan tidak diketahui (dan secara komputasi sulit ditemukan) meskipun pasangan tersebut pasti ada. Bukti keamanan untuk beberapa konstruksi berdasarkan resistensi tabrakan dari fungsi hash tetap bermakna bahkan ketika fungsi hash H yang tidak dikunci digunakan, selama bukti tersebut menunjukkan bahwa musuh efisien mana pun yang "mematahkan" primitif dapat digunakan untuk menemukan tabrakan secara efisien. H.

(Semua pembuktian dalam buku ini memenuhi syarat ini.) Dalam hal ini, penafsiran pembuktian keamanan adalah jika dalam praktiknya musuh dapat mematahkan skema tersebut, maka skema tersebut dapat digunakan untuk menemukan tumbukan dalam praktik.

Gagasan Keamanan yang Lebih Lemah

Dalam beberapa aplikasi, cukup mengandalkan persyaratan keamanan yang lebih lemah daripada ketahanan terhadap tabrakan. Ini termasuk:

Resistensi terhadap gambar awal kedua atau tabrakan target: Secara informal, fungsi hash tahan terhadap gambar awal kedua jika diberikan s dan x yang seragam, maka musuh PPT tidak mungkin menemukan $x' \neq x$ sehingga $H^s(x') = H^s(X)$.

Resistensi preimage: Secara informal, fungsi hash tahan terhadap preimage jika diberikan s dan y yang seragam, maka musuh PPT tidak mungkin menemukan nilai x sehingga $H^s(x) = y$. (Melihat ke Bab 7, ini pada dasarnya berarti bahwa H^s adalah satu arah.)

Fungsi hash apa pun yang tahan benturan juga tahan terhadap preimage kedua. Hal ini berlaku karena jika, dengan x yang seragam, suatu musuh dapat menemukan $x' \neq x$ yang mana $H^s(x') = H^s(x)$, maka ia dapat dengan jelas menemukan pasangan tumbukan x dan x'. Demikian pula, fungsi hash apa pun yang tahan terhadap preimage kedua juga tahan terhadap preimage. Hal ini disebabkan oleh fakta bahwa jika diketahui y, kita dapat menemukan x sedemikian rupa sehingga $H^s(x) = y$, maka kita juga dapat mengambil input tertentu x', hitung $y := H^s(x')$, dan lalu peroleh x dengan $H^s(x) = y$. Dengan probabilitas tinggi $x' \neq x$ (mengandalkan fakta bahwa H terkompresi, sehingga beberapa masukan dipetakan ke keluaran yang sama), dalam hal ini gambar awal kedua telah ditemukan.

Kami tidak secara formal mendefinisikan gagasan di atas atau membuktikan implikasi di atas, karena gagasan tersebut tidak digunakan di bagian lain buku ini. Anda diminta untuk meresmikan hal di atas dalam Latihan 5.1.

5.2 EKSTENSI DOMAIN: TRANSFORMASI MERKLE-DAMG'ARD

Fungsi hash sering kali dibuat dengan terlebih dahulu merancang fungsi kompresi tahan benturan yang menangani masukan dengan panjang tetap, dan kemudian menggunakan ekstensi domain untuk menangani masukan dengan panjang sembarang. Pada bagian ini, kami menunjukkan salah satu solusi untuk masalah ekstensi domain. Kita kembali ke pertanyaan merancang fungsi kompresi tahan benturan di Bagian 6.3.

Transformasi Merkle–Damg˚ard adalah pendekatan umum untuk memperluas fungsi kompresi menjadi fungsi hash lengkap, sambil mempertahankan properti ketahanan benturan dari fungsi kompresi sebelumnya. Ini digunakan secara luas dalam praktik untuk fungsi hash termasuk MD5 dan keluarga SHA (lihat Bagian 6.3). Adanya transformasi ini berarti bahwa ketika merancang fungsi hash yang tahan tabrakan, kita dapat membatasi perhatian kita pada kasus dengan panjang tetap. Hal ini, pada gilirannya, membuat pekerjaan merancang fungsi hash tahan benturan menjadi lebih mudah. Transformasi Merkle–Damg˚ard juga menarik dari sudut pandang teoritis karena ini menyiratkan bahwa mengompresi dengan satu bit sama mudahnya (atau sesulit) dengan mengompresi dengan jumlah yang berubah-ubah.

Agar lebih konkret, asumsikan fungsi kompresi (Gen, h) memampatkan masukannya hingga setengahnya; katakanlah panjang masukannya adalah $2n$ dan panjang keluarannya adalah n. (Konstruksinya dapat berjalan berapa pun panjang input/outputnya, selama h dikompres.) Kita membuat fungsi hash tahan benturan (Gen, H) yang memetakan input dengan panjang sembarang ke output dengan panjang n. (Gen tetap tidak berubah.) Transformasi Merkle–Damg˚ard didefinisikan dalam Konstruksi 5.3 dan digambarkan pada Gambar 5.1. Nilai z_0 yang digunakan pada langkah 2 konstruksi, disebut vektor inisialisasi atau IV, bersifat arbitrer dan dapat diganti dengan konstanta apa pun.

CONSTRUCTION 5.3

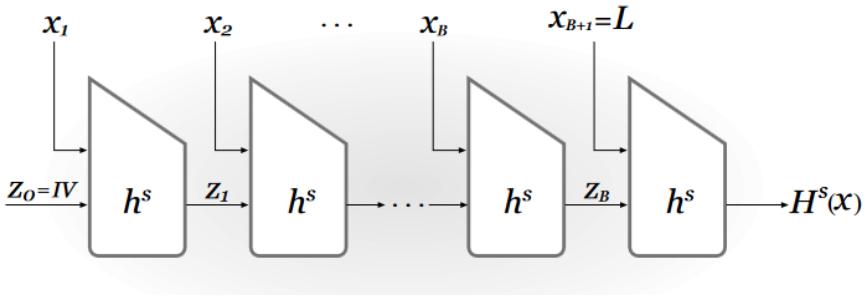
Let (Gen, h) be a fixed-length hash function for inputs of length $2n$ and with output length n. Construct hash function (Gen, H) as follows:

- Gen: remains unchanged.
- H: on input a key s and a string $x \in \{0,1\}^*$ of length $L < 2^n$, do the following:
  1. Set $B := \lceil \frac{L}{n} \rceil$ (i.e., the number of blocks in x). Pad x with zeros so its length is a multiple of n. Parse the padded result as the sequence of n-bit blocks $x_1, \ldots, x_B$. Set $x_{B+1} := L$, where L is encoded as an n-bit string.
  2. Set $z_0 := 0^n$. (This is also called the IV.)
  3. For $i = 1, \ldots, B+1$, compute $z_i := h^s(z_{i-1} \| x_i)$.
  4. Output z_{B+1}.

Transformasi Merkle – Damg˚ard.

TEOREMA 5.4 Jika (Kejadian, h) tahan benturan, maka (Kejadian, H) juga demikian.

BUKTI kita tunjukan beberapa s, bertabrakan di H^s hasil yang bertabrakan di h^s. Biarkan x dan x' menjadi dua perbedaan rangkaian panjang L dan L' maka dari itu, masing-masing $H^s(x) = H^s(x')$. Biarkan $x_1, \ldots, x_B$ menjadi blok B pada x dan



GAMBAR 5.1: Transformasi Merkle–Damgard.

misalkan $x'_1, \ldots, x'_{B'}$ menjadi blok B' dari x' yang empuk. Ingat bahwa $x_{B+1} = L$ dan $x'_{B'+1} = L'$. Ada dua kasus yang perlu dipertimbangkan:

1. Kasus 1: $L \neq L'$. Dalam hal ini langkah terakhir perhitungan $H^s(x)$ adalah $z_{B+1} := h^s(z_B \parallel L)$, dan langkah terakhir perhitungan $H^s(x')$ adalah $z'_{B'+1} := h^s(z'_{B'} \parallel L')$. Karena $H^s(x) = H^s(x')$ maka: $h^s(z_B \parallel L) = h^s(z'_{B'} \parallel L')$ Namun, $L \neq L'$ sehingga $z_B \parallel L$ dan $z'_{B'} \parallel L$ adalah dua string berbeda yang bertabrakan di bawah h^s.
2. Kasus 2: $L = L'$. Artinya $B = B'$. Biarkan $z_0, \ldots, z_{B+1}$ menjadi nilai yang ditentukan selama komputasi $H^s(x)$, let $I_i \stackrel{\text{def}}{=} z_{i-1} \| x_i$ nyatakan input ke-i ke h^s, dan set $I_{B+2} \stackrel{\text{def}}{=} z_{B+1}$. Mendefinisikan $I'_1, \ldots, I'_{B+2}$ analog terhadap x'. Misalkan N adalah indeks terbesar dimana $I_N \neq I'_N$. Karena $x \neq x'$ tetapi $x \neq x'$, terdapat i dengan $x_i \neq x'_i$ sehingga N pasti ada. Karena

$$I_{B+2} = z_{B+1} = H^s(x) = H^s(x') = z'_{B+1} = I'_{B+2},$$

kita mempunyai $N \leq B + 1$. Dengan maksimalitas N, kita mempunyai $I_{N+1} = I'_{N+1}$ dan in z_N tertentu $= z'_N$. Namun ini berarti I_N, I'_N merupakan tumbukan di h^s.

Kami membiarkannya sebagai latihan untuk mengubah hal di atas menjadi pengurangan formal.

5.3 OTENTIKASI PESAN MENGGUNAKAN FUNGSI HASH

Pada bab sebelumnya, kami menyajikan dua konstruksi kode autentikasi pesan untuk pesan dengan panjang sembarang. Pendekatan pertama bersifat umum, namun tidak efisien. Yang kedua, CBC-MAC, didasarkan pada fungsi pseudorandom. Di sini kita akan melihat pendekatan lain, yang kita sebut "hash-and-MAC," yang mengandalkan hashing tahan tabrakan bersama dengan kode otentikasi pesan apa pun. Kami kemudian membahas konstruksi standar dan banyak digunakan yang disebut HMAC yang dapat dipandang sebagai contoh spesifik dari pendekatan ini.

Hash-dan-MAC

Ide di balik pendekatan hash-dan-MAC sederhana saja. Pertama, pesan m yang panjangnya sembarang di-hash menjadi string dengan panjang tetap $H^s(m)$ menggunakan fungsi hash tahan benturan. Kemudian, MAC (panjang tetap) diterapkan pada hasilnya. Lihat Konstruksi 5.5 untuk penjelasan formal.

CONSTRUCTION 5.5

Let $\Pi = (\mathsf{Mac}, \mathsf{Vrfy})$ be a MAC for messages of length $\ell(n)$, and let $\Pi_H = (\mathsf{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Construct a MAC $\Pi' = (\mathsf{Gen}', \mathsf{Mac}', \mathsf{Vrfy}')$ for arbitrary-length messages as follows:

- Gen': on input 1^n, choose uniform $k \in \{0,1\}^n$ and run $\mathsf{Gen}_H(1^n)$ to obtain s; the key is $k' := \langle k, s \rangle$.
- Mac': on input a key $\langle k, s \rangle$ and a message $m \in \{0,1\}^*$, output $t \leftarrow \mathsf{Mac}_k(H^s(m))$.
- Vrfy': on input a key $\langle k, s \rangle$, a message $m \in \{0,1\}^*$, and a MAC tag t, output 1 if and only if $\mathsf{Vrfy}_k(H^s(m), t) \stackrel{?}{=} 1$.

Konstruksi 5.5 aman jika $\prod$ adalah MAC aman untuk pesan dengan panjang tetap dan (Gen, H) tahan benturan. Secara intuitif, karena fungsi hash tahan benturan, mengautentikasi $H^s(m)$ sama baiknya dengan mengautentikasi m itu sendiri: jika pengirim dapat memastikan bahwa penerima memperoleh nilai $H^s(m)$ yang benar, ketahanan benturan menjamin bahwa penyerang tidak dapat menemukan pesan berbeda m' yang di-hash dengan nilai yang sama. Secara lebih formal, katakanlah pengirim menggunakan Konstruksi 5.5 untuk mengautentikasi beberapa kumpulan pesan Q, dan penyerang kemudian $\mathcal{A}$ dapat memalsukan tag yang valid pada pesan baru $m^* \notin Q$. Ada dua kemungkinan kasus:
Kasus 1: terdapat pesan $Q \in m$ sehingga $H^s(m^*) = H^s(m)$. Kemudian ditemukan tumbukan pada H^s, bertentangan dengan ketahanan tumbukan (Gen, H).
Kasus 2: untuk setiap pesan $m \in Q$ dinyatakan bahwa $H^s(m^*) \neq H^s(m)$. Misalkan $H^s(Q) \stackrel{\text{def}}{=} \{ H^s(m) \mid m \in Q\}$. Maka $H^s(m^*) \notin H^s(Q)$. Dalam hal ini, telah memalsukan tag yang valid pada "pesan baru" $H^s(m^*)$ sehubungan dengan kode otentikasi pesan dengan panjang teta$\prod$p. Hal ini bertentangan dengan asumsi bahwa $\prod$ adalah MAC yang aman.
Kami sekarang mengubah hal di atas menjadi bukti formal.

TEOREMA 5.6 Jika $\prod$ adalah MAC aman untuk pesan dengan panjang ℓ dan $\prod_H$ tahan benturan, maka Konstruksi 5.5 adalah MAC aman (untuk pesan dengan panjang sembarang).

BUKTI Misalkan $\prod'$ menyatakan Konstruksi 5.5, dan misalkan $\mathcal{A}'$ menjadi musuh PPT yang menyerang $\prod'$. Dalam pelaksanaan percobaan Mac-forge$_{\mathcal{A}',\prod'}(n)$, misalkan $k' = \langle k, s\rangle$ menunjukkan kunci MAC, misalkan Q menunjukkan kumpulan pesan yang tagnya diminta oleh $\mathcal{A}'$, dan misalkan (m^*, t) menjadi keluaran akhir dari $\mathcal{A}'$. Kita asumsikan tanpa kehilangan keumuman bahwa $m^* \notin Q$. Definisikan coll sebagai kejadian di mana, dalam percobaan Mac-forge$_{\mathcal{A}',\prod'}(n)$, terdapat $m \in Q$ yang mana $H^s(m^*) = H^s(m)$. Kita punya

$$\begin{aligned}
&\Pr[\text{Mac-forge}_{\mathcal{A}',\Pi'}(n) = 1] \\
&= \Pr[\text{Mac-forge}_{\mathcal{A}',\Pi'}(n) = 1 \wedge \text{coll}] + \Pr[\text{Mac-forge}_{\mathcal{A}',\Pi'}(n) = 1 \wedge \overline{\text{coll}}] \\
&\leq \Pr[\text{coll}] + \Pr[\text{Mac-forge}_{\mathcal{A}',\Pi'}(n) = 1 \wedge \overline{\text{coll}}]. \qquad (5.1)
\end{aligned}$$

Kita tunjukkan bahwa kedua suku pada Persamaan (5.1) dapat diabaikan, sehingga melengkapi pembuktian. Secara intuitif, suku pertama dapat diabaikan karena ketahanan tumbukan sebesar $\prod_H$, dan suku kedua dapat diabaikan karena keamanan sebesar $\prod$.

Perhatikan algoritma C berikut untuk mencari tumbukan pada $\prod_H$:
Algoritma :
Algoritme diberikan s sebagai masukan (dengan n implisit).

- Pilih seragam $k \in \{0,1\}^n$.
- Jalankan $\mathcal{A}'(1^n)$. Ketika $\mathcal{A}'$ meminta tag pada pesan ke-i

$m_i \in \{0,1\}^*$, hitung $t_i \leftarrow \text{Mac}_k(H^s(m_i))$ dan berikan t_i ke $\mathcal{A}'$.

Ketika ’ menghasilkan (m^*, t), maka jika terdapat i yang mana $H^s(m^*) = H^s(m_i)$, keluaran (m^*, m_i).

Jelas bahwa C berjalan dalam waktu polinomial. Mari kita analisis perilakunya. Ketika masukan ke C dihasilkan dengan menjalankan $\text{Gen}_H(1^n)$ untuk mendapatkan s, tampilan $\mathcal{A}'$ ketika dijalankan sebagai subrutin oleh C didistribusikan secara identik dengan tampilan $\mathcal{A}'$ dalam eksperimen $\text{Mac-forge}_{\mathcal{A}',\Pi'}(n)$. Secara khusus, tag yang diberikan kepada $\mathcal{A}'$ oleh C memiliki distribusi yang sama dengan tag yang diterima $\mathcal{A}'$ di $\text{Mac-forge}_{\mathcal{A}',\Pi'}(n)$. Karena C mengeluarkan tabrakan tepat ketika coll terjadi, kita punya

$$\Pr[\text{Hash-coll}_{C,\Pi_H}(n) = 1] = \Pr[\text{coll}].$$

Karena ΠH tahan tumbukan, kami menyimpulkan bahwa Pr[coll] dapat diabaikan.
Kita sekarang melanjutkan untuk membuktikan bahwa suku kedua pada Persamaan (5.1) dapat diabaikan. Pertimbangkan musuh berikut $\mathcal{A}$ yang menyerang $\prod$ di $\text{Mac-forge}_{\mathcal{A}',\Pi'}(n)$:
Musuh $\mathcal{A}$:
Musuh diberikan akses ke oracle MAC $\text{Mac}_k(\cdot)$.

- Hitung $\text{Gen}_H(1^n)$ untuk memperoleh s.
- Jalankan $\mathcal{A}(1^n)$. Ketika $\mathcal{A}'$ meminta tag pada pesan ke-i $m_i \in \{0,1\}^*$, maka: (1) hitung $\hat{m}_i := H^s(m_i)$; (2) mendapatkan tag ti pada $\hat{m}_i$ dari oracle MAC; dan (3) berikan t_i pada $\mathcal{A}'$.
- Bila $\mathcal{A}'$ mengeluarkan (m^*, t), maka keluarkan $(H^s(m^*), t)$.

Jelas $\mathcal{A}$ berjalan dalam waktu polinomial. Pertimbangkan eksperimen $\text{Mac-forge}_{\mathcal{A},\Pi}(n)$. Dalam eksperimen tersebut, tampilan $\mathcal{A}'$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}$ didistribusikan secara identik dengan tampilannya dalam eksperimen $\text{Mac-forge}_{\mathcal{A}',\Pi'}(n)$. Selanjutnya, setiap kali $\text{Mac-forge}_{\mathcal{A}',\Pi'}(n) = 1$ dan coll tidak terjadi, akan menghasilkan pemalsuan $\mathcal{A}$ yang valid. (Dalam hal ini t adalah tag yang valid pada $H^s(m^*)$ dalam skema $\prod$ sehubungan dengan k. Fakta bahwa coll tidak terjadi berarti bahwa $H^s(m^*)$ tidak pernah ditanyakan oleh $\mathcal{A}$ ke oracle MAC-nya sendiri sehingga ini memang palsu.) Oleh karena itu,

$$\Pr[\text{Mac-forge}_{\mathcal{A},\Pi}(n) = 1] = \Pr[\text{Mac-forge}_{\mathcal{A}',\Pi'}(n) \wedge \overline{\text{coll}}],$$

dan keamanan $\prod$ menyiratkan bahwa probabilitas sebelumnya dapat diabaikan. Ini menyimpulkan pembuktian teorema tersebut.

HMAC

Semua konstruksi kode otentikasi pesan yang telah kita lihat sejauh ini pada akhirnya didasarkan pada beberapa blok sandi. Apakah mungkin untuk membuat MAC yang aman (untuk pesan dengan panjang sewenang-wenang) berdasarkan langsung pada fungsi hash? Pikiran pertama mungkin mendefinisikan $\text{Mac}_k(m) = H(k \| m)$; kita mungkin berharap bahwa jika H adalah fungsi hash yang “baik” maka akan sulit bagi penyerang untuk

memprediksi nilai $H(k \parallel m')$ mengingat nilai $H(k \parallel m)$, untuk setiap $m' \neq m$, dengan asumsi k dipilih secara acak (dan tidak diketahui penyerang). Sayangnya, jika H dibangun menggunakan transformasi Merkle–Damg˚ard—seperti kebanyakan fungsi hash di dunia nyata—maka MAC yang dirancang dengan cara ini sama sekali tidak aman, seperti yang diminta untuk Anda tunjukkan pada Latihan 5.10.

Sebagai gantinya, kita bisa mencoba menggunakan dua lapisan hashing. Lihat Konstruksi 5.7 untuk skema standar yang disebut HMAC berdasarkan ide ini.

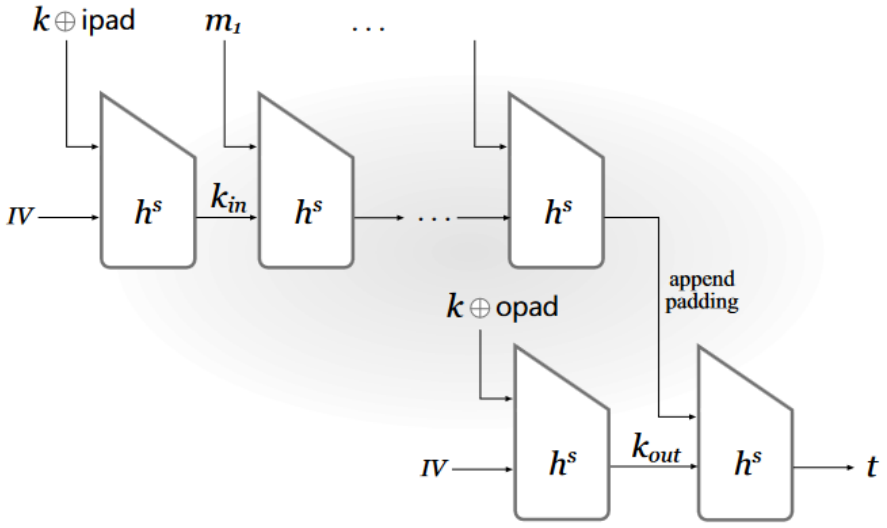
CONSTRUCTION 5.7

Let (Gen_H, H) be a hash function constructed by applying the Merkle–Damgård transform to a compression function (Gen_H, h) taking inputs of length $n + n'$. (See text.) Let opad and ipad be fixed constants of length n'. Define a MAC as follows:

- Gen: on input 1^n, run $\mathsf{Gen}_H(1^n)$ to obtain a key s. Also choose uniform $k \in \{0,1\}^{n'}$. Output the key $\langle s, k \rangle$.
- Mac: on input a key $\langle s, k \rangle$ and a message $m \in \{0,1\}^*$, output

$$t := H^s\Big((k \oplus \mathsf{opad}) \parallel H^s\big((k \oplus \mathsf{ipad}) \parallel m\big)\Big).$$

- Vrfy: on input a key $\langle s, k \rangle$, a message $m \in \{0,1\}^*$, and a tag t, output 1 if and only if $t \stackrel{?}{=} H^s\big((k \oplus \mathsf{opad}) \parallel H^s\big((k \oplus \mathsf{ipad}) \parallel m\big)\big)$.



GAMBAR 5.2: HMAC, secara bergambar.

Mengapa kita harus yakin bahwa HMAC aman? Salah satu alasannya adalah kita dapat melihat HMAC sebagai contoh spesifik dari paradigma hash-dan-MAC dari bagian sebelumnya. Untuk melihat hal ini, kita akan melihat "di balik terpal" apa yang terjadi ketika sebuah pesan diautentikasi; lihat Gambar 5.2. Kita juga harus menentukan parameter dengan lebih hati-hati dan menjelaskan lebih detail mengenai cara penerapan transformasi Merkle–Damg˚ard dalam praktiknya.

Katakanlah $(\text{Gen}_{H,}H)$ dibangun berdasarkan fungsi kompresi $(\text{Gen}_{H,}h)$ di mana h memetakan masukan dengan panjang $n + n'$ ke keluaran dengan panjang n (di mana, secara formal, n' adalah fungsi dari n). Saat kami mendeskripsikan transformasi Merkle–Damg˚ard di Bagian 5.2, kami berasumsi $n' = n$, namun hal tersebut tidak selalu terjadi. Kami juga mengatakan bahwa panjang pesan yang di-hash dikodekan sebagai blok pesan tambahan yang ditambahkan ke pesan tersebut. Dalam praktiknya, panjangnya dikodekan dalam sebagian blok menggunakan $\ell < n'$ bit. Artinya, perhitungan $H^s(x)$ dimulai dengan mengisi x dengan nol pada string yang panjangnya tepat ℓ kurang dari kelipatan n'; kemudian menambahkan panjangnya $L = |x|$, dikodekan menggunakan persis ℓ bit. Hash dari urutan yang dihasilkan dari blok n'-bit $x_1, \ldots$ kemudian dihitung seperti pada Konstruksi 5.3. Kita asumsikan bahwa $n + \ell \leq n'$. Artinya, khususnya, jika kita melakukan hash pada masukan x dengan panjang $n' + n$ maka hasil yang diisi (termasuk panjangnya) akan memiliki panjang tepat $2n'$ bit. Pembuktian Teorema 5.4 yang menunjukkan bahwa $(\text{Gen}_{H,}H)$ tahan tumbukan jika $(\text{Gen}_{H,}h)$ tahan tumbukan, tetap tidak berubah.

Kembali ke HMAC, dan melihat Gambar 5.2, kita dapat melihat bahwa bentuk umum HMAC melibatkan hashing pesan dengan panjang sembarang menjadi string pendek $y \stackrel{\text{def}}{=} H^s(\,(k \oplus \text{ipad}) \,\|\, m)$, dan kemudian menghitung (diam-diam keyed) fungsi $H^s(\,(k \oplus \text{opad}) \,\|\, y)$ dari hasilnya. Tapi kita bisa mengatakan lebih dari ini. Perhatikan dulu bahwa perhitungan "batin".

$$\tilde{H}^s(m) \stackrel{\text{def}}{=} H^s(\,(k \oplus \text{ipad}) \,\|\, m)$$

tahan benturan (dengan asumsi h), untuk nilai k $\oplus$ ipad apa pun. Selain itu, langkah pertama dalam perhitungan "luar" $H^s(\,(k \oplus \text{opad}) \,\|\, y)$ adalah menghitung nilai $k_{out} \stackrel{\text{def}}{=} h^s(IV \,\|\, k \oplus \text{opad}))$. Kemudian, kita mengevaluasi $h^s(k_{out} \,\|\, \hat{y}\,)$ di mana $\hat{y}$ mengacu pada nilai empuk dari y (yaitu, termasuk panjang $(k \oplus \text{opad}) \,\|\, y$, yang selalu $n' + n$ bit, dikodekan menggunakan tepat ℓ bit). Jadi, jika kita memperlakukan k_{out} sebagai seragam—kita akan menjelaskannya secara lebih formal di bawah—dan berasumsi demikian

$$\widetilde{\text{Mac}}_k(y) \stackrel{\text{def}}{=} h^s(k \,\|\, \hat{y}) \qquad (5.2)$$

adalah MAC dengan panjang tetap yang aman, maka HMAC dapat dilihat sebagai contoh pendekatan hash-dan-MAC dengan

$$\text{HMAC}_{s,k}(m) = \widetilde{\text{Mac}}_{k_{out}}(\tilde{H}^s(m)) \qquad (5.3)$$

(dimana $k_{out} = h^s(IV \,\|\, (k \oplus \text{opad}))$). Karena cara fungsi kompresi h dirancang secara khusus (lihat Bagian 6.3.1), asumsi bahwa $\widetilde{Mac}$ adalah MAC dengan panjang tetap yang aman adalah masuk akal.

Peran ipad dan opad. Mengingat hal di atas, orang mungkin bertanya-tanya mengapa k perlu dimasukkan dalam perhitungan "dalam" $H^s(\,(k \oplus \text{ipad}) \,\|\, m)$. (Khususnya, agar pendekatan hash-dan-MAC aman, kami memerlukan ketahanan tabrakan pada langkah pertama, yang tidak memerlukan kunci rahasia apa pun.) Alasannya adalah hal ini memungkinkan keamanan HMAC didasarkan pada potensi asumsi yang lebih lemah bahwa $(\text{Gen}_{H,}H)$ memiliki ketahanan tumbukan yang lemah, dengan ketahanan tumbukan yang lemah ditentukan oleh eksperimen berikut: kunci s dihasilkan menggunakan Gen_H dan kerabat rahasia yang seragam $k_{in} \in \{0,1\}^n$ dipilih. Kemudian musuh diizinkan untuk berinteraksi dengan "hash oracle" yang mengembalikan $H^s_{kin}(m)$ sebagai respons terhadap kueri m, di mana H^s_{kin} mengacu pada penghitungan H^s menggunakan transformasi Merkle–Damg˚ard yang diterapkan pada h^s, tetapi menggunakan nilai k_{in} rahasia saudara sebagai IV. (Lihat lagi Gambar 5.2.) Musuh berhasil jika dapat menghasilkan nilai yang berbeda m, m' sehingga $H^s_{kin}(m) = H^s_{kin}(m')$, dan kita katakan bahwa $(\text{Gen}_{H,}H)$ memiliki ketahanan tumbukan yang lemah jika setiap PPT $\mathcal{A}$ berhasil dalam eksperimen ini dengan probabilitas yang dapat diabaikan. Jika $(\text{Gen}_{H,}H)$ tahan benturan maka ketahanan benturannya jelas lemah; namun kondisi terakhir ini merupakan kondisi yang lebih lemah dan berpotensi lebih mudah untuk dipenuhi. Ini adalah contoh bagus dari rekayasa keamanan yang baik. Strategi desain defensif ini membuahkan hasil ketika ditemukan bahwa fungsi hash MD5 tidak tahan benturan. Serangan pencarian tabrakan pada MD5 tidak melanggar resistensi tabrakan yang lemah, dan HMAC-MD5 tidak rusak meskipun MD5 melanggarnya. Hal ini memberikan waktu bagi pengembang untuk mengganti MD5 dalam implementasi HMAC, tanpa rasa takut akan serangan. (Meskipun demikian, HMAC-MD5 seharusnya tidak lagi digunakan karena kelemahan MD5 sudah diketahui.)

Pembahasan di atas menyarankan bahwa kunci independen harus digunakan dalam perhitungan luar dan dalam. Untuk alasan efisiensi, satu kunci k digunakan untuk HMAC, namun kunci tersebut digunakan bersama dengan ipad dan opad untuk mendapatkan dua kunci lainnya. Mendefinisikan

$$G^s(k) \stackrel{\text{def}}{=} h^s\left(IV \| (k \oplus \text{opad})\right) \| h^s\left(IV \| (k \oplus \text{ipad})\right) = k_{out} \| k_{in}. \qquad (5.4)$$

Jika kita berasumsi bahwa G^s adalah generator pseudorandom untuk sembarang s, maka kout dan kin dapat dianggap sebagai kunci independen dan seragam ketika k seragam. Keamanan HMAC kemudian dikurangi menjadi keamanan konstruksi berikut:

$$\text{Mac}_{s,k_{in},k_{out}}(m) = h^s\left(k_{out} \,\|\, H^s_{k_{in}}(m)\right).$$

(Bandingkan dengan Persamaan (5.3).) Seperti disebutkan sebelumnya, konstruksi ini dapat dibuktikan aman (menggunakan varian pembuktian untuk pendekatan hash-dan-MAC) jika

H memiliki ketahanan benturan yang lemah dan MAC ditentukan dalam Persamaan (5.2) adalah MAC dengan panjang tetap yang aman.

TEOREMA 5.8 Asumsikan G^s seperti yang didefinisikan dalam Persamaan (5.4) adalah generator pseudo-acak untuk setiap s, MAC yang didefinisikan dalam Persamaan (5.2) adalah MAC dengan panjang tetap yang aman untuk pesan dengan panjang n, dan$(\text{Ge}n_{H,}H)$ lemah tahan benturan. Maka HMAC adalah MAC yang aman (untuk pesan dengan panjang sewenang-wenang).

HMAC dalam praktiknya. HMAC adalah standar industri dan banyak digunakan dalam praktik. Ini sangat efisien dan mudah diterapkan, serta didukung oleh bukti keamanan berdasarkan asumsi yang diyakini berlaku untuk fungsi hash praktis. Pentingnya HMAC sebagian disebabkan oleh ketepatan waktu kemunculannya. Sebelum HMAC diperkenalkan, banyak praktisi menolak menggunakan CBC-MAC (dengan klaim bahwa itu "terlalu lambat") dan malah menggunakan konstruksi heuristik yang tidak aman. HMAC menyediakan cara yang terstandarisasi dan aman dalam melakukan otentikasi pesan berdasarkan fungsi hash.

5.4 SERANGAN GENERIK PADA FUNGSI HASH

Keamanan apa yang terbaik yang bisa kita harapkan untuk disediakan oleh fungsi hash H? Kami mengeksplorasi pertanyaan ini dengan menunjukkan dua serangan yang bersifat umum dalam arti bahwa serangan tersebut diterapkan pada fungsi hash yang berubah-ubah. Adanya serangan-serangan ini menyiratkan batasan yang lebih rendah pada panjang keluaran H yang diperlukan untuk mencapai tingkat keamanan yang diinginkan, dan oleh karena itu mempunyai konsekuensi praktis yang penting.

Serangan Ulang Tahun untuk Menemukan Tabrakan

Misalkan $H : \{0,1\}^* \rightarrow \{0,1\}^\ell$ adalah fungsi hash. (Di sini dan di sisa bab ini, kami tidak menyebutkan secara eksplisit kunci hash s karena tidak relevan secara langsung. Kita juga dapat melihat s dihasilkan dan diperbaiki sebelum algoritme ini diterapkan.) Terdapat temuan tabrakan yang sepele serangan berjalan tepat waktu $\mathcal{O}(2^\ell)$: cukup evaluasi H pada $2^\ell + 1$ input berbeda; berdasarkan prinsip merpati, dua keluaran harus sama. Bisakah kita berbuat lebih baik?

Menggeneralisasi algoritma di atas, katakanlah kita memilih q input berbeda $x_1, \ldots, x_q$, hitung $y_i := H(x_i)$, dan periksa apakah salah satu dari dua nilai y_i sama. Berapa probabilitas algoritma ini menemukan tabrakan? Seperti yang baru saja kita katakan, jika $q > 2^\ell$ maka tumbukan terjadi dengan peluang 1. Berapa peluang tumbukan jika q lebih kecil? Agak sulit untuk menganalisis probabilitas ini secara tepat, sehingga kita akan menganalisis kasus ideal di mana H diperlakukan sebagai fungsi acak.[1] Artinya, untuk setiap i kita asumsikan bahwa nilai $y_i := H(x_i)$ terdistribusi secara merata di $\{0,1\}^\ell$ dan tidak bergantung pada nilai keluaran sebelumnya $\{y_j\}_{j<i}$ (ingat kita berasumsi semua $\{x_i\}$ berbeda). Oleh karena itu, kita telah mengurangi masalah kita menjadi masalah berikut: jika kita memilih nilai $y_1, \ldots, y_q \in \{0,1\}^\ell$ acak seragam, berapa peluang terdapat i, j yang berbeda dengan $y_i = y_j$?

Masalah ini telah dipelajari secara ekstensif, dan berkaitan dengan apa yang disebut masalah ulang tahun. Oleh karena itu, algoritma pencarian tabrakan yang telah kami jelaskan sering disebut serangan ulang tahun. Soal ulang tahunnya adalah sebagai berikut: jika ada q orang dalam satu ruangan, berapa peluang terambilnya dua orang yang mempunyai hari ulang tahun yang sama? (Asumsikan tanggal lahir terdistribusi secara seragam dan independen di antara 365 hari pada tahun non-kabisat.) Hal ini analog dengan permasalahan kita: jika y_i mewakili tanggal lahir orang i, maka kita mempunyai $y_1, \ldots, y_q \in \{1, \ldots, 365\}$ dipilih secara seragam, dan tanggal lahir yang cocok berhubungan dengan i, j yang berbeda dengan $y_i = y_j$ (yaitu, ulang tahun yang cocok berhubungan dengan tumbukan).

Kami menunjukkan bahwa untuk $y_1, \ldots, y_q$ dipilih secara seragam dalam $\{1, \ldots, N\}$, peluang terjadinya tumbukan kira-kira $1/2$ jika $q = \Theta(N^{1/2})$. Dalam hal ulang tahun, jika hanya terdapat 23 orang, probabilitas bahwa dua orang di antara mereka mempunyai hari ulang tahun yang sama adalah lebih besar dari $1/2$. Dalam pengaturan kita, ini berarti bahwa ketika fungsi hash memiliki panjang keluaran ℓ (sehingga rentangnya berukuran 2^ℓ), maka pengambilan $q = \Theta(2^{\ell/2})$ menghasilkan tabrakan dengan probabilitas sekitar $1/2$.

Dari perspektif keamanan konkrit, hal di atas berarti bahwa agar fungsi hash dapat menahan serangan pencarian tabrakan yang berjalan dalam waktu T (di mana kita meluangkan waktu untuk mengevaluasi H sebagai satuan waktu), panjang keluaran hash fungsinya harus minimal 2 bit $\log T$ (karena $2^{(2\log T)/2} = T$). Mengambil parameter tertentu, ini berarti bahwa jika kita ingin menemukan tabrakan sama sulitnya dengan pencarian menyeluruh pada kunci 128-bit, maka kita memerlukan panjang keluaran fungsi hash minimal 256 bit. Kami menekankan bahwa memiliki output selama ini hanyalah kondisi yang diperlukan, bukan kondisi yang cukup. Kami juga mencatat bahwa serangan ulang tahun hanya berfungsi untuk menemukan tabrakan. Tidak ada serangan umum untuk resistensi preimage kedua atau resistensi preimage dari fungsi hash H yang memerlukan evaluasi H kurang dari 2^ℓ (meskipun lihat Bagian 5.4).

Menemukan tabrakan yang bermakna. Serangan ulang tahun yang baru saja dijelaskan memberikan tabrakan yang belum tentu terlalu berguna. Namun gagasan yang sama juga dapat digunakan untuk menemukan tabrakan yang "bermakna". Asumsikan Alice ingin menemukan dua pesan x dan x' sehingga $H(x) = H(x')$, dan lebih jauh lagi x harus berupa surat dari majikannya yang menjelaskan mengapa dia dipecat dari pekerjaannya, sedangkan x' harus berupa surat yang menyanjung rekomendasi. (Hal ini memungkinkan Alice untuk memalsukan tag yang sesuai pada surat rekomendasi jika pendekatan hash-dan-MAC digunakan oleh perusahaannya untuk mengautentikasi pesan.) Pengamatannya adalah bahwa serangan ulang tahun hanya memerlukan input hash $x_1, \ldots, x_q$ untuk membedakan; mereka tidak perlu acak. Alice dapat melakukan serangan tipe ulang tahun dengan menghasilkan pesan $q = \Theta(2^{\ell/2})$ dari tipe pertama dan pesan q dari tipe kedua, dan kemudian mencari tabrakan antara pesan dari kedua tipe tersebut. Perubahan kecil pada analisis dari Lampiran A.4 menunjukkan bahwa hal ini memberikan tabrakan antara pesan-pesan dari jenis yang berbeda dengan probabilitas sekitar 1/2. Sedikit pemikiran menunjukkan bahwa mudah untuk menulis pesan yang sama dengan berbagai cara. Misalnya, pertimbangkan hal berikut:

Sulit/sulit/menantang/tidak mungkin membayangkan/percaya bahwa kita akan menemukan/mencari/mempekerjakan karyawan/orang lain yang memiliki kemampuan/keterampilan/karakter serupa dengan Alice. Dia telah melakukan pekerjaan yang hebat/super.

Kombinasi apa pun dari kata-kata yang dicetak miring dimungkinkan, dan mengungkapkan maksud yang sama. Jadi, kalimat tersebut dapat ditulis dengan $4^{23}.2^{32} = 2^{88}$ cara yang berbeda. Ini hanya satu kalimat sehingga mudah untuk menghasilkan pesan yang dapat ditulis ulang dalam 2^{64} cara berbeda—yang diperlukan hanyalah 64 kata dengan masing-masing satu sinonim. Alice dapat menyiapkan surat $2^{\ell/2}$ yang menjelaskan mengapa dia dipecat dan surat rekomendasi $2^{\ell/2}$ lainnya; dengan probabilitas yang baik akan ditemukan benturan antara kedua jenis huruf tersebut.

Serangan Ulang Tahun di Ruang Kecil

Serangan ulang tahun yang dijelaskan di atas memerlukan sejumlah besar memori; khususnya, mereka mengharuskan penyerang untuk menyimpan semua nilai $\mathcal{O}(q) = \mathcal{O}(2^{\ell/2})$ $\{y_i\}$, karena penyerang tidak mengetahui sebelumnya pasangan nilai mana yang akan menghasilkan tabrakan. Ini merupakan kelemahan yang signifikan karena memori, secara umum, merupakan sumber daya yang lebih langka dibandingkan waktu. Bisa dibilang lebih sulit mengalokasikan dan mengelola penyimpanan sebesar 2^{60} byte daripada menjalankan instruksi CPU 2^{60}. Lebih jauh lagi, kita selalu dapat membiarkan komputasi berjalan tanpa batas waktu, sedangkan kebutuhan memori suatu algoritma harus dipenuhi segera setelah jumlah memori tersebut dibutuhkan.

Di sini kami tunjukkan serangan ulang tahun yang lebih baik dengan kebutuhan memori yang berkurang secara drastis. Faktanya, ia memiliki kompleksitas waktu dan probabilitas keberhasilan yang sama seperti sebelumnya, namun hanya menggunakan memori konstan. Serangan dimulai dengan memilih nilai acak x_0 dan kemudian menghitung $x_i := H(x_{i-1})$ dan $x_{2i} := H(H(x_{2(i-1)}))$ untuk $i = 1, 2,$ (Perhatikan bahwa $x_i = H^{(i)}(x_0)$ untuk semua i, dimana $H^{(i)}$ mengacu pada iterasi kali lipat dari H.) Pada setiap langkah nilai x_i dan x_{2i-1} dibandingkan; jika keduanya sama maka terjadi tumbukan di suatu tempat pada barisan $x_0, x_1, \dots, x_{2i-1}$.

Algoritme kemudian menemukan nilai terkecil dari j yang $x_j = x_{j+i}$ (perhatikan bahwa $j \leq i$ karena $j = i$ berfungsi), dan menghasilkan x_{j-1}, x_{j+i-1} sebagai tumbukan. Serangan ini, yang dijelaskan secara formal sebagai Algoritma 5.9 dan dianalisis di bawah, hanya memerlukan penyimpanan dua nilai hash di setiap iterasi.

ALGORITHM 5.9
A small-space birthday attack

Input: A hash function $H : \{0,1\}^* \to \{0,1\}^\ell$
Output: Distinct x, x' with $H(x) = H(x')$

$x_0 \leftarrow \{0,1\}^{\ell+1}$
$x' := x := x_0$
for $i = 1, 2, \ldots$ **do:**
  $x := H(x)$
  $x' := H(H(x'))$
  // now $x = H^{(i)}(x_0)$ and $x' = H^{(2i)}(x_0)$
  if $x = x'$ **break**
$x' := x,\ x := x_0$
for $j = 1$ to i:
  if $H(x) = H(x')$ **return** x, x' and **halt**
  else $x := H(x),\ x' := H(x')$
  // now $x = H^{(j)}(x_0)$ and $x' = H^{(i+j)}(x_0)$

Berapa banyak iterasi loop pertama yang kita harapkan sebelum $x' = x$? Perhatikan barisan nilai $x_1, x_2, \ldots$, dimana $x_i = H^{(i)}(x_0)$ sebagaimana didefinisikan sebelumnya. Jika kita memodelkan H sebagai fungsi acak, maka masing-masing nilai ini terdistribusi secara seragam dan independen dalam 0,1 A hingga pengulangan pertama terjadi. Jadi, kita perkirakan akan terjadi pengulangan dengan probabilitas 1/2 pada suku $q = \Theta(2^{\ell/2})$ pertama dari barisan tersebut. Kami menunjukkan bahwa ketika terdapat pengulangan pada elemen q pertama, algoritme menemukan pengulangan paling banyak pada q iterasi pada loop pertama:

KLAIM 5.10 Misalkan $x_1, \ldots, x_q$ adalah barisan nilai dengan $x_m = H(x_{m-1})$.
Jika $x_I = x_J$ dengan $1 \le I < J \le q$, maka terdapat $i < J$ sehingga $x_i = x_{2i}$.

BUKTI Barisan $x_I, x_{I+1}, \ldots$ berulang dengan periode $\Delta \stackrel{\text{def}}{=} J - I$. Artinya, untuk semua $i \ge \ell$ dan $k \ge 0$ dinyatakan bahwa $x_i = x_{i+k.\Delta}$. Misal i adalah kelipatan terkecil dari Δ yang juga lebih besar atau sama dengan I. Kita mempunyai $i < J$ karena barisan nilai $\Delta\, I, I + 1, \ldots I + (\Delta - 1) = J - 1$ berisi kelipatan Δ. Karena $i \ge I$ dan $2i - i = i$ adalah kelipatan dari Δ, maka $x_i = x_{2i}$.
Jadi, jika ada nilai berulang pada barisan $x_1, \ldots, x_q$, maka ada beberapa $i < q$ yang $x_i = x_{2i}$. Namun kemudian pada iterasi i dari algoritma kita, kita mendapatkan $x = x'$ dan algoritma keluar dari loop pertama. Pada titik tersebut dalam algoritma, kita mengetahui bahwa $x_i = x_{i+i}$. Algoritme kemudian menetapkan $x_i := x\ (= x_i)$. dan $x := x_0$, dan melanjutkan untuk mencari $j \ge 0$ terkecil yang $x_j = x_{j+i}$. (Catatan $j \ne 0$ karena $|x_0| \ell + 1$.) Menghasilkan x_{j-1}, x_{j+i-1} sebagai tumbukan.

Menemukan tabrakan yang bermakna. Algoritme yang baru saja dijelaskan mungkin tampak tidak dapat menemukan tabrakan yang berarti karena tidak memiliki kendali atas elemen yang dijadikan sampel. Namun demikian, kami menunjukkan bagaimana menemukan tabrakan yang bermakna adalah mungkin. Caranya adalah dengan menemukan tumbukan pada fungsi yang tepat!

Asumsikan, seperti sebelumnya, bahwa Alice ingin menemukan tabrakan antara dua "tipe" pesan yang berbeda, misalnya, surat yang menjelaskan alasan Alice dipecat dan surat rekomendasi yang bagus bahwa keduanya di-hash dengan nilai yang sama. Kemudian, Alice menulis setiap pesan sehingga terdapat $\ell - 1$ kata yang dapat dipertukarkan di setiap pesan; yaitu, ada $2^{\ell-1}$ pesan untuk setiap jenis. Definisikan fungsi satu-ke-satu $g : \{0,1\}^{\ell} \rightarrow \{0,1\}^{*}$ sedemikian rupa sehingga bit input ke- ℓ memilih antara pesan-pesan bertipe 0 atau tipe 1, dan bit ke-i (untuk $1 \leq i \leq \ell - 1$) memilih di antara opsi-opsi untuk kata ke-enam yang dapat dipertukarkan dalam pesan-pesan dari jenis yang sesuai. Misalnya, perhatikan kalimat:

0: Bob adalah pekerja/karyawan yang baik/pekerja keras dan jujur/dapat dipercaya. 1: Bob adalah pekerja/karyawan yang sulit/bermasalah dan membebani/menjengkelkan.

Definisikan fungsi g yang mengambil masukan 4-bit, dengan bit terakhir menentukan jenis keluaran kalimat, dan tiga bit awal menentukan pilihan kata dalam kalimat tersebut. Misalnya:
$g(0000)$ = Bob adalah pekerja yang baik dan jujur.
$g(0000)$ = Bob adalah pekerja yang sulit dan melelahkan.
$g(0000)$ = Bob adalah karyawan yang pekerja keras dan jujur.
$g(0000)$ = Bob adalah karyawan yang bermasalah dan membebani.

Sekarang tentukan $f : \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell}$ dengan $f(x) \stackrel{\text{def}}{=} H(g(x))$. Alice dapat menemukan tabrakan di f menggunakan serangan ulang tahun ruang kecil yang ditunjukkan sebelumnya. Intinya di sini adalah setiap tumbukan x, x' di f menghasilkan dua pesan $g(x), g(x')$ yang bertabrakan di bawah H. Jika x, x' adalah tumbukan acak maka kita perkirakan dengan probabilitas 1/2 pesan yang bertabrakan $g(x), g(x')$ akan memiliki tipe yang berbeda (karena x dan x' berbeda dalam bit terakhirnya dengan probabilitas tersebut). Jika pesan yang bertabrakan tidak berbeda jenisnya, prosesnya dapat diulangi lagi dari awal.

***Pengorbanan Waktu/Ruang untuk Fungsi Pembalik**

Pada bagian ini kita membahas pertanyaan tentang resistensi preimage, yaitu, kita tertarik pada algoritma untuk masalah inversi fungsi. Di sini, suatu algoritma diberikan $y = H(x)$ untuk seragam x, dan tujuannya adalah untuk menemukan x' apa pun sehingga $H(x') = y$. Kita mulai dengan mengasumsikan bahwa panjang masukan dan keluaran H adalah sama, dan secara singkat membahas kasus yang lebih umum di bagian akhir.

Misalkan $H : \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell}$ suatu fungsi. Tanpa mengeksploitasi kelemahan H, menemukan gambaran awal titik y dapat dilakukan dalam waktu $\mathcal{O}(2^{\ell})$ melalui pencarian menyeluruh pada domain. Kami menunjukkan bahwa dengan pra-pemrosesan yang signifikan, dan jumlah memori yang relatif besar, perbaikan dapat dilakukan dengan lebih baik.

Untuk lebih jelasnya: kami memandang pra-pemrosesan sebagai operasi satu kali dan kami tidak akan terlalu memikirkan biayanya. Kami malah tertarik pada waktu online yang diperlukan untuk membalikkan H pada titik y, setelah pra-pemrosesan selesai. Hal ini dibenarkan jika biaya pra-pemrosesan dapat diamortisasi selama inversi banyak titik, atau jika

kita bersedia menginvestasikan sumber daya komputasi untuk pra-pemrosesan sebelum y diketahui demi keuntungan inversi yang lebih cepat setelahnya. Faktanya, menggunakan prapemrosesan untuk mengaktifkan inversi fungsi dalam waktu yang sangat singkat adalah hal yang sepele. Yang perlu kita lakukan hanyalah mengevaluasi H pada setiap titik selama fase prapemrosesan, dan kemudian menyimpan pasangan $\{(x, H(x))\}$ dalam sebuah tabel, diurutkan berdasarkan entri kedua. Setelah menerima poin y, gambar awal y dapat ditemukan dengan mudah dengan mencari pasangan dengan entri kedua y di tabel. Kelemahannya di sini adalah kita perlu mengalokasikan ruang untuk menyimpan pasangan $\mathcal{O}(2^{\ell})$ dalam tabel, yang dapat menjadi penghalang, atau bahkan tidak mungkin untuk ℓ yang besar (misalnya, $\ell = 80$).

Serangan brute force awal menggunakan memori konstan dan waktu $\mathcal{O}(2^{\ell})$, sedangkan serangan hanya mendeskripsikan penyimpanan titik $\mathcal{O}(2^{\ell})$ dan memungkinkan inversi dalam waktu yang pada dasarnya konstan. Kami sekarang menyajikan pendekatan yang memungkinkan penyerang menukar waktu dan memori. Secara khusus, kami menunjukkan cara menyimpan titik $\mathcal{O}(2^{2\ell/3})$ dan menemukan gambar awal dalam waktu $\mathcal{O}(2^{2\ell/3})$; pengorbanan lainnya mungkin terjadi.

Sebuah pemanasan. Kita mulai dengan mempertimbangkan kasus sederhana di mana fungsi H mendefinisikan sebuah siklus, artinya $x, H(x), H(H(x)), \ldots$ mencakup semua $\{0,1\}^{\ell}$ untuk setiap titik awal x (perhatikan bahwa sebagian besar fungsi tidak mendefinisikan siklus, namun kami berasumsi ini untuk mendemonstrasikan gagasan dalam kasus yang sangat sederhana). Untuk lebih jelasnya, misalkan $N = 2^{\ell}$ menunjukkan ukuran domain.

Pada fase prapemrosesan, penyerang menghabiskan seluruh siklus, dimulai dari titik awal sembarang x_0 dan menghitung $x_1 := H(x_0), x_2 := H(H(x_0))$, hingga $x_N = H^{(N)}(x_0)$, di mana $H^{(i)}$ mengacu pada evaluasi kali lipat dari H.

Misalkan $x_1 \stackrel{\text{def}}{=} H^{(i)}(x_0)$. Kita bayangkan mempartisi siklus menjadi $\sqrt{N}$ segmen dengan panjang masing-masing $\sqrt{N}$, dan meminta penyerang menyimpan titik di awal dan akhir setiap segmen tersebut. Artinya, attacker menyimpan dalam tabel pasangan bentuk $(x_{i.\sqrt{N}}, x_{(i+1).\sqrt{N}})$, untuk $i = 0$ hingga $\sqrt{N} - 1$ diurutkan berdasarkan komponen kedua dari setiap pasangan. Tabel yang dihasilkan berisi poin $\mathcal{O}(\sqrt{N})$.

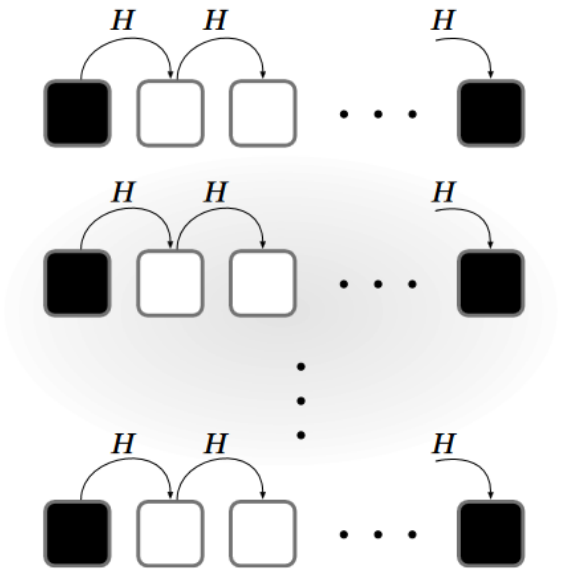
Ketika penyerang diberi titik y untuk dibalik dalam fase online, ia memeriksa yang mana dari $y, H(y), H^{(2)}(y), \ldots$ sesuai dengan titik akhir suatu segmen. (Setiap pemeriksaan hanya melibatkan pencarian tabel pada komponen kedua dari pasangan yang disimpan.) Karena y terletak di beberapa segmen, maka dijamin mencapai titik akhir dalam $\sqrt{N}$ langkah. Setelah titik akhir $x = x_{(i+1).\sqrt{N}}$ teridentifikasi, penyerang mengambil titik awal $x' = x_{i.\sqrt{N}}$ dari segmen terkait dan menghitung $H(x'), H^{(2)}(x'), \ldots$ sampai y tercapai; tVini segera memberikan gambar awal yang diinginkan. Perhatikan bahwa ini memerlukan paling banyak $\sqrt{N}$ evaluasi H.

Singkatnya, serangan ini menyimpan titik $\mathcal{O}(\sqrt{N})$ dan menemukan gambar awal dengan probabilitas 1 menggunakan perhitungan hash $\mathcal{O}(\sqrt{N})$.

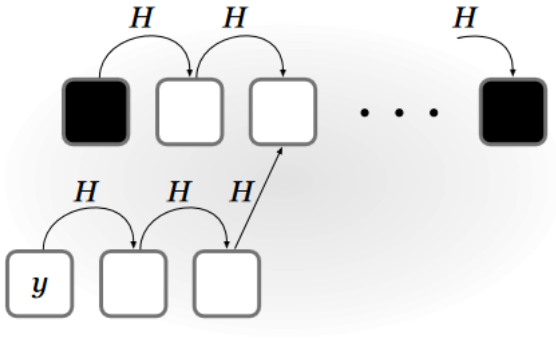
Pengorbanan ruang/waktu Hellman. Martin Hellman memperkenalkan tradeoff waktu/ruang yang lebih umum yang dapat diterapkan pada fungsi sembarang H (meskipun analisisnya memperlakukan H sebagai fungsi acak). Serangan Hellman masih menyimpan titik awal dan titik akhir dari beberapa segmen, namun dalam kasus ini segmen tersebut bersifat "independen" dan bukan menjadi bagian dari satu siklus besar. Lebih detailnya: misalkan s, t menjadi parameter yang akan kita atur nanti. Serangan pertama memilih titik awal yang seragam $SP_1, \ldots, SP_s \in \{0,1\}^{\ell}$. Untuk setiap titik SP_i tersebut, ia menghitung titik akhir EP_i yang sesuai $:= H^{(t)}(SP_i)$ menggunakan penerapan t-fold dari H. (Lihat Gambar 5.3.) Penyerang kemudian menyimpan nilai $\{(SP_i, EP_i)\}_{i=1}^{s}$ di dalam tabel, diurutkan berdasarkan entri kedua dari setiap pasangan.

Setelah menerima nilai y untuk dibalik, serangan berlanjut seperti pada kasus sederhana yang dibahas sebelumnya. Secara khusus, ia memeriksa apakah ada dari $y, H(y), \ldots, H^{(t-1)}(y)$ sama dengan titik akhir suatu segmen (berhenti segera setelah kecocokan pertama ditemukan). Ada kemungkinan bahwa tidak satu pun dari nilai-nilai ini yang sama dengan titik akhir (seperti yang kita bahas di bawah). Namun, jika $H^{(j)}(y) = , EP_i = H^{(t)}(SP_i)$ untuk beberapa i, j, maka penyerang menghitung $H^{(t-j-1)}(SP_i)$ dan memeriksa apakah ini merupakan preimage dari y. Seluruh proses memerlukan paling banyak t evaluasi H.

Tampaknya ini berhasil, namun ada beberapa kehalusan yang kami abaikan. Pertama, mungkin saja tidak ada satu pun dari $y, H(y), \ldots, H^{(t-1)}(y)$ adalah titik akhir suatu segmen. Hal ini dapat terjadi jika y tidak ada dalam kumpulan nilai $s.t$ (tidak termasuk titik awal) yang diperoleh pada proses awal pembuatan tabel. Kita dapat mengatur $s \cdot t \geq N$ dalam upaya memasukkan setiap string ℓ-bit ke dalam tabel, namun hal ini tidak menyelesaikan masalah karena dapat terjadi tabrakan di dalam tabel itu sendiri—pada kenyataannya, untuk $s.t \geq N^{1/2}$ dari tabel sebelumnya Analisis terhadap masalah hari ulang tahun memberi tahu kita bahwa kemungkinan terjadi tabrakan—yang akan mengurangi jumlah titik berbeda dalam kumpulan nilai. Masalah kedua, yang muncul meskipun y ada dalam tabel, adalah meskipun kita menemukan titik akhir yang cocok, sehingga $H^{(j)}(y) = , EP_i = H^{(t)}(SP_i)$ untuk beberapa i, j, ini tidak menjamin bahwa $H^{(t-j-1)}(SP_i)$ adalah gambar awal dari y. Masalahnya di sini adalah segmen $y, H(y), \ldots, H^{(t-1)}(y)$ mungkin akan bertabrakan dengan segmen ke-i meskipun y sendiri tidak berada di segmen tersebut; lihat Gambar 5.4. (Bahkan jika y terletak di suatu segmen, titik akhir pertama yang cocok mungkin tidak berada di segmen tersebut.) Kita menyebutnya positif palsu. Orang mungkin berpikir hal ini tidak mungkin terjadi jika H tahan benturan; Namun, sekali lagi, kita berhadapan dengan situasi yang melibatkan lebih dari $\sqrt{N}$ titik sehingga tabrakan menjadi mungkin terjadi.



GAMBAR 5.3: Pembuatan tabel. Hanya pasangan (SP_i, EP_i) yang disimpan.



GAMBAR 5.4: Bertabrakan dalam fase on-line.

Masalah positif palsu dapat diatasi dengan memodifikasi algoritma sehingga selalu menghitung seluruh barisan $y, H(y), \ldots, H^{(t-1)}(y)$ dan memeriksa apakah $H^{(t-j-1)}(SP_i)$ merupakan preimage dari y untuk setiap i, j sehingga $H^{(j)}(y) = EP_i$. Hal ini dijamin untuk menemukan preimage selama y ada dalam kumpulan nilai (tidak termasuk titik awal) yang dihasilkan selama preprocessing. Kekhawatirannya saat ini adalah waktu berjalan algoritme mungkin meningkat, karena setiap positif palsu menimbulkan evaluasi hash tambahan $\mathcal{O}(t)$. Dapat ditunjukkan bahwa jumlah positif palsu yang diharapkan adalah $\mathcal{O}(st^2/N)$. (Terdapat nilai t pada barisan $y, H(y), \ldots, H^{(t-1)}(y)$ dan paling banyak st titik berbeda dalam tabel. Perlakukan H sebagai fungsi acak, probabilitas bahwa titik mana pun dalam urutan yang sama dengan beberapa titik dalam tabel adalah $1/N$. Jadi, jumlah positif palsu yang diharapkan adalah $t.st.1/N = st^2/N$.) Jadi, selama $st^2 \approx N$, yang akan kami pastikan untuk alasan lain di bawah, jumlah positif palsu yang diharapkan adalah konstan dan menangani positif palsu diperkirakan hanya memerlukan $\mathcal{O}(t)$ komputasi hash tambahan.

Mengingat modifikasi di atas, probabilitas pembalikan $y = H(x)$ setidaknya adalah probabilitas bahw x a berada dalam kumpulan titik (tidak termasuk titik akhir) yang dihasilkan selama prapemrosesan. Kami sekarang menurunkan batas probabilitas ini, mengambil alih keacakan tahap prapemrosesan serta pilihan x yang seragam, memperlakukan H sebagai fungsi acak dalam analisis. Pertama-tama kita menghitung jumlah titik berbeda yang diharapkan dalam tabel. Pertimbangkan apa yang terjadi ketika baris ke-i dari tabel dibuat. Titik awal SPi adalah seragam dan terdapat paling banyak $(i-1)\cdot$ t titik berbeda (tidak termasuk titik akhir) dalam tabel, sehingga kemungkinan bahwa SP_i adalah "baru" (yaitu, tidak sama dengan nilai sebelumnya) setidaknya adalah $1-(i-1)\cdot t/N$. Berapa peluang terambilnya $H(SP_i)$ baru? Jika SP_i bukanlah sesuatu yang baru, maka hampir pasti juga $H(SP_i)$. Di sisi lain, jika SP_i baru maka $H(SP_i)$ seragam (karena kita memperlakukan H sebagai fungsi acak) dan juga baru dengan probabilitas paling sedikit $1-((i-1)\cdot t+1)/N$. (Sekarang kita mempunyai poin tambahan SP_i.) Jadi, probabilitas bahwa $H(SP_i)$ adalah yang baru setidaknya adalah

$$\begin{aligned}\Pr[SP_i \text{ is new}] \cdot \Pr[H(SP_i) \text{ is new} \mid SP_i \text{ is new}] \\ \geq \left(1-\frac{(i-1)\cdot t}{N}\right)\cdot\left(1-\frac{(i-1)\cdot t+1}{N}\right) \\ > \left(1-\frac{(i-1)\cdot t+1}{N}\right)^2.\end{aligned}$$

Melanjutkan cara ini, probabilitas $H^{t-1}(SP_i)$. adalah baru setidaknya adalah

$$\left(1-\frac{i\cdot t}{N}\right)^t = \left[\left(1-\frac{i\cdot t}{N}\right)^{\frac{N}{i\cdot t}}\right]^{\frac{i\cdot t^2}{N}} \approx e^{-it^2/N}.$$

Hal yang perlu diperhatikan di sini adalah ketika $it^2 \leq N/2$, probabilitasnya setidaknya 1/2; sebaliknya, jika $it^2 > N$ kemungkinannya agak kecil. Mengingat baris terakhir, ketika $i = s$, ini berarti kita tidak akan mendapatkan banyak cakupan tambahan jika $st^2 > N$. Oleh karena itu, pengaturan parameter yang baik adalah $st^2 = N/2$. Dengan asumsi ini, jumlah titik berbeda yang diharapkan dalam tabel adalah

$$\sum_{i=1}^{s}\sum_{j=0}^{t-1}\Pr\left[H^{(j)}(SP_i) \text{ is new}\right] \geq \sum_{i=1}^{s}\sum_{j=0}^{t-1}\frac{1}{2} = \frac{st}{2}.$$

Probabilitas bahwa x "tercakup" setidaknya $\frac{st}{2N} = \frac{1}{4t}$.

Hal ini memberikan trade-off waktu/ruang yang lemah, di mana kita dapat menggunakan lebih banyak ruang (dan akibatnya lebih sedikit waktu) dengan mengorbankan penurunan kemungkinan pembalikan y. Namun kita bisa melakukan lebih baik dengan membuat tabel "independen" $T = 4t$. (Hal ini meningkatkan ruang dan waktu sebesar faktor

T .) Selama kita dapat memperlakukan probabilitas x berada di masing-masing tabel terkait sebagai independen, probabilitas bahwa setidaknya salah satu tabel ini berisi x adalah

$$1 - \Pr[\text{no table contains } x] = 1 - \left(1 - \frac{1}{4t}\right)^{4t} \approx 1 - e^{-1} = 0.63.$$

Satu-satunya pertanyaan yang tersisa adalah bagaimana cara menghasilkan tabel independen. (Perhatikan bahwa membuat tabel persis seperti sebelumnya sama dengan menambahkan s baris tambahan ke tabel asli kita, yang telah kita lihat tidak membantu.) Kita dapat melakukan ini untuk tabel ke-i dengan menerapkan beberapa fungsi F_i setelah setiap evaluasi tabel. H, dimana $F_1, \ldots, F_T$ semuanya berbeda. (Pilihan yang baik adalah dengan menetapkan $F_i(x) = x \oplus c_i$ untuk beberapa konstanta tetap c_i yang berbeda di setiap tabel.) Misal $H_i \stackrel{\text{def}}{=} F_i \circ H$, yaitu $H_i(x) = F_i(H(x))$. Kemudian untuk tabel ke-i kita kembali memilih s titik awal secara acak, namun untuk setiap titik tersebut sekarang kita menghitung $H_i(SP)$, $H_i^{(2)}(SP)$, dan seterusnya. Setelah menerima nilai $y = H(x)$ untuk dibalik, penyerang terlebih dahulu menghitung $y' = F_i(y)$ dan kemudian memeriksa apakah ada di antara $y', H_i(y'), \ldots, H_i^{(t-1)}(y')$ berhubungan dengan titik akhir pada tabel ke-i; ini diulangi untuk $i = 1, \ldots, T$. (Kami menghilangkan rincian lebih lanjut.) Meskipun sulit untuk memperdebatkan independensi secara formal, pendekatan ini memberikan hasil yang baik dalam praktiknya.

Memilih parameter. Meringkas pembahasan di atas, kita melihat bahwa selama $st^2 = N/2$ kita mempunyai algoritma yang menyimpan titik $\mathcal{O}(s \cdot T) = \mathcal{O}(s \cdot t) = \mathcal{O}(N/t)$ selama fase prapemrosesan, dan kemudian dapat membalikkan y dengan probabilitas konstan dalam waktu $\mathcal{O}(t\,T) = \mathcal{O}(t^2)$. Salah satu pengaturan parameternya adalah $t = N^{1-3} = 2^{\ell/3}$, dalam hal ini kita memiliki algoritma yang menyimpan titik $\mathcal{O}(2^{2\ell/3})$ yang menemukan gambar awal menggunakan komputasi hash $\mathcal{O}(2^{2\ell/3})$. Jika fungsi hash dengan output 80 bit digunakan, maka hal ini layak dilakukan dalam praktiknya.

Menangani domain dan jangkauan yang berbeda. Dalam praktiknya, sering kali kita dihadapkan pada situasi di mana domain dan range H berbeda. Salah satu contohnya adalah dalam konteks peretasan kata sandi (lihat Bagian 5.6), dimana penyerang memiliki $H(pw)$ tetapi $|pw| \ll \ell$. Dalam kasus umum, katakanlah x dipilih dari beberapa domain D yang mungkin lebih besar atau lebih kecil dari $\{0,1\}^\ell$. Meskipun tentu saja dimungkinkan untuk memperluas domain/rentang secara artifisial agar cocok, hal ini tidak akan berguna untuk serangan yang dijelaskan di atas. Untuk mengetahui alasannya, pertimbangkan contoh kata sandi. Agar serangan berhasil, kami ingin pw berada di tabel nilai yang dihasilkan selama prapemrosesan. Jika kita menghasilkan setiap baris tabel hanya dengan menghitung $H(SP), H^{(2)}(SP), \ldots$, untuk $SP \in D$, maka tidak satupun dari nilai tersebut (kecuali mungkin SP itu sendiri) yang akan sama dengan pw.

Kita dapat mengatasi hal ini dengan menerapkan fungsi F_i, seperti sebelumnya, di antara setiap evaluasi H, meskipun sekarang kita memilih pemetaan $F_i\{0,1\}^\ell$ ke D. Ini memecahkan masalah di atas, karena $F_i(H(SP)), (F_i \circ H)^{(2)}(SP), \ldots$ sekarang semua terletak pada D.

Aplikasi untuk serangan pemulihan kunci. Pengorbanan waktu/ruang memberikan serangan terhadap primitif kriptografi selain fungsi hash. Salah satu aplikasi kanonik—pada kenyataannya, aplikasi yang awalnya dipertimbangkan oleh Hellman—adalah serangan terhadap cipher blok F yang sewenang-wenang yang mengarah pada pemulihan kunci. Definisikan $H(k) \stackrel{\text{def}}{=} F_k(x)$ di mana x adalah masukan sembarang namun tetap yang akan digunakan untuk membuat tabel. Jika seorang penyerang dapat memperoleh $F_k(x)$ untuk kunci k yang tidak diketahui—baik melalui serangan teks biasa yang dipilih atau dengan memilih x sehingga $F_k(x)$ kemungkinan besar diperoleh dalam serangan teks biasa yang diketahui—maka dengan membalikkan H penyerang belajar (nilai kandidat untuk) k. Perhatikan bahwa mungkin saja panjang kunci F berbeda dari panjang bloknya, namun dalam kasus ini kita dapat menggunakan teknik yang baru saja dijelaskan untuk menangani H dengan domain dan jangkauan berbeda.

5.5 MODEL ORACLE ACAK

Ada beberapa contoh konstruksi berdasarkan fungsi hash kriptografi yang tidak dapat dibuktikan aman hanya berdasarkan asumsi bahwa fungsi hash tahan terhadap tabrakan atau preimage. (Kita akan melihat beberapa di bagian berikut.) Dalam banyak kasus, tampaknya tidak ada asumsi sederhana dan masuk akal mengenai fungsi hash yang akan cukup untuk membuktikan keamanan konstruksi.

Menghadapi situasi ini, ada beberapa pilihan. Salah satunya adalah mencari skema yang terbukti aman berdasarkan asumsi masuk akal tentang fungsi hash yang mendasarinya. Ini adalah pendekatan yang baik, namun hal ini menyisakan pertanyaan tentang apa yang harus dilakukan sampai skema tersebut ditemukan. Selain itu, konstruksi yang terbukti aman mungkin kurang efisien dibandingkan pendekatan lain yang belum terbukti aman. (Ini adalah masalah utama yang akan kita temui dalam pengaturan kriptografi kunci publik.)

Kemungkinan lain, tentu saja, adalah menggunakan sistem kriptografi yang sudah ada meskipun sistem tersebut tidak memiliki pembenaran atas keamanannya selain, mungkin, fakta bahwa perancangnya mencoba menyerangnya dan tidak berhasil. Hal ini bertentangan dengan semua yang telah kami katakan tentang pentingnya pendekatan kriptografi yang ketat dan modern, dan harus jelas bahwa hal ini tidak dapat diterima.

Sebuah pendekatan yang sangat sukses dalam praktiknya, dan menawarkan “jalan tengah” antara bukti keamanan yang sangat ketat di satu sisi dan tanpa bukti apa pun di sisi lain, adalah dengan memperkenalkan model ideal untuk membuktikan keamanan suatu produk. skema kriptografi. Meskipun idealisasi tersebut mungkin tidak mencerminkan kenyataan secara akurat, setidaknya kita dapat memperoleh tingkat keyakinan terhadap kelayakan rancangan suatu skema berdasarkan bukti dalam model yang diidealkan. Selama modelnya masuk akal, pembuktian seperti itu tentu lebih baik daripada tidak ada pembuktian sama sekali.

Contoh paling populer dari pendekatan ini adalah model random-oracle, yang memperlakukan fungsi hash kriptografi H sebagai fungsi yang benar-benar acak. (Kita telah melihat contohnya dalam diskusi kita tentang trade-off waktu/ruang, meskipun di sana kita

menganalisis sebuah serangan dan bukan sebuah konstruksi.) Lebih khusus lagi, model random-oracle mengemukakan keberadaan fungsi publik dan acak H yang dapat dievaluasi hanya dengan "menanyakan" sebuah oracle—yang dapat dianggap sebagai "kotak hitam"—yang mengembalikan $H(x)$ ketika diberikan masukan x. (Kita akan membahas bagaimana hal ini ditafsirkan pada bagian berikut.) Untuk membedakannya, model yang kita gunakan sampai saat ini (di mana tidak ada ramalan acak) sering disebut "model standar."

Tidak ada seorang pun yang menyatakan bahwa ramalan acak itu ada, meskipun ada saran bahwa ramalan acak dapat diimplementasikan dalam praktik dengan menggunakan pihak yang tepercaya (misalnya, beberapa server di Internet). Sebaliknya, model random-oracle menyediakan metodologi formal yang dapat digunakan untuk merancang dan memvalidasi skema kriptografi menggunakan pendekatan dua langkah berikut:

1. Pertama, skema dirancang dan terbukti aman dalam model random-oracle. Artinya, kami berasumsi bahwa dunia berisi ramalan acak, dan membangun serta menganalisis skema kriptografi dalam model ini. Asumsi kriptografi standar seperti yang telah kita lihat sampai sekarang dapat digunakan dalam pembuktian keamanan juga.
2. Saat kita ingin mengimplementasikan skema tersebut di dunia nyata, oracle acak tidak tersedia. Sebaliknya, oracle acak dibuat dengan fungsi hash kriptografi yang dirancang dengan tepat $\widehat{H}$. (Kita kembali ke titik ini di akhir bagian ini.) Artinya, pada setiap titik di mana skema menentukan bahwa salah satu pihak harus menanyakan nilai $H(x)$ kepada Oracle, pihak tersebut malah menghitung $\widehat{H}(x)$ sendiri.

Harapannya adalah bahwa fungsi hash kriptografi yang digunakan pada langkah kedua "cukup baik" dalam meniru oracle acak, sehingga bukti keamanan yang diberikan pada langkah pertama akan diterapkan pada contoh skema di dunia nyata. Kesulitannya di sini adalah bahwa tidak ada pembenaran teoritis untuk harapan ini, dan pada kenyataannya ada skema (yang dibuat-buat) yang dapat dibuktikan aman dalam model random-oracle namun tidak aman tidak peduli bagaimana random oracle dipakai pada langkah kedua. Selain itu, tidak jelas (secara matematis atau heuristik) apa arti fungsi hash menjadi "cukup baik" dalam meniru oracle acak, juga tidak jelas apakah ini merupakan tujuan yang dapat dicapai.

Secara khusus, tidak ada contoh nyata $\widehat{H}$ yang dapat berperilaku seperti fungsi acak, karena $\widehat{H}$ bersifat deterministik dan tetap. Oleh karena itu, bukti keamanan dalam model random-oracle harus dipandang sebagai bukti bahwa suatu skema tidak memiliki "kelemahan desain bawaan", namun bukan merupakan bukti kuat bahwa penerapan skema di dunia nyata aman. Diskusi lebih lanjut tentang bagaimana menafsirkan bukti dalam model random-oracle diberikan di Bagian 5.5.

Model Random-Oracle secara Detail

Sebelum melanjutkan, mari kita jelaskan apa yang dimaksud dengan model oracle acak. Cara yang baik untuk memikirkan model oracle acak adalah sebagai berikut: "Oracle" hanyalah sebuah kotak yang mengambil string biner sebagai masukan dan mengembalikan string biner sebagai keluaran. Cara kerja bagian dalam kotak itu tidak diketahui dan tidak dapat dipahami. Setiap orang—baik pihak yang jujur maupun pihak yang bermusuhan—dapat berinteraksi dengan kotak, di mana interaksi tersebut terdiri dari memasukkan string biner x

sebagai masukan dan menerima string biner y sebagai keluaran; kami menyebutnya sebagai menanyakan oracle pada x, dan menyebut x sendiri sebagai kueri yang dibuat untuk oracle. Kueri ke oracle diasumsikan bersifat pribadi sehingga jika beberapa pihak menanyakan oracle pada input x maka tidak ada orang lain yang mempelajari x, atau bahkan mengetahui bahwa pihak tersebut menanyakan oracle sama sekali. Hal ini masuk akal, karena panggilan ke oracle berhubungan (dalam contoh dunia nyata) dengan evaluasi lokal dari fungsi hash kriptografi. Sifat penting dari "kotak" ini adalah konsistensinya. Artinya, jika kotak tersebut mengeluarkan keluaran y untuk masukan x tertentu, maka kotak tersebut selalu mengeluarkan jawaban y yang sama ketika diberi masukan x yang sama lagi. Ini berarti bahwa kita dapat melihat kotak tersebut sebagai implementasi fungsi H yang terdefinisi dengan baik; yaitu, kita mendefinisikan fungsi H dalam kaitannya dengan karakteristik masukan/keluaran kotak. Untuk kenyamanan, kita berbicara tentang "mengkueri H" daripada menanyakan kotak. Tidak ada yang "mengetahui" keseluruhan fungsi H (kecuali kotak itu sendiri); paling banter, yang diketahui hanyalah nilai H pada string yang telah ditanyakan secara eksplisit sejauh ini.

Kita telah membahas di Bab 3 apa artinya memilih fungsi acak H. Kami hanya menegaskan kembali di sini bahwa ada dua cara yang setara untuk memikirkan pemilihan seragam H: gambar H dipilih "dalam satu pengambilan" secara seragam dari himpunan dari semua fungsi pada domain dan rentang tertentu, atau bayangkan menghasilkan output untuk H "on-the-fly," sesuai kebutuhan. Khususnya, dalam kasus kedua kita dapat melihat fungsi didefinisikan oleh tabel yang awalnya kosong. Saat oracle menerima kueri x, ia terlebih dahulu memeriksa apakah $x = x_i$ untuk beberapa pasangan (x_i, y_i) dalam tabel; jika demikian, nilai yang sesuai $y_i$dikembalikan. Jika tidak, string seragam $y \in \{0,1\}^{\ell}$ dipilih (untuk beberapa ℓ tertentu), jawaban y dikembalikan, dan Oracle menyimpan (x, y) dalam tabelnya. Sudut pandang kedua ini seringkali secara konseptual lebih mudah untuk dipikirkan, dan juga secara teknis lebih mudah untuk ditangani jika H didefinisikan pada domain tak terbatas (misalnya, $\{0,1\}^*$).

Ketika kami mendefinisikan fungsi pseudorandom di Bagian 3.5, kami juga mempertimbangkan algoritma yang memiliki akses oracle ke fungsi acak. Agar tidak terjadi kebingungan, kami perhatikan bahwa penggunaan fungsi acak di sana sangat berbeda dengan penggunaan fungsi acak di sini. Di sana, fungsi acak digunakan sebagai cara untuk mendefinisikan apa artinya fungsi berkunci (konkret) menjadi pseudorandom. Sebaliknya, dalam model random-oracle, fungsi acak digunakan sebagai bagian dari konstruksi itu sendiri dan entah bagaimana harus dipakai di dunia nyata jika kita menginginkan realisasi konkrit dari konstruksi tersebut. Fungsi pseudorandom bukanlah sebuah ramalan acak karena ia hanya bersifat pseudorandom jika kuncinya rahasia. Namun, dalam model random-oracle semua pihak harus mampu menghitung fungsinya; jadi tidak mungkin ada kunci rahasia.

Definisi dan Pembuktian dalam Model Random-Oracle

Definisi dalam model random-oracle sedikit berbeda dari definisi dalam model standar karena ruang probabilitas yang dipertimbangkan dalam setiap kasus tidak sama. Dalam model standar, skema $\prod$ aman jika untuk semua musuh PPT $\mathcal{A}$, probabilitas suatu kejadian berada di bawah ambang batas tertentu, di mana probabilitas ini diambil alih oleh pilihan acak dari

pihak-pihak yang menjalankan $\prod$ dan pihak-pihak dari pihak lawan $\mathcal{A}$. Dengan asumsi pihak jujur yang menggunakan $\prod$ di dunia nyata membuat pilihan acak seperti yang diarahkan oleh skema, memenuhi definisi semacam ini menjamin keamanan untuk penggunaan $\prod$ di dunia nyata. Sebaliknya, dalam model oracle acak, skema $\prod$ mungkin bergantung pada oracle H. Seperti sebelumnya, $\prod$ aman jika untuk semua musuh PPT $\mathcal{A}$ probabilitas suatu kejadian berada di bawah ambang batas tertentu, namun sekarang probabilitas ini diambil atas pilihan acak H serta pilihan acak dari pihak yang menjalankan $\prod$ dan pihak lawan $\mathcal{A}$. Saat menggunakan $\prod$ di dunia nyata, beberapa (instansiasi) H harus diperbaiki. Sayangnya, keamanan $\prod$ tidak dijamin untuk pilihan H tertentu. Hal ini menunjukkan salah satu alasan mengapa sulit untuk berargumentasi bahwa setiap contoh nyata dari oracle H dengan fungsi deterministik menghasilkan skema yang aman. (Kesulitan tambahan yang bersifat teknis adalah ketika fungsi konkrit H diperbaiki, musuh tidak lagi terbatas pada menanyakan H sebagai oracle tetapi malah dapat melihat dan menggunakan kode H dalam serangannya.)

Pembuktian dalam model oracle acak dapat mengeksploitasi fakta bahwa H dipilih secara acak, dan bahwa satu-satunya cara untuk mengevaluasi $H(x)$ adalah dengan menanyakan x ke H secara eksplisit. Tiga properti khususnya sangat berguna; kami membuat sketsanya secara informal di sini, dan menunjukkan beberapa penerapan sederhananya di bawah dan di bagian berikutnya, namun perlu diingat bahwa pemahaman penuh mungkin harus menunggu sampai kami menyajikan bukti formal dalam model random-oracle di bab-bab selanjutnya.

Properti pertama yang berguna dari model random-oracle adalah:

Jika x belum ditanyakan ke H, maka nilai $H(x)$ seragam.

Hal ini mungkin tampak mirip dengan jaminan yang diberikan oleh generator pseudo-acak, namun sebenarnya jauh lebih kuat. Jika G adalah generator pseudorandom maka $G(x)$ adalah pseudorandom bagi pengamat dengan asumsi x dipilih secara acak dan sama sekali tidak diketahui oleh pengamat. Namun, jika H adalah ramalan acak, maka $H(x)$ benar-benar seragam bagi pengamat selama pengamat tersebut tidak menanyakan x. Hal ini berlaku meskipun x diketahui, atau jika x tidak seragam namun sulit ditebak. (Misalnya, jika x adalah string n-bit yang bagian pertama dari x diketahui dan bagian terakhirnya acak, maka $G(x)$mungkin mudah dibedakan dari acak tetapi $H(x)$ tidak akan mudah.)

Dua properti lainnya $\mathcal{A}$ berhubungan secara eksplisit dengan pembuktian dengan reduksi dalam model random-oracle. (Di sini mungkin berguna untuk meninjau Bagian 3.3) Sebagai bagian dari reduksi, ramalan acak yang berinteraksi dengan musuh harus disimulasikan. Yaitu: $\mathcal{A}$ akan mengirimkan pertanyaan ke, dan menerima jawaban dari, apa yang diyakini sebagai oracle, namun reduksi itu sendiri sekarang harus menjawab pertanyaan ini. Hal ini ternyata memberikan kekuatan yang besar. Sebagai permulaan:

Jika $\mathcal{A}$ menanyakan x ke H, reduksi dapat melihat kueri ini dan mempelajari x.

Hal ini terkadang disebut “ekstraksi.” (Hal ini tidak bertentangan dengan fakta, yang telah disebutkan sebelumnya, bahwa kueri terhadap oracle acak bersifat “pribadi”. Meskipun hal ini berlaku dalam model oracle acak itu sendiri, di sini kita menggunakan $\mathcal{A}$ sebagai subrutin dalam reduksi yang menyimulasikan acak oracle untuk $\mathcal{A}$.) Akhirnya:

Pengurangan dapat mengatur nilai $H(x)$ (yaitu, respons terhadap kueri x) ke nilai pilihannya, selama nilai ini terdistribusi dengan benar, yaitu seragam. Ini disebut “kemampuan terprogram.” Tidak ada tandingannya terhadap kemampuan ekstraksi atau kemampuan program setelah H dipakai dengan fungsi konkrit apa pun.

Ilustrasi Sederhana Model Random-Oracle

Pada titik ini beberapa contoh mungkin bisa membantu. Contoh yang diberikan di sini relatif sederhana, dan tidak menggunakan kekuatan penuh yang diberikan oleh model random-oracle. Sebaliknya, contoh-contoh ini disajikan hanya untuk memberikan pengenalan singkat terhadap model tersebut. Berikut ini, kita asumsikan oracle acak memetakan masukan ℓ_{in}-bit ke keluaran ℓ_{out}-bit, di mana ℓ_{in}, $\ell_{out} > n$, parameter keamanan (jadi ℓ_{in}, ℓ_{out} adalah fungsi dari n).

Oracle acak sebagai generator pseudorandom. Pertama-tama kami tunjukkan bahwa, untuk $\ell_{out} > \ell_{in}$, oracle acak dapat digunakan sebagai generator pseudorandom. (Kami tidak mengatakan bahwa oracle acak adalah generator pseudorandom, karena oracle acak bukanlah fungsi tetap.) Secara formal, kami mengklaim bahwa untuk setiap musuh PPT $\mathcal{A}$, ada fungsi yang dapat diabaikan sehingga

$$\left|\Pr[\mathcal{A}^{H(\cdot)}(y)=1]-\Pr[\mathcal{A}^{H(\cdot)}(H(x))=1]\right| \leq \mathsf{negl}(n),$$

dimana dalam kasus pertama probabilitas diambil alih pilihan seragam H, pilihan seragam $y \in \{0,1\}^{\ell_{out}(n)}$, dan keacakan $\mathcal{A}$, dan dalam kasus kedua probabilitas diambil alih pilihan seragam H, pilihan seragam $x \in \{0,1\}^{\ell_{out}(n)}$, dan keacakan $\mathcal{A}$. Kami telah secara eksplisit menunjukkan $\mathcal{A}$ bahwa memiliki akses oracle ke H dalam setiap kasus; setelah H dipilih maka $\mathcal{A}$ dapat dengan bebas menanyakannya.

Misalkan S menyatakan himpunan titik yang ditanyakan H; tentu saja, $|S|$ adalah polinomial di n. Perhatikan bahwa dalam kasus kedua, probabilitas $x \in S$ dapat diabaikan. Hal ini berlaku karena dimulai tanpa informasi tentang x (perhatikan bahwa $H(x)$ dengan sendirinya tidak mengungkapkan apa pun tentang x karena H adalah fungsi acak), dan karena S secara eksponensial lebih kecil dari $\{0,1\}^{\ell in}$. Selain itu, dikondisikan pada $x \notin S$ dalam kasus kedua, masukan $\mathcal{A}$ dalam setiap kasus adalah string seragam yang tidak tergantung pada jawaban pertanyaan $\mathcal{A}$.

Oracle acak sebagai fungsi hash tahan benturan. Jika $\ell_{out} < \ell_{in}$, oracle acak tahan benturan. Artinya, probabilitas keberhasilan musuh PPT $\mathcal{A}$ dalam percobaan berikut dapat diabaikan:

1. Fungsi acak H dipilih.
2. $\mathcal{A}$ berhasil jika menghasilkan x, x' yang berbeda dengan H(x) = H(x').

Untuk melihat hal ini, asumsikan tanpa kehilangan keumuman bahwa $\mathcal{A}$ hanya mengeluarkan nilai x, x' yang sebelumnya telah ditanyakan ke oracle, dan bahwa $\mathcal{A}$ tidak pernah membuat ueri yang sama ke oracle dua kali. Membiarkan kueri Oracle dari $\mathcal{A}$ menjadi $x_1, \ldots, x_q$, dengan $q = \text{poli}(n)$, jelas bahwa peluang keberhasilan $\mathcal{A}$ dibatasi atas oleh peluang $H(xi) = H(xj)$ untuk beberapa $i \neq j$. Tapi ini sama persis dengan probabilitas jika kita memilih string $q\, y_1, \ldots, y_q \in \{0,1\}^{\ell out}$ secara independen dan seragam secara acak, kita mempunyai $y_1 = y_j$ untuk beberapa $i \neq j$. Kita mendapatkan bahwa $\mathcal{A}$ berhasil dengan probabilitas yang dapat diabaikan $O(q/2^{\ell in})$.

Membangun fungsi pseudorandom dari oracle acak. Membangun fungsi pseudorandom dalam model random-oracle juga cukup mudah. Misalkan $\ell_{in}(n) = 2n$ dan $\ell_{out}(n) = n$, dan tentukan

$$F_k(x) \stackrel{\text{def}}{=} H(k\|x)$$

dimana $|k| = |x| = n$. Dalam Latihan 5.11 Anda diminta untuk menunjukkan bahwa ini adalah fungsi pseudorandom, yaitu, untuk setiap waktu polinomial, probabilitas $\mathcal{A}$ dalam percobaan berikut tidak lebih dari 1/2 ditambah fungsi yang dapat diabaikan:

1. Fungsi H dan nilai $k \in \{0,1\}^n$ dan $b \in \{0,1\}$ dipilih secara seragam.
2. Jika $b = 0$, musuh diberi akses ke oracle untuk $F_k(\cdot) = H(k\|\cdot)$. Jika $b = 1$, maka diberikan akses ke fungsi acak yang memetakan input n-bit ke output n-bit. (Fungsi acak ini tidak bergantung pada H.)
3. $\mathcal{A}$ mengeluarkan sedikit b', dan berhasil jika $b' = b$.

Pada langkah 2, dapat mengakses H selain fungsi Oracle yang disediakan oleh eksperimen. (Fungsi pseudorandom dalam model oracle acak harus tidak dapat dibedakan dari fungsi acak yang tidak bergantung pada H.)

Aspek yang menarik dari semua klaim di atas adalah bahwa klaim tersebut tidak membuat asumsi komputasi; mereka berlaku bahkan untuk musuh yang tidak terbatas secara komputasi selama musuh tersebut dibatasi untuk membuat banyak pertanyaan secara polinomial ke oracle. Hal ini tidak ada bandingannya di dunia nyata, di mana kita telah melihat bahwa asumsi komputasi diperlukan.

Metodologi Random-Oracle?

Skema yang dirancang dalam model random-oracle diimplementasikan di dunia nyata dengan menginstansiasi H dengan beberapa fungsi konkret. Dengan mekanisme model random-oracle di belakang kita, kita beralih ke pertanyaan yang lebih mendasar:

> *Apa yang dijamin oleh bukti keamanan dalam model oracle acak sejauh keamanan instanisasi dunia nyata?*

Pertanyaan ini tidak memiliki jawaban yang pasti: saat ini ada perdebatan dalam komunitas kriptografi mengenai bagaimana menafsirkan bukti dalam model oracle acak, dan bidang penelitian yang aktif adalah untuk menentukan apa tepatnya bukti keamanan dalam

model oracle acak. model Oracle menyiratkan vis-a-vis dunia nyata. Kami hanya bisa berharap dapat memberikan gambaran mengenai kedua sisi perdebatan tersebut.

Keberatan terhadap model random-oracle. Titik awal argumen yang menentang penggunaan oracle acak adalah sederhana: seperti yang telah kita catat, tidak ada pembenaran formal atau ketat untuk meyakini bahwa bukti keamanan untuk beberapa skema Π dalam model oracle acak menjelaskan apa pun tentang keamanan Π di dunia nyata, setelah oracle acak H dibuat dengan fungsi hash tertentu $\widehat{H}$. Ini lebih dari sekedar kegelisahan teoritis. Sedikit pemikiran menunjukkan bahwa tidak ada fungsi hash konkrit yang dapat bertindak sebagai ramalan acak yang "benar". Misalnya, dalam model random-oracle, nilai $H(x)$ adalah "sepenuhnya acak" jika x tidak ditanyakan secara eksplisit. Rekannya akan melakukannya menjadi mengharuskan $\widehat{H}(x)$ acak (atau pseudorandom) jika $\widehat{H}$ tidak dievaluasi secara eksplisit pada x. Bagaimana kita menafsirkan hal ini di dunia nyata? Bahkan tidak jelas apa yang dimaksud dengan "mengevaluasi secara eksplisit" $\widehat{H}$: bagaimana jika musuh mengetahui jalan pintas untuk menghitung $\widehat{H}$ yang tidak melibatkan menjalankan kode sebenarnya untuk $\widehat{H}$? Terlebih lagi, $\widehat{H}(x)$ tidak mungkin acak (atau bahkan pseudorandom) karena begitu musuh mempelajari deskripsi $\widehat{H}$, nilai fungsi tersebut pada semua input segera ditentukan.

Keterbatasan model random-oracle menjadi lebih jelas setelah kita memeriksa teknik pembuktian yang diperkenalkan sebelumnya. Ingatlah bahwa salah satu teknik pembuktian adalah dengan menggunakan fakta bahwa pengurangan dapat "melihat" pertanyaan yang diajukan musuh terhadap oracle acak. Jika kita mengganti oracle acak dengan fungsi hash tertentu $\widehat{H}$, ini berarti kita harus memberikan deskripsi $\widehat{H}$ kepada musuh di awal percobaan. Namun kemudian $\mathcal{A}$ dapat mengevaluasi $\widehat{H}$ sendiri, tanpa membuat pertanyaan eksplisit apa pun, sehingga reduksi tidak lagi memiliki kemampuan untuk "melihat" pertanyaan apa pun yang dibuat oleh $\mathcal{A}$. (Faktanya, seperti disebutkan sebelumnya, $\mathcal{A}$ melakukan evaluasi eksplisit terhadap $\widehat{H}$ mungkin tidak benar dan tentu saja tidak dapat didefinisikan secara formal.) Demikian pula, bukti keamanan dalam model random-oracle memungkinkan reduksi untuk memilih output dari H sebagaimana yang diharapkan. keinginannya, sesuatu yang jelas tidak mungkin dilakukan bila fungsi konkrit digunakan.

Bahkan jika kita ingin mengabaikan permasalahan teoretis di atas, masalah praktisnya adalah saat ini kita tidak memiliki pemahaman yang baik tentang apa artinya fungsi hash konkret menjadi "cukup baik" dalam membuat instance oracle acak. Agar lebih konkrit, misalkan kita ingin membuat instance oracle acak menggunakan beberapa modifikasi SHA-1 yang sesuai. Sementara untuk beberapa skema tertentu Π mungkin masuk akal untuk berasumsi bahwa Π aman ketika dipakai menggunakan SHA-1, jauh lebih tidak masuk akal untuk berasumsi bahwa SHA-1 dapat menggantikan oracle acak di setiap skema yang dirancang dalam skema acak- model ramalan. Memang benar, seperti yang telah kami katakan sebelumnya, kita tahu bahwa SHA-1 bukanlah oracle sembarangan. Dan tidak sulit untuk merancang skema yang aman dalam model oracle acak, namun tidak aman ketika oracle acak digantikan oleh SHA-1.

Kami menekankan bahwa asumsi bentuk "SHA-1 bertindak seperti oracle acak" secara kualitatif berbeda dari asumsi seperti "SHA-1 tahan benturan" atau "AES adalah fungsi

pseudorandom." Permasalahannya antara lain terletak pada kenyataan bahwa tidak ada definisi yang memuaskan mengenai arti dari pernyataan pertama, sementara kita mempunyai definisi yang sama untuk dua pernyataan terakhir.

Oleh karena itu, penggunaan model random-oracle untuk membuktikan keamanan suatu skema secara kualitatif berbeda dengan, misalnya, memperkenalkan asumsi kriptografi baru untuk membuktikan keamanan skema dalam model standar. Oleh karena itu, bukti keamanan pada model random-oracle kurang memuaskan dibandingkan bukti keamanan pada model standar.

Dukungan untuk model oracle acak. Mengingat semua masalah dengan model random-oracle, mengapa menggunakannya? Lebih penting lagi: mengapa model random-oracle begitu berpengaruh dalam perkembangan kriptografi modern (terutama penggunaan kriptografi praktis saat ini), dan mengapa model ini terus digunakan secara luas? Seperti yang akan kita lihat, model random-oracle memungkinkan perancangan skema yang jauh lebih efisien dibandingkan skema yang kita ketahui cara membangunnya dalam model standar. Dengan demikian, hanya ada sedikit (jika ada) sistem kriptografi kunci publik yang digunakan saat ini yang memiliki bukti keamanan dalam model standar, sementara ada banyak skema yang diterapkan yang memiliki bukti keamanan dalam model random-oracle. Selain itu, pembuktian dalam model random-oracle hampir secara universal diakui memberikan kepercayaan terhadap keamanan skema yang sedang dipertimbangkan untuk standardisasi.

Alasan mendasarnya adalah keyakinan bahwa:

> *Bukti keamanan dalam model random-oracle secara signifikan lebih baik daripada tidak ada bukti sama sekali.*

Meskipun ada yang tidak setuju, kami menawarkan hal berikut untuk mendukung pernyataan ini:

> *Bukti keamanan untuk skema dalam model random-oracle menunjukkan bahwa desain skema tersebut "baik", dalam arti bahwa satu-satunya serangan yang mungkin terjadi pada contoh skema di dunia nyata adalah serangan yang muncul karena kelemahan dalam sistem. fungsi hash yang digunakan untuk membuat instance Oracle acak. Jadi, jika fungsi hash yang "cukup baik" digunakan untuk membuat instance oracle acak, kita harus yakin dengan keamanan skema tersebut. Selain itu, jika contoh skema tertentu berhasil diserang, kita cukup mengganti fungsi hash yang digunakan dengan fungsi yang "lebih baik".*

Yang penting, belum ada serangan di dunia nyata yang berhasil terhadap skema yang terbukti aman dalam model oracle acak, ketika oracle acak dibuat dengan benar. Hal ini memberikan bukti mengenai kegunaan model ramalan acak dalam merancang skema praktis.

Namun demikian, hal di atas pada akhirnya hanya mewakili spekulasi intuitif mengenai kegunaan pembuktian dalam model ramalan acak dan—semua hal lainnya dianggap sama—bukti tanpa ramalan acak lebih disukai.

Membuat Instansiasi Oracle Acak

Membuat contoh ramalan acak dengan benar adalah hal yang tidak kentara, dan diskusi lengkap berada di luar cakupan buku ini. Di sini kami hanya memperingatkan pembaca bahwa menggunakan fungsi hash kriptografi yang "siap pakai" tanpa modifikasi, secara umum, bukanlah pendekatan yang tepat. Salah satu alasannya adalah sebagian besar fungsi hash kriptografi dibuat menggunakan paradigma Merkle–Damg˚ard, yang dapat dengan mudah dibedakan dari oracle acak ketika input dengan panjang variabel diperbolehkan. Selain itu, dalam beberapa konstruksi, output dari oracle acak harus berada pada kisaran tertentu (misalnya, oracle harus mengeluarkan elemen dari beberapa grup), yang mengakibatkan komplikasi tambahan.

5.6 PENERAPAN FUNGSI HASH TAMBAHAN

Kami menyimpulkan bab ini dengan diskusi singkat tentang beberapa penerapan tambahan fungsi hash kriptografi dalam kriptografi dan keamanan komputer.

Sidik Jari dan Deduplikasi

Saat menggunakan fungsi hash tahan benturan H, hash (atau intisari) suatu file berfungsi sebagai pengidentifikasi unik untuk file tersebut. (Jika ada file lain yang ditemukan memiliki pengenal yang sama, ini berarti tabrakan di H). Hash $H(x)$ dari file x seperti sidik jari, dan seseorang dapat memeriksa apakah dua file sama dengan membandingkan intisarinya. Ide sederhana ini memiliki banyak penerapan.

Sidik jari virus: Pemindai virus mengidentifikasi virus dan memblokir atau mengkarantinanya. Salah satu langkah paling mendasar untuk mencapai tujuan ini adalah dengan menyimpan database yang berisi hash dari virus yang dikenal, dan kemudian mencari hash dari aplikasi yang diunduh atau lampiran email dalam database ini. Karena hanya string pendek yang perlu dicatat (dan/atau didistribusikan) untuk setiap virus, biaya tambahan yang diperlukan dapat dilakukan.

Deduplikasi: Deduplikasi data digunakan untuk menghilangkan duplikat salinan data, terutama dalam konteks penyimpanan cloud di mana banyak pengguna mengandalkan satu layanan cloud untuk menyimpan data mereka. Pengamatannya di sini adalah jika beberapa pengguna ingin menyimpan file yang sama (misalnya video populer), maka file tersebut hanya perlu disimpan satu kali dan tidak perlu diunggah secara terpisah oleh setiap pengguna. Deduplikasi dapat dilakukan dengan terlebih dahulu meminta pengguna mengunggah hash dari file baru yang ingin mereka simpan; jika file dengan hash ini sudah disimpan di cloud, maka penyedia penyimpanan cloud cukup menambahkan pointer ke file yang ada untuk menunjukkan bahwa pengguna tertentu juga telah menyimpan file ini. Hal ini menghemat komunikasi dan penyimpanan, dan kesehatan metodologi dihasilkan dari ketahanan fungsi hash terhadap benturan.

Berbagi file peer-to-peer (P2P): Dalam sistem berbagi file P2P, tabel disimpan oleh server untuk menyediakan layanan pencarian file. Tabel ini berisi hash dari file yang tersedia, sekali lagi memberikan pengidentifikasi unik tanpa menggunakan banyak memori.

Mungkin mengejutkan bahwa intisari kecil dapat mengidentifikasi secara unik setiap file di dunia. Namun ini adalah jaminan yang diberikan oleh fungsi hash anti-tabrakan, yang membuatnya berguna dalam pengaturan di atas.

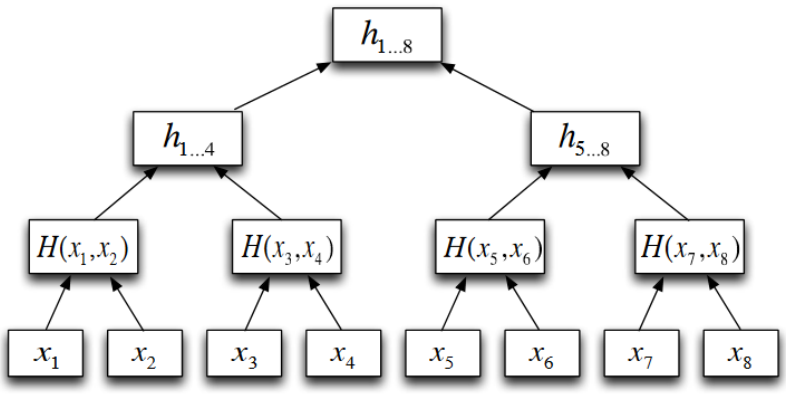
Pohon Merkle

Pertimbangkan klien yang mengunggah file x ke server. Ketika klien kemudian mengambil x, ia ingin memastikan bahwa server mengembalikan file asli yang tidak dimodifikasi. Klien dapat dengan mudah menyimpan x dan memeriksa apakah file yang diambil sama dengan x, tetapi hal tersebut menggagalkan tujuan penggunaan server. Kami mencari solusi di mana penyimpanan klien kecil.

Solusi alaminya adalah dengan menggunakan pendekatan "sidik jari" yang dijelaskan di atas. Klien dapat menyimpan ringkasan singkat secara lokal $h := H(x)$; ketika server mengembalikan file kandidat x' klien hanya perlu memeriksa bahwa $H(x') \stackrel{?}{=} h$.

Apa yang terjadi jika kita ingin memperluas solusi ini ke beberapa file $x_1, \ldots, x_t$? Ada dua cara yang jelas untuk melakukan hal ini. Salah satunya adalah dengan melakukan hash pada setiap file secara independen; klien akan menyimpan intisari $h_1, \ldots, h_t$?, dan verifikasi file yang diambil seperti sebelumnya. Kerugiannya adalah penyimpanan klien tumbuh secara linier sebesar t. Kemungkinan lainnya adalah meng-hash semua file secara bersamaan. Artinya, klien dapat menghitung $h := H(x_1, \ldots, x_t)$ dan hanya menyimpan h. Kekurangannya sekarang adalah ketika klien ingin mengambil dan memverifikasi kebenaran file ke-i x_i, klien perlu mengambil semua file untuk menghitung ulang intisari.

Pohon Merkle, yang diperkenalkan oleh Ralph Merkle, memberikan trade-off di antara kedua ekstrem ini. Pohon Merkle dihitung berdasarkan nilai masukan $x_1, \ldots, x_t$ hanyalah pohon biner dengan log kedalaman t yang inputnya ditempatkan di daun, dan nilai setiap node internal adalah hash dari nilai kedua anaknya; lihat Gambar 5.5. (Kami berasumsi t adalah pangkat 2; jika tidak, maka kami dapat menetapkan beberapa nilai masukan menjadi nol atau menggunakan pohon biner yang tidak lengkap, bergantung pada aplikasinya.)



GAMBAR 5.5: Pohon Merkle.

Memperbaiki beberapa fungsi hash H, kami menyatakan dengan $\mathcal{MT}_t$ fungsi yang mengambil t nilai input $x_1, \ldots, x_t$, menghitung pohon Merkle yang dihasilkan, dan mengeluarkan nilai akar pohon. (Fungsi hash yang dikunci menghasilkan fungsi yang dikunci $\mathcal{MT}_t$ dengan cara yang jelas.) Kita punya:

TEOREMA 5.11 Misalkan (Gen_H, H) tahan tumbukan. Lalu $((\mathrm{Gen}_H, t)$ juga tahan benturan untuk t tetap apa pun.

Dengan demikian, pohon Merkle memberikan alternatif terhadap transformasi Merkle – Damg˚ard untuk mencapai ekstensi domain untuk fungsi hash yang tahan benturan. (Namun, seperti yang dijelaskan, pohon Merkle tidak tahan benturan jika jumlah nilai masukan t dibiarkan bervariasi.)

Pohon Merkle memberikan solusi efisien untuk masalah awal kita, karena memungkinkan verifikasi input t asli mana pun menggunakan komunikasi $\mathrm{O}(\log t)$. Klien menghitung $h := t(x_1, \ldots, x_t)$, mengunggah $x_1, \ldots, x_t$ ke server, dan menyimpan h (bersama dengan jumlah file t) secara lokal. Ketika klien mengambil file ke-i, server mengirimkan x_i bersama dengan "bukti" π_i bahwa ini adalah nilai yang benar. Bukti ini terdiri dari nilai-nilai node pada pohon Merkle yang berdekatan dengan jalur dari x_i ke akar. Dari nilai-nilai ini klien dapat menghitung ulang nilai root dan memverifikasi bahwa nilai tersebut sama dengan nilai yang disimpan h. Sebagai contoh, perhatikan pohon Merkle pada Gambar 5.5. Klien menghitung $h_{1\ldots8} := \mathcal{MT}_8(x_1, \ldots, x_8)$, mengunggah $x_1, \ldots, x_8$ ke server, dan menyimpan $h_{1\ldots8}$ secara lokal. Ketika klien mengambil x_3, server mengirimkan x_3 bersama dengan x_4, $h_{1\ldots2} = H(x_1, x_2)$, dan $h_{5\ldots8} = H(H(x_5, x_6), H(x_7, x_8))$. (Jika file berukuran besar, kami mungkin ingin menghindari pengiriman file apa pun selain yang diminta klien. Hal ini dapat dengan mudah dilakukan jika kami mendefinisikan pohon Merkle di atas hash file, bukan file itu sendiri. Kami menghilangkan detailnya.) Klien menghitung $h'_{1\ldots4} := H(h_{1\ldots2}, H(x_3, x_4))$ dan $h'_{1\ldots8} := H(h_{1\ldots4}, h_{5\ldots8})$, dan kemudian memverifikasi bahwa $h_{1\ldots8} := h'_{1\ldots8}$

Jika H tahan tabrakan, server tidak mungkin mengirim file yang salah (dan bukti apa pun) yang akan menyebabkan verifikasi berhasil. Dengan menggunakan pendekatan ini, penyimpanan lokal klien bersifat konstan (tidak tergantung pada jumlah file t), dan komunikasi dari server ke klien sebanding dengan $log\ t$.

Pencirian Kata Sandi

Salah satu kegunaan fungsi hash yang paling umum dan penting dalam keamanan komputer adalah untuk perlindungan kata sandi. Bayangkan seorang pengguna mengetikkan kata sandi sebelum menggunakan laptopnya. Untuk mengautentikasi pengguna, beberapa bentuk kata sandi pengguna harus disimpan di suatu tempat di laptop mereka. Jika kata sandi pengguna disimpan dengan jelas, maka musuh yang mencuri laptop dapat membaca kata sandi pengguna dari hard drive dan kemudian login sebagai pengguna tersebut. (Sepertinya tidak ada gunanya mencoba menyembunyikan kata sandi seseorang dari penyerang yang sudah dapat membaca isi hard drive. Namun, file di hard drive mungkin dienkripsi dengan kunci yang berasal dari kata sandi pengguna, dan dengan demikian hanya dapat diakses

setelah kata sandi dimasukkan. Selain itu, pengguna kemungkinan besar akan menggunakan kata sandi yang sama di situs lain.)

Risiko ini dapat dikurangi dengan menyimpan hash kata sandi, bukan kata sandi itu sendiri. Artinya, hard drive menyimpan nilai $hpw = H(pw)$ dalam file kata sandi; nanti, ketika pengguna memasukkan kata sandinya pw, sistem operasi memeriksa apakah $H(pw) \stackrel{?}{=} hpw$ sebelum memberikan akses. Pendekatan dasar yang sama juga digunakan untuk otentikasi berbasis kata sandi di web. Sekarang, jika penyerang mencuri hard drive (atau membobol server web), yang diperolehnya hanyalah hash kata sandi dan bukan kata sandi itu sendiri.

Jika kata sandi dipilih dari ruang D kemungkinan yang relatif kecil (misalnya, D mungkin merupakan kamus kata-kata bahasa Inggris, dalam hal ini $|D| \approx 80,000$), penyerang dapat menghitung semua kemungkinan kata sandi $pw_1, pw_2, \ldots \in D$ dan, untuk setiap kandidat pw_i, periksa apakah $H(pw_i) = hpw$. Kami ingin menyatakan bahwa penyerang tidak bisa berbuat lebih baik dari ini. (Ini juga akan memastikan bahwa musuh tidak dapat mengetahui kata sandi pengguna mana pun yang memilih kata sandi yang kuat dari ruang yang besar.) Sayangnya, resistensi preimage (yaitu, satu arah) dari H tidak cukup untuk menyiratkan apa yang kita inginkan. Untuk satu hal, resistensi pragambar hanya mengatakan bahwa $H(x)$ sulit untuk dibalik ketika x dipilih secara seragam dari domain besar seperti $\{0,1\}^n$. Ia tidak menjelaskan apa pun tentang kekerasan pembalikan H ketika x dipilih dari ruang lain, atau ketika x dipilih berdasarkan distribusi lain. Selain itu, resistensi preimage tidak menjelaskan apa pun tentang jumlah waktu konkrit yang diperlukan untuk menemukan preimage. Misalnya, fungsi hash H yang menghitung $x \in \{0,1\}^n$ mengingat $H(x)$ memerlukan waktu $2^{n/2}$ masih dapat memenuhi syarat sebagai tahan preimage, namun ini berarti bahwa kata sandi 30-bit dapat dipulihkan hanya dalam waktu 2^{15}.

Jika kita memodelkan H sebagai oracle acak, maka kita dapat membuktikan secara formal keamanan yang kita inginkan, yaitu memulihkan pw dari hpw (dengan asumsi pw dipilih secara seragam dari D) memerlukan evaluasi $|D|/2$ dari H, rata-rata.

Pembahasan di atas mengasumsikan tidak ada preprocessing yang dilakukan oleh penyerang. Namun, prapemrosesan dapat digunakan untuk menghasilkan tabel besar yang memungkinkan inversi (bahkan fungsi acak!) lebih cepat daripada pencarian menyeluruh. Hal ini merupakan kekhawatiran yang signifikan dalam praktiknya: bahkan jika pengguna memilih kata sandinya sebagai kombinasi acak dari 8 karakter alfanumerik—memberikan ruang kata sandi berukuran $N = 62^8 \approx 2^{47.6}$—ada serangan yang menggunakan ruang dan waktu $N^{2/3} \approx 2^{32}$yang akan menjadi sangat efektif. Tabel hanya perlu dibuat satu kali, dan dapat digunakan untuk memecahkan ratusan ribu kata sandi jika terjadi pelanggaran server. Serangan semacam ini rutin dilakukan dalam praktiknya.

Mitigasi. Kami menjelaskan secara singkat dua mekanisme yang digunakan untuk mengurangi ancaman peretasan kata sandi; diskusi lebih lanjut dapat ditemukan dalam teks tentang keamanan komputer. Salah satu tekniknya adalah dengan menggunakan fungsi hash "lambat", atau memperlambat fungsi hash yang ada dengan menggunakan beberapa iterasi (yaitu, menghitung $H^{(I)}(pw)$ untuk $I \gg 1$). Hal ini berdampak memperlambat pengguna yang

sah sebesar faktor I, yang tidak menjadi masalah jika I disetel ke nilai "sedang" (misalnya, 1.000).

Di sisi lain, hal ini berdampak signifikan pada musuh yang mencoba memecahkan ribuan kata sandi sekaligus.

Mekanisme kedua adalah dengan memasukkan garam. Saat pengguna mendaftarkan kata sandinya, laptop/server akan menghasilkan nilai acak panjang s ("garam") yang unik untuk pengguna tersebut, dan menyimpan $(s, hpw = H(s, pw))$ alih-alih hanya menyimpan $H(pw)$ seperti sebelumnya. Karena s tidak diketahui oleh penyerang sebelumnya, pra-pemrosesan tidak efektif dan hal terbaik yang dapat dilakukan penyerang adalah menunggu sampai ia mendapatkan file kata sandi dan kemudian melakukan pencarian menyeluruh dalam waktu linier pada domain D seperti yang dibahas sebelumnya. Perhatikan juga bahwa karena garam yang berbeda digunakan untuk setiap kata sandi yang disimpan, pencarian brute force terpisah diperlukan untuk memulihkan setiap kata sandi.

Derivasi Kunci

Semua kriptosistem kunci simetris yang telah kita lihat memerlukan bit-string yang terdistribusi secara seragam untuk kunci rahasianya. Namun, sering kali lebih mudah bagi dua pihak untuk mengandalkan informasi bersama seperti kata sandi atau data biometrik yang tidak terdistribusi secara merata. (Selanjutnya, di Bab 10 kita akan melihat bagaimana pihak-pihak dapat berinteraksi untuk menghasilkan rahasia bersama dengan entropi tinggi yang tidak terdistribusi secara merata.) Para pihak dapat mencoba menggunakan informasi bersama mereka secara langsung sebagai kunci rahasia, namun secara umum hal ini tidak akan aman (karena, misalnya, semua skema kunci pribadi mengasumsikan kunci terdistribusi secara seragam). Selain itu, data yang dibagikan mungkin tidak memiliki format yang benar untuk digunakan sebagai kunci rahasia (misalnya, mungkin terlalu panjang).

Memotong rahasia bersama, atau memetakannya dengan cara ad hoc lainnya ke string dengan panjang yang benar, dapat kehilangan sejumlah besar entropi. (Kami mendefinisikan satu gagasan entropi secara lebih formal di bawah ini, namun untuk saat ini kita dapat menganggap entropi sebagai logaritma dari ruang rahasia bersama.) Misalnya, bayangkan dua pihak berbagi kata sandi yang terdiri dari 28 huruf besar acak, dan ingin menggunakan sistem kriptografi dengan kunci 128-bit. Karena ada 26 kemungkinan untuk setiap karakter, maka ada $26^{28} > 2^{130}$ kemungkinan kata sandi. Jika kata sandi dibagikan dalam format ASCII, setiap karakter disimpan menggunakan 8 bit, sehingga total panjang kata sandi adalah 224 bit. Jika para pihak memotong kata sandinya menjadi 128 bit pertama, mereka hanya akan menggunakan 16 karakter pertama kata sandinya. Namun, ini bukan string 128-bit yang terdistribusi secara merata! Faktanya, representasi ASCII dari huruf A–Z terletak antara $0x41$ dan $0x5A$; khususnya, 3 bit pertama dari setiap byte selalu 010. Ini berarti bahwa 37,5% bit kunci yang dihasilkan akan diperbaiki, dan kunci 128-bit yang diperoleh pihak-pihak tersebut hanya akan memiliki sekitar 75 bit entropi (yaitu, hanya ada 2^{75} atau lebih kemungkinan untuk kuncinya).

Yang kita butuhkan adalah solusi umum untuk mendapatkan kunci dari rahasia bersama dengan entropi tinggi (tetapi tidak harus seragam). Sebelum melanjutkan, kami mendefinisikan pengertian entropi yang kami pertimbangkan di sini.

DEFINISI 5.12 Suatu distribusi probabilitas X mempunyai m bit entropi min jika untuk setiap nilai tetap x dinyatakan bahwa $Pr_{X \leftarrow X}[X = x] \leq 2^{-m}$. Artinya, bahkan hasil yang paling mungkin terjadi dengan probabilitas paling banyak 2^{-m}

Distribusi seragam pada himpunan berukuran s memiliki entropi min $\log s$. Distribusi yang mana satu elemen muncul dengan probabilitas 1/10 dan 90 elemen masing-masing muncul dengan probabilitas 1/100 memiliki entropi min $\log 10 \approx 3.3$. Min-entropi suatu distribusi mengukur probabilitas penyerang dapat menebak nilai yang diambil sampelnya dari distribusi tersebut; strategi terbaik penyerang adalah menebak nilai yang paling mungkin, sehingga jika distribusi memiliki entropi min, penyerang akan menebak dengan benar dengan probabilitas paling banyak 2^{-m}. Hal ini menjelaskan mengapa min-entropi (dibandingkan pengertian entropi lainnya) berguna dalam konteks kita. Perpanjangan dari entropi-min, yang disebut entropi-min komputasional, didefinisikan seperti di atas, kecuali bahwa distribusi tersebut hanya diperlukan agar secara komputasi tidak dapat dibedakan dari distribusi dengan entropi-min yang diberikan. (Gagasan ketidakmampuan komputasi didefinisikan secara formal di Bagian 7.8.)

Fungsi derivasi kunci menyediakan cara untuk mendapatkan string yang terdistribusi secara merata dari distribusi mana pun dengan entropi min (komputasi) yang tinggi. Tidak sulit untuk melihat bahwa jika kita memodelkan fungsi hash H sebagai oracle acak, maka H berfungsi sebagai fungsi turunan kunci yang baik. Pertimbangkan ketidakpastian penyerang tentang $H(x)$, di mana X diambil sampelnya dari distribusi dengan entropi min (sebagai poin teknis, kami mengharuskan distribusi tersebut tidak bergantung pada H). Setiap pertanyaan penyerang terhadap H dapat dilihat sebagai "tebakan" untuk nilai X; dengan asumsi entropi min distribusi, penyerang yang membuat q kueri ke H akan menanyakan $H(x)$ dengan probabilitas paling banyak $q \cdot 2^{-m}$. Jika penyerang tidak menanyakan $x \; ke \; H$, maka $H(x)$ adalah string seragam.

Dimungkinkan juga untuk merancang fungsi turunan kunci, tanpa bergantung pada model oracle acak, menggunakan fungsi hash berkunci yang disebut ekstraktor (kuat). Kunci ekstraktor harus seragam, namun tidak perlu dirahasiakan. Salah satu standar untuk hal ini disebut HKDF; lihat referensi di akhir bab.

Skema Komitmen

Skema komitmen memungkinkan satu pihak untuk "berkomitmen" pada pesan m dengan mengirimkan nilai komitmen, sambil memperoleh sifat-sifat yang tampaknya bertentangan berikut ini:

- *Menyembunyikan:* komitmen tidak mengungkapkan apa pun tentang m.

- *Mengikat:* tidak mungkin bagi pembuat komitmen untuk mengeluarkan komitmen com yang nantinya dapat "dibuka" sebagai dua pesan berbeda m, m'. (Dalam hal ini, com benar-benar "mengikatkan" pelaku pada suatu nilai yang terdefinisi dengan baik.)

Skema komitmen dapat dilihat sebagai amplop digital: menyegel pesan di dalam amplop dan menyerahkannya kepada pihak lain memberikan privasi (sampai amplop dibuka) dan mengikat (sejak amplop disegel).

Secara formal, skema komitmen (non-interaktif) ditentukan oleh algoritma acak Gen yang mengeluarkan param parameter publik dan algoritma Com yang mengambil param dan pesan $m \in \{0,1\}^n$ dan mengeluarkan komitmen com; kita akan membuat keacakan yang digunakan oleh Com menjadi eksplisit, dan menyatakannya dengan r, Pengirim berkomitmen pada m dengan memilih seragam r, menghitung $\mathrm{com} := \mathrm{Com}\,(\mathrm{params},\ m;\ r)$, dan mengirimkannya ke penerima. Pengirim nantinya dapat menonaktifkan com dan mengungkapkan m dengan mengirimkan m, r ke penerima; penerima memverifikasi ini dengan memeriksa bahwa $\mathrm{Com}(\mathrm{params}, m;\ r)\ =?\ \mathrm{com}$.

Menyembunyikan, secara informal, berarti com tidak mengungkapkan apa pun tentang m; mengikat artinya tidak mungkin menghasilkan komitmen com yang dapat dibuka dengan dua cara berbeda. Kami mendefinisikan properti ini secara formal sekarang.

Eksperimen menyembunyikan komitmen $\mathrm{Hiding}_{A,\mathrm{Com}}(n)$:

1. Parameter $\mathrm{parameter} \leftarrow \mathrm{Gen}(1^n)$ dihasilkan.
2. Musuh $\mathcal{A}$ diberi parameter masukan, dan mengeluarkan sepasang pesan $m_0, m_1 \in \{0,1\}^n$.
3. Seragam $b \in \{0,1\}$ dipilih dan $\mathrm{com} \leftarrow \mathrm{Com}(\mathrm{params}, m_b; \mathrm{r})$ dihitung.
4. Musuh $\mathcal{A}$ diberi com dan mengeluarkan sedikit b'.
5. Hasil percobaan adalah 1 jika dan hanya jika $b' = b$.

Eksperimen pengikatan komitmen $\mathrm{Binding}_{\mathcal{A},\mathrm{Com}}(n)$:

1. Parameter $\mathrm{parameter} \leftarrow \mathrm{Gen}(1^n)$ dihasilkan.
2. A diberikan parameter input dan output $(\mathrm{com}, m, r, m', r')$.
3. Keluaran percobaan ditetapkan 1 jika dan hanya jika $m \neq m'$ dan $\mathrm{Com}(\mathrm{params}, m;\ r) = \mathrm{com} = \mathrm{Com}(\mathrm{params}, m';\ r')$.

DEFINISI 5.13 Skema komitmen Com aman jika untuk semua musuh PPT $\mathcal{A}$ terdapat pengabaian fungsi yang dapat diabaikan sehingga

$$\Pr\left[\mathrm{Hiding}_{A,\mathrm{Com}}(n) = 1\right] \leq \frac{1}{2} + \mathrm{neg}|(n)$$

Dan

$$\Pr\left[\mathrm{Binding}_{A,\mathrm{Com}}(n) = 1\right] \leq \mathrm{neg}|(n)$$

Sangat mudah untuk membangun skema komitmen aman dari oracle acak H. Untuk berkomitmen pada pesan m, pengirim memilih seragam $r \in \{0,1\}^n$ dan keluaran $com := H(m\|r)$. (Dalam model oracle acak, Gen dan Param tidak diperlukan karena H, pada

dasarnya, berfungsi sebagai parameter publik skema.) Secara intuitif, penyembunyian mengikuti fakta bahwa musuh menanyakan $H(*\|r)$ dengan probabilitas yang dapat diabaikan (karena r adalah string *n*-bit yang seragam); jika ia tidak pernah membuat kueri dalam bentuk ini maka $H(m\|r)$ tidak mengungkapkan apa pun tentang m. Pengikatan mengikuti fakta bahwa H tahan benturan.

Skema komitmen dapat dibangun tanpa ramalan acak (pada kenyataannya, dari fungsi satu arah), namun rinciannya berada di luar cakupan buku ini.

BAB 6
KONSTRUKSI PRAKTIS PRIMITIF KUNCI SIMETRIS

Dalam bab sebelumnya kita telah menunjukkan bagaimana skema enkripsi aman dan kode otentikasi pesan dapat dibangun dari kriptografi primitif seperti generator pseudorandom, permutasi pseudorandom, dan fungsi hash. Namun, satu pertanyaan yang belum kami jawab adalah bagaimana primitif kriptografi ini dibangun, atau bahkan apakah mereka ada atau tidak! Pada bab berikutnya kita akan mempelajari pertanyaan ini dari sudut pandang teoritis, dan menunjukkan konstruksi generator pseudorandom dan permutasi pseudorandom berdasarkan asumsi yang cukup lemah. (Ternyata fungsi hash lebih sulit untuk dibangun, dan tampaknya memerlukan asumsi yang lebih kuat) Dalam bab ini, fokus kita adalah pada heuristik komparatif , namun jauh lebih efisien, konstruksi primitif yang banyak digunakan dalam praktik.

Seperti telah disebutkan, konstruksi yang akan kita bahas dalam bab ini bersifat heuristik dalam arti bahwa konstruksi tersebut tidak dapat dibuktikan aman berdasarkan asumsi yang lebih lemah. Namun konstruksi ini didasarkan pada sejumlah prinsip desain, beberapa di antaranya dapat dibenarkan melalui analisis teoritis. Mungkin yang lebih penting, banyak dari konstruksi ini telah bertahan selama bertahun-tahun dalam pengawasan publik dan upaya pembacaan sandi, dan mengingat hal ini, cukup masuk akal untuk mengasumsikan keamanan konstruksi ini.

Dalam beberapa hal, tidak ada perbedaan mendasar antara asumsi, katakanlah, bahwa pemfaktoran itu sulit dan asumsi bahwa AES (sebuah cipher blok yang akan kita pelajari secara rinci nanti dalam bab ini) adalah permutasi pseudorandom. Namun terdapat perbedaan kualitatif yang signifikan antara asumsi-asumsi ini. Perbedaan utama adalah bahwa asumsi sebelumnya lebih dapat dipercaya karena tampaknya berkaitan dengan persyaratan yang lebih lemah: asumsi bahwa bilangan bulat besar sulit untuk difaktorkan bisa dibilang lebih wajar dibandingkan asumsi tersebut. bahwa AES dengan kunci seragam tidak dapat dibedakan dari permutasi acak. Perbedaan relevan lainnya antara asumsi-asumsi tersebut adalah bahwa pemfaktoran telah dipelajari lebih lama dibandingkan dengan masalah membedakan AES dari permutasi acak, dan telah dikenali sebagai masalah yang sulit jauh sebelum munculnya skema kriptografi yang berdasarkan pada asumsi tersebut.

Ringkasnya, masuk akal untuk berasumsi bahwa konstruksi yang direkomendasikan yang dijelaskan dalam bab ini aman, dan masyarakat merasa nyaman mengandalkan asumsi tersebut dalam praktiknya. Namun, akan lebih baik jika mendasarkan keamanan kriptografi primitif pada asumsi yang lebih lemah dan lebih lama. Seperti yang akan kita lihat di Bab 7, hal ini (secara prinsip) mungkin; sayangnya, konstruksi yang akan kita lihat ternyata kurang efisien dibandingkan konstruksi yang dijelaskan di sini, dan oleh karena itu tidak berguna dalam praktik.

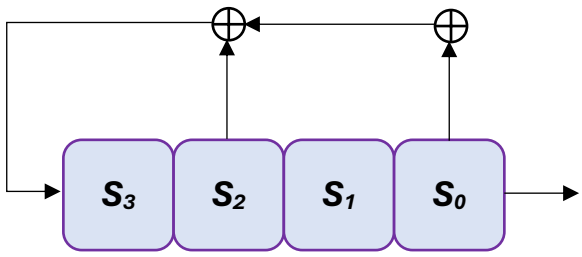
6.1 SANDI ALIRAN

Ingat dari Bagian 3.3 bahwa stream cipher ditentukan oleh dua algoritma deterministik (Init, GetBits). Algoritma Init mengambil kunci k dan vektor inisialisasi (opsional) IV sebagai masukan dan mengembalikan beberapa keadaan awal **st.** Algoritma GetBits dapat digunakan untuk menghasilkan aliran bit $y_1, y_2, \ldots$ berdasarkan **st.** Persyaratan utama dari stream cipher adalah bahwa ia harus berperilaku seperti generator pseudorandom, yaitu ketika k dipilih secara seragam secara acak, urutan yang dihasilkan adalah y_1, y_2,... harus tidak dapat dibedakan dari rangkaian bit yang seragam dan independen oleh penyerang yang dibatasi secara komputasi.

Kami telah menunjukkan pada bab sebelumnya, bahwa stream cipher dapat dibuat dengan mudah dari block cipher, yang merupakan cipher primitif yang lebih kuat. Motivasi utama untuk menggunakan konstruksi stream-cipher khusus yang diperkenalkan pada bagian ini adalah efisiensi, terutama dalam lingkungan dengan sumber daya terbatas (misalnya, dalam perangkat keras di mana mungkin ada keinginan untuk menjaga jumlah gerbang tetap kecil). Namun, serangan telah ditunjukkan terhadap beberapa konstruksi stream cipher baru-baru ini, dan keamanannya tampaknya jauh lebih lemah dibandingkan dengan kasus cipher blok. Oleh karena itu kami merekomendasikan penggunaan cipher blok (mungkin dalam mode stream-cipher) bila memungkinkan.

Register Pergeseran Umpan Balik Linier

Kita mulai dengan membahas register geser umpan balik linier (LFSR). Ini telah digunakan secara historis untuk menghasilkan bilangan pseudorandom, karena sangat efisien untuk diterapkan pada perangkat keras, dan menghasilkan keluaran yang memiliki sifat statistik yang baik. Namun, mereka tidak memberikan generator pseudorandom yang kuat secara kriptografis, dan pada kenyataannya kami akan menunjukkan serangan pemulihan kunci yang mudah pada LFSR. Meskipun demikian, LFSR dapat digunakan sebagai komponen dalam membangun stream cipher dengan keamanan yang lebih baik.



GAMBAR 6.1: Register geser umpan balik linier.

LFSR terdiri dari array n register $s_{n-1}, \ldots, s_0$ bersama dengan loop umpan balik yang ditentukan oleh sekumpulan n koefisien umpan balik $c_{n-1}, \ldots, c_0$. (Lihat Gambar 6.1.) Ukuran array disebut derajat LFSR. Setiap register menyimpan satu bit, dan keadaan st dari LFSR pada suatu waktu hanyalah kumpulan bit yang terdapat dalam register. Keadaan LFSR diperbarui dalam setiap rangkaian "clock ticks" dengan menggeser nilai-nilai di semua register ke kanan, dan menetapkan nilai baru dari register paling kiri sama dengan XOR dari beberapa subset dari

register saat ini. register, dengan subset ditentukan oleh koefisien umpan balik. Artinya, jika keadaan pada suatu waktu t adalah $S_{n-1}^{(t)}, \ldots, S_0^{(t)}$ maka keadaan setelah jam berikutnya adalah

$$S_i^{(t+1)} := S_{i+i}^{(t)}$$
$$i = 0, \ldots, n-2$$
$$S_{n-1}^{(t+1)} := \oplus_{i=0}^{n-1} c_i S_i^{(t)}$$

Gambar 6.1 menunjukkan LFSR derajat-4 dengan $c_0 = c_2 = 1$ dan $c_1 = c_3 = 0$.

Pada setiap detak jam, LFSR mengeluarkan nilai register paling kanan s0. Jika keadaan awal LFSR adalah $s_{n-1}^{(0)}, \ldots, s_0^{(0)}$, n bit pertama dari aliran keluaran persis $s_0^{(0)}, \ldots, s_{n-1}^{(0)}$. Bit keluaran selanjutnya adalah $S_{n-1}^{(1)} = \oplus_{i=0}^{n-1} c_i s_i^{(0)}$
Jika kita menyatakan bit keluaran dengan $y_1, y_2, \ldots$, dimana $y_1 = s_0^{(i-1)}$, maka

$$y_i = s_{i-1}^{(0)}, \qquad i = 1, \ldots, n$$
$$y_i = \bigoplus_{j=0}^{n-1} c_j \, y_{i-n+j-1} \qquad i > n.$$

Sebagai contoh penggunaan LFSR pada Gambar 6.1, jika keadaan awal adalah `(0,0,1,1)` maka keadaan pada beberapa periode waktu pertama adalah

```
(0,0,1,1)
(1,0,0,1)
(1,1,0,0)
(1,1,1,0)
(1,1,1,1)
```

dan keluarannya (yang dapat dibaca dari kolom paling kanan di atas) adalah aliran bit `1,1,0,0,1,...`.

Keadaan LFSR terdiri dari n bit; dengan demikian, LFSR dapat melakukan siklus melalui paling banyak 2^n kemungkinan keadaan sebelum mengulanginya. Ketika keadaan berulang, bit keluaran akan berulang, dan ini berarti urutan keluaran akan mulai berulang setelah paling banyak 2^n bit keluaran dihasilkan. LFSR dengan panjang maksimum berputar melalui semua 2^{n-1} keadaan bukan nol sebelum mengulanginya. (Perhatikan bahwa jika keadaan semua-0 terwujud maka LFSR akan tetap berada dalam keadaan itu selamanya, itulah sebabnya kami mengecualikannya.) Apakah LFSR memiliki panjang maksimal atau tidak hanya bergantung pada koefisien umpan balik; jika panjangnya maksimal maka, setelah diinisialisasi dalam keadaan bukan nol, ia akan melewati semua 2^{n-1} keadaan bukan nol. Cara mengatur koefisien

umpan balik untuk mendapatkan LFSR dengan panjang maksimal telah dipahami dengan baik, meskipun rinciannya berada di luar cakupan buku ini.

Serangan rekonstruksi. Output dari LFSR dengan panjang maksimal derajat n memiliki sifat statistik yang baik; misalnya, setiap string n-bit muncul dengan frekuensi yang kira-kira sama di aliran keluaran LFSR. Namun demikian, LFSR bukanlah generator pseudorandom yang baik untuk tujuan kriptografi karena keluarannya dapat diprediksi. Hal ini mengikuti fakta bahwa penyerang dapat merekonstruksi seluruh keadaan LFSR derajat-n setelah mengamati paling banyak 2^n bit keluaran. Untuk melihat hal ini, asumsikan keadaan awal dan koefisien umpan balik dari beberapa LFSR tidak diketahui. N bit keluaran pertama $y_1, \ldots,$ dan LFSR secara tepat mengungkapkan keadaan awal. Diberikan n bit keluaran berikutnya $y_{n+1}, \ldots, y_{2n}$, penyerang dapat membuat sistem n persamaan linier dalam n yang tidak diketahui

$$\begin{gathered} c_0, \ldots, c_{n-1}: \\ y_{n+1} = c_{n-1}\, y_n \oplus \cdots \oplus\ c_0 y_1 \\ \vdots \\ y_{2n} = c_{n-1} y_{2n-1} \oplus \cdots \oplus\ c_0 y_n \end{gathered}$$

Dapat ditunjukkan bahwa persamaan di atas bebas linier (modulo 2) untuk LFSR dengan panjang maksimal, sehingga secara unik menentukan koefisien umpan balik. (Solusinya dapat ditemukan secara efisien menggunakan aljabar linier.) Dengan mengetahui koefisien umpan balik, semua bit keluaran berikutnya dari LFSR dapat dengan mudah dihitung.

Menambahkan Nonlinier

Hubungan linier antara bit keluaran LFSR memungkinkan terjadinya serangan yang mudah. Untuk menggagalkan serangan tersebut, kita harus memperkenalkan beberapa nonlinier, yaitu beberapa operasi selain XOR. Ada beberapa pendekatan berbeda untuk melakukan hal ini, dan kami hanya mengeksplorasi beberapa di antaranya di sini.

Umpan balik nonlinier. Salah satu cara yang jelas untuk memodifikasi LFSR adalah dengan membuat putaran umpan balik menjadi nonlinier. Register geser umpan balik nonlinier (FSR) sekali lagi akan terdiri dari array register, masing-masing berisi satu bit. Seperti sebelumnya, status FSR diperbarui di setiap rangkaian detak jam dengan menggeser nilai di semua register ke kanan; namun sekarang, nilai baru dari register paling kiri adalah fungsi nonlinier dari register saat ini. Dengan kata lain, jika keadaan pada suatu waktu t adalah $s_0^{(t)}, \ldots, s_{n-1}^{(t)}$ maka keadaan setelah jam berikutnya adalah $s_0^{(t+1)}, \ldots, s_{n-1}^{(t+1)}$dengan $s_i^{(t+1)} := s_{i+1}^{(t)}$, $i = 0, \ldots, n-2$

$$s_{n-1}^{(t-1)} := g\left(s_0^{(t)}, \ldots, s_{n-1}^{(t)}\right)$$

untuk beberapa fungsi nonlinier g. Seperti sebelumnya, FSR mengeluarkan nilai register paling kanan s_0 pada setiap detak jam.

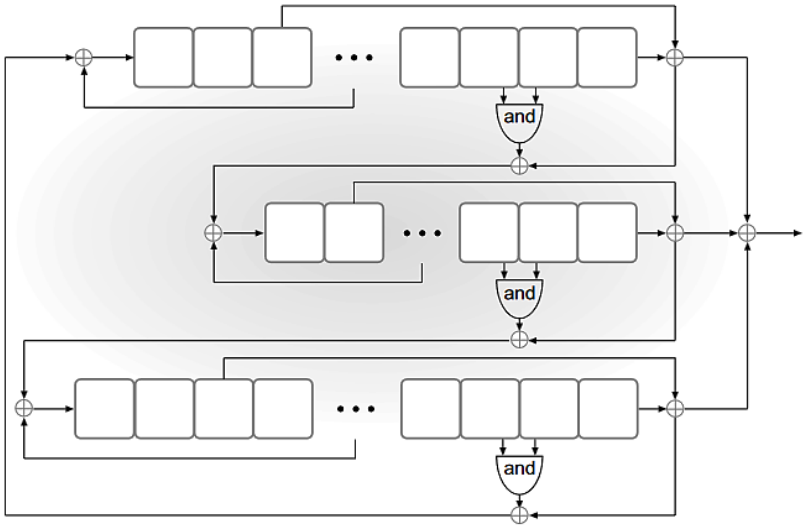
FSR nonlinier dapat dirancang dengan panjang maksimal dan keluarannya memiliki sifat statistik yang baik.

Generator kombinasi nonlinier. Pendekatan lain adalah dengan memperkenalkan nonlinier dalam urutan keluaran. Dalam kasus paling dasar, kita dapat memiliki LFSR seperti sebelumnya (di mana nilai baru dari register paling kiri dihitung lagi sebagai fungsi linier dari register saat ini), tetapi keluaran pada setiap detak jam adalah fungsi nonlinier g dari semua register saat ini, bukan hanya register paling kanan. Penting di sini bahwa g harus seimbang dalam artian bahwa $Pr[g(s_0, \ldots, s_{n-1}) = 1] \approx 1/2$ (di mana probabilitasnya melebihi pilihan seragam $s_0, \ldots, s_{n-1}$); jika tidak, meskipun mungkin sulit untuk merekonstruksi seluruh keadaan LFSR berdasarkan keluaran, aliran keluaran akan menjadi bias dan karenanya mudah dibedakan dari seragam.

Varian dari cara di atas adalah dengan menggunakan beberapa LFSR (dengan masing-masing aliran keluaran dihitung, seperti sebelumnya, hanya dengan mengambil nilai register paling kanan dari setiap LFSR), dan untuk menghasilkan aliran keluaran aktual dengan menggabungkan keluaran dari LFSR individu dalam beberapa cara nonlinier. Ini menghasilkan apa yang dikenal sebagai generator kombinasi (nonlinier). LFSR individu tidak perlu mempunyai derajat yang sama, dan pada kenyataannya panjang siklus generator kombinasi akan maksimal jika mereka tidak memiliki derajat yang sama. Di sini, kehati-hatian harus diberikan untuk memastikan bahwa aliran keluaran dari generator kombinasi tidak terlalu berkorelasi tinggi dengan aliran keluaran mana pun dari masing-masing LFSR; korelasi yang tinggi dapat menyebabkan serangan pada masing-masing LFSR, sehingga menggagalkan tujuan penggunaan beberapa LFSR dalam konstruksi.

Trivium

Untuk mengilustrasikan ide-ide dari bagian sebelumnya, kami menjelaskan secara singkat stream cipher Trivium. Stream cipher ini dipilih sebagai bagian dari portofolio proyek eSTREAM, sebuah upaya Eropa yang diselesaikan pada tahun 2008 yang tujuannya adalah untuk mengidentifikasi stream cipher baru. Trivium dirancang untuk memiliki deskripsi sederhana dan memungkinkan implementasi perangkat keras yang ringkas.



GAMBAR 6.2: Ilustrasi skema Trivium dengan (dari atas ke bawah) tiga FSR nonlinier A, B, dan C yang digabungkan.

Trivium menggunakan tiga FSR nonlinier berpasangan yang dilambangkan dengan A, B, dan C dan masing-masing memiliki derajat 93, 84, dan 111. (Lihat Gambar 6.2.) Status **st** dari Trivium hanyalah 288 bit yang terdiri dari nilai-nilai di semua register FSR ini. Algoritme GetBits untuk Trivium bekerja sebagai berikut: Pada setiap detak jam, output dari setiap FSR adalah XOR dari register paling kanan dan satu register tambahan; keluaran Trivium adalah XOR dari bit keluaran ketiga FSR. FSR digabungkan: pada setiap detak jam, nilai baru dari register paling kiri dari setiap FSR dihitung sebagai fungsi dari salah satu register di FSR yang sama dan subset register dari FSR kedua. (Fungsi umpan balik untuk A bergantung pada satu register A dan empat register C; fungsi umpan balik untuk B bergantung pada satu register B dan empat register A; dan fungsi umpan balik untuk C bergantung pada satu register C dan empat register dari B.) Fungsi umpan balik dalam setiap kasus adalah nonlinier.

Algoritme Init Trivium menerima kunci 80-bit dan *IV* 80-bit. Kunci dimuat ke dalam 80 register paling kiri di A, dan *IV* dimuat ke dalam 80 register paling kiri di B. Register sisanya disetel ke 0, kecuali untuk tiga register paling kanan di C, yaitu disetel ke 1. Kemudian GetBits dijalankan 4.288 kali (dengan output dibuang), dan status yang dihasilkan diambil sebagai st0. Sampai saat ini, tidak ada serangan kriptanalitik yang lebih baik daripada pencarian kunci menyeluruh yang diketahui terhadap sandi Trivium lengkap.

RC4

LFSR efisien ketika diimplementasikan pada perangkat keras, namun memiliki kinerja yang buruk pada perangkat lunak. Untuk alasan ini, desain alternatif dari stream cipher telah dieksplorasi. Contoh yang menonjol adalah RC4, yang dirancang oleh Ron Rivest pada tahun 1987. RC4 luar biasa karena kecepatan dan kesederhanaannya, dan tahan terhadap serangan serius selama beberapa tahun. Ini banyak digunakan saat ini, dan kami mendiskusikannya karena alasan ini; namun kami memperingatkan pembaca bahwa serangan baru-baru ini telah menunjukkan kelemahan kriptografi yang serius pada RC4 dan sebaiknya tidak digunakan lagi.

ALGORITHM 6.1
Init algorithm for RC4

Input: 16-byte key k
Output: Initial state (S, i, j)
(Note: All addition is done modulo 256)

for $i = 0$ to 255:
  $S[i] := i$
  $k[i] := k[i \bmod 16]$
$j := 0$
for $i = 0$ to 255:
  $j := j + S[i] + k[i]$
  Swap $S[i]$ and $S[j]$
$i := 0, \quad j := 0$
return (S, i, j)

Keadaan RC4 adalah array 256-byte S, yang selalu berisi permutasi elemen 0, . . . , 255, beserta dua nilai $i, j \in \{0, \ldots, 255\}$. Algoritma Init untuk RC4 disajikan sebagai Algoritma 6.1.

Untuk mempermudah kita asumsikan kunci k berukuran 16-byte (128-bit), meskipun algoritma dapat menangani kunci dengan panjang antara 1 byte dan 256 byte. Kami mengindeks byte S sebagai $S[0], \ldots, S[255]$, dan byte kunci sebagai $k[0], \ldots, k[15]$.

Selama inisialisasi, S pertama kali diatur ke permutasi identitas (yaitu, dengan $S[i] = i$ untuk semua i) dan k diperluas hingga 256 byte dengan pengulangan. Kemudian setiap entri S ditukar setidaknya satu kali dengan entri S lainnya di lokasi "acak semu". Indeks i, j diatur ke 0, dan (S, i, j) dikeluarkan sebagai keadaan awal.

Keadaan tersebut kemudian digunakan untuk menghasilkan urutan bit keluaran, seperti yang ditunjukkan pada Algoritma 6.2. Indeks i hanya bertambah (modulo 256), dan j diubah dalam beberapa cara "acak semu". Entri $S[i]$ dan $S[j]$ ditukar, dan nilai S pada posisi $S[i] + S[j]$ (sekali lagi dihitung modulo 256) adalah output. Perhatikan bahwa setiap entri S ditukar dengan beberapa entri S lainnya (mungkin itu sendiri) setidaknya sekali setiap 256 iterasi, memastikan "pencampuran" yang baik dari permutasi S.

ALGORITHM 6.2
GetBits algorithm for RC4

Input: Current state (S, i, j)
Output: Output byte y; updated state (S, i, j)
(Note: All addition is done modulo 256)

$i := i + 1$
$j := j + S[i]$
Swap $S[i]$ and $S[j]$
$t := S[i] + S[j]$
$y := S[t]$
return $(S, i, j), y$

RC4 tidak dirancang untuk menggunakan infus sebagai masukan; namun, dalam praktiknya, IV sering kali digabungkan hanya dengan menggabungkannya dengan kunci sebenarnya k' sebelum inisialisasi. Artinya, IV acak dengan panjang yang diinginkan dipilih, k diatur sama dengan rangkaian IV dan k' (ini dapat dilakukan dengan menambahkan atau menambahkan IV), dan kemudian Init dijalankan seperti pada Algoritma 6.1 untuk menghasilkan keadaan awal. Bit keluaran kemudian diproduksi menggunakan Algoritma 6.2 persis seperti sebelumnya. Dengan asumsi RC4 digunakan dalam mode tidak tersinkronisasi, IV kemudian akan dikirim secara jelas ke penerima—yang mungkin sudah memiliki kunci sebenarnya k'—sehingga memungkinkan mereka untuk menghasilkan keadaan awal yang sama dan karenanya aliran keluaran yang sama. Metode penggabungan IV ini digunakan dalam standar enkripsi Wired Equivalent Privacy (WEP) untuk melindungi komunikasi di jaringan nirkabel 802.11.

Kita harus prihatin dengan cara yang relatif ad hoc dalam memodifikasi RC4 untuk menerima infus. Bahkan jika RC4 adalah stream cipher yang aman ketika menggunakan (hanya) kunci seperti yang dirancang semula, tidak ada alasan untuk percaya bahwa itu harus

aman ketika dimodifikasi untuk menggunakan IV dengan cara ini. Memang, bertentangan dengan kuncinya, IV diungkapkan kepada penyerang (karena dikirim dengan jelas); lebih jauh lagi, menggunakan IV yang berbeda dengan kunci tetap yang sama k'—seperti yang akan dilakukan saat menggunakan RC4 dalam mode tidak tersinkronisasi—berarti nilai terkait k sedang digunakan untuk menginisialisasi status RC4. Seperti yang akan kita lihat di bawah, kedua masalah ini menyebabkan serangan ketika RC4 digunakan dengan cara ini.

Serangan pada RC4. Meskipun RC4 tersebar luas di sistem modern, berbagai serangan terhadap RC4 telah diketahui selama beberapa tahun. Oleh karena itu, RC4 tidak boleh digunakan lagi; sebagai gantinya, stream cipher atau block cipher yang lebih modern harus digunakan sebagai gantinya.

Kami mulai dengan mendemonstrasikan serangan statistik sederhana pada RC4 yang tidak bergantung pada penggunaan infus oleh pihak yang jujur. Serangan ini mengeksploitasi fakta bahwa byte keluaran kedua RC4 (sedikit) bias ke arah 0. Misalkan S_t menunjukkan keadaan array S setelah t iterasi GetBits, dengan S_0 menunjukkan keadaan awal. Memperlakukan S_0 (secara heuristik) sebagai permutasi seragam dari $\{0,\ldots,255\}$, dengan probabilitas $1/256 \cdot (1 — 1/255) \approx 1/256$, menyatakan bahwa $S_0[2] = 0$ dan $X \stackrel{\text{def}}{=} S_0[0] \neq 2$. Asumsikan sejenak bahwa ini adalah kasus. Pada iterasi pertama GetBits, nilai i bertambah menjadi 1, dan j diset sama dengan $S_0[i] = S_0[1] = X$. Kemudian $S_0[1]$ dan $S_0[X]$ ditukar, sehingga pada di akhir iterasi kita mempunyai $S_1[X] = S_0[0] = X$. Pada iterasi kedua, i bertambah menjadi 2 dan j diberi nilai

$$j + S_1[i] = X + S_1[2] = X + S_0[2] = X$$

karena $S_0[2] = 0$. Kemudian $S_1[2]$ dan $S_1[X]$ ditukar, sehingga $S_2[X] = S_1[2] = S_0[2] = 0$ dan $S_2[2] = S_1[X] = X$. Terakhir, nilai S_2 pada posisi $S_2[i] + S_2[j] = S_2[2] + S_2[x] = X$ adalah keluaran; inilah tepatnya nilai $S_2[X] = 0$. Ketika $S_0[2] = 0$ byte keluaran kedua terdistribusi secara merata. Maka secara keseluruhan, probabilitas byte keluaran kedua adalah 0 adalah

$$\Pr[S_0[2] = 0 \text{ dan } S_0[1] \neq 2] + \frac{1}{256} \cdot (1 - \Pr[S_0[2] = 0 \text{ dan } S_0[1] \neq 2])$$
$$= \frac{1}{256} + \frac{1}{256} \cdot \left(1 - \frac{1}{256}\right) \approx \frac{2}{256}$$

atau dua kali lipat dari apa yang diharapkan untuk nilai yang seragam.

Hal di atas mungkin tidak dipandang sebagai serangan yang serius, meskipun tampaknya menunjukkan adanya masalah struktural yang mendasari RC4. Serangan yang lebih serius terhadap RC4 mungkin terjadi ketika IV dimasukkan dengan menambahkannya ke kunci. Serangan ini dapat digunakan untuk memulihkan kunci, berapa pun panjangnya, dan karenanya lebih serius dibandingkan serangan pembeda seperti yang dijelaskan di atas. Yang penting, serangan ini dapat digunakan untuk sepenuhnya melanggar standar enkripsi WEP yang disebutkan sebelumnya, dan berpengaruh dalam penggantian standar tersebut.

Inti serangannya adalah cara untuk memperluas pengetahuan dari n byte pertama k ke pengetahuan $(n + 1)$ byte pertama dari k. Perhatikan bahwa ketika IV ditambahkan ke kunci sebenarnya k' (jadi $k = IV||k'$), beberapa byte pertama k diberikan kepada penyerang secara gratis! Jika IV panjangnya n byte, maka musuh dapat menggunakan serangan ini untuk memulihkan $(n + 1)$ byte pertama dari k (yang merupakan byte pertama dari kunci sebenarnya k'), lalu byte berikutnya dari k, dan seterusnya, hingga menyimpulkan seluruh kunci.

Asumsikan IV panjangnya 3 byte, seperti halnya WEP. Penyerang menunggu hingga dua byte pertama IV memiliki bentuk tertentu. Serangan dapat dilakukan dengan beberapa kemungkinan untuk dua byte pertama dari IV, tetapi kita melihat kasus di mana IV mengambil bentuk $IV = (3, 255, X)$ untuk X byte yang berubah-ubah. Artinya, $k[0] = 3, k[1] = 255$, dan $k[2] = X$ pada Algoritma 6.1. Kita dapat memeriksa bahwa setelah empat iterasi pertama dari loop kedua Init, kita sudah mendapatkannya

Persamaan 6.1

$$S[0] = 3, S[1] = 0, S[3] = X + 6 + k[3]$$

Dalam 252 iterasi berikutnya dari algoritma Init, i selalu lebih besar dari 3. Jadi nilai $S[0], S[1]$, dan $S[3]$ selanjutnya tidak diubah selama j tidak pernah mengambil nilai 0 , 1, atau 3. Jika kita (secara heuristik) memperlakukan j mengambil nilai yang seragam di setiap iterasi, ini berarti $S[0], S[1]$, dan $S[3]$ selanjutnya tidak dimodifikasi dengan probabilitas $(253/256)^{252} \approx 0{,}05$, atau 5% dari keseluruhan. Dengan asumsi demikian, keluaran byte pertama oleh GetBits adalah $S[3] = X + 6 + k[3]$; karena X diketahui, maka diperoleh $k[3]$.

Jadi, penyerang mengetahui bahwa 5% byte pertama keluaran berkorelasi dengan $k[3]$ seperti dijelaskan di atas. (Ini jauh lebih baik daripada tebakan acak, yang benar $1/256 = 0{,}4\%$ dari keseluruhan waktu.) Jadi, dengan mengumpulkan cukup banyak sampel dari byte pertama keluaran—untuk beberapa IV yang memiliki bentuk yang benar—penyerang mendapat perkiraan keyakinan tinggi untuk $k[3]$.

6.2 SANDI BLOK

Ingat dari Bagian 3.5 bahwa cipher blok adalah permutasi berkunci yang efisien F : $\{0,1\}^n \times \{0,1\}^\ell \to \{0,1\}^\ell$. Ini berarti fungsi F_k yang didefinisikan oleh $F_k(x) \stackrel{\text{def}}{=} F(k, x)$ adalah sebuah bijeksi (yaitu, permutasi), dan terlebih lagi F_k dan invers k-nya F_k^{-1} dapat dihitung secara efisien jika diberikan k. Kami menyebut n sebagai panjang kunci dan ℓ sebagai panjang blok F , dan di sini kami secara eksplisit mengizinkan keduanya berbeda. Panjang kunci dan panjang blok sekarang merupakan konstanta tetap, sedangkan di Bab 3 keduanya dipandang sebagai fungsi parameter keamanan. Hal ini menempatkan kita pada kondisi keamanan konkrit dibandingkan keamanan asimtotik. Persyaratan keamanan konkrit untuk cipher blok cukup ketat, dan cipher blok umumnya hanya dianggap "baik" jika serangan yang paling diketahui (tanpa pra-pemrosesan) mempunyai waktu. kompleksitasnya kira-kira setara dengan pencarian kunci secara brute force. Jadi, jika sebuah sandi dengan panjang kunci $n = 256$ dapat dipecahkan pada waktu 2^{128}, sandi tersebut (umumnya) dianggap tidak aman

meskipun serangan 2^{128} kali masih tidak dapat dilakukan. Sebaliknya, dalam keadaan asimtotik, serangan dengan kompleksitas $2^{n/2}$ tidak dianggap efisien karena memerlukan waktu eksponensial (dan dengan demikian, sandi yang memungkinkan terjadinya serangan mungkin masih memenuhi definisi permutasi pseudorandom). Namun, dalam situasi nyata, kita harus mengkhawatirkan kompleksitas serangan yang sebenarnya (bukannya perilaku tanpa gejala). Selain itu, terdapat kekhawatiran bahwa adanya serangan semacam itu dapat mengindikasikan kelemahan mendasar dalam desain sandi.

Cipher blok dirancang untuk berperilaku, setidaknya, sebagai permutasi pseudo-acak (kuat); lihat Definisi 3.28. Pemodelan cipher blok sebagai permutasi pseudorandom memungkinkan pembuktian keamanan untuk konstruksi berdasarkan cipher blok, dan juga memperjelas persyaratan yang diperlukan dari cipher blok. Pemahaman yang kuat tentang apa yang ingin dicapai oleh cipher blok sangat penting dalam desainnya. Pandangan bahwa cipher blok harus dimodelkan sebagai permutasi pseudorandom, setidaknya di masa lalu, memberikan pengaruh besar dalam desainnya. Sebagai contoh, permintaan proposal untuk Standar Enkripsi Lanjutan (AES) terkini yang akan kita temui nanti di bab ini menyatakan kriteria evaluasi berikut:

Keamanan yang diberikan oleh suatu algoritma adalah faktor yang paling penting. . . . Algoritma akan dinilai berdasarkan faktor-faktor berikut. . .

Sejauh mana keluaran algoritma tidak dapat dibedakan dari permutasi acak. . .

Cipher blok modern cocok untuk semua konstruksi yang menggunakan permutasi pseudorandom (atau fungsi pseudorandom) yang telah kita lihat dalam buku ini.

Seringkali, cipher blok dirancang (dan diasumsikan) untuk memenuhi sifat keamanan yang lebih kuat. Terlepas dari kenyataan bahwa cipher blok bukanlah skema enkripsi, terminologi standar untuk serangan terhadap cipher blok F adalah:

- Dalam serangan teks biasa yang diketahui, penyerang diberikan pasangan input/output $\{x_i, F_k(x_i)\}$ (untuk kunci k yang tidak diketahui), dengan $\{x_i\}$ di luar kendali penyerang.
- Dalam serangan teks biasa yang dipilih, penyerang diberikan $\{F_k(x_i)\}$ (sekali lagi, untuk kunci k yang tidak diketahui) untuk serangkaian masukan $\{x_i\}$ yang dipilih oleh penyerang.
- Dalam serangan teks sandi terpilih, penyerang diberikan $\{F_k(x_i)\}$ untuk $\{x_i\}$ yang dipilih penyerang, serta $\{F_k^{-1}(y_i)\}$ untuk $\{y_i\}$ yang dipilih.

Selain menggunakan cara di atas untuk membedakan F_k dari permutasi seragam, kita juga akan tertarik pada serangan pemulihan kunci di mana penyerang dapat memulihkan kunci k setelah berinteraksi dengan F_k. (Ini lebih kuat daripada membedakan F_k dari seragam.)

Sehubungan dengan taksonomi ini, permutasi pseudorandom tidak dapat dibedakan dari permutasi seragam dalam serangan teks biasa terpilih, sedangkan permutasi pseudorandom kuat tidak dapat dibedakan bahkan dalam serangan teks sandi terpilih.

Jaringan Substitusi-Permutasi

Sebuah cipher blok harus berperilaku seperti permutasi acak. Ada $2^{\ell}!$ permutasi pada string ℓ-bit, jadi merepresentasikan permutasi arbitrer dengan panjang blok ℓ-bit memerlukan $\log(2^{\ell}!) \approx \ell \cdot 2^{\ell}$ bit. Hal ini tidak praktis untuk $\ell > 20$ dan tidak mungkin untuk $\ell > 50$. (Ke depan, cipher blok modern memiliki panjang blok $\ell \geq 128$.) Tantangan ketika merancang sebuah cipher blok adalah membangun sekumpulan permutasi dengan deskripsi yang ringkas (yaitu, kunci pendek) yang berperilaku seperti permutasi acak. Secara khusus, seperti mengevaluasi permutasi acak pada dua masukan yang hanya berbeda satu bit akan menghasilkan dua keluaran (hampir) independen (keduanya tidak sepenuhnya independen karena tidak bisa sama), demikian pula dengan mengubah satu bit masukan ke $F_k(\cdot)$, dimana k seragam dan tidak diketahui oleh penyerang, akan memberikan hasil yang (hampir) independen. Ini menyiratkan bahwa perubahan satu bit pada masukan harus "mempengaruhi" setiap bit keluaran. (Perhatikan bahwa ini tidak berarti bahwa semua bit keluaran akan diubah—ini akan menjadi perilaku yang berbeda dari yang diharapkan untuk permutasi acak. Sebaliknya, yang kami maksudkan secara informal adalah bahwa setiap bit keluaran diubah dengan probabilitas kira-kira setengahnya.) Ini membutuhkan usaha untuk mencapainya.

Paradigma kebingungan-difusi. Selain karyanya tentang kerahasiaan sempurna, Shannon juga memperkenalkan paradigma dasar untuk menyusun permutasi yang ringkas dan tampak acak. Ide dasarnya adalah membuat permutasi F yang tampak acak dengan panjang blok besar dari banyak permutasi acak (atau tampak acak) kecil f_i dengan panjang blok kecil. Mari kita lihat cara kerjanya pada tingkat paling dasar. Katakanlah kita ingin F memiliki panjang blok 128 bit. Kita dapat mendefinisikan F sebagai berikut: kunci k untuk F akan menentukan 16 permutasi $f_1, \ldots, f_{16}$ yang masing-masing memiliki panjang blok 8-bit (1-byte). Diberikan masukan $x \in \{0,1\}^{128}$, kita parsing sebagai 16 byte $x_1 \ldots \ldots x_{16}$lalu atur (Persamaan 6.2)

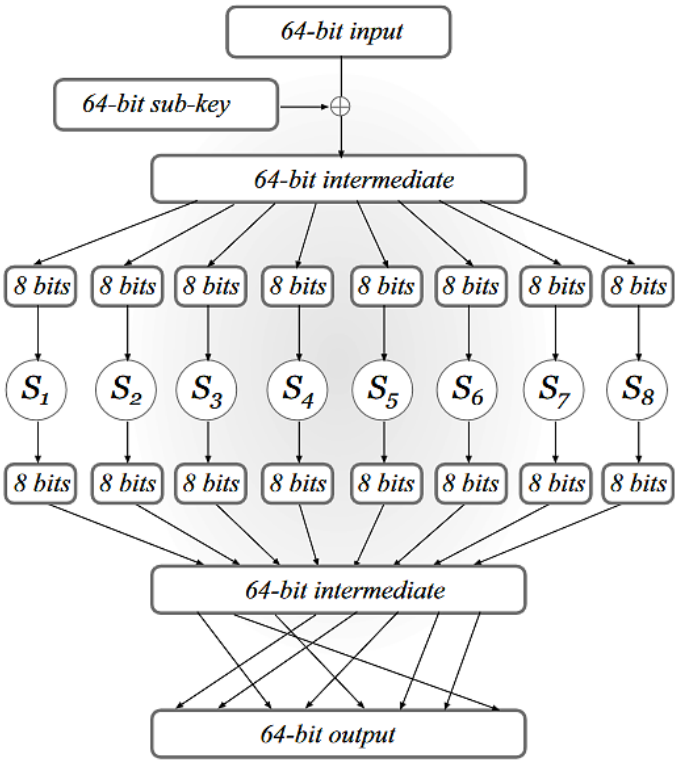
$$F_k(x) = f_1(x_1) \| \cdots \| f_{16}(x_{16})$$

Fungsi bulat ini $\{f_i\}$ dikatakan menimbulkan kebingungan pada F .

Namun harus segera jelas bahwa F sebagaimana didefinisikan di atas bukanlah pseudorandom. Khususnya, jika x dan x' hanya berbeda pada bit pertamanya, maka $F_k(x)$ dan $F_k(x')$ hanya akan berbeda pada byte pertamanya (apa pun kuncinya k). Sebaliknya, jika F adalah permutasi yang benar-benar acak maka perubahan bit pertama masukan diperkirakan akan mempengaruhi semua byte keluaran.

Untuk alasan ini, langkah difusi diperkenalkan dimana bit-bit keluaran di permutasi, atau "dicampur," menggunakan permutasi pencampuran. Hal ini mempunyai efek menyebarkan perubahan lokal (misalnya perubahan pada byte pertama) ke seluruh blok. Langkah-langkah kebingungan/difusi—bersama-sama disebut putaran—diulangi beberapa kali. Hal ini membantu memastikan bahwa mengubah satu bit masukan akan mempengaruhi semua bit keluaran.

Sebagai contoh, cipher blok dua putaran yang mengikuti pendekatan ini akan beroperasi sebagai berikut. Pertama, kebingungan terjadi dengan menghitung hasil antara $f_1(x_1)\|\cdots\|f_{16}(x_{16})$ seperti pada Persamaan (6.2). Potongan-potongan hasilnya kemudian "dikocok", atau diurutkan ulang, untuk menghasilkan x'. Kemudian $f'_1(x'_1)\|\cdots\|f'_{16}(x'_{16})$ dihitung (di mana $x' = x'_1 \cdots x'_{16}$), menggunakan fungsi yang mungkin berbeda f'_1, dan bit-bit hasilnya diijinkan untuk menghasilkan keluaran x''. $\{f_i\}$, $\{f'_i\}$ dan permutasi pencampuran bisa acak dan bergantung pada kunci, seperti yang telah kami jelaskan di atas. Namun dalam praktiknya, mereka dirancang dan diperbaiki secara khusus, dan kuncinya digabungkan dengan cara yang berbeda, seperti yang akan kami jelaskan di bawah.



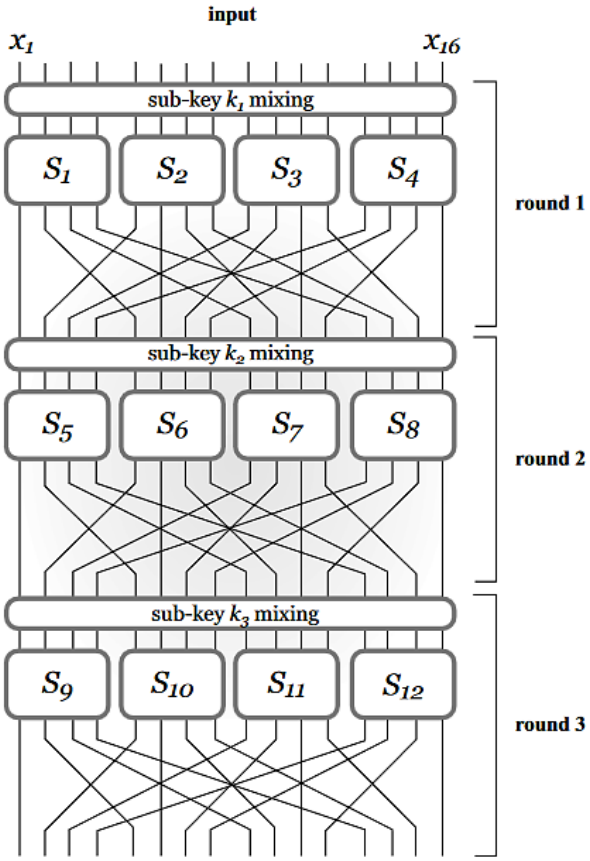
GAMBAR 6.3: Satu putaran jaringan substitusi-permutasi.

Jaringan substitusi-permutasi. Jaringan substitusi-permutasi (SPN) dapat dipandang sebagai implementasi langsung dari paradigma difusi kebingungan. Perbedaannya adalah sekarang fungsi bulat mempunyai bentuk tertentu dan tidak dipilih dari himpunan semua kemungkinan permutasi pada domain tertentu. Secara khusus, daripada meminta (sebagian dari) kunci k menentukan permutasi sembarang f, kita malah menetapkan "fungsi substitusi" publik (yaitu, permutasi) S yang disebut S-box, dan kemudian biarkan k mendefinisikan fungsi tersebut f diberikan oleh $f(x) = S(k \oplus x)$.

Untuk melihat cara kerjanya secara nyata, pertimbangkan SPN dengan panjang blok 64-bit berdasarkan kumpulan S-box 8-bit (1-byte) $S_1, \ldots, S_8$ (Lihat Gambar 6.3.) Evaluasi cipher berlangsung dalam serangkaian putaran, di mana pada setiap putaran kita menerapkan urutan operasi berikut ke masukan 64-bit x pada putaran tersebut (masukan ke putaran pertama hanyalah masukan ke sandi):

1. Pencampuran kunci: Himpunan $x := x \oplus k$, dimana k adalah subkunci putaran saat ini;
2. Substitusi: Himpunan $x := S_1(x_1)\|\cdots\|S_8(x_8)$, dimana $x_1$adalah byte ke-i dari x;
3. Permutasi: Mengurutkan bit-bit x untuk mendapatkan keluaran putaran.

Output dari setiap putaran dijadikan masukan untuk putaran berikutnya. Setelah putaran terakhir ada langkah pencampuran kunci terakhir, dan hasilnya adalah keluaran sandi. (Dengan prinsip Kerckhoffs, kami berasumsi bahwa S-box dan permutasi pencampuran bersifat publik dan diketahui oleh penyerang mana pun. Ini berarti bahwa tanpa langkah pencampuran kunci akhir, langkah substitusi dan permutasi terakhir tidak akan memberikan tambahan apa pun. keamanan karena tidak bergantung pada kunci.) Gambar 6.4 menunjukkan struktur tingkat tinggi SPN dengan panjang blok 16-bit dan kumpulan S-box 4-bit berbeda yang digunakan dalam setiap putaran.



GAMBAR 6.4: Jaringan substitusi-permutasi.

Sub-kunci yang berbeda (atau tombol bulat) digunakan di setiap putaran. Kunci sebenarnya dari cipher blok kadang-kadang disebut kunci master. Sub-kunci bulat diturunkan dari kunci master menurut jadwal kunci. Penjadwalan kunci seringkali sederhana dan dapat bekerja hanya dengan mengambil subset bit kunci master yang berbeda, meskipun penjadwalan yang lebih kompleks juga dapat ditentukan. SPN putaran-r mempunyai r (penuh) putaran pencampuran kunci, substitusi kotak-S, dan penerapan permutasi pencampuran, diikuti dengan langkah pencampuran kunci terakhir. (Ini berarti bahwa dalam SPN putaran-r, sub-kunci $r + 1$ digunakan.)

SPN apa pun dapat dibalik (berikan kuncinya). Untuk melihat ini, kami menunjukkan bahwa dengan keluaran SPN dan kuncinya, masukan tersebut dapat dipulihkan. Hal ini cukup untuk menunjukkan bahwa satu putaran dapat dibalik; ini berarti seluruh SPN dapat dibalik dengan bekerja dari putaran terakhir kembali ke awal. Namun membalikkan satu putaran itu mudah: permutasi pencampuran dapat dengan mudah dibalik karena ini hanyalah pengurutan ulang bit-bit. Karena S-box adalah permutasi (yaitu satu-ke-satu), maka S-box juga dapat dibalik. Hasilnya kemudian dapat di-XOR dengan sub-kunci yang sesuai untuk mendapatkan masukan asli. Karena itu:

PROPOSISI 6.3 Misalkan F adalah fungsi berkunci yang didefinisikan oleh SPN yang semua S-boxnya merupakan permutasi. Maka terlepas dari jadwal kunci dan jumlah putaran, F_k adalah permutasi untuk sembarang k.

Jumlah putaran, bersama dengan pilihan yang tepat dari S-box, permutasi pencampuran, dan jadwal kunci, adalah hal-hal yang pada akhirnya menentukan apakah suatu cipher blok tertentu mudah dipecahkan atau sangat aman. Kita sekarang membahas prinsip dasar di balik desain S-box dan permutasi pencampuran.

Efek longsoran salju. Seperti disebutkan berulang kali, properti penting dalam setiap blok cipher adalah bahwa perubahan kecil pada masukan harus "mempengaruhi" setiap bit keluaran. Kami menyebutnya sebagai efek longsoran salju. Salah satu cara untuk menimbulkan *efek avalanche* dalam jaringan substitusi-permutasi adalah dengan memastikan bahwa dua properti berikut berlaku (dan cukup banyak putaran yang digunakan):

1. S-box dirancang sedemikian rupa sehingga mengubah satu bit masukan ke S-box akan mengubah setidaknya dua bit keluaran S-box.
2. Permutasi pencampuran dirancang sedemikian rupa sehingga bit keluaran dari S-box tertentu digunakan sebagai masukan ke beberapa S-box pada putaran berikutnya.

Untuk melihat bagaimana hal ini menghasilkan efek avalanche, setidaknya secara heuristik, asumsikan bahwa S-box semuanya sedemikian rupa sehingga mengubah satu bit input dari S-box akan menghasilkan perubahan tepat dua bit output dari S-box. kotak, dan permutasi pencampuran dipilih seperti yang disyaratkan di atas. Untuk lebih konkritnya, asumsikan S-box mempunyai ukuran input/output 8 bit, dan panjang blok cipher adalah 128 bit. Pertimbangkan sekarang apa yang terjadi ketika cipher blok diterapkan pada dua input yang berbeda dalam satu bit:

1. Setelah putaran pertama, nilai antara berbeda tepat dua posisi bit. Hal ini karena XOR pada sub-kunci saat ini mempertahankan perbedaan 1-bit pada nilai antara, sehingga input ke semua S-box kecuali satu adalah identik. Dalam satu S-box yang masukannya berbeda, keluaran S-box menyebabkan perbedaan 2-bit. Permutasi pencampuran yang diterapkan pada hasil mengubah posisi perbedaan ini, namun mempertahankan perbedaan 2-bit.
2. Permutasi pencampuran yang diterapkan pada akhir putaran pertama menyebarkan dua posisi bit dimana hasil antara berbeda ke dalam dua kotak S yang berbeda pada

putaran kedua. Hal ini tetap berlaku bahkan setelah subkunci yang sesuai di-XOR dengan hasil putaran sebelumnya. Jadi, pada putaran kedua sekarang ada dua S-box yang menerima input berbeda satu bit. Jadi, pada akhir putaran kedua, nilai antara berbeda 4 bit.

3. Melanjutkan argumen yang sama, kami memperkirakan 8 bit nilai antara akan terpengaruh setelah putaran ke-3, 16 bit akan terpengaruh setelah putaran ke-4, dan seluruh 128 bit keluaran akan terpengaruh pada akhir putaran ke-7.

Poin terakhir kurang tepat dan kemungkinan besar akan terdapat lebih sedikit perbedaan dari yang diperkirakan pada akhir suatu putaran. (Faktanya, hal ini harus terjadi karena outputnya juga tidak boleh berbeda di semua bitnya.) Karena alasan ini, biasanya menggunakan lebih dari 7 putaran. Namun, analisis di atas memberikan batas bawah pada jumlah putaran: jika kurang dari 7 putaran yang digunakan maka harus ada beberapa set bit keluaran yang tidak terpengaruh oleh perubahan satu bit pada masukan, yang menyiratkan bahwa hal itu akan terjadi. mungkin untuk membedakan sandi dari permutasi acak.

Kita mungkin berharap bahwa cara "terbaik" untuk merancang S-box adalah dengan memilihnya secara acak (dengan batasan bahwa S-box tersebut merupakan permutasi). Menariknya, hal ini ternyata tidak terjadi, setidaknya jika kita ingin memenuhi kriteria desain yang disebutkan sebelumnya. Pertimbangkan kasus S-box yang beroperasi pada input 4-bit dan misalkan x dan x' menjadi dua nilai yang berbeda. Misalkan $y = S(x)$, dan sekarang pertimbangkan untuk memilih seragam $y' \neq y$ sebagai nilai $S(x')$. Ada 4 string yang berbeda dari y hanya dalam 1 bit, sehingga dengan probabilitas 4/15 kita akan memilih y' yang tidak berbeda dari y dalam dua bit atau lebih. Masalahnya menjadi lebih rumit ketika kita mempertimbangkan semua pasangan input yang berbeda dalam satu bit.

Kami menyimpulkan berdasarkan contoh ini bahwa, sebagai aturan umum, S-box harus dirancang dengan hati-hati daripada dipilih secara acak. S-box acak juga tidak bagus untuk bertahan dari serangan seperti yang akan kita tunjukkan di Bagian 6.2.

Jika sebuah cipher blok juga harus bersifat pseudorandom kuat, maka efek avalanche juga harus diterapkan pada kebalikannya. Artinya, mengubah satu bit keluaran akan mempengaruhi setiap bit masukan. Untuk ini akan berguna jika S-box dirancang sedemikian rupa sehingga mengubah satu bit output dari S-box akan mengubah setidaknya dua bit input ke S-box. Mencapai efek longsoran salju di kedua arah adalah alasan lain untuk semakin meningkatkan jumlah putaran.

Menyerang SPN Putaran Tereduksi

Pengalaman, bersama dengan upaya kriptanalitik selama bertahun-tahun, menunjukkan bahwa jaringan substitusi-permutasi adalah pilihan yang baik untuk membangun permutasi pseudorandom selama pemilihan S-box, permutasi pencampuran, dan jadwal kunci dilakukan dengan hati-hati. Standar Enkripsi Lanjutan, memiliki struktur yang mirip dengan jaringan substitusi-permutasi yang dijelaskan di atas, dan secara luas diyakini sebagai permutasi pseudorandom yang kuat.

Kekuatan sandi F yang dibuat dengan cara ini sangat bergantung pada jumlah putaran. Untuk mendapatkan lebih banyak wawasan tentang jaringan substitusi-permutasi, kami akan

mendemonstrasikan serangan terhadap SPN yang mempunyai putaran yang sangat sedikit. Serangan-serangan ini bersifat langsung, namun layak untuk dilihat karena menunjukkan secara meyakinkan mengapa diperlukan sejumlah besar peluru.

Kita pertama-tama mempertimbangkan kasus sepele dimana F terdiri dari satu putaran penuh dan tidak ada langkah pencampuran kunci akhir. Kami menunjukkan bahwa musuh yang hanya diberi satu pasangan input/output (x, y) dapat dengan mudah mempelajari kunci rahasia k yang mana $y = F_k(x)$. Musuh memulai dengan nilai keluaran y dan kemudian membalikkan permutasi pencampuran dan kotak-S. Hal ini dapat dilakukan, seperti disebutkan sebelumnya, karena spesifikasi lengkap dari permutasi pencampuran dan S-box bersifat publik. Nilai antara yang dihitung musuh adalah tepat $x \oplus k$ (dengan asumsi, tanpa kehilangan keumuman, bahwa kunci master digunakan sebagai sub-kunci dalam satu-satunya putaran jaringan). Karena musuh juga mengetahui masukan x, ia dapat segera memperoleh kunci rahasia k. Oleh karena itu, ini adalah istirahat total. Meskipun serangan ini sepele, serangan ini menunjukkan bahwa dalam jaringan substitusi-permutasi mana pun tidak ada keamanan yang diperoleh dengan melakukan substitusi S-box atau menerapkan permutasi pencampuran setelah pencampuran sub-kunci terakhir.

Menyerang SPN satu putaran. Sekarang kita mempunyai satu putaran penuh diikuti dengan langkah pencampuran kunci. Untuk lebih konkretnya, kami mengasumsikan panjang blok 64-bit dan S-box dengan panjang input/output 8-bit (1-byte). Kami mengasumsikan sub-kunci 64-bit independen k_1, k_2 digunakan untuk dua langkah pencampuran kunci, sehingga kunci master $k_1 \| k_2$ dari SPN memiliki panjang 128 bit.

Pengamatan pertama adalah kita dapat memperluas serangan dari kasus sepele di atas untuk memberikan serangan pemulihan kunci di sini dengan menggunakan kurang dari 2^{128} pekerjaan. Idenya adalah sebagai berikut: Dengan adanya pasangan input/output (x, y) seperti sebelumnya, penyerang menghitung semua nilai yang mungkin untuk sub-kunci k_2 putaran kedua. Untuk setiap nilai tersebut, penyerang dapat membalikkan langkah pencampuran kunci terakhir untuk mendapatkan kandidat nilai antara y'. Kita telah melihat di atas bahwa dengan masukan x dan keluaran y' dari putaran SPN (penuh), kemungkinan sub-kunci unik k_1 dapat dengan mudah diidentifikasi. Jadi, untuk setiap pilihan k_2 yang mungkin, penyerang mendapatkan k_1 unik yang bersesuaian dimana $k_1 \| k_2$ dapat menjadi kunci utama. Dengan cara ini, penyerang memperoleh (dalam 2^{64} kali) daftar 2^{64} kemungkinan kunci master. Hal ini dapat dipersempit dengan menggunakan pasangan input/output tambahan dalam sekitar 2^{64} waktu tambahan; lihat juga di bawah.

Serangan yang lebih baik dimungkinkan dengan mencatat bahwa bit-bit individual dari output hanya bergantung pada sebagian dari kunci master. Perbaiki beberapa pasangan input/output (x, y) seperti sebelumnya. Sekarang, musuh akan menghitung semua nilai yang mungkin untuk byte pertama k_2. Ia dapat meng-XOR setiap nilai tersebut dengan byte pertama y untuk mendapatkan nilai kandidat untuk keluaran S-box pertama. Dengan membalikkan S-box ini, penyerang mempelajari nilai kandidat untuk masukan ke S-box tersebut. Karena input ke S-box tersebut adalah XOR dari 8 bit x dan 8 bit k_1 (di mana posisi

bit-bit tersebut bergantung pada permutasi pencampuran putaran pertama dan diketahui oleh penyerang), maka ini menghasilkan nilai kandidat untuk 8 bit k_1.

Ringkasnya: untuk setiap nilai kandidat byte pertama k_2, terdapat kemungkinan nilai unik yang sesuai untuk 8 bit k_1. Dengan kata lain, ini berarti bahwa untuk 16 bit kunci master, penyerang telah mengurangi jumlah nilai yang mungkin untuk bit tersebut dari 2^{16} menjadi 2^8. Penyerang dapat membuat tabulasi semua nilai yang memungkinkan tersebut dalam 2^8 waktu. Hal ini dapat diulangi untuk setiap byte k_2, menghasilkan 8 daftar—masing-masing berisi 2^8 nilai—yang bersama-sama mencirikan nilai yang mungkin dari seluruh kunci master. Penyerang kemudian mengurangi jumlah kunci master yang mungkin menjadi $(2^8)^8 = 2^{64}$ seperti pada serangan sebelumnya. Namun, total waktu untuk melakukan hal ini sekarang adalah $8 \cdot 2^8 = 2^{11}$ suatu peningkatan yang dramatis.

Penyerang dapat menggunakan pasangan input/output tambahan untuk mengurangi ruang kemungkinan kunci. Pertimbangkan daftar 2^8 nilai yang layak untuk beberapa set 16 bit kunci master. Penyerang mengetahui bahwa nilai yang benar dari daftar tersebut harus konsisten dengan pasangan input/output tambahan yang dipelajari penyerang. Secara heuristik, setiap nilai yang salah dari daftar konsisten dengan beberapa pasangan masukan/keluaran tambahan (x', y') dengan probabilitas yang tidak lebih baik daripada tebakan acak; karena setiap nilai 16-bit dari tabel dapat digunakan untuk menghitung 1 byte keluaran dengan masukan x', kita berharap bahwa nilai yang salah akan konsisten dengan keluaran sebenarnya y' dengan probabilitas 2^{-8}. Sejumlah kecil pasangan masukan/keluaran tambahan akan cukup untuk mempersempit semua tabel menjadi hanya satu nilai saja, sehingga seluruh kunci utama diketahui.

Ada pelajaran penting yang bisa dipetik di sini. Serangan ini mungkin terjadi karena bagian kunci yang berbeda dapat diisolasi dari bagian lain. Oleh karena itu, difusi lebih lanjut diperlukan untuk memastikan bahwa semua bit kunci mempengaruhi semua bit keluaran. Diperlukan beberapa putaran agar hal ini dapat terjadi.

Menyerang SPN dua putaran. Ide di atas dapat diperluas untuk memberikan serangan yang lebih baik daripada serangan brute force pada SPN dua putaran menggunakan sub-kunci independen di setiap putaran; kami meninggalkan ini sebagai latihan.

Sebaliknya, kami hanya mencatat bahwa SPN dua putaran tidak akan menjadi permutasi pseudo-acak yang baik. Di sini kita mengandalkan fakta, yang disebutkan sebelumnya, bahwa efek longsoran salju tidak terjadi hanya setelah dua putaran (tentu saja, ini tergantung pada panjang blok cipher dan panjang input/output S-box, tetapi dengan parameter yang masuk akal. ini akan terjadi). Seorang penyerang dapat membedakan SPN dua putaran dari permutasi seragam jika ia mengetahui hasil evaluasi SPN pada dua input yang berbeda dalam satu bit: dalam SPN dua putaran, banyak bit dari dua output akan sama, sesuatu yang tidak diharapkan terjadi karena permutasi acak.)

Jaringan Feistel

Jaringan Feistel menawarkan pendekatan lain untuk membangun cipher blok. Keuntungan jaringan Feistel dibandingkan jaringan substitusi-permutasi adalah bahwa fungsi dasar yang digunakan dalam jaringan Feistel—berbeda dengan S-box yang digunakan di SPN—

tidak perlu dibalik. Jaringan Feistel memberikan cara untuk membangun fungsi yang dapat dibalik dari komponen yang tidak dapat dibalik. Hal ini penting karena cipher blok yang baik harus memiliki perilaku "tidak terstruktur" (sehingga terlihat acak), namun memerlukan semua komponen konstruksi yang dapat dibalik dan secara inheren memperkenalkan struktur. Memerlukan invertibilitas juga menimbulkan kendala tambahan pada S-box, sehingga lebih sulit untuk dirancang.

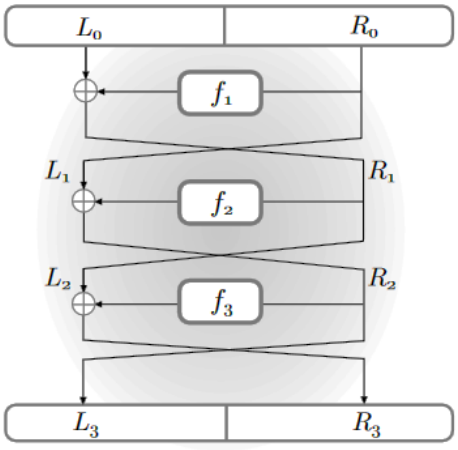
Jaringan Feistel beroperasi dalam serangkaian putaran. Di setiap putaran, fungsi putaran berkunci diterapkan dengan cara yang dijelaskan di bawah ini. Fungsi bulat tidak perlu dapat dibalik. Mereka biasanya dibangun dari komponen seperti S-box dan permutasi pencampuran, namun jaringan Feistel dapat menangani fungsi bulat apa pun terlepas dari desainnya.

Dalam jaringan Feistel yang seimbang (satu-satunya tipe yang akan kita pertimbangkan), fungsi putaran ke-i $\hat{f}_i$ mengambil sub-kunci k_i dan string $\ell/2$-bit sebagai masukan dan menghasilkan string ℓ /2-bit sebagai masukan. Seperti dalam kasus SPN, kunci master k digunakan untuk mendapatkan subkunci untuk setiap putaran. Ketika beberapa kunci master dipilih, sehingga menentukan setiap sub-kunci k , kami mendefinisikannya $f_i : \{0,1\}^{\ell/2} \rightarrow \{0,1\}^{\ell/2}$ via $f_i(R) \stackrel{\text{def}}{=} \hat{f}_i(k_i, R)$. Perhatikan bahwa fungsi putaran $\hat{f}_i$ bersifat tetap dan diketahui publik, namun f_i bergantung pada kunci master sehingga tidak diketahui oleh penyerang.

Putaran ke-*i* dari jaringan Feistel beroperasi sebagai berikut. Masukan ke putaran dibagi menjadi dua bagian yang dilambangkan dengan L_{i-1} dan, R_{i-1} (masing-masing bagian "kiri" dan "kanan"). Jika panjang blok sandi adalah ℓ bit, maka L_{i-1} dan, R_{i-1} masing-masing memiliki panjang $\ell/2$. Output $(L_{i-1}, R_{i-1)}$ dari putaran tersebut adalah

$$L_i := R_{i-1} \quad \text{and} \quad R_i := L_{i-1} \oplus f_i(R_{i-1}). \qquad (6.3)$$

Dalam jaringan Feistel putaran-r, masukan ℓ -bit ke jaringan diurai sebagai (L_0, R_0), dan keluarannya adalah nilai ℓ -bit (L_r, R_r) yang diperoleh setelah menerapkan semua putaran r. Jaringan Feistel tiga putaran ditunjukkan pada Gambar 6.5.



GAMBAR 6.5: Jaringan Feistel tiga putaran.

Membalikkan jaringan Feistel. Jaringan Feistel dapat dibalik terlepas dari $\{f_i\}$ (dan dengan demikian terlepas dari fungsi putarannya $\{\hat{f}_i\}$). Untuk menunjukkan hal ini kita hanya perlu menunjukkan bahwa setiap putaran jaringan dapat dibalik jika $\{f_i\}$ diketahui. Diketahui keluaran (L_i, R_i) dari putaran ke-*i*, kita dapat menghitung (L_{-1}, R_{-1}) sebagai berikut: himpunan pertama $(R_{i-1} := L_i)$. Kemudian hitung

$$L_{i-1} := R_i \oplus f_i(R_{i-1}).$$

Hal ini memberikan nilai L_{i-1}, R_{i-1} yang merupakan masukan dari putaran ini (yaitu, menghitung kebalikan dari Persamaan (6.3)). Perhatikan bahwa f_i dievaluasi hanya dalam arah maju, sehingga tidak perlu dibalik. Dengan demikian kami memiliki:

PROPOSISI 6.4 Misalkan F adalah fungsi berkunci yang didefinisikan oleh jaringan Feistel. Maka terlepas dari fungsi putaran $\{\hat{f}_i\}$ dan jumlah putarannya, f_k adalah permutasi yang dapat dibalik secara efisien untuk semua k.

Seperti dalam kasus jaringan substitusi-permutasi, serangan terhadap jaringan Feistel mungkin terjadi ketika jumlah putaran terlalu sedikit. Kita akan melihat serangan seperti itu ketika kita membahas DES di bagian selanjutnya.

DES – Standar Enkripsi Data

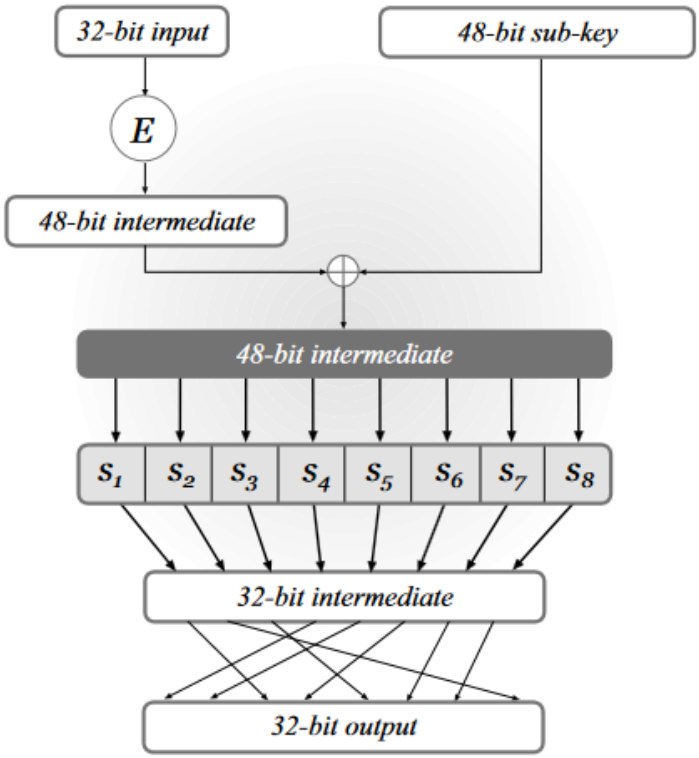
Standar Enkripsi Data, atau DES, dikembangkan pada tahun 1970-an oleh IBM (dengan bantuan Badan Keamanan Nasional) dan diadopsi pada tahun 1977 sebagai Standar Pemrosesan Informasi Federal untuk AS. Dalam bentuk dasarnya, DES tidak lagi dianggap aman karena panjang kuncinya yang pendek yaitu 56 bit, sehingga rentan terhadap serangan brute force. Namun demikian, hal ini masih digunakan secara luas saat ini dalam bentuk triple-DES yang diperkuat.

DES memiliki makna sejarah yang besar. Algoritma ini telah mengalami pengawasan intensif dalam komunitas kriptografi, dan bisa dibilang lebih banyak dibandingkan algoritma kriptografi lainnya dalam sejarah. Konsensus umum adalah, terlepas dari panjang kuncinya, DES merupakan sandi yang dirancang dengan sangat baik. Memang benar, bahkan setelah bertahun-tahun, serangan paling terkenal terhadap DES dalam praktiknya adalah pencarian menyeluruh terhadap 256 kemungkinan kunci. (Seperti yang akan kita lihat, ada serangan teoritis penting terhadap DES yang memerlukan komputasi lebih sedikit; namun, serangan ini mengasumsikan kondisi tertentu yang tampaknya sulit untuk diwujudkan dalam praktiknya.)
Pada bagian ini, kami memberikan gambaran umum tingkat tinggi tentang komponen utama DES. Kami menekankan bahwa kami tidak akan memberikan spesifikasi lengkap yang benar di setiap detailnya, dan beberapa bagian desain akan dihilangkan dari deskripsi kami. Tujuan kami adalah untuk menyajikan ide-ide dasar yang mendasari pembangunan DES, dan bukan semua rincian tingkat rendah; pembaca yang tertarik dengan detail tersebut dapat melihat referensi di akhir bab ini.

Desain DES

Cipher blok DES adalah jaringan Feistel 16 putaran dengan panjang blok 64 bit dan panjang kunci 56 bit. Fungsi putaran yang sama $\hat{f}$ digunakan di masing-masing dari 16 putaran. Fungsi putaran mengambil sub-kunci 48-bit dan, seperti yang diharapkan untuk jaringan Feistel (seimbang), input 32-bit (yaitu, setengah blok). Jadwal kunci DES digunakan untuk memperoleh urutan sub-kunci 48-bit $k_1, \ldots, k_{16}$ dari kunci master 56-bit. Jadwal kunci DES relatif sederhana, dengan masing-masing sub-kunci ki menjadi subset permutasi dari 48 bit kunci master. Untuk tujuan kita, cukup dicatat bahwa 56 bit kunci master dibagi menjadi dua bagian—"setengah kiri" dan "setengah kanan"—masing-masing berisi 28 bit. (Pembagian ini terjadi setelah permutasi awal diterapkan pada kunci, namun kita mengabaikan hal ini dalam uraian kita.) Dalam setiap putaran, 24 bit paling kiri dari subkunci diambil sebagai subset dari 28 bit di sebelah kiri. setengah dari kunci master, dan 24 bit paling kanan dari subkunci bulat diambil sebagai subset dari 28 bit di paruh kanan kunci master. Kami menekankan bahwa seluruh jadwal kunci (termasuk cara bit dibagi menjadi bagian kiri dan kanan, dan bit mana yang digunakan dalam membentuk setiap sub-kunci k_i) bersifat tetap dan bersifat publik, dan satu-satunya rahasia adalah kunci master itu sendiri. .

Fungsi putaran DES. Fungsi putaran DES $\hat{f}$ —terkadang disebut fungsi DES mangler—dibangun menggunakan paradigma yang telah kita analisis sebelumnya: fungsi ini (pada dasarnya) hanyalah jaringan substitusi-permutasi! Secara lebih rinci, perhitungan $\hat{f}(k_i, R)$ dengan $k_i \in \{0,1\}^{48}$ dan $R \in \{0,1\}^{32}$ berlangsung sebagai berikut: pertama, R diperluas ke nilai 48-bit R'. Hal ini dilakukan hanya dengan menduplikasi setengah bit R; kami menyatakannya dengan $R' := E(R)$ dimana



GAMBAR 6.6: Fungsi mangkuk DES.

E disebut fungsi ekspansi. Setelah ini, perhitungan berjalan persis seperti pada pembahasan kita sebelumnya tentang SPN: Nilai yang diperluas R' di-XOR dengan k_i, yang juga panjangnya 48 bit, dan nilai yang dihasilkan dibagi menjadi 8 blok, yang masing-masing blok panjangnya 6 bit. . Setiap blok dilewatkan melalui S-box (berbeda) yang mengambil masukan 6-bit dan menghasilkan keluaran 4-bit; menggabungkan output dari 8 S-box memberikan hasil 32-bit. Permutasi pencampuran kemudian diterapkan pada bit-bit hasil ini untuk mendapatkan hasil akhir. Lihat Gambar 6.6.

Satu perbedaan dibandingkan dengan pembahasan awal kita tentang SPN adalah bahwa S-box di sini tidak dapat dibalik; memang, hal-hal tersebut tidak dapat dibalik karena masukannya lebih panjang daripada keluarannya. Pembahasan lebih lanjut mengenai rincian struktur S-box diberikan di bawah ini.

Kami menekankan sekali lagi bahwa segala sesuatu dalam uraian di atas (termasuk S-box itu sendiri serta permutasi pencampurannya) diketahui publik. Satu-satunya rahasia adalah kunci master yang digunakan untuk mendapatkan semua sub-kunci.

S-box dan permutasi pencampuran. Delapan S-box yang membentuk "inti" $\hat{f}$ merupakan elemen penting dalam konstruksi DES dan dirancang dengan sangat cermat. Studi terhadap DES menunjukkan bahwa jika S-box sedikit dimodifikasi, DES akan jauh lebih rentan terhadap serangan.

Hal ini harus menjadi peringatan bagi siapapun yang ingin merancang sebuah blok cipher: pilihan yang tampaknya sewenang-wenang tidaklah sewenang-wenang sama sekali, dan jika tidak dilakukan dengan benar dapat membuat keseluruhan konstruksi menjadi tidak aman.

Ingatlah bahwa setiap S-box memetakan masukan 6-bit ke keluaran 4-bit. Setiap S-box dapat dilihat sebagai tabel dengan 4 baris dan 16 kolom, dimana setiap sel tabel berisi entri 4-bit. Input 6-bit dapat dilihat sebagai pengindeksan salah satu dari $2^6 = 64 = 4 \times 16$ sel tabel dengan cara berikut: Bit input pertama dan terakhir digunakan untuk memilih baris tabel, dan bit 2–5 digunakan untuk memilih baris tabel. pilih kolom tabel. Entri 4-bit pada beberapa posisi tabel mewakili nilai output untuk input yang terkait dengan posisi tersebut. S-box DES memiliki properti berikut (antara lain):

1. Setiap S-box adalah fungsi 4-ke-1. (Artinya, tepat 4 masukan dipetakan ke setiap kemungkinan keluaran.) Ini mengikuti properti di bawah ini.
2. Setiap baris dalam tabel berisi masing-masing dari 16 kemungkinan string 4-bit tepat satu kali.
3. Mengubah satu bit masukan apa pun ke S-box selalu mengubah setidaknya dua bit keluaran.

Permutasi pencampuran juga dirancang dengan cermat. Secara khusus ia mempunyai sifat bahwa empat bit keluaran dari setiap kotak S akan mempengaruhi masukan ke enam kotak S pada putaran berikutnya. (Hal ini dimungkinkan karena fungsi ekspansi yang diterapkan pada putaran berikutnya sebelum S-box dihitung.)

Efek longsoran DES. Desain fungsi mangkuk memastikan bahwa DES menunjukkan efek longsoran salju yang kuat. Untuk melihatnya, kita akan menelusuri perbedaan antara nilai

antara dalam perhitungan DES dari dua input yang berbeda hanya satu bit. Mari kita nyatakan dua masukan pada sandi dengan (L_0, R_0) dan (L'_0, R'_0), di mana kita berasumsi bahwa $(R_0 = R'_0)$ sehingga perbedaan bit tunggal terjadi di separuh kiri masukan (itu dapat membantu dengan mengacu pada Persamaan (6.3) dan Gambar 6.6 berikut). Setelah putaran pertama nilai antara (L_1, R_1) dan (L'_1, R'_1)masih berbeda hanya satu bit, meskipun sekarang perbedaannya berada di separuh kanan. Pada putaran kedua DES, separuh kanan setiap masukan dijalankan melalui $\hat{f}$. Dengan asumsi bahwa bit dimana $(R_1 \text{ dan } R'_1)$ berbeda tidak diduplikasi pada langkah ekspansi, nilai antara sebelum penerapan S-box masih berbeda hanya satu bit.

Dengan menggunakan Ppoperti 3 dari S-box, nilai antara setelah perhitungan S-box berbeda setidaknya dalam dua bit. Hasilnya adalah nilai antara (L_2, R_2) dan (L'_2, R'_2) berbeda dalam tiga bit: terdapat perbedaan 1-bit antara L_2 dan L'_2 (dibawa dari perbedaan antara R_1dan R'_1) dan perbedaan 2-bit antara R_2 dan R'_2.

Permutasi pencampuran menyebarkan perbedaan dua bit antara R_2 dan R'_2 sedemikian rupa sehingga, pada putaran berikutnya, masing-masing dari dua bit tersebut digunakan sebagai input ke S-box yang berbeda, sehingga menghasilkan perbedaan minimal 4 bit di sebelah kanan. setengah dari nilai antara. (Jika salah satu atau kedua bit yang berbeda R_2 dan R'_2 diduplikasi oleh E, perbedaannya mungkin lebih besar.) Sekarang juga terdapat perbedaan 2-bit di bagian kiri. Seperti halnya jaringan substitusi-permutasi, kita mempunyai efek eksponensial dan setelah 7 putaran kita berharap semua 32 bit di bagian kanan akan terpengaruh (dan setelah 8 putaran, semua 32 bit di bagian kiri akan terpengaruh juga).

DES memiliki 16 putaran, sehingga efek longsoran terjadi sangat awal dalam komputasi. Hal ini memastikan bahwa penghitungan DES pada masukan serupa menghasilkan keluaran yang terlihat independen.

Serangan pada DES Putaran Dikurangi

Latihan yang berguna untuk memahami lebih lanjut tentang konstruksi DES dan keamanannya adalah dengan melihat perilaku DES hanya dalam beberapa putaran. Kami akan menunjukkan serangan pada varian DES satu, dua, dan tiga putaran (ingat bahwa DES sebenarnya memiliki 16 putaran). DES dengan tiga putaran atau kurang tidak dapat menjadi fungsi pseudorandom karena tiga putaran tidak cukup untuk terjadinya efek longsoran. Dengan demikian, kita akan tertarik untuk mendemonstrasikan serangan pemulihan kunci yang lebih sulit (dan lebih merusak) yang menghitung kunci k hanya dengan menggunakan sejumlah kecil pasangan masukan/keluaran yang dihitung menggunakan kunci tersebut. Beberapa serangan serupa dengan yang kita lihat dalam konteks jaringan substitusi-permutasi; Namun di sini, kita akan melihat bagaimana penerapannya pada cipher blok konkrit dan bukan pada desain abstrak.

Serangan di bawah ini adalah serangan teks biasa yang diketahui dimana musuh mengetahui beberapa pasangan teks biasa/teks tersandi $\{(x_i, y_i)\}$ dengan $y_i = DES_k(x_i)$ untuk beberapa kunci rahasia k. Saat kami mendeskripsikan serangan, kami akan fokus pada pasangan input/output tertentu (x, y) dan akan menjelaskan informasi tentang

kunci yang dapat diperoleh musuh dari pasangan ini. Melanjutkan penggunaan notasi yang telah dikembangkan sebelumnya, kita menyatakan bagian kiri dan kanan masukan x masing-masing sebagai L_0 dan R_0, dan misalkan L_i, R_i menunjukkan bagian kiri dan kanan setelah putaran ke-i. Ingatlah bahwa E menunjukkan fungsi ekspansi DES, $k_i$menunjukkan sub-kunci yang digunakan pada putaran ke-i, dan $f_i(R) = \hat{f}(k_i, R)$ menunjukkan fungsi sebenarnya yang diterapkan dalam jaringan Feistel pada putaran ke-i.

DES satu putaran. Katakanlah kita diberi pasangan input/output (x, y). Dalam DES satu putaran, kita mempunyai $y = (L_i, R_i)$, dimana $L_1 = R_0$ dan $R_1 = L_0 \oplus f_1(R_0)$. Oleh karena itu kita mengetahui pasangan input/output untuk f_1; secara spesifik, kita mengetahui bahwa $f_1(R_0) = R_1 \oplus L_0$. Dengan menerapkan kebalikan dari permutasi pencampuran pada keluaran $R_1 \oplus L_0$, diperoleh nilai antara yang terdiri dari keluaran semua S-box, dimana 4 bit pertama adalah keluaran dari S-box pertama, 4 bit berikutnya adalah keluaran dari S-box kedua, dan seterusnya.

Pertimbangkan keluaran 4-bit (yang diketahui) dari S -box pertama. Karena setiap S-box adalah fungsi 4 banding 1, ini berarti terdapat tepat empat kemungkinan masukan ke S-box ini yang akan menghasilkan keluaran tertentu, dan demikian pula untuk semua S-box lainnya; setiap masukan tersebut panjangnya 6 bit. Input ke S-box hanyalah XOR dari $E(R_0)$ dengan sub-kunci k_1. Karena R_0, dan karenanya $E(R_0)$, diketahui, kita dapat menghitung empat kemungkinan nilai untuk setiap bagian 6-bit dari k_1. Ini berarti kita telah mengurangi jumlah kemungkinan kunci k_1 dari 2^{48} menjadi $4^{48/6} = 4^4 = 2^{16}$ (karena ada empat kemungkinan untuk masing-masing dari delapan bagian 6-bit k_1). Ini sudah merupakan angka yang kecil sehingga kita dapat mencoba semua kemungkinan pada pasangan input/output yang berbeda (x', y') untuk menemukan kunci yang tepat. Dengan demikian, kami memperoleh kunci hanya dengan menggunakan dua teks biasa yang diketahui dalam waktu sekitar 2^{16}.

DES dua putaran. Pada DES dua putaran, keluaran y sama dengan (L_2, R_2) dimana

$$L_1 = R_0$$
$$R_1 = L_0 \oplus f_1(R_0)$$
$$L_2 = R_1 = L_0 \oplus f_1(R_0)$$
$$R_2 = L_1 \oplus f_2(R_1).$$

L_0, R_0, L_2 dan R_2 diketahui dari pasangan input/output yang diberikan (x, y), sehingga kita juga mengetahui $L_1 = R_0$ dan $R_1 = L_2$. Ini berarti bahwa kita mengetahui input/output dari f_1 dan f_2, sehingga metode yang sama yang digunakan dalam serangan pada DES satu putaran dapat digunakan di sini untuk menentukan k_1 dan k_2, dalam waktu kira-kira $2 - 2^{16}$. Serangan ini berhasil bahkan jika k_1 dan k_2 merupakan kunci yang sepenuhnya independen, meskipun pada kenyataannya jadwal kunci DES memastikan bahwa banyak bit k_1 dan k_2 adalah sama (yang dapat digunakan untuk lebih mempercepat serangan).

DES tiga putaran. Mengacu pada Gambar 6.5, nilai keluaran y sekarang sama dengan (L_3, R_3). Karena $L_1 = R_0$ dan $R_2 = L_3$., satu-satunya nilai yang tidak diketahui pada gambar adalah R_1 dan L_2.

Sekarang kita tidak lagi mempunyai input/output ke fungsi putaran f_i. Misalnya, nilai keluaran f_2 sama dengan $L_1 \oplus R_2$, dimana kedua nilai tersebut diketahui. Namun kita tidak mengetahui nilai R_1 yang dimasukkan ke f_2. Demikian pula, kita dapat menentukan masukan ke f_1 dan f_3 tetapi tidak dapat menentukan keluaran dari fungsi tersebut. Oleh karena itu, serangan yang kami gunakan untuk mematahkan DES satu putaran dan dua putaran tidak akan berhasil di sini.

Daripada mengandalkan pengetahuan penuh tentang input dan output dari salah satu fungsi putaran, kita akan menggunakan pengetahuan tentang hubungan tertentu antara input dan output dari f_1 dan f_3. Perhatikan bahwa keluaran dari f_1 sama dengan $L_0 \oplus R_1 = L_1 \oplus L_2$, dan keluaran dari f_3 sama dengan $L_2 \oplus R_3$. Karena itu,

$$f_1(R_0) \oplus f_3(R_2) = (L_0 \oplus L_2) \oplus (L_2 \oplus R_3) = L_0 \oplus R_3,$$

dimana L_0 dan R_3 diketahui. Artinya, XOR keluaran f_1 dan f_3 diketahui. Selanjutnya masukan ke f_1 adalah R_0 dan masukan ke f_3 adalah L_3, keduanya diketahui. Kita menyimpulkan bahwa kita dapat menentukan masukan ke f_1 dan f_3, serta XOR keluarannya. Kami sekarang menjelaskan serangan yang menemukan kunci rahasia berdasarkan informasi ini.

Ingatlah bahwa jadwal kunci DES memiliki sifat bahwa kunci master dibagi menjadi "setengah kiri", yang dilambangkan dengan k_L, dan "setengah kanan" k_R, masing-masing berisi 28 bit. Selanjutnya 24 bit paling kiri dari subkunci yang digunakan pada setiap putaran diambil hanya dari k_L, dan 24 bit paling kanan dari setiap subkunci diambil hanya dari k_R. Artinya k_L hanya mempengaruhi masukan ke empat S-box pertama pada setiap putaran, sedangkan k_R hanya mempengaruhi masukan ke empat S-box terakhir. Karena permutasi pencampuran diketahui, kita juga mengetahui bit keluaran setiap fungsi putaran yang keluar dari setiap S-box.

Gagasan di balik serangan ini adalah untuk secara terpisah melintasi ruang kunci untuk masing-masing setengah dari kunci master, memberikan serangan dengan kompleksitas kira-kira $2 \cdot 2^{28}$ daripada kompleksitas 2^{56}. Serangan seperti itu akan mungkin terjadi jika kita dapat memverifikasi tebakan setengah dari kunci master, dan sekarang kami menunjukkan bagaimana hal ini dapat dilakukan. Katakanlah kita menebak beberapa nilai k_L, bagian kiri dari kunci master. Kita mengetahui input R_0 dari f_1, dan dengan menggunakan tebakan k_L kita dapat menghitung input ke empat S-box pertama. Ini berarti bahwa kita dapat menghitung setengah bit keluaran dari f_1 (permutasi pencampuran menyebarkan bit-bit yang kita ketahui, tetapi karena permutasi pencampuran diketahui, kita mengetahui secara pasti bit-bit tersebut). Demikian pula, kita dapat menghitung lokasi yang sama pada keluaran f_3 dengan menggunakan masukan yang diketahui L_3 hingga f_3 dan tebakan yang sama untuk k_L. Terakhir, kita dapat menghitung XOR dari nilai keluaran ini dan memeriksa apakah nilai tersebut cocok dengan bit yang sesuai dalam nilai XOR keluaran f_1 dan f_3 yang diketahui. Jika tidak sama, maka tebakan kita untuk k_L salah. Tebakan yang benar untuk k_L akan selalu lulus pengujian ini, sehingga tidak akan dihilangkan, namun tebakan yang salah diharapkan lulus pengujian ini hanya dengan probabilitas kira-kira 2^{-16} (karena kita memeriksa kesetaraan 16 bit dalam dua nilai yang dihitung). Ada 2^{28} kemungkinan nilai k_L, jadi jika setiap nilai yang

salah tetap menjadi kandidat yang layak dengan probabilitas 2^{-16} maka kita perkirakan hanya tersisa $2^{28} \cdot 2^{-16} = 2^{12}$ kemungkinan k_L setelah hal di atas.

Dengan melakukan hal di atas untuk setiap separuh kunci master, kita memperoleh dalam waktu $2 \cdot 2^{28}$ kira-kira 2^{12} calon untuk separuh kiri dan 2^{12} calon untuk separuh kanan. Karena setiap kombinasi separuh kiri dan separuh kanan dimungkinkan, kami memiliki 2^{24} kandidat kunci secara keseluruhan dan dapat menjalankan penelusuran brute force pada himpunan ini menggunakan pasangan masukan/keluaran tambahan (x', y'). (Pendekatan alternatif yang lebih efisien adalah dengan mengulangi serangan sebelumnya dengan menggunakan 2^{12} kandidat yang tersisa untuk setiap setengah kunci.) Kompleksitas waktu serangan tersebut kira-kira $2 \cdot 2^{28} + 2^{24} < 2^{30}$ yang jauh lebih kecil dari 2^{56}.

Keamanan DES

Setelah hampir 30 tahun melakukan studi intensif, serangan praktis yang paling terkenal terhadap DES masih berupa pencarian menyeluruh melalui ruang kuncinya. (Kami membahas beberapa serangan teoretis penting di Bagian 6.2. Serangan ini memerlukan pasangan input/output dalam jumlah besar, yang akan sulit diperoleh dalam serangan pada sistem dunia nyata mana pun yang menggunakan DES.) Sayangnya, versi 56-bit panjang kunci DES cukup pendek sehingga pencarian menyeluruh melalui 2^{56} kemungkinan kunci sekarang dapat dilakukan. Pada akhir tahun 1970-an terdapat keberatan yang kuat terhadap pilihan kunci pendek untuk DES. Saat itu, keberatan tersebut bersifat teoritis, karena kekuatan komputasi yang diperlukan untuk mencari kunci-kunci tersebut pada umumnya tidak tersedia. Namun, kepraktisan serangan brute-force terhadap DES ditunjukkan pada tahun 1997 ketika tantangan pertama dari serangkaian tantangan DES diatur oleh RSA Keamanan diselesaikan oleh proyek DESCHALL menggunakan ribuan komputer yang dikoordinasikan melalui Internet; perhitungannya memakan waktu 96 hari. Tantangan kedua dipecahkan pada tahun berikutnya hanya dalam 41 hari oleh proyek distribution.net. Sebuah terobosan signifikan terjadi pada tahun 1998 ketika tantangan ketiga diselesaikan hanya dalam waktu 56 jam. Prestasi mengesankan ini dicapai melalui mesin pemecah DES dengan tujuan khusus yang disebut Deep Crack yang dibuat oleh Electronic Frontier Foundation dengan biaya Rp. 2.500.000.000. Pada tahun 1999, tantangan DES diselesaikan hanya dalam waktu 22 jam sebagai upaya gabungan dari Deep Crack dan distribution.net. Yang paling canggih saat ini adalah kotak cracking DES dari PICO Computing, yang menggunakan 48 FPGA dan dapat menemukan kunci DES dalam waktu sekitar 23 jam.

Pertukaran waktu/ruang yang dibahas di Bagian 5.4 menunjukkan bahwa serangan pencarian kunci yang menyeluruh dapat dipercepat menggunakan pra-komputasi dan memori tambahan. Karena pendeknya panjang kunci DES, trade-off waktu/ruang bisa menjadi sangat efektif. Secara khusus, dengan menggunakan pra-pemrosesan, dimungkinkan untuk menghasilkan tabel berukuran beberapa terabyte yang kemudian memungkinkan pemulihan kunci DES dengan probabilitas tinggi dari satu pasangan input/output menggunakan sekitar 2^{38} evaluasi DES (yang dapat dihitung hanya dalam hitungan menit). Intinya adalah DES memiliki kunci yang terlalu pendek, dan tidak dapat dianggap aman untuk aplikasi serius apa pun saat ini.

Penyebab kekhawatiran kedua adalah panjang blok DES yang relatif pendek. Panjang blok yang pendek merupakan masalah karena keamanan konkrit dari banyak konstruksi yang didasarkan pada cipher blok bergantung pada panjang blok—bahkan jika cipher yang digunakan "sempurna." Misalnya, bukti keamanan untuk mode CTR (lihat Teorema 3.32) menunjukkan bahwa meskipun fungsi acak digunakan, penyerang dapat merusak keamanan skema enkripsi ini dengan probabilitas $2q^2/2^\ell$ jika ia memperoleh q teks biasa/teks sandi berpasangan. Dalam kasus DES di mana $\ell = 64$, ini berarti bahwa jika penyerang hanya memperoleh $q = 2^{30}$ pasangan teks biasa/teks sandi, kemungkinan besar keamanan akan terganggu. Mendapatkan pasangan teks biasa/teks sandi relatif mudah jika musuh menguping enkripsi pesan yang berisi header yang diketahui, redundansi, dll.

Ketidakamanan DES tidak ada hubungannya dengan desainnya sendiri, namun lebih disebabkan oleh pendeknya panjang kunci (dan, pada tingkat yang lebih rendah, panjang bloknya yang pendek). Ini merupakan penghargaan besar bagi para perancang DES, yang tampaknya telah berhasil membangun sebuah cipher blok yang hampir "sempurna" (selain kuncinya yang terlalu pendek). Karena DES sendiri tampaknya tidak memiliki kelemahan struktural yang signifikan, maka masuk akal untuk menggunakan DES sebagai blok penyusun untuk membangun cipher blok dengan kunci yang lebih panjang. Kami membahas hal ini lebih lanjut di Bagian 6.2.

Pengganti DES—Standar Enkripsi Lanjutan (AES), yang dibahas nanti dalam bab ini—secara eksplisit dirancang untuk mengatasi kekhawatiran mengenai pendeknya panjang kunci dan panjang blok DES. AES mendukung kunci 128-, 192-, atau 256-bit, dan panjang blok 128 bit.

Serangan yang lebih baik daripada serangan brute force terhadap DES pertama kali ditunjukkan pada awal tahun 1990an oleh Biham dan Shamir, yang mengembangkan teknik yang disebut kriptanalisis diferensial. Serangan mereka memerlukan waktu 2^{37} dan membutuhkan 2^{47} teks biasa yang dipilih. Meskipun serangan tersebut merupakan terobosan dari sudut pandang teoritis, hal ini tampaknya tidak menimbulkan banyak kekhawatiran praktis karena sulit untuk membayangkan skenario realistis di mana musuh dapat memperoleh enkripsi sebanyak ini dari teks biasa yang dipilih.

Menariknya, karya Biham dan Shamir menunjukkan bahwa DES S-box telah dirancang khusus agar tahan terhadap kriptanalisis diferensial, yang menunjukkan bahwa teknik kriptanalisis diferensial telah diketahui (tetapi tidak diungkapkan secara terbuka) oleh para perancang DES. Setelah Biham dan Shamir mengumumkan hasilnya, kecurigaan ini terbukti.

Kriptanalisis linier dikembangkan oleh Matsui pada pertengahan 1990an dan juga berhasil diterapkan pada DES. Keuntungan dari serangan ini adalah ia menggunakan teks biasa yang diketahui dibandingkan teks biasa yang dipilih. Meski demikian, jumlah pasangan plaintext/ciphertext yang dibutuhkan—sekitar 2^{43}—masih sangat banyak.

3DES: Menambah Panjang Kunci dari Block Cipher

Kelemahan utama DES adalah kuncinya yang pendek. Oleh karena itu masuk akal untuk mencoba merancang cipher blok dengan panjang kunci yang lebih besar menggunakan DES sebagai blok penyusunnya. Beberapa pendekatan untuk melakukan hal tersebut dibahas dalam bagian ini. Meskipun kami sering mengacu pada DES sepanjang diskusi, dan DES adalah

blok cipher yang paling menonjol dimana teknik ini telah diterapkan, semua yang kami katakan di sini berlaku secara umum untuk semua blok cipher.

Modifikasi internal vs. konstruksi "kotak hitam". Ada dua pendekatan umum yang dapat diambil untuk membangun sandi lain berdasarkan DES. Pendekatan pertama adalah dengan memodifikasi struktur internal DES, sekaligus meningkatkan panjang kunci. Misalnya, seseorang dapat membiarkan fungsi putaran tidak tersentuh dan cukup menggunakan kunci master 128-bit dengan jadwal kunci yang berbeda (masih memilih sub-kunci 48-bit di setiap putaran). Atau, seseorang dapat mengubah S-boxnya sendiri dan menggunakan sub-kunci yang lebih besar di setiap putaran. Kerugian dari pendekatan tersebut adalah dengan memodifikasi DES—bahkan dengan cara yang paling kecil sekalipun—kita kehilangan kepercayaan yang kita peroleh terhadap DES karena fakta bahwa DES tetap tahan terhadap serangan selama bertahun-tahun. Konstruksi kriptografi sangat sensitif, dan bahkan perubahan kecil sekalipun, perubahan yang tampaknya tidak signifikan dapat membuat konstruksi menjadi tidak aman. Oleh karena itu, mengubah bagian dalam cipher blok tidak disarankan.

Pendekatan alternatif yang tidak mengalami masalah di atas adalah dengan menggunakan DES sebagai "kotak hitam" dan tidak menyentuh struktur internalnya sama sekali. Dalam pendekatan ini kami memperlakukan DES sebagai cipher blok "sempurna" dengan kunci 56-bit, dan membangun cipher blok baru yang hanya memanggil DES asli yang tidak dimodifikasi. Karena DES sendiri tidak diubah, pendekatan ini jauh lebih bijaksana, dan inilah yang akan kita terapkan di sini.

Enkripsi Ganda

Misalkan F adalah sebuah cipher blok dengan panjang kunci n-bit dan panjang blok ℓ-bit. Kemudian cipher blok baru F' dengan panjang kunci $2n$ dapat didefinisikan dengan

$$F'_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_2}(F_{k_1}(x)),$$

dimana k_1 dan k_2 adalah kunci independen. Untuk kasus di mana F adalah DES, kita memperoleh sandi F' yang disebut 2DES yang menggunakan kunci 112-bit; jika pencarian kunci menyeluruh adalah serangan terbaik yang tersedia, panjang kunci 112 bit sudah cukup karena serangan yang memerlukan waktu 2^{112} benar-benar di luar jangkauan. Sayangnya, kami sekarang menunjukkan serangan terhadap F' yang berjalan dalam waktu sekitar 2^n, jauh lebih kecil dari waktu 2^{2n} yang diharapkan diperlukan untuk melakukan pencarian menyeluruh untuk kunci $2n$-bit. Artinya, cipher blok baru pada dasarnya tidak lebih baik dari cipher lama, meskipun memiliki kunci yang dua kali lebih panjang.

Serangan ini disebut "serangan pertemuan di tengah-tengah," karena alasan yang akan segera diketahui. Katakanlah musuh diberi pasangan masukan/keluaran tunggal (x, y), dengan $y = F'_{k_1^*,k_2^*}(x) = F_{k_2^*}(F_{k_1^*}(x))$ untuk k_1^*, k_2^* yang tidak diketahui. Musuh dapat mempersempit kumpulan kemungkinan kunci dengan cara berikut:

1. Untuk setiap $k_1 \in \{0,1\}^n$, hitung $z := F_{k_1}(x)$ dan simpan (z, k_1) dalam daftar L.
2. Untuk setiap $k_2 \in \{0,1\}^n$ hitung $z := F_{k_2}^{-1}(y)$ dan simpan (z, k_2) dalam daftar L'.

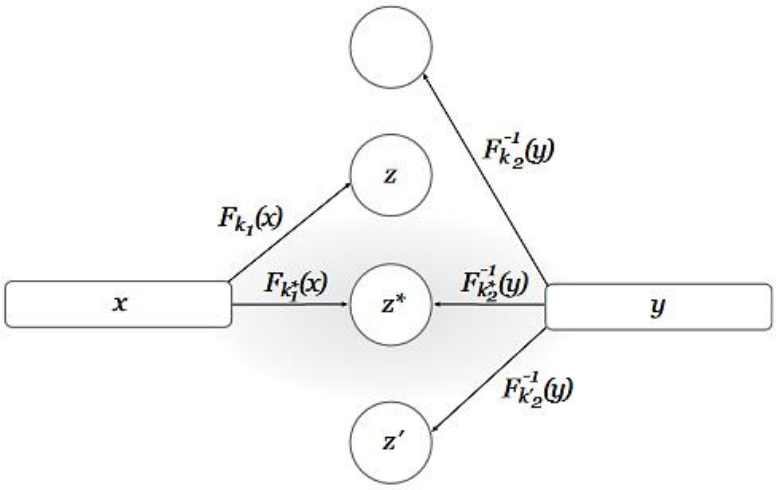
3. Entri $(z_1, k_1) \in L$ dan $(z_2, k_2) \in L'$ cocok jika $z_1 = z_2$. Untuk setiap kecocokan tersebut, tambahkan (k_1, k_2) ke himpunan S. (Kecocokan dapat ditemukan dengan mudah setelah mengurutkan L dan L' berdasarkan komponen pertamanya.)

Lihat Gambar 6.7 untuk gambaran grafis serangan tersebut.

Serangannya membutuhkan waktu $O(n \cdot 2^n)$, dan membutuhkan ruang $O((n + \ell) \cdot 2^n)$. Keluaran himpunan S oleh algoritma ini berisi nilai-nilai tersebut (k_1, k_2) yang tepat

$$F_{k_1}(x) = F_{k_2}^{-1}(y) \qquad (6.4)$$

atau, setara dengan $y = F'_{k_1,k_2}(x)$. Secara khusus, $k_1^*, k_2^* \in$ S. Sebaliknya, pasangan $k_1, k_2 = k_1^*, k_2^*$ (secara heuristik) diharapkan memenuhi Persamaan (6.4) dengan probabilitas $2^{-\ell}$ jika kita memperlakukan $F_{k_1}(x)$ dan $F_{k_2}^{-1}(y)$ sebagai string ℓ-bit yang seragam, sehingga ukuran S yang diharapkan adalah $2^{2n} \cdot 2^{-\ell} = 2^{2n-\ell}$ Dengan menggunakan beberapa pasangan input/output lainnya, dan mengambil perpotongan dari himpunan yang diperoleh, k_1^*, k_2^* yang benar dapat diidentifikasi dengan probabilitas yang sangat tinggi.



GAMBAR 6.7: Serangan temu di tengah.

Enkripsi Tiga Kali Lipat

Generalisasi yang jelas dari pendekatan sebelumnya adalah dengan menerapkan block cipher tiga kali berturut-turut. Dua varian dari pendekatan ini umum terjadi:

Varian 1: tiga kunci. Pilih tiga kunci independen k_1, k_2, k_3 dan tentukan

$$F''_{k_1,k_2,k_3}(x) \stackrel{\text{def}}{=} F_{k_3}(F_{k_2}^{-1}(F_{k_1}(x))).$$

Varian 2: dua kunci. Pilih dua kunci independen k_1, k_2 lalu tentukan

$$F''_{k_1,k_2}(x) \stackrel{\text{def}}{=} F_{k_1}(F_{k_2}^{-1}(F_{k_1}(x))).$$

Sebelum membandingkan keamanan kedua alternatif, kami mencatat bahwa pemanggilan tengah F adalah terbalik. Jika F merupakan sandi yang cukup baik, maka tidak ada bedanya dalam hal keamanan, karena jika F adalah permutasi pseudorandom yang kuat maka F^{-1} juga pasti demikian. Alasan untuk membalikkan penerapan F yang kedua adalah untuk

mendapatkan kompatibilitas ke belakang: jika seseorang menetapkan $k_1 = k_2 = k_3$ fungsi yang dihasilkan setara dengan pemanggilan tunggal F menggunakan kunci k_1.

Keamanan varian pertama. Panjang kunci dari varian pertama adalah $3n$, jadi kita berharap serangan terbaik pada sandi ini memerlukan waktu 2^{3n}. Namun, sandi rentan terhadap serangan bertemu di tengah-tengah seperti halnya enkripsi ganda, meskipun serangan tersebut sekarang memerlukan waktu 2^{2n}. Ini adalah serangan yang paling terkenal. Jadi, meskipun varian ini tidak seaman yang diharapkan, varian ini memperoleh keamanan yang cukup untuk semua tujuan praktis bahkan untuk $n = 56$ (dengan asumsi, tentu saja, sandi asli F tidak memiliki kelemahan).

Keamanan varian kedua. Panjang kunci varian ini adalah $2n$ sehingga harapan terbaik kami adalah keamanan terhadap serangan yang berjalan dalam waktu 2^{2n}. Tidak ada serangan yang diketahui memiliki kompleksitas waktu yang lebih baik ketika musuh hanya diberikan sejumlah kecil pasangan input/output. (Lihat Latihan 6.13 untuk serangan menggunakan $2n$ teks biasa yang dipilih.) Dengan demikian, enkripsi rangkap tiga dua kunci adalah pilihan yang masuk akal dalam praktiknya.

Tiga-DES (3DES). Triple-DES (atau 3DES) didasarkan pada pemanggilan tiga kali DES menggunakan dua atau tiga kunci, seperti dijelaskan di atas. 3DES distandarisasi pada tahun 1999, dan digunakan secara luas saat ini. Kelemahan utamanya adalah panjang blok yang relatif kecil dan fakta bahwa ia relatif lambat karena memerlukan 3 operasi cipher blok penuh. Karena panjang kunci minimum yang direkomendasikan saat ini adalah 128 bit, 3DES 2 kunci tidak lagi direkomendasikan (karena panjang kuncinya hanya 112 bit). Kelemahan ini menyebabkan penggantian DES/triple-DES dengan Standar Enkripsi Lanjutan, yang disajikan di bagian berikutnya.

AES – Standar Enkripsi Tingkat Lanjut

Pada bulan Januari 1997, Institut Standar dan Teknologi Nasional Amerika Serikat (NIST) mengumumkan bahwa mereka akan mengadakan kompetisi untuk memilih cipher blok baru—yang disebut Standar Enkripsi Lanjutan, atau AES—untuk menggantikan DES. Kompetisi dimulai dengan panggilan terbuka bagi tim untuk menyerahkan calon blok cipher untuk evaluasi. Sebanyak 15 algoritma berbeda dikirimkan dari seluruh dunia, termasuk kontribusi dari banyak kriptografer dan cryptanalyst terbaik. Kandidat sandi setiap tim dianalisis secara intensif oleh anggota NIST, publik, dan (terutama) tim lainnya. Dua lokakarya diadakan, satu pada tahun 1998 dan satu lagi pada tahun 1999, untuk membahas dan menganalisis berbagai masukan. Setelah lokakarya kedua, NIST mempersempit peserta menjadi 5 “finalis” dan putaran kedua kompetisi dimulai. Lokakarya AES ketiga diadakan pada bulan April 2000, mengundang pengawasan tambahan terhadap lima finalis. Pada bulan Oktober 2000, NIST mengumumkan bahwa algoritma pemenangnya adalah Rijndael (sebuah cipher blok yang dirancang oleh kriptografer Belgia Vincent Rijmen dan Joan Daemen), meskipun NIST mengakui bahwa salah satu dari 5 finalis akan membuat pilihan yang sangat baik. Secara khusus, tidak ada kerentanan keamanan serius yang ditemukan di salah satu dari 5 finalis, dan pemilihan “pemenang” sebagian didasarkan pada properti seperti efisiensi, kinerja perangkat keras, fleksibilitas, dll.

Proses pemilihan AES sangat cerdik karena setiap kelompok yang mengirimkan suatu algoritma, dan oleh karena itu tertarik untuk mengadopsi algoritma tersebut, mempunyai motivasi yang kuat untuk menemukan serangan terhadap pengajuan lainnya. Dengan cara ini, para cryptanalyst terbaik di dunia memusatkan perhatian mereka untuk menemukan kelemahan sekecil apa pun dalam kandidat cipher yang dikirimkan ke kompetisi. Hanya dalam beberapa tahun, masing-masing algoritma kandidat telah dipelajari secara intensif, sehingga meningkatkan kepercayaan kami terhadap keamanan algoritma pemenang. Tentu saja, semakin lama algoritma tersebut digunakan dan dipelajari tanpa dilanggar, maka rasa percaya diri kita akan terus tumbuh. Saat ini, AES digunakan secara luas dan tidak ditemukan kelemahan keamanan yang signifikan.

Konstruksi AES. Pada bagian ini, kami menyajikan struktur tingkat tinggi Rijndael/AES. (Secara teknis, Rijndael dan AES bukanlah hal yang sama tetapi perbedaannya tidak penting untuk pembahasan kita di sini.) Seperti halnya DES, kami tidak akan menyajikan spesifikasi lengkap dan uraian kami tidak boleh digunakan sebagai dasar penerapan. Tujuan kami hanya memberikan gambaran umum tentang cara kerja algoritme.

Cipher blok AES memiliki panjang blok 128-bit dan dapat menggunakan kunci 128-, 192-, atau 256-bit. Panjang kunci mempengaruhi jadwal kunci (yaitu, sub-kunci yang digunakan dalam setiap putaran) serta jumlah putaran, namun tidak mempengaruhi struktur tingkat tinggi setiap putaran.

Berbeda dengan DES yang menggunakan struktur Feistel, AES pada dasarnya adalah jaringan substitusi-permutasi. Selama komputasi algoritma AES, array byte berukuran 4 kali 4 yang disebut status dimodifikasi dalam serangkaian putaran. Keadaan awalnya diatur sama dengan input ke cipher (perhatikan bahwa inputnya adalah 128 bit, yaitu tepatnya 16 byte). Operasi berikut kemudian diterapkan pada negara bagian dalam serangkaian empat tahap selama setiap putaran:

Tahap 1 – AddRoundKey: Di setiap putaran AES, sub-kunci 128-bit diturunkan dari kunci master, dan diinterpretasikan sebagai array byte berukuran 4 kali 4. Array status diperbarui dengan melakukan XOR dengan subkunci ini.

Tahap 2 – SubBytes: Pada langkah ini, setiap byte dari array keadaan digantikan oleh byte lain sesuai dengan tabel pencarian tetap S. Tabel substitusi ini (atau S-box) adalah bijection atas $\{0,1\}^n$.

Tahap 3 – ShiftRows: Pada langkah ini, byte di setiap baris array status digeser ke kiri sebagai berikut: baris pertama array tidak disentuh, baris kedua digeser satu tempat ke kiri, baris ketiga digeser digeser dua tempat ke kiri, dan baris keempat digeser tiga tempat ke kiri. Semua pergeseran bersifat siklik sehingga, misalnya, pada baris kedua, byte pertama menjadi byte keempat.

Tahap 4 – MixColumns: Pada langkah ini, transformasi yang dapat dibalik diterapkan pada empat byte di setiap kolom. (Secara teknis, ini adalah transformasi linier—yaitu, perkalian matriks—pada bidang yang sesuai.) Transformasi ini memiliki sifat bahwa jika dua masukan berbeda dalam $b > 0$ byte, penerapan transformasi tersebut akan menghasilkan dua keluaran yang berbeda setidaknya 5 — b byte.

Di babak final, MixColumns diganti dengan AddRoundKey. Hal ini mencegah musuh membalikkan tiga tahap terakhir, yang tidak bergantung pada kuncinya.

Dengan melihat tahap 3 dan 4 bersama-sama sebagai langkah "pencampuran", kita melihat bahwa setiap putaran AES memiliki struktur jaringan substitusi-permutasi: sub-kunci putaran pertama kali di-XOR dengan masukan ke putaran saat ini; selanjutnya, fungsi kecil yang dapat dibalik diterapkan pada "potongan" dari nilai yang dihasilkan; akhirnya, potongan-potongan hasilnya dicampur untuk mendapatkan difusi. Satu-satunya perbedaan adalah, tidak seperti penjelasan kami sebelumnya tentang jaringan substitusi-permutasi, di sini langkah pencampuran tidak terdiri dari pengocokan bit yang sederhana namun dilakukan dengan menggunakan pengocokan ditambah transformasi linier yang dapat dibalik. (Sedikit menyederhanakan dan melihat contoh 3-bit yang sepele, mengacak bit-bit dari $x = x_1 \| x_2 \| x_3$ mungkin, misalnya, memetakan $x \text{ ke } x' = x_2 \| x_1 \| x_3$Transformasi linier yang dapat dibalik mungkin memetakan $x \text{ ke } x_1 \oplus x_2 \| x_2 \oplus x_3 \| x_1 \oplus x_2 \oplus x_3$)

Jumlah putaran tergantung pada panjang kunci. Sepuluh putaran digunakan untuk kunci 128-bit, 12 putaran untuk kunci 192-bit, dan 14 putaran untuk kunci 256-bit.

Keamanan AES. Seperti yang telah kami sebutkan, sandi AES menjadi sasaran pengawasan ketat selama proses seleksi dan terus dipelajari sejak saat itu. Sampai saat ini, tidak ada serangan kriptanalitik praktis yang secara signifikan lebih baik daripada pencarian kunci secara menyeluruh.

Kami menyimpulkan bahwa, saat ini, AES merupakan pilihan yang sangat baik untuk skema kriptografi apa pun yang memerlukan permutasi pseudorandom (kuat). Ini gratis, terstandarisasi, efisien, dan sangat aman.

***Kriptanalisis Diferensial dan Linier**

Block cipher relatif rumit, sehingga sulit untuk dianalisis. Meskipun demikian, seseorang tidak boleh terkecoh dengan berpikir bahwa sandi yang rumit sulit dipecahkan. Sebaliknya, sangat sulit untuk membuat cipher blok yang aman dan sangat mudah untuk menemukan serangan pada sebagian besar konstruksi (tidak peduli betapa rumitnya serangan tersebut). Hal ini harus menjadi peringatan bahwa non-ahli tidak boleh mencoba membuat sandi baru. Mengingat ketersediaan triple-DES dan AES, sulit untuk membenarkan penggunaan hal lain.

Pada bagian ini kami menjelaskan dua alat yang sekarang menjadi bagian standar dari kotak peralatan kriptanalis. Tujuan kami di sini adalah memberikan gambaran beberapa analisis kripto tingkat lanjut, serta memperkuat gagasan bahwa merancang cipher blok yang aman memerlukan pemilihan komponen-komponennya secara cermat.

Kriptanalisis diferensial. Teknik ini, yang dapat mengarah pada serangan teks biasa terpilih pada cipher blok, pertama kali diperkenalkan pada akhir tahun 1980an oleh Biham dan Shamir, yang menggunakannya untuk menyerang DES pada tahun 1993. Ide dasar di balik serangan ini adalah untuk membuat tabulasi perbedaan spesifik dalam blok cipher. masukan yang menyebabkan perbedaan tertentu dalam keluaran dengan probabilitas lebih besar dari yang diharapkan untuk permutasi acak. Secara khusus, misalkan diferensial (Δ_x, Δ_y) terjadi pada beberapa permutasi kunci F' dengan probabilitas p jika untuk input seragam $x_1$dan

x_2 memuaskan $x_1 \oplus x_2 = \Delta_x$, dan pemilihan kunci seragam k, probabilitas bahwa $F'_k(x_1) \oplus F'_k(x_1) = \Delta_y$ adalah p. Untuk sembarang tetap (Δ_x, Δ_y) dan (x_1, x_2) yang memenuhi $x_1 \oplus x_2 = \Delta_x$, jika kita memilih fungsi seragam $f: \{0,1\}^\ell \to \{0,1\}^\ell$, kita mempunyai $\Pr[f(x_1) \oplus f(x_2) = \Delta_y] = 2^{-\ell}$. Namun, dalam cipher blok yang lemah, mungkin ada perbedaan yang terjadi dengan probabilitas yang jauh lebih tinggi. Hal ini dapat dimanfaatkan untuk memberikan serangan pemulihan kunci penuh, seperti yang sekarang kami tunjukkan untuk SPN.

Kami menjelaskan ide dasarnya, dan kemudian mengerjakan contoh konkrit. Misalkan F adalah sebuah cipher blok dengan panjang blok ℓ-bit yang merupakan SPN putaran r, dan misalkan $F'_k(x)$ menunjukkan hasil antara dalam perhitungan $F_k(x)$ setelah menerapkan langkah pencampuran kunci pada putaran r. (Artinya, F' tidak termasuk substitusi S-box dan permutasi pencampuran pada putaran terakhir, serta langkah pencampuran kunci terakhir.) Katakanlah ada diferensial (Δ_x, Δ_y) dalam F' yang terjadi dengan probabilitas $p \gg 2^{-\ell}$. Dimungkinkan untuk mengeksploitasi diferensial probabilitas tinggi ini untuk mempelajari bit-bit dari sub-kunci pencampuran akhir k_{r+1}. Ide tingkat tinggi adalah sebagai berikut: misalkan $\{(x_1^i, x_2^i)\}_{i=1}^L$ adalah himpunan L pasang input acak dengan diferensial Δ_x, yakni dengan $x_1^i \oplus x_2^i = \Delta_x$ untuk semua i. Dengan menggunakan serangan teks biasa terpilih, dapatkan nilai $y_1^i = F_k(x_1^i)$ dan $y_2^i = F_k(x_2^i)$ untuk semua i. Sekarang, untuk semua bitstring yang mungkin $k^* \in \{0,1\}^\ell$, hitung $y_1^{\sim i} = F'_k(x_1^i)$ dan $y_2^{\sim i} = F'_k(x_2^i)$, dengan asumsi nilai sub-kunci terakhir k_{r+1} adalah k^*. Hal ini dilakukan dengan membalik langkah pencampuran kunci terakhir menggunakan k^*, dan kemudian membalik permutasi pencampuran dan S-box dari putaran r, yang tidak bergantung pada kunci master. Jika $k^* = k_{r+1}$, kita perkirakan pecahan p dari pasangan tersebut akan memenuhi $y_1^{\sim i} \oplus y_2^{\sim i} = \Delta_y$. Di sisi lain, ketika $k^* \neq k_{r+1}$ kita mungkin secara heuristik mengharapkan hanya pecahan $2^{-\ell}$ dari pasangan yang menghasilkan diferensial ini. Dengan mengatur L cukup besar, nilai yang benar dari subkunci akhir k_{r+1} dapat ditentukan.

Ini berfungsi, tetapi tidak terlalu efisien karena dalam setiap langkah kita menghitung lebih dari 2^ℓ nilai yang mungkin. Kita dapat melakukan lebih baik dengan menebak porsi k_{r+1} sekaligus. Lebih konkretnya, asumsikan S-box di F memiliki panjang input/output 1 byte, dan fokus pada byte pertama Δ_y, yang kita asumsikan bukan nol. Dimungkinkan untuk memverifikasi apakah diferensial bertahan dalam byte tersebut dengan menebak hanya 8 bit k_{r+1}, yaitu 8 bit yang sesuai (setelah permutasi pencampuran putaran-r) dengan keluaran S-box pertama. Jadi, dengan melanjutkan seperti di atas, kita dapat mempelajari 8 bit ini dengan menghitung semua nilai yang mungkin untuk bit tersebut, dan melihat nilai mana yang menghasilkan diferensial yang diinginkan pada byte pertama dengan probabilitas tertinggi. Tebakan yang salah untuk 8 bit tersebut menghasilkan perbedaan yang diharapkan dalam byte tersebut dengan probabilitas (heuristik) 2^{-8}, tetapi tebakan yang benar akan menghasilkan perbedaan yang diharapkan dengan probabilitas kira-kira $p + 2^{-8}$; ini karena dengan probabilitas p diferensial tersebut berlaku di seluruh blok (khususnya untuk byte pertama), dan jika hal ini tidak terjadi maka kita dapat memperlakukan diferensial pada byte pertama

sebagai acak. Perhatikan bahwa perbedaan yang berbeda mungkin diperlukan untuk mempelajari bagian k_{r+1} yang berbeda.

Dalam praktiknya, berbagai pengoptimalan dilakukan untuk meningkatkan efektivitas pengujian di atas atau, lebih khusus lagi, untuk meningkatkan kesenjangan antara probabilitas bahwa tebakan yang salah akan menghasilkan perbedaan vs. Salah satu optimasinya adalah dengan menggunakan diferensial berbobot rendah di mana Δ_y memiliki banyak byte nol pada posisi yang masuk ke S-box pada putaran r. Setiap pasangan $\tilde{y}_1, \tilde{y}_2$ yang memenuhi diferensial tersebut mempunyai nilai yang sama yang memasuki banyak kotak S pada putaran r, dan dengan demikian akan menghasilkan nilai keluaran y_1, y_2 yang sama pada posisi bit yang sesuai (tergantung pada permutasi pencampuran akhir). Ini berarti bahwa ketika melakukan pengujian yang dijelaskan sebelumnya, seseorang dapat dengan mudah membuang pasangan apa pun (y_1^i, y_2^i) yang tidak cocok pada posisi bit tersebut (karena nilai antara yang bersangkutan $(\tilde{y}_1, \tilde{y}_2)$ tidak mungkin memenuhi diferensial, untuk setiap pilihan subkunci terakhir). Hal ini secara signifikan meningkatkan efektivitas serangan.

Setelah k_{r+1} diketahui, penyerang dapat "mengupas" langkah pencampuran kunci terakhir, serta langkah permutasi pencampuran dan substitusi S-box pada putaran r (karena ini tidak bergantung pada kunci master), dan kemudian menerapkan serangan yang sama—menggunakan diferensial yang berbeda—untuk menemukan sub-kunci k_r putaran ke-r, dan seterusnya, hingga ia mempelajari semua sub-kunci (atau, setara, seluruh kunci master).

Kami mengerjakan contoh "mainan", yang juga mengilustrasikan bagaimana perbedaan yang baik dapat ditemukan. Kami menggunakan SPN empat putaran dengan panjang blok 16 bit, berdasarkan pada satu S-box dengan panjang input/output 4-bit. S-box didefinisikan sebagai berikut (tabel menunjukkan bagaimana setiap masukan 4-bit dipetakan ke keluaran 4-bit):

Input:	0000	0001	0010	0011	0100	0101	0110	0111
Output:	0000	1011	0101	0001	0110	1000	1101	0100
Input:	1000	1001	1010	1011	1100	1101	1110	1111
Output:	1111	0111	0010	1100	1001	0011	1110	1010

Permutasi pencampuran, yang menunjukkan ke mana setiap bit dipindahkan untuk masing-masing 16 bit dalam satu blok, adalah sebagai berikut:

In:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Out:	7	2	3	8	12	5	11	9	10	1	14	13	4	6	16	15

x	S(x)	x ⊕ 1111	S(x ⊕ 1111)	S(x) ⊕ S(x ⊕ 1111)
0000	0000	1111	1010	**1010**
0001	1011	1110	1110	0101
0010	0101	1101	0011	0110
0011	0001	1100	1001	1000
0100	0110	1011	1100	**1010**
0101	1000	1010	0010	**1010**
0110	1101	1001	0111	**1010**
0111	0100	1000	1111	1011
1000	1111	0111	0100	1011
1001	0111	0110	1101	**1010**
1010	0010	0101	1000	**1010**
1011	1100	0100	0110	**1010**
1100	1001	0011	0001	1000
1101	0011	0010	0101	0110
1110	1110	0001	1011	0101
1111	1010	0000	0000	**1010**

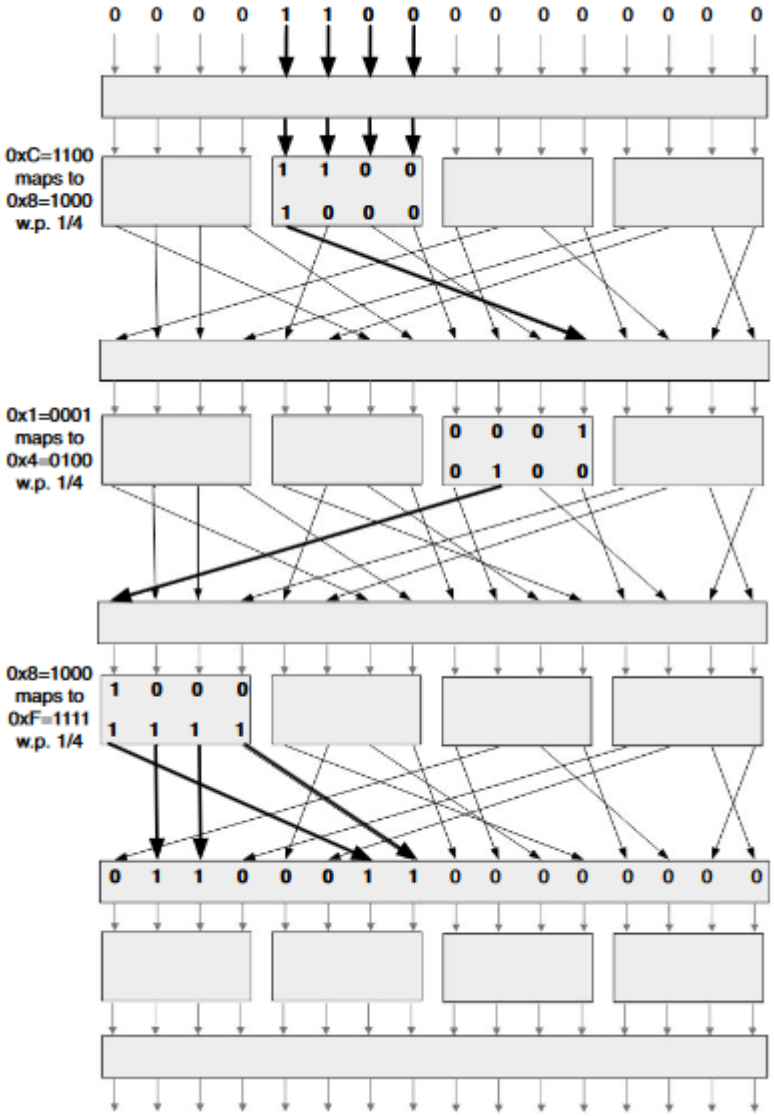
GAMBAR 6.8: Pengaruh perbedaan masukan Δ_x= 1111 pada S-box kita.

Pertama-tama kita temukan perbedaannya di S-box. Misalkan $S(x)$ menyatakan keluaran S-box pada masukan x. Pertimbangkan diferensial Δ_x = 1111. Maka, misalnya, kita mempunyai $S(0000) \oplus S(1111) = 0000 \oplus 1010 = 1010$ dan dalam hal ini perbedaan masukan sebesar 1111 menyebabkan perbedaan keluaran sebesar 1010. Mari kita lihat apakah hubungan ini sering terjadi. Kita mempunyai $S(0001) = 1011$ dan $S(0001 \oplus 1111) = S(1110) = 1110$, jadi di sini perbedaan masukan sebesar 1111 tidak menyebabkan perbedaan keluaran sebesar 1010. Namun, $S(0100) = 0110$ dan $S(0100 \oplus 1111) = S(1011) = 1100$ dan dalam kasus ini, selisih masukan sebesar 1111 menghasilkan selisih keluaran sebesar 1010. Pada Gambar 6.8 kita telah membuat tabulasi hasil untuk semua masukan yang mungkin. Kita melihat bahwa separuh waktu perbedaan masukan sebesar 1111 menghasilkan selisih keluaran sebesar 1010. Jadi, (1111, 1010) adalah diferensial S yang terjadi dengan probabilitas 1/2.

Input Difference Δ_x \ Output Difference Δ_y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	4	0	0	0	2	2	2	2	0	0	4	0
2	0	0	0	0	0	2	0	2	0	2	2	4	2	2	0	0
3	0	2	2	4	0	4	0	0	0	0	0	0	0	2	2	0
4	0	0	0	2	2	2	6	0	2	0	0	0	2	0	0	0
5	0	2	2	0	0	0	0	0	4	0	0	0	4	2	2	0
6	0	2	0	2	0	0	0	0	0	2	0	2	0	4	0	4
7	0	2	0	0	2	4	2	2	0	2	0	0	0	2	0	0
8	0	0	0	2	0	0	0	2	0	0	0	2	2	2	2	4
9	0	2	0	2	2	2	0	4	0	2	2	0	0	0	0	0
A	0	0	4	0	2	0	2	4	2	0	2	0	0	0	0	0
B	0	0	2	0	0	0	2	0	0	2	0	0	4	2	4	0
C	0	0	0	0	0	0	0	0	4	4	0	4	0	0	0	4
D	0	4	2	2	0	0	2	2	0	0	0	0	0	0	0	4
E	0	2	4	2	4	0	0	0	0	0	0	0	2	0	2	0
F	0	0	0	0	0	2	2	0	2	0	**8**	2	0	0	0	0

GAMBAR 6.9: Diferensial pada S-box kita.

Proses yang sama dapat dilakukan untuk 2^4 perbedaan masukan Δ_x untuk menghitung probabilitas setiap diferensial. Yaitu, untuk setiap pasangan (Δ_x, Δ_y) kita mentabulasikan jumlah input 4-bit x yang mana $S(x) \oplus S(x \oplus \Delta_x) = \Delta_y$. Kita telah melakukan ini untuk contoh S-box pada Gambar 6.9. (Agar ringkasnya kami merepresentasikan (Δ_x, Δ_y) menggunakan notasi heksadesimal.) Tabelnya harus dibaca sebagai berikut: entri (i, j) menghitung berapa banyak input dengan perbedaan i yang dipetakan ke output dengan perbedaan j. Perhatikan, misalnya, ada 8 masukan dengan selisih $0 \times \mathrm{F} = 1111$ yang dipetakan ke keluaran $0 \times \mathrm{A} = 1010$, seperti yang telah kami tunjukkan di atas. Ini adalah diferensial dengan probabilitas tertinggi (terlepas dari diferensial sepele $(0,0)$). Namun terdapat perbedaan menarik lainnya: perbedaan masukan sebesar $0 \times 4 = 0100$ dipetakan ke perbedaan keluaran sebesar $0 \times 6 = 0110$dengan probabilitas $6/16 = 3/8$, dan terdapat beberapa perbedaan dengan probabilitas $4/16 = 1/4$.



GAMBAR 6.10: Menelusuri perbedaan melalui SPN empat putaran yang menggunakan S-box dan permutasi pencampuran yang diberikan dalam teks.

Kami sekarang memperluas ini untuk menemukan perbedaan yang baik untuk tiga putaran pertama SPN. Pertimbangkan untuk mengevaluasi SPN pada dua masukan yang memiliki diferensial 0000 1100 0000 0000, dan menelusuri perbedaan antara nilai-nilai antara pada setiap langkah evaluasi ini. (Lihat Gambar 6.10, yang menunjukkan empat putaran penuh SPN ditambah langkah pencampuran kunci terakhir. Untuk lebih jelasnya, gambar tersebut menghilangkan permutasi pencampuran pada putaran ke-4; bahwa permutasi pencampuran hanya mempunyai efek pengocokan bit-bitnya. dari diferensial, sehingga dapat dengan mudah diperhitungkan dalam serangan.) Langkah pencampuran kunci pada putaran pertama tidak mempengaruhi diferensial, sehingga input ke S-box kedua pada putaran pertama memiliki diferensial 1100. Kita melihat dari Gambar 6.9 bahwa selisih $0 \times \mathrm{C} = 1100$ pada masukan ke S-box menghasilkan selisih $0 \times 8 = 1000$ pada keluaran S-box dengan probabilitas 1/4. Jadi dengan probabilitas 1/4 diferensial keluaran S-box ke-2 setelah putaran ke-1 adalah satu bit yang digerakkan oleh permutasi pencampuran dari posisi ke-5 ke posisi

ke-12. (Masukan ke S-box yang lain adalah sama, jadi keluarannya sama dan selisih keluarannya adalah 0000.) Dengan asumsi demikian, selisih masukan ke S-box ketiga pada putaran kedua adalah $0 \times 1 = 0001$ (sekali lagi, langkah pencampuran kunci pada putaran kedua tidak mempengaruhi diferensial); menggunakan Gambar 6.9 kita mendapatkan bahwa dengan probabilitas 1/4 perbedaan keluaran dari S-box tersebut adalah $0 \times 4 = 0100$. Jadi, sekali lagi hanya ada satu bit keluaran yang berbeda, dan dipindahkan dari posisi ke-10 ke posisi pertama. posisi dengan permutasi pencampuran. Terakhir, dengan melihat kembali Gambar 6.9, kita melihat bahwa selisih masukan sebesar $0 \times 8 = 1000$ pada S-box menghasilkan selisih keluaran sebesar $0 \times \mathrm{F} = 1111$ dengan probabilitas 1/4. Bit-bit pada posisi 1, 2, 3, dan 4 kemudian dipindahkan dengan permutasi pencampuran ke posisi 7, 2, 3, dan 8.

Jadi, secara keseluruhan, kita melihat bahwa selisih masukan sebesar $\Delta_x = 0000\ 1100\ 0000\ 0000$ menghasilkan selisih keluaran $\Delta_y = 0110\ 0011\ 0000\ 0000$ setelah tiga putaran dengan probabilitas paling sedikit $\frac{1}{4} \cdot \frac{1}{4} \cdot \frac{1}{4} = \frac{1}{64} \cdot ^7$ (Kita kalikan probabilitasnya karena kita mengasumsikan independensi setiap putaran secara heuristik.) Untuk fungsi acak, probabilitas terjadinya diferensial tertentu hanyalah $2^{-16} = 1/65536$. Jadi, diferensial yang kami temukan muncul dengan probabilitas yang jauh lebih tinggi daripada yang diharapkan untuk fungsi acak. Amati juga bahwa kami telah menemukan perbedaan bobot rendah.

Kita dapat menggunakan diferensial ini untuk menemukan 8 bit pertama dari subkunci terakhir k_5. Seperti yang telah dibahas sebelumnya, kita mulai dengan membiarkan $\{(x_1^i, x_2^i)\}_{i=1}^{L}$ menjadi himpunan L pasang input acak dengan diferensial Δ_x. Dengan menggunakan serangan teks biasa terpilih, kita kemudian memperoleh nilai $y_1^i = F_k(x_1^i)$ dan $y_2^i = F_k(x_2^i)$ untuk semua i. Sekarang, untuk semua nilai yang mungkin untuk 8 bit awal k_5, kita menghitung 8 bit awal $(y_1^{\sim i}, y_2^{\sim i})$, nilai antara setelah langkah pencampuran kunci pada putaran ke-4. (Kita dapat melakukan ini karena kita hanya perlu membalik dua S-box paling kiri pada putaran ke-4 untuk mendapatkan 8 bit tersebut.) Ketika kita menebak nilai yang benar untuk 8 bit awal k_5, kita mengharapkan 8 bit -bit differential 0110 0011 terjadi dengan probabilitas minimal 1/64. Secara heuristik, tebakan yang salah menghasilkan perbedaan yang diharapkan hanya dengan probabilitas $2^{-8} = 1/256$. Dengan menetapkan L cukup besar, kita dapat (dengan probabilitas tinggi) mengidentifikasi nilai yang benar.

Serangan diferensial dalam praktiknya. Kriptanalisis diferensial sangat kuat dan telah digunakan untuk menyerang sandi yang sebenarnya. Contoh yang menonjol adalah FEAL-8, yang diusulkan sebagai alternatif DES pada tahun 1987. Ditemukan serangan diferensial pada FEAL-8 yang hanya membutuhkan 1.000 teks biasa yang dipilih. Pada tahun 1991, dibutuhkan waktu kurang dari 2 menit menggunakan serangan ini untuk menemukan seluruh kuncinya. Saat ini, setiap sandi yang diusulkan diuji ketahanannya terhadap kriptanalisis diferensial.

Serangan diferensial adalah serangan pertama pada DES yang membutuhkan waktu lebih sedikit dibandingkan pencarian brute force sederhana. Meskipun merupakan hasil teoritis yang menarik, serangan ini tidak menimbulkan kekhawatiran yang signifikan dalam praktiknya karena memerlukan 2^{47} teks biasa yang dipilih. Sangat sulit bagi penyerang untuk

mendapatkan pasangan teks biasa/teks sandi sebanyak ini di sebagian besar aplikasi dunia nyata. Menariknya, modifikasi kecil pada S-box DES membuat sandi lebih rentan terhadap serangan diferensial. Kesaksian pribadi para perancang DES (setelah serangan diferensial ditemukan di dunia luar) telah mengkonfirmasi bahwa S-box DES dirancang khusus untuk menggagalkan serangan diferensial.

Kriptanalisis linier. Kriptanalisis linier dikembangkan oleh Matsui pada awal tahun 1990an. Kami hanya akan menjelaskan tekniknya pada tingkat tinggi. Ide dasarnya adalah untuk mempertimbangkan hubungan linier antara masukan dan keluaran yang memiliki probabilitas lebih tinggi daripada yang diharapkan untuk fungsi acak. Secara lebih rinci, katakanlah posisi bit $i_1, \ldots, i_{in}$ dan $i_1, \ldots, i_{out}$ memiliki bias linier ε jika, untuk x seragam dan k, dan $y \stackrel{\text{def}}{=} F_k(x)$, maka dinyatakan bahwa:

$$\left|\Pr\left[x_{i_1} \oplus \ldots \oplus x_{i_{in}} \oplus y_{i\prime_1} \oplus \ldots \oplus y_{i\prime_{out}} = 0\right] - \frac{1}{2}\right| = \varepsilon$$

Dimana x_i, y_i menunjukkan bit ke-i dari x dan y. Untuk fungsi acak dan set posisi bit tetap, kita perkirakan biasnya mendekati 0. Matsui menunjukkan cara menggunakan bias yang cukup besar dalam sandi F untuk menemukan kunci rahasia. Selain memberikan metode lain untuk menyerang cipher, fitur penting dari serangan ini adalah bahwa serangan ini tidak memerlukan teks biasa yang dipilih, melainkan teks biasa yang diketahui saja sudah cukup. Hal ini sangat penting, karena file yang dienkripsi dapat menyediakan sejumlah besar teks biasa yang diketahui, sedangkan pengumpulan enkripsi dari teks biasa yang dipilih jauh lebih sulit. Matsui menunjukkan bahwa DES dapat dipecahkan hanya dengan 2^{43} pasangan teks biasa/teks sandi yang diketahui.

Dampak pada desain blok-cipher. Cipher blok modern dirancang dan dievaluasi berdasarkan, sebagian, pada ketahanannya terhadap analisis sandi diferensial dan linier. Saat membuat cipher blok, perancang memilih S-box dan komponen lainnya untuk meminimalkan probabilitas diferensial dan bias linier. Kami mencatat bahwa tidak mungkin untuk menghilangkan semua perbedaan probabilitas tinggi dalam S-box: setiap S-box akan memiliki beberapa diferensial yang lebih sering muncul dibandingkan yang lain. Meski demikian, penyimpangan-penyimpangan tersebut masih bisa diminimalisir. Selain itu, meningkatkan jumlah putaran (dan memilih permutasi pencampuran dengan hati-hati) dapat mengurangi probabilitas diferensial serta mempersulit kriptanalis untuk menemukan perbedaan apa pun untuk dieksploitasi.

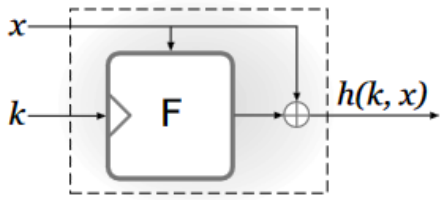
6.3 FUNGSI HASH

Ingat dari Bab 5 bahwa persyaratan keamanan utama untuk fungsi hash H adalah ketahanan terhadap tabrakan; artinya, akan sulit untuk menemukan tumbukan, atau masukan berbeda x, x' sedemikian rupa sehingga $H(x) = H(x')$. (Kami tidak menyebutkan kunci apa pun di sini, karena fungsi hash dunia nyata umumnya tidak memiliki kunci.) Jika fungsi hash memiliki panjang keluaran ℓ-bit, maka hal terbaik yang dapat kita harapkan adalah tidak mungkin menemukan tabrakan menggunakan secara substansial kurang dari $2^{\ell/2}$ pemanggilan H. Kami juga ingin fungsi hash mencapai ketahanan preimage (kedua) terhadap

serangan yang berjalan dalam waktu kurang dari 2^{ℓ}, meskipun kami tidak mempertimbangkan serangan seperti itu di kami diskusi di sini. Fungsi hash umumnya dibangun dalam dua langkah. Pertama, fungsi kompresi (yaitu, fungsi hash dengan panjang tetap) h dirancang; selanjutnya, beberapa mekanisme digunakan untuk memperluas h sehingga dapat menangani panjang masukan yang berubah-ubah. Di Bagian 5.2 kami telah menunjukkan satu pendekatan—transformasi Merkle–Damg˚ard—untuk langkah kedua. Di sini, kami mengeksplorasi teknik untuk merancang fungsi kompresi yang mendasarinya. Kami juga mendiskusikan beberapa fungsi hash yang digunakan dalam praktik. Konstruksi teoritis fungsi kompresi berdasarkan asumsi teori bilangan diberikan pada Bagian 8.4.

Fungsi Hash dari Block Cipher

Mungkin mengejutkan, adalah mungkin untuk membangun fungsi kompresi tahan benturan dari cipher blok yang memenuhi properti tambahan tertentu. Ada beberapa cara untuk melakukan hal ini; salah satu yang paling umum adalah melalui konstruksi Davies – Meyer. Misalkan F adalah sebuah cipher blok dengan panjang kunci n-bit dan panjang blok ℓ-bit. Kita kemudian dapat mendefinisikan fungsi kompresi $h : \{0,1\}^{n+\ell} \rightarrow \{0,1\}^{\ell}$ dengan $h(k,x) \stackrel{\text{def}}{=} F_k(x) \oplus x$ (Lihat Gambar 6.11.)



GAMBAR 6.11: Konstruksi Davies–Meyer.

Kita tidak tahu bagaimana membuktikan ketahanan tumbukan dari fungsi kompresi yang dihasilkan hanya berdasarkan asumsi bahwa F adalah permutasi pseudorandom kuat, dan faktanya ada alasan untuk meyakini bahwa pembuktian seperti itu tidak mungkin dilakukan. Namun, kita dapat membuktikan ketahanan tumbukan jika kita ingin memodelkan F sebagai sandi yang ideal. Model sandi ideal merupakan penguatan dari model oracle acak (lihat Bagian 5.5), yang mana kita mengandaikan bahwa semua pihak mempunyai akses ke oracle untuk permutasi kunci acak $F: \{0,1\}^n \times \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell}$ serta inversnya F^{-1} (yaitu, sehingga $F^{-1}\big(k, F(k,x)\big) = x$ untuk semua k, x). Cara lain untuk memikirkan hal ini adalah bahwa setiap kunci $k \in \{0,1\}^n$ menentukan permutasi seragam dan independen $F\,(k,\ \cdot)$ pada string ℓ-bit. Seperti dalam model oracle acak, satu-satunya cara untuk menghitung F (atau F^{-1}) adalah dengan menanyakan oracle secara eksplisit dengan (k,x) dan menerima kembali $F(k,x)$ (atau $F^{-1}(k,x)$).

Menganalisis konstruksi dalam model sandi ideal disertai dengan semua kelebihan dan kekurangan bekerja dalam model oracle acak, seperti yang dibahas panjang lebar di Bagian 5.5. Kami hanya menambahkan di sini bahwa model sandi ideal menyiratkan tidak adanya serangan kunci terkait, dalam arti bahwa (seperti yang baru saja kami katakan) permutasi $F(k,\cdot)$ dan $F(k',\cdot)$ harus berperilaku independen bahkan jika, misalnya, k dan k' berbeda

hanya dalam satu bit. Selain itu, tidak ada "kunci lemah" k (katakanlah, kunci semua-0) yang mana $F(k,\cdot)$ mudah dibedakan dari acak. Ini juga berarti bahwa $F(k,\cdot)$ harus "berperilaku acak" bahkan ketika k diketahui. Untuk sandi F di dunia nyata mana pun, sifat-sifat ini belum tentu berlaku (dan bahkan tidak terdefinisi dengan baik) bahkan jika F adalah permutasi pseudorandom yang kuat, dan pembaca mungkin memperhatikan bahwa kami belum membahas sifat-sifat ini dalam analisis kami. konstruksi block-cipher dunia nyata. (Faktanya, DES dan triple-DES tidak memenuhi sifat-sifat ini.) Setiap blok cipher yang digunakan untuk membuat instance cipher ideal harus dievaluasi sehubungan dengan persyaratan yang lebih ketat ini.

Kita buktikan teorema berikut dalam keadaan konkrit, namun pembuktiannya juga dapat diadaptasi dengan mudah untuk keadaan asimtotik.

TEOREMA 6.5 Jika F dimodelkan sebagai sandi ideal, maka konstruksi Davies–Meyer menghasilkan fungsi kompresi tahan tumbukan. Konkritnya, penyerang mana pun yang membuat kueri $q < 2^{\ell/2}$ ke oracle cipher idealnya akan menemukan tabrakan dengan probabilitas paling banyak $q^2/2^{\ell}$.

BUKTI. Agar lebih jelas, di sini kita pertimbangkan eksperimen probabilistik di mana F diambil sampelnya secara acak (lebih tepatnya, untuk setiap $k \in \{0,1\}^n$ fungsi $F(k,\cdot) : \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell}$dipilih secara seragam dari atur Perm_{ℓ} permutasi pada string ℓ-bit) dan kemudian penyerang diberikan akses oracle ke F dan F^{-1}. Penyerang kemudian mencoba menemukan pasangan yang bertabrakan $(k,x),(k',x')$, yaitu $F(k,x)\oplus x = F(k',x')\oplus x'$. Tidak ada batasan komputasi yang ditempatkan pada penyerang selain membatasi jumlah kueri Oracle yang dibuatnya. Kami berasumsi bahwa jika penyerang menghasilkan pasangan yang bertabrakan $(k,x),(k',x')$, maka sebelumnya ia telah membuat kueri oracle yang diperlukan untuk menghitung nilai $F(k,x)$ dan $F(k',x')$. Kami juga berasumsi bahwa penyerang tidak pernah membuat kueri yang sama lebih dari satu kali, dan tidak pernah menanyakan $F^{-1}(k,y)$ setelah mengetahui bahwa $y = F(k,x)$ (dan sebaliknya). Semua asumsi ini tidak kehilangan sifat umum.

Pertimbangkan permintaan ke$-i$ yang dibuat penyerang terhadap oracle-nya. Kueri (k_i,x_i) ke F hanya mengungkapkan nilai hash $h \stackrel{\text{def}}{=} h(k_i,x_i) = F(k_i,x_i) \oplus x_i$; sama halnya, kueri ke F^{-1} memberikan hasil $x_i = F^{-1}(k_i,x_i)$ hanya menghasilkan nilai hash $h \stackrel{\text{def}}{=} h(k_i,x_i) = y_i \oplus F^{-1}(k_i,x_i)$. Penyerang tidak mendapatkan tabrakan kecuali $h_i = h_j$ untuk beberapa $i \neq j$.

Perbaiki i,j dengan $i > j$ dan pertimbangkan probabilitas $h_i = h_j$. Pada saat query ke-i, nilai h_j adalah tetap. Tabrakan antara h_i dan h_j diperoleh pada kueri ke-i hanya jika penyerang menanyakan (k_i,x_i) ke F dan memperoleh hasil $F(k_i,x_i) = h_j\oplus x_i$, atau menanyakan (k_i,y_i) ke F^{-1} dan memperoleh hasil $F^{-1}(k_i,y_i) = h_j\oplus y_i$. Salah satu peristiwa terjadi dengan probabilitas paling banyak $1/(2^{\ell} - (i-1))$ karena, misalnya, $F(k_i,x_i)$ seragam pada $\{0,1\}^{\ell}$ kecuali bahwa ia tidak dapat sama dengan nilai apa pun $F(k_i,x)$ sudah ditentukan oleh (paling banyak) $i-1$ kueri Oracle sebelumnya milik penyerang menggunakan kunci k_i. Karena $i \leq q < 2^{\ell/2}$, peluang terambilnya $h_i = h_j$ paling banyak adalah $2/2^{\ell}$,

Mengambil gabungan terikat pada semua $\binom{q}{2} < q^2/2$ pasangan berbeda i, j memberikan hasil yang dinyatakan dalam teorema.

Davies – Meyer dan DES. Seperti yang telah kami sebutkan di atas, seseorang harus berhati-hati saat membuat contoh konstruksi Davies – Meyer dengan cipher blok beton apa pun, karena cipher tersebut harus memenuhi properti tambahan (selain permutasi pseudorandom yang kuat) agar konstruksi yang dihasilkan aman. Dalam Latihan 6.21 kita mengeksplorasi apa yang salah ketika DES digunakan dalam konstruksi Davies–Meyer.

Hal ini harus menjadi peringatan bahwa bukti keamanan konstruksi Davies-Meyer dalam model sandi ideal tidak selalu berarti keamanan nyata ketika dipakai dengan sandi sebenarnya. Namun demikian, seperti yang akan kami jelaskan di bawah, paradigma ini telah digunakan untuk membangun fungsi hash praktis yang tahan terhadap serangan (tetapi khususnya ketika cipher blok di tengah konstruksi dirancang khusus untuk tujuan ini).

Kesimpulannya, konstruksi Davies-Meyer adalah paradigma yang berguna untuk membangun fungsi kompresi tahan tumbukan. Namun, ini tidak boleh diterapkan pada cipher blok yang dirancang untuk enkripsi, seperti DES dan AES.

MD5

MD5 adalah fungsi hash dengan panjang keluaran 128-bit. Ini dirancang pada tahun 1991 dan untuk beberapa waktu diyakini tahan benturan. Selama jangka waktu beberapa tahun, berbagai kelemahan mulai ditemukan pada MD5 namun hal ini tampaknya tidak memberikan cara yang mudah untuk menemukan tabrakan. Yang mengejutkan, pada tahun 2004 sebuah tim cryptanalyst Tiongkok menyajikan metode baru untuk menemukan tabrakan di MD5; mereka dengan mudah dapat meyakinkan orang lain bahwa pendekatan mereka benar dengan menunjukkan benturan yang jelas! Sejak itu, serangannya telah ditingkatkan dan saat ini tabrakan dapat ditemukan dalam waktu kurang dari satu menit di PC desktop. Selain itu, serangan telah diperluas sehingga bahkan “tabrakan terkontrol” (misalnya, dua file postscript yang menghasilkan konten yang dapat dilihat secara sewenang-wenang) dapat ditemukan. Karena serangan ini, MD5 tidak boleh digunakan di mana pun keamanan kriptografi diperlukan. Kami menyebutkan MD5 hanya karena masih ditemukan dalam kode lama.

SHA-0, SHA-1, dan SHA-2

Algoritma Hash Aman (SHA) mengacu pada serangkaian fungsi hash kriptografi yang distandarisasi oleh NIST. Mungkin yang paling terkenal adalah SHA-1, yang diperkenalkan pada tahun 1995. Algoritma ini memiliki panjang keluaran 160-bit dan menggantikan pendahulunya yang disebut SHA-0, yang ditarik karena ditemukan kelemahan yang tidak ditentukan dalam algoritma tersebut.

Pada saat tulisan ini dibuat, tabrakan eksplisit belum ditemukan di SHA-1. Namun, analisis teoritis selama beberapa tahun terakhir menunjukkan bahwa tabrakan di SHA-1 dapat ditemukan dengan menggunakan evaluasi fungsi hash yang jauh lebih sedikit dari 280 yang diperlukan untuk menggunakan serangan ulang tahun, dan diperkirakan bahwa tabrakan akan segera ditemukan. Oleh karena itu disarankan untuk bermigrasi ke SHA-2, yang saat ini

tampaknya tidak memiliki kelemahan yang sama. SHA-2 terdiri dari dua fungsi terkait: SHA-256 dan SHA-512, dengan panjang keluaran masing-masing 256 dan 512-bit.

Semua fungsi hash dalam keluarga SHA dibangun menggunakan desain dasar yang sama, yang menggabungkan komponen-komponen yang telah kita lihat: Fungsi kompresi pertama-tama didefinisikan dengan menerapkan konstruksi Davies-Meyer pada cipher blok, dan ini kemudian diperluas untuk mendukung input panjang sewenang-wenang menggunakan transformasi Merkle – Damg˚ard. Satu hal yang menarik di sini adalah bahwa blok cipher dalam setiap kasus dirancang khusus untuk membangun fungsi kompresi. Faktanya, hanya secara surut komponen yang mendasari fungsi kompresi diisolasi dan dianalisis sebagai cipher blok SHACAL-1 (untuk SHA-1) dan SHACAL-2 (untuk SHA-2). Cipher ini sendiri menarik, karena memiliki panjang blok yang besar (masing-masing 160 dan 256 bit) dan kunci 512-bit.

SHA-3 (Keccak)

Sebagai buntut dari serangan tabrakan pada MD5 dan kelemahan teoretis yang ditemukan pada SHA-1, NIST pada akhir tahun 2007 mengumumkan kompetisi publik untuk merancang fungsi hash kriptografi baru yang disebut SHA-3. Algoritme yang dikirimkan harus mendukung setidaknya panjang keluaran 256 dan 512-bit. Seperti halnya kompetisi AES sekitar 10 tahun sebelumnya, kompetisi ini sepenuhnya terbuka dan transparan; siapa pun dapat mengajukan algoritma untuk dipertimbangkan, dan masyarakat diundang untuk menyampaikan pendapatnya mengenai kandidat mana pun. 51 kandidat putaran pertama dipersempit menjadi 14 pada bulan Desember 2008, dan selanjutnya dikurangi menjadi lima finalis pada tahun 2010. Kandidat yang tersisa harus diawasi secara ketat oleh komunitas kriptografi selama dua tahun berikutnya. Pada bulan Oktober 2012, NIST mengumumkan pemilihan Keccak sebagai pemenang kompetisi. Pada saat penulisan ini, algoritma ini sedang menjalani standarisasi sebagai pengganti SHA-2 generasi berikutnya.

Keccak tidak biasa dalam beberapa hal. (Menariknya, salah satu alasan Keccak dipilih adalah karena strukturnya sangat berbeda dengan SHA-1 dan SHA-2.) Pada intinya, ini didasarkan pada permutasi tanpa kunci f dengan panjang blok besar 1600 bit; ini sangat berbeda dari, misalnya, konstruksi Davies – Meyer, yang mengandalkan permutasi kunci. Selain itu, Keccak tidak menggunakan transformasi Merkle–Damg˚ard untuk menangani panjang masukan yang berubah-ubah. Sebaliknya, ia menggunakan pendekatan baru yang disebut konstruksi spons. Keccak—dan konstruksi spons secara lebih umum—dapat dianalisis dalam model permutasi acak di mana kita mendalilkan bahwa pihak-pihak mempunyai akses ke oracle untuk permutasi acak $f: \{0,1\}^{\ell} \rightarrow \{0,1\}^{\ell}$ (dan mungkin kebalikannya). Ini lebih lemah dari model sandi ideal; memang, kita dapat dengan mudah mendapatkan permutasi acak dalam model sandi ideal hanya dengan menetapkan kunci sandi menjadi nilai konstan apa pun.

Akan sangat menarik untuk menyaksikan standar hash baru berkembang, dan melihat seberapa cepat pengembang beradaptasi dari SHA-1/SHA-2 ke SHA-3 yang lebih baru.

BAB 7
KONSTRUKSI TEORITIS PRIMITIF KUNCI SIMETRIS

Pada Bab 3 kami memperkenalkan gagasan keacakan semu dan mendefinisikan beberapa primitif kriptografi dasar termasuk generator, fungsi, dan permutasi pseudorandom. Kami telah menunjukkan di Bab 3 dan 4 bahwa primitif ini berfungsi sebagai blok bangunan untuk semua kriptografi kunci privat. Oleh karena itu, sangatlah penting untuk memahami primitif ini dari sudut pandang teoritis. Dalam bab ini kami secara formal memperkenalkan konsep fungsi satu arah—fungsi yang, secara informal, mudah dihitung namun sulit dibalik—dan menunjukkan bagaimana generator, fungsi, dan permutasi pseudorandom dapat dibangun dengan asumsi tunggal bahwa ada fungsi satu arah.1 Selain itu, kita akan melihat bahwa fungsi satu arah diperlukan untuk kriptografi kunci privat yang "non-trivial". Artinya: keberadaan fungsi satu arah setara dengan keberadaan semua kriptografi kunci privat (non-trivial). Ini adalah salah satu kontribusi besar kriptografi modern.

Konstruksi yang kami tunjukkan dalam bab ini harus dipandang sebagai pelengkap konstruksi stream cipher dan block cipher yang dibahas dalam bab sebelumnya. Fokus bab sebelumnya adalah bagaimana berbagai kriptografi primitif saat ini direalisasikan dalam praktik, dan tujuan bab tersebut adalah untuk memperkenalkan beberapa pendekatan dasar dan prinsip desain yang digunakan. Namun yang agak mengecewakan adalah kenyataan bahwa tidak satu pun konstruksi yang kami tunjukkan dapat terbukti aman berdasarkan asumsi yang lebih lemah (yaitu lebih masuk akal). Sebaliknya, dalam bab ini kita akan membuktikan bahwa membangun permutasi pseudorandom dapat dimulai dari asumsi yang sangat ringan bahwa fungsi satu arah ada. Asumsi ini lebih masuk akal daripada asumsi, katakanlah, bahwa AES adalah permutasi pseudorandom, karena asumsi kualitatifnya lebih lemah dan juga karena kita mempunyai sejumlah kandidat, fungsi satu arah teori bilangan yang telah dipelajari banyak orang. tahun, bahkan sebelum munculnya kriptografi. (Lihat bagian awal Bab 6 untuk pembahasan lebih lanjut mengenai hal ini.) Namun kelemahannya adalah konstruksi yang kami tunjukkan di sini kurang efisien dibandingkan dengan Bab 6, sehingga tidak benar-benar digunakan. Masih menjadi tantangan penting bagi para kriptografer untuk "menjembatani kesenjangan ini" dan mengembangkan konstruksi generator pseudorandom, fungsi, dan permutasi yang terbukti aman yang efisiensinya sebanding dengan stream cipher dan block cipher terbaik yang tersedia.

Fungsi hash tahan benturan. Berbeda dengan bab sebelumnya, di sini kita tidak mempertimbangkan fungsi hash yang tahan benturan. Alasannya adalah konstruksi fungsi hash dari fungsi satu arah tidak diketahui dan, pada kenyataannya, terdapat bukti yang menunjukkan bahwa konstruksi seperti itu tidak mungkin dilakukan. Kita akan beralih ke konstruksi fungsi hash tahan benturan yang dapat dibuktikan.

Catatan tentang bab ini. Materi dalam bab ini agak lebih maju dibandingkan materi dalam bagian selanjutnya dalam buku ini. Materi ini tidak digunakan secara eksplisit di tempat lain, sehingga bab ini dapat dilewati jika diinginkan. Oleh karena itu, kami telah mencoba

menyajikan materi sedemikian rupa sehingga dapat dimengerti (dengan susah payah) bagi mahasiswa tingkat sarjana atau pascasarjana tingkat lanjut. Kami mendorong semua pembaca untuk membaca dengan teliti Bagian 7.1 dan 7.2, yang memperkenalkan fungsi satu arah dan memberikan gambaran keseluruhan bab ini. Kami percaya bahwa pemahaman terhadap setidaknya beberapa topik yang dibahas di sini cukup penting untuk menjamin upaya tersebut.

7.1 FUNGSI SATU ARAH

Pada bagian ini kita secara formal mendefinisikan fungsi satu arah, dan kemudian membahas secara singkat beberapa kandidat yang diyakini secara luas memenuhi definisi ini. (Kita akan melihat lebih banyak contoh dugaan fungsi satu arah di Bab 8.) Selanjutnya kita perkenalkan gagasan predikat inti, yang dapat dilihat sebagai merangkum kesulitan dalam membalikkan fungsi satu arah dan akan digunakan secara luas dalam konstruksi yang mengikuti bagian berikutnya.

Definisi

Fungsi satu arah $f : \{0,1\}^* \to \{0,1\}^*$ mudah dihitung, namun sulit dibalik. Kondisi pertama mudah untuk diformalkan: kita hanya memerlukan agar f dapat dihitung dalam waktu polinomial. Karena kami pada akhirnya tertarik untuk membangun skema kriptografi yang sulit ditembus oleh musuh waktu polinomial probabilistik kecuali dengan probabilitas yang dapat diabaikan, kami akan memformalkan kondisi kedua dengan mensyaratkan bahwa algoritma waktu polinomial probabilistik mana pun tidak mungkin untuk membalikkan f — bahwa adalah, untuk mencari gambaran awal dari nilai tertentu y—kecuali dengan probabilitas yang dapat diabaikan. Poin teknisnya adalah probabilitas ini diambil alih dari eksperimen di mana y dihasilkan dengan memilih elemen seragam x dari domain f dan kemudian menetapkan $y := f(x)$ (daripada memilih y secara seragam dari rentang f) . Alasannya harus jelas dari konstruksi yang akan kita lihat di sisa bab ini.

Misalkan $f : \{0,1\}^* \to \{0,1\}^*$ adalah suatu fungsi. Pertimbangkan eksperimen berikut yang ditentukan untuk algoritma apa pun $\mathcal{A}$ dan nilai n apa pun untuk parameter keamanan:

Eksperimen pembalik $\mathrm{Invert}_{\mathcal{A},f}(n)$

1. Pilih seragam $x \in \{0,1\}^n$, dan hitung $y := f(x)$.
2. $\mathcal{A}$ diberikan 1^n dan y sebagai masukan, dan keluaran x'.
3. Keluaran percobaan ditetapkan 1 jika $f(x') = y$, dan 0 jika tidak.

Kami menekankan bahwa tidak perlu menemukan gambar awal x yang asli; cukup untuk mencari nilai x' yang mana $f(x') = y = f(x)$. Kami memberikan parameter keamanan 1^n pada langkah kedua untuk menekankan yang mungkin berjalan dalam polinomial waktu dalam parameter keamanan n, terlepas dari panjang y.

Sekarang kita dapat mendefinisikan arti suatu fungsi f menjadi satu arah.

DEFINISI 7.1 Suatu fungsi $f : \{0,1\}^* \to \{0,1\}^*$ bersifat satu arah jika memenuhi dua kondisi berikut:

1. (Mudah dihitung :) Terdapat algoritma waktu polinomial M_f komputasi f; yaitu, $M_f(x) = f(x)$ untuk semua .
2. (Sulit untuk dibalik :) Untuk setiap algoritma waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\text{Invert}_{\mathcal{A},f}(n) = 1] \leq \text{negl}(n)$$

Notasi. Dalam bab ini kita akan sering membuat ruang probabilitas lebih eksplisit dengan mensubskripkannya (sebagian) ke dalam notasi probabilitas. Misalnya, kita dapat secara ringkas menyatakan persyaratan kedua dalam definisi di atas sebagai berikut: Untuk setiap algoritma waktu polinomial probabilistik, terdapat fungsi yang dapat diabaikan sehingga

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) \in f^{-1}(f(x))] \leq \text{negl}(n).$$

(Ingat bahwa $x \leftarrow \{0,1\}^n$ berarti x dipilih secara seragam dari $\{0,1\}^n$) Probabilitas di atas juga diambil alih keacakan yang digunakan oleh $\mathcal{A}$, yang di sini dibiarkan implisit.

Inversi fungsi satu arah yang berhasil. Suatu fungsi yang tidak bersifat satu arah belum tentu mudah untuk dibalik sepanjang waktu (atau bahkan "sering"). Sebaliknya, kebalikan dari kondisi kedua Definisi 7.1 adalah terdapat algoritma waktu polinomial probabilistik $\mathcal{A}$ dan fungsi yang tidak dapat diabaikan γ sehingga membalikkan $f(x)$ dengan probabilitas paling sedikit $\gamma(n)$ (dimana probabilitas diambil atas pilihan seragam $x \in \{0,1\}^n$ dan keacakan). Hal ini berarti, pada gilirannya, terdapat polinomial positif p( ) sehingga untuk banyak nilai n yang tak terhingga, algoritma akan membalikkan f dengan probabilitas paling sedikit $1/p(n)$. Jadi, jika ada sebuah yang membalikkan f dengan probabilitas n^{-10} untuk semua nilai genap dari n (tetapi selalu gagal untuk membalikkan f ketika n ganjil), maka f bukan satu arah—walaupun hanya berhasil pada setengah nilai dari n, dan hanya berhasil dengan probabilitas n^{-10} (untuk nilai n yang berhasil).

Inversi waktu eksponensial. Fungsi satu arah apa pun dapat diinversi di titik mana saja y dalam waktu eksponensial, cukup dengan mencoba semua nilai $x \in \{0,1\}^n$hingga ditemukan nilai x sehingga $f(x) = y$. Dengan demikian, keberadaan fungsi satu arah pada dasarnya merupakan asumsi tentang kompleksitas komputasi dan kekerasan komputasi. Artinya, menyangkut masalah yang pada prinsipnya dapat diselesaikan tetapi dianggap sulit diselesaikan secara efisien.

Permutasi satu arah. Kita sering kali tertarik pada fungsi satu arah dengan sifat struktural tambahan. Kita mengatakan suatu fungsi f mempertahankan panjang jika $|f(x)| = |x|$ untuk semua x. Fungsi satu arah yang mempertahankan panjang dan fungsi satu-ke-satu disebut permutasi satu arah. Jika f adalah permutasi satu arah, maka nilai apa pun y mempunyai preimage unik $x = F^{-1}(y)$. Meskipun demikian, masih sulit menemukan x dalam waktu polinomial.

Kelompok fungsi/permutasi satu arah. Definisi fungsi satu arah dan permutasi di atas cukup tepat karena mempertimbangkan fungsi tunggal pada domain dan rentang tak terhingga. Namun, sebagian besar kandidat fungsi dan permutasi satu arah tidak cocok dengan kerangka ini. Sebaliknya, ada algoritma yang menghasilkan beberapa set parameter I

yang mendefinisikan suatu fungsi f_I ; kesatu arah di sini pada dasarnya berarti bahwa f_I haruslah satu arah dengan semua probabilitas yang dapat diabaikan atas pilihan I. Karena setiap nilai I mendefinisikan fungsi yang berbeda, kita sekarang mengacu pada kelompok fungsi satu arah (resp., permutasi). Kami memberikan definisinya sekarang, dan mengarahkan pembaca ke bagian berikutnya untuk mendapatkan contoh nyata. (Lihat juga Bagian 8.4)

DEFINISI 7.2 Tupel $\Pi = (\mathrm{Gen}, \mathrm{Samp}, \mathrm{f})$ dari algoritma waktu polinomial probabilistik merupakan suatu kelompok fungsi jika hal berikut berlaku:

1. Algoritma pembangkitan parameter Gen, pada input 1^n, mengeluarkan parameter I dengan $|I| \geq n$. Setiap nilai keluaran I oleh Gen mendefinisikan himpunan D_I dan R_I yang masing-masing membentuk domain dan rentang fungsi f_I.
2. Algoritma pengambilan sampel Samp, pada masukan I, mengeluarkan elemen D_I yang terdistribusi secara merata.
3. Algoritma evaluasi deterministik f, pada masukan I dan $x \in D_I$, mengeluarkan elemen $y \in R_I$. Kami menulis ini sebagai $y := f_I(x)$.

Π adalah keluarga permutasi jika untuk setiap nilai I keluaran $\mathrm{Gen}(1^n)$, dinyatakan bahwa $D_I = R_I$ dan fungsinya $f_I = D_I \to D_I$ adalah sebuah bijection.

Biarkan Π menjadi keluarga fungsi. Berikut ini adalah analogi alami dari eksperimen yang diperkenalkan sebelumnya.

Eksperimen pembalik $\mathrm{Invert}_{\mathcal{A},\Pi}(n)$:

1. $\mathrm{Gen}(1^n)$, dijalankan untuk memperoleh I, kemudian $\mathrm{Samp}(I)$, dijalankan untuk memperoleh $x \in D_I$ yang seragam. Akhirnya, $y := f_I(x)$ dihitung.
2. $\mathcal{A}$ diberikan I dan y sebagai masukan, dan keluaran x'.
3. Hasil percobaan adalah 1 jika $f_I(x') = y$

DEFINISI 7.3 Suatu keluarga fungsi/permutasi Π = (Gen, Samp, ƒ) adalah satu arah jika untuk semua algoritma waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathrm{Invert}_{\mathcal{A},\Pi}(n) = 1] \leq \mathrm{negl}(n).$$

Sepanjang bab ini kita bekerja dengan fungsi/permutasi satu arah pada domain tak terbatas (seperti dalam Definisi 7.1), daripada bekerja dengan keluarga fungsi/permutasi satu arah. Hal ini terutama untuk kenyamanan, dan tidak mempengaruhi hasil apa pun secara signifikan. (Lihat Latihan 7.7.)

Fungsi Satu Arah Kandidat

Fungsi satu arah hanya menarik jika ada. Kita tidak tahu bagaimana membuktikan keberadaan mereka tanpa syarat (ini akan menjadi terobosan besar dalam teori kompleksitas), jadi kita harus menduga atau mengasumsikan keberadaan mereka. Dugaan tersebut didasarkan pada fakta bahwa beberapa masalah komputasi alami telah mendapat banyak perhatian, namun masih belum memiliki algoritma waktu polinomial untuk menyelesaikannya. Mungkin masalah yang paling terkenal adalah faktorisasi bilangan bulat,

yaitu mencari faktor prima dari bilangan bulat besar. Sangat mudah untuk mengalikan dua bilangan dan memperoleh hasil kali, tetapi sulit untuk mengambil suatu bilangan dan mencari faktor-faktornya. Hal ini mengarahkan kita untuk mendefinisikan fungsi $f_{mult}(x, y) = x \cdot y$. Jika kita tidak membatasi panjang x dan y, maka fmult mudah untuk dibalik: kemungkinan besar $x \cdot y$ akan genap, dalam hal ini $(2, xy/2)$ adalah invers. Masalah ini dapat diatasi dengan membatasi domain fmult pada bilangan prima x dan y yang panjangnya sama. Kita kembali ke gagasan ini di Bagian 8.2.

Kandidat fungsi satu arah lainnya, yang tidak bergantung langsung pada teori bilangan, didasarkan pada masalah subset-sum dan didefinisikan oleh

$$f_{ss}(x_1, \ldots, x_n, J) = \left(x_1, \ldots, x_n, \left[\sum_{j \in J} x_j \bmod 2^n\right]\right)$$

di mana masing-masing x_i adalah string n-bit yang diinterpretasikan sebagai bilangan bulat, dan J adalah string n-bit yang diinterpretasikan sebagai subset dari $\{1, \ldots, n\}$. Membalikkan fss pada keluaran $(x_i, \ldots, x_n, y)$ memerlukan pencarian subset $J' \subseteq \{1, \ldots, n\}$ sehingga $\sum_{j \in J'} x_j = y \bmod 2^n$. Siswa yang telah mempelajari $\mathcal{NP}$-tuntas mungkin ingat bahwa soal ini adalah N P-tuntas. Tetapi bahkan $\mathcal{P} \neq \mathcal{NP}$ tidak berarti bahwa f_{ss} adalah satu arah: $\mathcal{P} \neq \mathcal{NP}$ berarti bahwa setiap algoritma waktu polinomial gagal menyelesaikan masalah subset-sum pada setidaknya satu input, sedangkan untuk f_{ss} menjadi algoritma satu- fungsi cara ini mengharuskan setiap algoritme waktu polinomial hampir selalu gagal menyelesaikan masalah jumlah subset (setidaknya untuk parameter tertentu). Jadi, keyakinan kami bahwa fungsi di atas adalah satu arah didasarkan pada kurangnya algoritma yang diketahui untuk menyelesaikan masalah ini bahkan dengan probabilitas "kecil" pada input acak, dan bukan hanya pada fakta bahwa masalahnya adalah $\mathcal{NP}$ -lengkap.

Kita menyimpulkan dengan menunjukkan suatu kelompok permutasi yang diyakini bersifat satu arah. Misalkan Gen adalah algoritma waktu polinomial probabilistik yang, pada masukan 1^n, menghasilkan n-bit bilangan prima p bersama dengan elemen khusus$g \in \{2, \ldots, p-1\}$. (Elemen g harus menjadi generator Z_p^*; lihat Bagian 8.3.) Misalkan $Samp$ adalah algoritma yang, jika diketahui p dan g, menghasilkan bilangan bulat seragam $x \in \{2, \ldots, p-1\}$. Terakhir, tentukan

$$f_{p,g}(x) = [g^x \bmod p].$$

(Fakta bahwa $f_{p,g}$ dapat dihitung secara efisien mengikuti hasil). Dapat ditunjukkan bahwa fungsi ini adalah fungsi satu-ke-satu, dan dengan demikian merupakan permutasi. Dugaan kesulitan dalam membalikkan fungsi ini didasarkan pada perkiraan kekerasan masalah logaritma diskrit; kami akan membahas lebih banyak lagi mengenai hal ini di Bagian 8.3.

Terakhir, kami menyatakan bahwa fungsi satu arah yang sangat efisien dapat diperoleh dari konstruksi kriptografi praktis seperti SHA-1 atau AES dengan asumsi masing-masing tahan terhadap benturan atau permutasi pseudorandom; lihat Latihan 7.4 dan 7.5. (Secara teknis, mereka tidak dapat memenuhi definisi satu-arah karena mereka memiliki input/output yang panjangnya tetap sehingga kita tidak dapat melihat perilaku asimtotiknya. Namun demikian,

masuk akal untuk menduga bahwa mereka adalah satu-arah dalam hal ini. pengertian yang konkrit.)

Predikat Inti Keras

Menurut definisinya, fungsi satu arah sulit untuk dibalik. Dinyatakan secara berbeda: mengingat $y = f(x)$, nilai x tidak dapat dihitung secara keseluruhan dengan algoritma waktu polinomial mana pun (kecuali dengan probabilitas yang dapat diabaikan; kita mengabaikannya di sini). Orang mungkin mendapat kesan bahwa x tidak dapat ditentukan dari $f(x)$ dalam waktu polinomial. Hal ini belum tentu terjadi. Memang benar bahwa $f(x)$ dapat "membocorkan" banyak informasi tentang x meskipun f bersifat satu arah. Sebagai contoh sederhana, misalkan g adalah fungsi satu arah dan tentukan $f(x_1, x_2) \stackrel{\text{def}}{=} \big(x_1(g, x_2)\big)$, dengan $|x_1| = |x_2|$. Sangat mudah untuk menunjukkan bahwa f juga merupakan fungsi satu arah (dibiarkan sebagai latihan), meskipun ia mengungkapkan separuh masukannya.

Untuk aplikasi kita, kita perlu mengidentifikasi informasi spesifik tentang x yang "disembunyikan" oleh $f(x)$. Hal ini memotivasi gagasan predikat hard-core. Predikat inti $hc : \{0,1\}^* \to \{0,1\}$ dari suatu fungsi f memiliki sifat yang $hc(x)$ sulit dihitung dengan probabilitas yang jauh lebih baik daripada $1/2$ yang diberikan $f(x)$. (Karena hc adalah fungsi boolean, $hc(x)$ selalu dapat dihitung dengan probabilitas $1/2$ dengan menebak secara acak.) Secara formal:

DEFINISI 7.4 Suatu fungsi $hc : \{0,1\}^* \to \{0,1\}$ adalah predikat inti dari suatu fungsi f jika hc dapat dihitung dalam waktu polinomial, dan untuk algoritma waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga

$$\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}(1^n, f(x)) = \mathrm{hc}(x)] \leq \frac{1}{2} + \mathrm{negl}(n),$$

dimana probabilitasnya diambil alih pilihan seragam x di $\{0,1\}^n$ dan keacakan $\mathcal{A}$.

Kami menekankan bahwa $hc(x)$ dapat dihitung secara efisien jika diberikan x (karena fungsi hc dapat dihitung dalam waktu polinomial); definisi tersebut mensyaratkan bahwa $hc(x)$ sulit dihitung mengingat $f(x)$. Definisi di atas tidak mengharuskan f bersifat satu arah; namun jika f merupakan permutasi, maka predikat tersebut tidak dapat memiliki predikat inti kecuali predikat satu arah. (Lihat Latihan 7.13.)

Ide sederhana tidak akan berhasil. Perhatikan predikatnya $hc(x) \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i$ di mana $x_1, \ldots, x_n$ menunjukkan bit dari x. Kita mungkin berharap bahwa ini adalah predikat inti dari fungsi satu arah apa pun f: jika f tidak dapat dibalik, maka $f(x)$ harus menyembunyikan setidaknya satu bit x_i dari gambar awal x, yang sepertinya menyiratkan hal itu eksklusif-atau dari semua bit x sulit dihitung. Meskipun ada banding, argumen ini tidak benar. Untuk melihatnya, misalkan g adalah fungsi satu arah dan definisikan $f(x) \stackrel{\text{def}}{=} (g(x), \bigoplus_{i=1}^{n} x_i)$. Tidak sulit untuk menunjukkan bahwa f bersifat satu arah. Namun jelas bahwa $f(x)$ tidak menyembunyikan nilai $hc(x) = \bigoplus_{i=1}^{n} x_i$karena ini adalah bagian dari keluarannya; oleh karena itu, $hc(x)$ bukan predikat inti dari f. Dengan memperluas hal ini, kita dapat

menunjukkan bahwa untuk setiap predikat tetap hc, terdapat fungsi satu arah f yang mana hc bukan merupakan predikat inti dari f.

Predikat hard-core yang sepele. Beberapa fungsi memiliki predikat hard-core yang "sepele". Misalnya, f adalah fungsi yang menghilangkan bit terakhir inputnya (yaitu, $f(x_1,\ldots,x_n) = x_1,\ldots,x_{n-1}$. Sulit untuk menentukan xn mengingat $f(x)$ karena x_n tidak bergantung pada keluaran; dengan demikian, $hc(x) = x_n$ adalah predikat inti dari f. Namun, f tidak bersifat satu arah. Ketika kita menggunakan predikat hard-core untuk konstruksi kita, akan menjadi jelas mengapa predikat hard-core yang sepele seperti ini tidak ada gunanya.

7.2 DARI FUNGSI SATU ARAH KE KEACAKAN SEMU

Tujuan dari bab ini adalah untuk menunjukkan bagaimana membangun generator, fungsi, dan permutasi pseudorandom berdasarkan fungsi/permutasi satu arah. Pada bagian ini, kami memberikan gambaran umum tentang konstruksi tersebut. Rinciannya diberikan pada bagian berikutnya.

Predikat inti dari fungsi satu arah mana pun. Langkah pertama adalah menunjukkan bahwa predikat hard-core ada untuk fungsi satu arah. Sebenarnya, masih belum diketahui apakah hal ini benar; kami menunjukkan sesuatu yang lebih lemah yang cukup untuk tujuan kami. Yaitu, kami menunjukkan bahwa dengan fungsi satu arah f kita dapat membuat fungsi satu arah yang berbeda g bersama dengan predikat inti g.

TEOREMA 7.5 (Teorema Goldreich–Levin) Asumsikan ada fungsi satu arah (resp., permutasi). Lalu terdapat fungsi satu arah (resp., permutasi) g dan predikat inti hc dari g.

Misalkan f merupakan fungsi satu arah. Fungsi g dan hc dibuat sebagai berikut: himpunan $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$ untuk $|x| = |r|$, dan tentukan

$$\mathrm{hc}(x,r) \stackrel{\text{def}}{=} \bigoplus_{i=1}^{n} x_i \cdot r_i,$$

dimana xi (resp., ri) menunjukkan bit ke-i dari x (resp., r). Perhatikan bahwa jika r seragam, maka $hc(x,r)$ mengeluarkan subset acak-eksklusif dari bit-bit x. (Ketika $r_i = 1$ bit x_i dimasukkan dalam XOR, dan sebaliknya tidak.) Teorema Goldreich–Levin pada dasarnya menyatakan bahwa jika f adalah fungsi satu arah maka $f(x)$ menyembunyikan fungsi eksklusif atau acak bagian dari bit x.

Generator acak semu dari permutasi satu arah. Langkah selanjutnya adalah menunjukkan bagaimana predikat hard-core dari permutasi satu arah dapat digunakan untuk membuat generator pseudorandom. (Diketahui bahwa predikat inti dari fungsi satu arah sudah cukup, namun pembuktiannya sangat rumit dan jauh di luar cakupan buku ini.) Secara khusus, kami menunjukkan:

TEOREMA 7.6 Misalkan ƒ adalah permutasi satu arah dan hc adalah predikat inti dari ƒ. Maka, $G(s) \stackrel{\text{def}}{=} (f(s)\|\text{hc}(s))$ adalah generator pseudorandom dengan faktor ekspansi $\ell(n) = n + 1$.

Sebagai intuisi mengapa G sebagaimana didefinisikan dalam teorema merupakan generator pseudorandom, perhatikan terlebih dahulu bahwa n bit awal dari output $G(s)$ (yaitu, bit dari $f(s)$) benar-benar terdistribusi secara seragam ketika s adalah terdistribusi secara merata, karena f merupakan permutasi. Selanjutnya, fakta bahwa hc adalah predikat inti dari f berarti bahwa $hc(s)$ "terlihat acak"—yaitu, adalah pseudo-acak—bahkan jika diberikan $f(s)$ (dengan asumsi sekali lagi bahwa s seragam). Dengan menggabungkan pengamatan ini, kita melihat bahwa seluruh keluaran G adalah pseudorandom.

Generator acak semu dengan ekspansi sewenang-wenang. Keberadaan generator pseudorandom yang merenggangkan benihnya bahkan hanya satu bit saja (seperti yang baru saja kita lihat) sudah sangat tidak sepele. Namun untuk aplikasi (misalnya, untuk enkripsi pesan besar yang efisien seperti pada Bagian 3.3), kita memerlukan generator pseudorandom dengan ekspansi yang jauh lebih besar. Untungnya, kita bisa mendapatkan faktor ekspansi polinomial apa pun yang kita inginkan:

TEOREMA 7.7 Jika terdapat generator pseudorandom dengan faktor ekspansi $\ell(n) = n + 1$ maka untuk polinomial mana pun terdapat generator pseudorandom dengan faktor ekspansi $\text{poli}(n)$.

Kami menyimpulkan bahwa generator pseudorandom dengan ekspansi sewenang-wenang (polinomial) dapat dibangun dari permutasi satu arah.

Fungsi/permutasi pseudorandom dari generator pseudorandom. Generator pseudorandom cukup untuk membangun skema enkripsi kunci pribadi yang aman dengan EAV. Namun, untuk mencapai enkripsi kunci pribadi yang aman terhadap CPA (belum lagi kode autentikasi pesan), kami mengandalkan fungsi acak semu. Hasil berikut menunjukkan bahwa yang terakhir dapat dibangun dari yang pertama:

TEOREMA 7.8 Jika terdapat generator pseudorandom dengan faktor ekspansi $\ell(n) = 2n$, maka terdapat fungsi pseudorandom.

Faktanya, kita dapat melakukan lebih banyak lagi:

TEOREMA 7.9 Jika terdapat fungsi pseudorandom, maka terdapat permutasi pseudorandom kuat.

Menggabungkan semua teorema di atas, serta hasil dari Bab 3 dan 4, kita mendapatkan akibat wajar berikut:

KOROLLARIS 7.10 Dengan asumsi adanya permutasi satu arah, terdapat generator pseudorandom dengan faktor ekspansi polinomial, fungsi pseudorandom, dan permutasi pseudorandom kuat.

KOROLLARIS 7.11 Dengan asumsi adanya permutasi satu arah, terdapat skema enkripsi kunci privat yang aman dengan CCA dan kode autentikasi pesan yang aman.

Seperti disebutkan sebelumnya, semua hasil ini dapat diperoleh hanya berdasarkan keberadaan fungsi satu arah.

7.3 PREDIKAT INTI KERAS DARI FUNGSI SATU ARAH

Pada bagian ini, kita buktikan Teorema 7.5 dengan menunjukkan hal berikut:

TEOREMA 7.12 Misalkan ƒ adalah fungsi satu arah dan definisikan g dengan $g(x,r) \stackrel{\text{def}}{=} (f(x),r)$, dimana $|x| = |r|$. Mendefinisikan $\text{gl}(x,r) \stackrel{\text{def}}{=} (\bigoplus_{i=1}^{n} x_i \cdot r_i)$. Dimana $x = x_1,\ldots,x_n$ dan $r = r_1,\ldots,r_n$. Maka gl merupakan predikat inti dari g.

Karena rumitnya pembuktian, kami membuktikan tiga hasil yang lebih kuat secara berturut-turut yang berpuncak pada apa yang diklaim dalam teorema.

Kasus Sederhana

Pertama-tama kita tunjukkan bahwa jika terdapat musuh dengan waktu polinomial yang selalu menghitung $\text{gl}(x,r)$ dengan benar jika diketahui $g(x,r) = (\text{ƒ}(x),r)$, maka ƒ dalam polinomial dapat dibalik waktu. Dengan asumsi bahwa ƒ adalah fungsi satu arah, maka tidak ada lawan $\mathcal{A}$ yang seperti itu.

PROPOSISI 7.13 Misalkan ƒ dan gl seperti pada Teorema 7.12. Jika terdapat algoritma waktu polinomial $\mathcal{A}$ sehingga $\mathcal{A}(f(x),r) = \text{gl}(x,r)$ untuk semua n dan semua $x,r \in \{0,1\}^n$, maka terdapat algoritma waktu polinomial algoritma $\mathcal{A}'$ sehingga $\mathcal{A}'(1^n f(x)) = x$ untuk semua n dan semua $x \in \{0,1\}^n$.

BUKTI Kita buat $\mathcal{A}'$ sebagai berikut. $\mathcal{A}'(1^n, y)$menghitung $x_i := \mathcal{A}(y, e^i)$ untuk $i = 1,\ldots,n$, di mana e^i menunjukkan string n-bit dengan 1 di posisi ke-i dan 0 di tempat lain. Maka $\mathcal{A}'$ keluaran $x = x_1,\ldots,x_n$. Jelas $\mathcal{A}'$ berjalan dalam waktu polinomial.

Dalam pelaksanaan $\mathcal{A}'(1^n, f(\hat{x}))$, nilai x_1 yang dihitung oleh $\mathcal{A}'$ memenuhi

$$x_i = \mathcal{A}(f(\hat{x}), e^i) = \text{gl}(\hat{x}, e^i) = \bigoplus_{j=1}^{n} \hat{x}_j \cdot e^i_j = \hat{x}_i.$$

Jadi, $x_i = \hat{x_i}$ untuk semua i sehingga $\mathcal{A}'$ menghasilkan invers yang benar $x = \hat{x}$.

Jika ƒ satu arah, mustahil bagi algoritma waktu polinomial probabilistik mana pun untuk membalikkan ƒ dengan probabilitas yang tidak dapat diabaikan. Jadi, kami menyimpulkan bahwa tidak ada algoritme waktu polinomial yang selalu menghitung $\mathrm{gl}(x,r)$ dari $(f(x),r)$ dengan benar. Ini adalah hasil yang agak lemah dan sangat jauh dari tujuan akhir kami untuk menunjukkan bahwa $gl(x,r)$ tidak dapat dihitung (dengan probabilitas yang jauh lebih baik daripada 1/2) jika diberikan $(f(x),r)$.

Kasus yang Lebih Terlibat

Kami sekarang menunjukkan bahwa sulit bagi algoritma waktu polinomial probabilistik $\mathcal{A}$ untuk menghitung $\mathrm{gl}(x,r)$ dari $(f(x),r)$ dengan probabilitas yang jauh lebih baik daripada 3/4. Kami akan kembali menunjukkan bahwa $\mathcal{A}$ menyiratkan adanya algoritma waktu polinomial $\mathcal{A}'$ yang membalikkan ƒ dengan probabilitas yang tidak dapat diabaikan. Perhatikan bahwa strategi dalam pembuktian Proposisi 7.13 gagal di sini karena mungkin strategi tersebut $\mathcal{A}$ tidak pernah berhasil ketika $r = e^i$ (walaupun mungkin berhasil, katakanlah, pada semua nilai r lainnya). Selanjutnya, dalam kasus ini $\mathcal{A}'$ tidak mengetahui apakah hasil $\mathcal{A}(f(x),r)$ sama dengan $\mathrm{gl}(x,r)$ atau tidak; satu-satunya hal yang diketahui $\mathcal{A}'$adalah bahwa dengan probabilitas tinggi, algoritmanya $\mathcal{A}$ benar. Hal ini semakin memperumit pembuktian.

PROPOSISI 7.14 Misalkan ƒ dan gl seperti pada Teorema 7.12. Jika terdapat algoritma waktu polinomial probabilistik $\mathcal{A}$ dan polinomial p(·) sedemikian rupa

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\mathrm{gl}(x,r)\right]\geq\frac{3}{4}+\frac{1}{p(n)}$$

untuk nilai n yang tak terhingga banyaknya, maka terdapat algoritma waktu polinomial probabilistik $\mathcal{A}'$ sedemikian rupa sehingga

$$\Pr_{x\leftarrow\{0,1\}^n}\left[\mathcal{A}'(1^n,f(x))\in f^{-1}(f(x))\right]\geq\frac{1}{4\cdot p(n)}$$

untuk banyak nilai n yang tak terhingga.

BUKTI Pengamatan utama yang mendasari pembuktian proposisi ini adalah bahwa untuk setiap $r\in\{0,1\}^n$, nilai $\mathrm{gl}(x,r\oplus e^i)$ dan $gl(x,r)$ bersama-sama dapat digunakan untuk menghitung bit ke-i dari x. (Ingatlah bahwa e^i menunjukkan string n-bit dengan 0 di mana pun kecuali posisi ke-i.) Hal ini benar karena

$$\begin{aligned}&\mathrm{gl}(x,r)\oplus\mathrm{gl}(x,r\oplus e^i)\\&=\left(\bigoplus_{j=1}^{n}x_j\cdot r_j\right)\oplus\left(\bigoplus_{j=1}^{n}x_j\cdot(r_j\oplus e^i_j)\right)=x_i\cdot r_i\oplus\left(x_i\cdot\bar{r}_i\right)=x_i\,,\end{aligned}$$

dimana $\hat{r}_i$ adalah komplemen dari $\hat{r}_i$, dan persamaan kedua disebabkan oleh fakta bahwa untuk $j\neq i$, nilai $x_1\cdot x_n$ muncul pada kedua penjumlahan sehingga dihilangkan. Perlihatkan di atas bahwa jika $\mathcal{A}$ menjawab dengan benar pada kedua $(f(x),r)$ dan $f((x),r\oplus e^i)$, maka $\mathcal{A}'$ dapat menghitung x_i dengan benar. Sayangnya, $\mathcal{A}'$ tidak mengetahui kapan $\mathcal{A}$ menjawab dengan benar dan kapan tidak; $\mathcal{A}'$ hanya mengetahui bahwa $\mathcal{A}$ menjawab dengan benar

dengan probabilitas "tinggi". Karena alasan ini, $\mathcal{A}'$ akan menggunakan beberapa nilai acak dari r, menggunakan masing-masing nilai tersebut untuk mendapatkan estimasi x_i, dan kemudian akan mengambil estimasi yang sering terjadi sebagai tebakan akhir untuk x_i.

Sebagai langkah awal, kami menunjukkan bahwa untuk banyak x, probabilitas menjawab benar untuk $(f(x), r)$ dan $(f(x), r \oplus e^i)$, ketika r seragam, cukup tinggi. Hal ini memungkinkan kita untuk memperbaiki x dan kemudian fokus hanya pada pilihan r yang seragam, sehingga membuat analisis lebih mudah.

KLAIM 7.15 Misalkan n sedemikian rupa

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right]\geq\frac{3}{4}+\frac{1}{p(n)}.$$

Maka terdapat himpunan $S_n \subseteq \{0,1\}^n$ dengan ukuran paling sedikit $\frac{1}{2p(n)} \cdot 2^n$ sehingga untuk setiap $x \in S_n$ berlaku bahwa

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r)=\text{gl}(x,r)]\geq\frac{3}{4}+\frac{1}{2p(n)}.$$

BUKTI Misalkan $\varepsilon(n) = 1/p(n)$, dan tentukan $S_n \subseteq \{0,1\}^n$ sebagai himpunan semua x yang

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r)=\text{gl}(x,r)]\geq\frac{3}{4}+\frac{\varepsilon(n)}{2}.$$

Kita punya

$$\begin{aligned}
\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right] &= \frac{1}{2^n}\sum_{x\in\{0,1\}^n}\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right] \\
&= \frac{1}{2^n}\sum_{x\in S_n}\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right] \\
&\quad + \frac{1}{2^n}\sum_{x\notin S_n}\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right] \\
&\leq \frac{|S_n|}{2^n}+\frac{1}{2^n}\cdot\sum_{x\notin S_n}\left(\frac{3}{4}+\frac{\varepsilon(n)}{2}\right) \\
&\leq \frac{|S_n|}{2^n}+\left(\frac{3}{4}+\frac{\varepsilon(n)}{2}\right).
\end{aligned}$$

Sejak

$$\tfrac{3}{4}+\varepsilon(n)\leq\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right]$$

aljabar lugas memberi

$$|S_n|\geq\tfrac{\varepsilon(n)}{2}\cdot 2^n.$$

Berikut ini adalah konsekuensi mudahnya.

KLAIM 7.16 Misalkan n sedemikian rupa sehingga

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r)=\text{gl}(x,r)\right]\geq\frac{3}{4}+\frac{1}{p(n)}.$$

Maka terdapat himpunan $S_n \subseteq \{0,1\}^n$ dengan ukuran minimal $\frac{1}{2p(n)} \cdot 2^n$ sehingga untuk setiap $x \in S_n$ dan setiap i berlaku bahwa

$$\Pr_{r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r) \bigwedge \mathcal{A}(f(x), r \oplus e^i) = \mathrm{gl}(x, r \oplus e^i)\right] \geq \frac{1}{2} + \frac{1}{p(n)}.$$

BUKTI Misalkan $\varepsilon(n) = 1/p(n)$, dan ambil S_n sebagai himpunan yang dijamin oleh klaim sebelumnya. Kita tahu bahwa untuk setiap $x \in Sn$ yang kita miliki

$$\Pr_{r \leftarrow \{0,1\}^n}[\mathcal{A}(f(x),r) \neq \mathrm{gl}(x,r)] \leq \frac{1}{4} - \frac{\varepsilon(n)}{2}.$$

Perbaiki $i \in \{1,\ldots,n\}$. Jika r terdistribusi secara merata maka $r \oplus e^i$ juga demikian; dengan demikian

$$\Pr_{r \leftarrow \{0,1\}^n}[\mathcal{A}(f(x), r \oplus e^i) \neq \mathrm{gl}(x, r \oplus e^i)] \leq \frac{1}{4} - \frac{\varepsilon(n)}{2}.$$

Kami tertarik pada probabilitas batas bawah yang menghasilkan jawaban yang benar untuk $\mathrm{gl}(x,r)$ dan $\mathrm{gl}(x, r \oplus e^i)$; sama halnya, kami ingin meningkatkan probabilitas yang gagal menghasilkan jawaban yang benar dalam salah satu kasus ini. Perhatikan bahwa r dan $r \oplus e^i$ tidak independen, jadi kita tidak bisa hanya mengalikan probabilitas kegagalan. Namun, kita dapat menerapkan ikatan gabungan (lihat Proposisi A.7) dan menjumlahkan probabilitas kegagalan. Artinya, peluang $\mathcal{A}$ salah pada $\mathrm{gl}(x,r)$ dan $\mathrm{gl}(x, r \oplus e^i)$; paling banyak adalah

$$\left(\frac{1}{4} - \frac{\varepsilon(n)}{2}\right) + \left(\frac{1}{4} - \frac{\varepsilon(n)}{2}\right) = \frac{1}{2} - \varepsilon(n),$$

dan benar pada $\mathrm{gl}(x,r)$ dan $\mathrm{gl}(x, r \oplus e^i)$; dengan probabilitas paling sedikit $1/2 + \varepsilon(n)$. Ini membuktikan klaim tersebut.

Untuk pembuktian selebihnya kita menetapkan $\varepsilon(n) = 1/p(n)$ dan hanya mempertimbangkan nilai n yang mana

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)\right] \geq \frac{3}{4} + \varepsilon(n). \qquad (7.1)$$

Klaim sebelumnya menyatakan bahwa untuk pecahan ε(n)/2 dari input x, dan i apa pun, algoritma menjawab dengan benar pada kedua $f(x,r)$ dan $f(x, r \oplus e^i)$; dengan probabilitas sebesar paling sedikit $1/2 + \varepsilon(n)$ pada pilihan r yang seragam, dan mulai sekarang kita hanya fokus pada nilai x tersebut. Kami membangun algoritma waktu polinomial probabilistik ' yang membalikkan $f(x)$ dengan probabilitas setidaknya 1/2 ketika $x \in S_n$. Ini cukup untuk membuktikan Proposisi 7.14 sejak saat itu, untuk banyak nilai n yang tak terhingga,

$$\begin{aligned}
&\Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}'(1^n, f(x)) \in f^{-1}(f(x))] \\
&\quad \geq \Pr_{x \leftarrow \{0,1\}^n}[\mathcal{A}'(1^n, f(x)) \in f^{-1}(f(x)) \mid x \in S_n] \cdot \Pr_{x \leftarrow \{0,1\}^n}[x \in S_n] \\
&\quad \geq \frac{1}{2} \cdot \frac{\varepsilon(n)}{2} = \frac{1}{4p(n)}.
\end{aligned}$$

Algoritma A', diberikan sebagai masukan 1^n dan y, bekerja sebagai berikut:

1. Untuk $i = 1,\ldots,n$ dan lakukan:

- Berulang kali pilih seragam $r \in \{0,1\}^n$ dan hitung $A(y,r) \oplus A(y,r \oplus ei)$ sebagai "perkiraan" untuk bit ke-i dari gambar awal y. Setelah melakukan hal ini berkali-kali (lihat di bawah), misalkan x_i adalah "perkiraan" yang sering muncul.

2. Keluaran $x = x_1 \cdots x_n$.

Kami membuat sketsa analisis probabilitas bahwa $\mathcal{A}'$ dengan benar membalikkan input y yang diberikan. (Kita biarkan diri kita menjadi sedikit ringkas, karena bukti lengkap untuk kasus yang lebih sulit diberikan di bagian berikut.) Katakanlah $y = f(\hat{x})$ dan ingat bahwa kita asumsikan di sini bahwa n sedemikian rupa sehingga Persamaan (7.1) berlaku dan $\hat{x} \in S_n$. Perbaiki beberapa i. Klaim sebelumnya menyiratkan bahwa estimasi $\mathcal{A}(y,r) \oplus \mathcal{A}(y, r \oplus e^i)$ sama dengan $\text{gl}(\hat{x}, e^i)$ dengan probabilitas paling sedikit $\frac{1}{2} + \varepsilon(n)$ atas pilihan r. Dengan memperoleh cukup banyak estimasi dan membiarkan xi menjadi nilai mayoritas, $\mathcal{A}'$ dapat memastikan bahwa x_i sama dengan $\text{gl}(\hat{x}, e^i)$ dengan probabilitas minimal $1 - \frac{1}{2n}$. Tentu saja, kita perlu memastikan bahwa banyak perkiraan polinomial sudah cukup. Untungnya, karena $\varepsilon(n) = 1/p(n)$ untuk beberapa polinomial p dan nilai independen r digunakan untuk mendapatkan setiap estimasi, ikatan Chernoff (lih. Proposisi A.14) menunjukkan bahwa banyak estimasi polinomial sudah mencukupi.

Ringkasnya, kita mendapatkan bahwa untuk setiap i nilai xi yang dihitung oleh $\mathcal{A}'$ salah dengan probabilitas paling banyak $\frac{1}{2n}$. Ikatan gabungan dengan demikian menunjukkan bahwa $\mathcal{A}'$salah untuk beberapa i dengan probabilitas paling banyak $n \cdot \frac{1}{2n} = \frac{1}{2}$. Artinya, $\mathcal{A}'$ benar untuk semua i—dan dengan demikian membalikkan y dengan benar—dengan probabilitas paling sedikit $1 - \frac{1}{2} = \frac{1}{2}$. Ini melengkapi pembuktian Proposisi 7.14.

Akibat wajar dari Proposisi 7.14 adalah jika f adalah fungsi satu arah, maka untuk algoritma waktu polinomial apa pun, probabilitas untuk menebak dengan benar $\text{gl}(x,r)$ ketika diberikan $(f(x),r)$ paling banyak dapat diabaikan lebih dari 3/4.

Bukti Lengkap

Kami berasumsi bahwa kami sudah familiar dengan bukti-bukti yang disederhanakan di bagian sebelumnya, dan mengembangkan ide-ide yang dikembangkan di sana.
Kita buktikan proposisi berikut yang mengimplikasikan Teorema 7.12:

PROPOSISI 7.17 Misalkan ƒ dan gl seperti pada Teorema 7.12. Jika terdapat algoritma waktu polinomial probabilistik $\mathcal{A}$ dan polinomial $p(\cdot)$ sedemikian rupa

$$\Pr_{x,r \leftarrow \{0,1\}^n}\left[\mathcal{A}(f(x),r) = \text{gl}(x,r)\right] \geq \frac{1}{2} + \frac{1}{p(n)}$$

untuk nilai n yang tak terhingga banyaknya, maka terdapat algoritma waktu polinomial probabilistik $\mathcal{A}'$ dan polinomial $p'(\cdot)$ sedemikian rupa sehingga

$$\Pr_{x \leftarrow \{0,1\}^n}\left[\mathcal{A}'(1^n, f(x)) \in f^{-1}(f(x))\right] \geq \frac{1}{p'(n)}$$

untuk banyak nilai n yang tak terhingga.

BUKTI Sekali lagi kita menetapkan $\varepsilon(n) = 1/p(n)$ dan hanya mempertimbangkan nilai n yang mana

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)\right] \geq \tfrac{1}{2} + \tfrac{1}{p(n)}.$$

Berikut ini analog dengan Klaim 7.15 dan dibuktikan dengan cara yang sama.

KLAIM 7.18 Misalkan n sedemikian rupa sehingga

$$\Pr_{x,r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)\right] \geq \frac{1}{2} + \varepsilon(n).$$

Maka terdapat himpunan $S_n \subseteq \{0,1\}^n$ dengan ukuran paling sedikit $\frac{\varepsilon(n)}{2} \cdot 2^n$ sehingga untuk setiap $x \in S_n$ berlaku bahwa

$$\Pr_{r\leftarrow\{0,1\}^n}[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r)] \geq \frac{1}{2} + \frac{\varepsilon(n)}{2}. \qquad (7.2)$$

Jika kita mulai dengan mencoba membuktikan analogi Klaim 7.16, hal terbaik yang bisa kita klaim di sini adalah ketika $x \in S_n$ kita punya

$$\Pr_{r\leftarrow\{0,1\}^n}\left[\mathcal{A}(f(x),r) = \mathrm{gl}(x,r) \bigwedge \mathcal{A}(f(x),r\oplus e^i) = \mathrm{gl}(x,r\oplus e^i)\right] \geq \varepsilon(n)$$

untuk setiap saya. Jadi, jika kita mencoba menggunakan $\mathcal{A}(f(x),r) \oplus \mathcal{A}(f(x),r\oplus e^i)$ sebagai pendugaan untuk x_i, yang dapat kita klaim hanyalah bahwa pendugaan ini benar dengan probabilitas paling sedikit $\varepsilon(n)$, yang mungkin tidak lebih baik daripada menebak secara acak! Kami juga tidak dapat mengklaim bahwa membalik hasil akan memberikan perkiraan yang baik.

Sebagai gantinya, kita merancang $\mathcal{A}'$ sehingga menghitung $\mathrm{gl}(x,r)$ dan $\mathrm{gl}(x,r\oplus e^i)$; dengan memanggil $\mathcal{A}$ hanya sekali. Kita melakukan $\mathcal{A}'$ dengan menjalankan $\mathcal{A}(x,r\oplus e^i)$ dan $\mathcal{A}'$ hanya "menebak" nilai $\mathrm{gl}(x,r)$ itu sendiri. Cara naif untuk melakukan hal ini adalah dengan memilih r secara independen, seperti sebelumnya, dan meminta $\mathcal{A}'$ membuat tebakan independen terhadap $\mathrm{gl}(x,r)$ untuk setiap nilai r. Namun probabilitas bahwa semua tebakan tersebut benar—yang, seperti akan kita lihat, diperlukan jika $\mathcal{A}'$ menghasilkan invers yang benar—akan diabaikan karena banyak r yang digunakan secara polinomial.

Pengamatan penting dari bukti ini adalah bahwa $\mathcal{A}'$ dapat menghasilkan r secara berpasangan-independen dan membuat tebakannya dengan cara tertentu sehingga dengan probabilitas yang tidak dapat diabaikan smua tebakannya benar. Secara khusus, untuk menghasilkan nilai m dari r, kita memiliki $\mathcal{A}'$ pilih $\ell = [log(m+1)]$ string independen dan terdistribusi seragam $s^1,\ldots,s^\ell \in \{0,1\}^n$. Kemudian, untuk setiap himpunan bagian tak kosong $I \subseteq \{1,\ldots,\ell\}$, kita tetapkan $r^I := \bigoplus_{i\in I} s^i$. Karena ada $2^\ell - 1$ himpunan bagian yang tidak kosong, ini mendefinisikan kumpulan string $2^{[\log(m+1)]} - 1 \geq m$. Setiap string tersebut didistribusikan secara merata. Senarnya tidak independen, namun independen berpasangan. Untuk melihat hal ini, perhatikan bahwa untuk setiap dua himpunan bagian $I \neq J$ terdapat indeks $j \in I \cup J$ sehingga $j \notin I \cap J$. Tanpa kehilangan keumumannya, asumsikan $j \notin I$. Maka nilai s^J seragam dan tidak bergantung pada nilai r^I . Karena s^J termasuk dalam XOR yang mendefinisikan r^J , hal ini menyiratkan bahwa r^J juga seragam dan tidak bergantung pada r^I.

Kami sekarang memiliki dua pengamatan penting berikut:

1. Diberikan $\text{gl}(x, s^1), \ldots, \text{gl}(x, s^\ell)$, dimungkinkan untuk menghitung $gl(x, r^I)$ untuk setiap subset $I \subseteq \{1, \ldots, \ell\}$. hal ini dikarenakan

$$\text{gl}(x, r^I) = \text{gl}(x, \oplus_{i \in} Is^i) = \oplus_{i \in} I \text{ gl}(x, s^i)$$

2. Jika $\mathcal{A}'$ cukup menebak nilai $\text{gl}(x, s^1), \ldots, \text{gl}\left(x, s^\ell\right)$ dengan memilih bit yang seragam untuk masing-masing bit, maka semua tebakan ini akan benar dengan probabilitas $1/2^\ell$. Jika m adalah polinomial dalam parameter keamanan n, maka $1/2^\ell$ tidak dapat diabaikan, sehingga dengan probabilitas yang tidak dapat diabaikan $\mathcal{A}'$ menebak dengan benar semua nilai $\text{gl}(x, s^1), \ldots, \text{gl}(x, s^\ell)$.

Menggabungkan hasil di atas akan menghasilkan $m = \text{poly}(n)$ string yang seragam dan tidak bergantung berpasangan $\{r^I\}$ serta nilai yang benar untuk $\{\text{gl}(x, r^I)\}$ dengan probabilitas yang tidak dapat diabaikan. Nilai-nilai ini kemudian dapat digunakan untuk menghitung x_i dengan cara yang sama seperti pada pembuktian Proposisi 7.14. Detailnya menyusul.

Algoritma inversi $\mathcal{A}'$. Kami sekarang memberikan penjelasan lengkap tentang algoritma $\mathcal{A}'$ yang menerima input 1^n, y dan mencoba menghitung invers dari y. Algoritmenya berlangsung sebagai berikut:

1. Tetapkan $\ell := \lceil \log(2n/\mathcal{E}(n)^2 + 1) \rceil$.
2. Pilih yang seragam, mandiri $s^1, \ldots, s^\ell \in \{0,1\}^n$ dan $\sigma^1, \ldots, \sigma^\ell \in \{0,1\}$.
3. Untuk setiap himpunan bagian tak kosong $I \subseteq \{1, \ldots, \ell\}$, hitung $r^I := \oplus i \in Is^i$ dan $\sigma^I := \oplus i \in I\sigma^i$.
4. Untuk i = 1, . . . , n lakukan:
   (a) Untuk setiap himpunan bagian tak kosong $\text{I} \subseteq \{1, \ldots, \ell\}$, atur

$$x_i^I := \sigma^I \oplus \mathcal{A}(y, r^I \oplus e^i).$$

   (b) Tetapkan $x_i :=$ mayoritas $I\{x_i^I\}$ (yaitu, ambil bit yang sering muncul pada langkah sebelumnya).
5. Keluaran $x = x_1 \ldots x_n$.

Tinggal menghitung probabilitas bahwa $\mathcal{A}'$ menghasilkan $x \in f^{-1}(y)$. Seperti pada pembuktian Proposisi 7.14, kita hanya fokus pada n seperti pada Klaim 7.18 dan mengasumsikan $y = f(\hat{x})$ untuk beberapa $\hat{x} \in S_n$. Setiap σ^i mewakili "tebakan" untuk nilai $\text{gl}(\hat{x}, s^i)$ Seperti disebutkan sebelumnya, dengan probabilitas yang tidak dapat diabaikan, semua tebakan ini benar; kami tunjukkan bahwa dikondisikan pada kejadian ini, $\mathcal{A}'$ keluaran $x = \hat{x}$ dengan probabilitas paling sedikit $1/2$.

Asumsikan $\sigma^i = \text{gl}(\hat{x}, s^i)$ untuk semua i. Maka $\sigma^I = \text{gl}(\hat{x}, r^I)$ untuk semua I. Tentukan indeks $i \in \{1, \ldots, n\}$ dan pertimbangkan probabilitas bahwa $\mathcal{A}'$ memperoleh nilai yang benar $x_i = \hat{x}_i$. Untuk I yang tidak kosong kita mempunyai $\mathcal{A}(y, r^I \oplus e^i) = \text{gl}(\hat{x}, r^I \oplus e^i)$ dengan probabilitas paling sedikit $\frac{1}{2} + \varepsilon(n)/2$ atas pilihan r; hal ini terjadi karena $\hat{x} \in S_n$ dan

$r^I \oplus e^i$ terdistribusi secara merata. Jadi, untuk himpunan bagian tak kosong I kita mempunyai $Pr[x_i^I = \hat{x_i}] \geq \frac{1}{2} + \varepsilon(n)/2$. Selain itu, $\{x_i^I\}I \subseteq \{1, \ldots, \ell\}$ independen berpasangan karena $\{r^I\}I \subseteq \{1, \ldots, \ell\}$ (dan karenanya $\{r^I \oplus e^i\}I \subseteq \{1, \ldots, \ell\}$) independen berpasangan. Karena x_i didefinisikan sebagai nilai yang sering muncul di antara $\{x_i^I\}I \subseteq \{1, \ldots, \ell\}$, kita dapat menerapkan Proposisi A.13 untuk memperoleh

$$\begin{aligned}\Pr[x_i \neq \hat{x}_i] &\leq \frac{1}{4 \cdot (\varepsilon(n)/2)^2 \cdot (2^\ell - 1)} \\ &\leq \frac{1}{4 \cdot (\varepsilon(n)/2)^2 \cdot (2n/\varepsilon(n)^2)} \\ &= \frac{1}{2n}.\end{aligned}$$

Persamaan di atas berlaku untuk semua i, jadi dengan menerapkan ikatan gabungan kita melihat bahwa probabilitas $x_i \neq \hat{x}_i$ untuk beberapa i adalah paling banyak 1/2. Artinya, $x_i = \hat{x}_i$ untuk semua i (dan karenanya $x = \hat{x}$) dengan probabilitas paling sedikit 1/2.

Gabungkan semuanya: Misalkan n seperti pada Klaim 7.18 dan $\mathcal{Y} = f(\hat{x})$. Dengan probabilitas paling sedikit $\varepsilon(n)/2$ kita mempunyai $\hat{x} \in S_n$. Semua tebakan σ^i benar dengan probabilitas paling sedikit

$$\frac{1}{2^\ell} \geq \frac{1}{2 \cdot (2n/\varepsilon(n)^2 + 1)} > \frac{\varepsilon(n)^2}{5n}$$

untuk n cukup besar. Dikondisikan pada kedua hal di atas, $\mathcal{A}'$ menghasilkan $x = \hat{x}$ dengan probabilitas minimal 1/2. Probabilitas keseluruhan dimana $\mathcal{A}'$ membalikkan masukannya setidaknya adalah $\varepsilon(\frac{n)^3}{20n} = 1/(20np(n)^3)$ untuk banyak n yang tak terhingga. Karena $20np(n)^3$ adalah polinomial dalam n, hal ini membuktikan Proposisi 7.17.

7.4 MEMBANGUN GENERATOR PSEUDORANDOM

Pertama-tama kami menunjukkan cara membuat generator pseudorandom yang memperluas inputnya sebanyak satu bit, dengan asumsi adanya permutasi satu arah. Kami kemudian menunjukkan cara memperluasnya untuk mendapatkan faktor ekspansi polinomial apa pun.

Generator Pseudorandom dengan Ekspansi Minimal

Misalkan ƒ merupakan permutasi satu arah dengan predikat hard-core hc. Artinya hc(s) "tampak acak" jika diberikan $f(s)$, ketika s seragam. Lebih lanjut, karena ƒ adalah permutasi, $f(s)$ sendiri terdistribusi secara seragam. (Menerapkan permutasi pada nilai yang terdistribusi secara seragam akan menghasilkan nilai yang terdistribusi secara seragam.) Jadi jika s adalah string n-bit yang seragam, maka string $(n+1)$-bit $f(s)||\text{hc}(s)$ terdiri dari n-bit yang seragam. bit string ditambah pada bit tambahan yang terlihat seragam bahkan dikondisikan pada n bit awal; dengan kata lain, string bit $(n+1)$ ini adalah pseudorandom. Jadi, algoritma G yang didefinisikan oleh $G(s) = f(s)||\text{hc}(s)$ adalah generator pseudorandom.

TEOREMA 7.19 Misalkan f adalah permutasi satu arah dengan predikat hard-core hc. Maka algoritma G yang didefinisikan oleh $G(s) = f(s)||\text{hc}(s)$ merupakan generator pseudorandom dengan faktor ekspansi $\ell(n) = n + 1$.

BUKTI Biarkan D menjadi algoritma waktu polinomial probabilistik. Kita buktikan bahwa ada fungsi yang dapat diabaikan sehingga

$$\Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r)=1] - \Pr_{s\leftarrow\{0,1\}^{n}}[D(G(s))=1] \leq \text{negl}(n). \qquad (7.3)$$

Argumen serupa menunjukkan bahwa ada fungsi negl’ yang dapat diabaikan

$$\Pr_{s\leftarrow\{0,1\}^{n}}[D(G(s))=1] - \Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r)=1] \leq \text{negl}'(n),$$

yang melengkapi buktinya.
Amati dulu itu

$$\begin{aligned}
\Pr_{r\leftarrow\{0,1\}^{n+1}}[D(r)=1] &= \Pr_{r\leftarrow\{0,1\}^{n},\, r'\leftarrow\{0,1\}}[D(r\|r')=1] \\
&= \Pr_{s\leftarrow\{0,1\}^{n},r'\leftarrow\{0,1\}}[D(f(s)\|r')=1] \\
&= \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^{n}}[D(f(s)\|\text{hc}(s))=1] \\
&\quad + \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^{n}}[D(f(s)\|\overline{\text{hc}}(s))=1],
\end{aligned}$$

menggunakan fakta bahwa f adalah permutasi untuk persamaan kedua, dan bit seragam r' sama dengan $\text{hc}(s)$ dengan probabilitas tepat 1/2 untuk persamaan ketiga. Sejak

$$\Pr_{s\leftarrow\{0,1\}^{n}}[D(G(s))=1] = \Pr_{s\leftarrow\{0,1\}^{n}}[D(f(s)\|\text{hc}(s))=1]$$

(menurut definisi G), ini berarti Persamaan (7.3) ekuivalen dengan

$$\frac{1}{2}\cdot\left(\Pr_{s\leftarrow\{0,1\}^{n}}[D(f(s)\|\overline{\text{hc}}(s))=1] - \Pr_{s\leftarrow\{0,1\}^{n}}[D(f(s)\|\text{hc}(s))=1]\right) \leq \text{negl}(n).$$

Perhatikan algoritma berikut yang memberikan masukan nilai $y = f(s)$ dan mencoba memprediksi nilai $\text{hc}(s)$:

1. Pilih seragam $r' \in \{0,1\}$.

2. Jalankan $D(\mathcal{Y}||r')$. Jika D menghasilkan 0, menghasilkan r'; jika tidak, keluarkan $\bar{r}'$. Jelas $\mathcal{A}$ berjalan dalam waktu polinomial. Menurut definisi $\mathcal{A}$, kita punya

$$\begin{aligned}
&\Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s)) = \mathsf{hc}(s)] \\
&= \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s)) = \mathsf{hc}(s) \mid r' = \mathsf{hc}(s)] \\
&\quad + \frac{1}{2}\cdot \Pr_{s\leftarrow\{0,1\}^n}[\mathcal{A}(f(s)) = \mathsf{hc}(s) \mid r' \neq \mathsf{hc}(s)] \\
&= \frac{1}{2}\cdot \left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s)) = 0] + \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s)) = 1]\right) \\
&= \frac{1}{2}\cdot \left(\left(1 - \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\mathsf{hc}(s)) = 1]\right) + \Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s)) = 1]\right) \\
&= \frac{1}{2} + \frac{1}{2}\cdot \left(\Pr_{s\leftarrow\{0,1\}^n}[D(f(s)\|\overline{\mathsf{hc}}(s)) = 1] - \Pr_{s\leftarrow\{0,1\}^n}[D\,(f(s)\|\mathsf{hc}(s)) = 1]\right).
\end{aligned}$$

Karena hc adalah predikat inti dari f, maka terdapat fungsi yang dapat diabaikan yang mana

$$\frac{1}{2}\cdot \left(\Pr_{s\leftarrow\{0,1\}^n}[D\,(f(s)\|\overline{\mathsf{hc}}(s)) = 1] - \Pr_{s\leftarrow\{0,1\}^n}[D\,(f(s)\|\mathsf{hc}(s)) = 1]\right) \leq \mathsf{negl}(n),$$

Meningkatkan Faktor Ekspansi

Kami sekarang menunjukkan bahwa faktor ekspansi generator pseudorandom dapat ditingkatkan dengan jumlah (polinomial) yang diinginkan. Artinya konstruksi sebelumnya, dengan faktor muai $\ell(n) = n + 1$, cukup untuk membuat generator pseudorandom dengan faktor muai arbitrer (polinomial).

TEOREMA 7.20 Jika terdapat generator pseudorandom G dengan faktor ekspansi $n + 1$, maka untuk polinomial mana pun terdapat generator pseudorandom $\hat{G}$ dengan faktor ekspansi poli(n).

BUKTI Pertama-tama kita mempertimbangkan untuk membuat generator pseudorandom $\hat{G}$ yang menghasilkan $n + 2$ bit.

$\hat{G}$ berfungsi sebagai berikut: Diberikan seed awal $s \in \{0,1\}^n$, ia menghitung $t_1 := G(s)$ untuk mendapatkan $n + 1$ bit pseudorandom. n bit awal dari t_1 kemudian digunakan kembali sebagai seed untuk $G; n + 1$ bit yang dihasilkan, digabungkan dengan bit terakhir t_1, menghasilkan keluaran $(n + 2)$-bit. (Lihat Gambar 7.1.) Penerapan kedua dari G menggunakan benih pseudorandom, bukan benih acak. Bukti keamanan yang kami berikan selanjutnya menunjukkan bahwa hal ini tidak berdampak pada keacakan semu pada keluaran.

Sekarang kita buktikan bahwa $\hat{G}$ adalah generator pseudorandom. Tentukan tiga barisan distribusi $\{H_n^0\}_{n=1,\ldots}$, $\{H_n^1\}_{n=1,\ldots}$, dan $\{H_n^2\}_{n=1,\ldots}$, dimana masing-masing H_n^0, H_n^1, dan H_n^2 adalah distribusi string dengan panjang $n + 2$. Dalam distribusi H_n^0, string seragam $t_0 \in \{0,1\}^n$ dipilih dan keluarannya adalah $\hat{G}(t_0)$. Dalam distribusi H_n^1, string seragam $t_1 \in \{0,1\}^{n+1}$

dipilih dan diuraikan sebagai $s_1 \parallel \sigma_1$ (dimana s_1 adalah n bit awal dari t_1 dan σ_1 adalah bit akhir). Outputnya adalah $t_2 := G(s_1) \parallel \sigma_1$. Dalam distribusi H_n^2, keluarannya berupa string seragam $t_2 \in \{0,1\}^{n+2}$. Kami menyatakan dengan $t_2 \leftarrow H_n^i$ proses menghasilkan $(n+2)$-bit string t_2 menurut distribusi H_n^i.

Perbaiki pembeda waktu polinomial probabilistik sembarang D. Pertama-tama kita klaim bahwa ada fungsi yang dapat diabaikan negl' sehingga

$$\left| \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] \right| \leq \mathrm{negl}'(n). \qquad (7.4)$$

Untuk melihatnya, pertimbangkan pembeda waktu polinomial D' yang, pada masukan $t_1 \in \{0,1\}^{n+1}$, menguraikan t_1 sebagai $s_1 \parallel \sigma_1$ dengan $|s_1| = n$, menghitung $t_2 := G(s_1) \parallel \sigma_1$, dan keluaran $D(t_2)$. Jelasnya D' berjalan dalam waktu polinomial. Perhatikan bahwa:

1. Jika t_1 seragam, maka distribusi pada t_2 yang dihasilkan oleh D' sama persis dengan distribusi H_n^1. Dengan demikian,

$$\Pr_{t_1 \leftarrow \{0,1\}^{n+1}}[D'(t_1) = 1] = \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1].$$

2. Jika $t_1 = G(s)$ untuk seragam $s \in \{0,1\}^n$, distribusi pada t_2 yang dihasilkan oleh D' sama persis dengan distribusi H_n^0. Itu adalah,

$$\Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] = \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1].$$

Keacakan semu G menyiratkan bahwa ada fungsi yang dapat diabaikan dengan negl'

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D'(G(s)) = 1] - \Pr_{t_1 \leftarrow \{0,1\}^{n+1}}[D'(t_1) = 1] \right| \leq \mathrm{negl}'(n).$$

Persamaan (7.4) berikut. Selanjutnya kita menyatakan bahwa terdapat fungsi yang dapat diabaikan negl'' sehingga

$$\left| \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1] \right| \leq \mathrm{negl}''(n). \qquad (7.5)$$

Untuk melihatnya, pertimbangkan pembeda waktu polinomial D'' yang, pada masukan $w \in \{0,1\}^{n+1}$, pilih seragam $\sigma_1 \in \{0,1\}$, himpunan $t_2 := w \parallel \sigma_1$, dan keluaran $D(t_2)$. Jika w seragam maka t_2 juga seragam; dengan demikian,

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D''(w) = 1] = \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1].$$

Sebaliknya, jika $w = G(s)$ untuk seragam $s \in \{0,1\}^n$, maka t_2 terdistribusi tepat menurut H_n^1 dan seterusnya

$$\Pr_{s \leftarrow \{0,1\}^n}[D''(G(s)) = 1] = \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1].$$

Seperti sebelumnya, pseudorandomness G menyiratkan Persamaan (7.5).
Menyatukan semuanya, kita punya

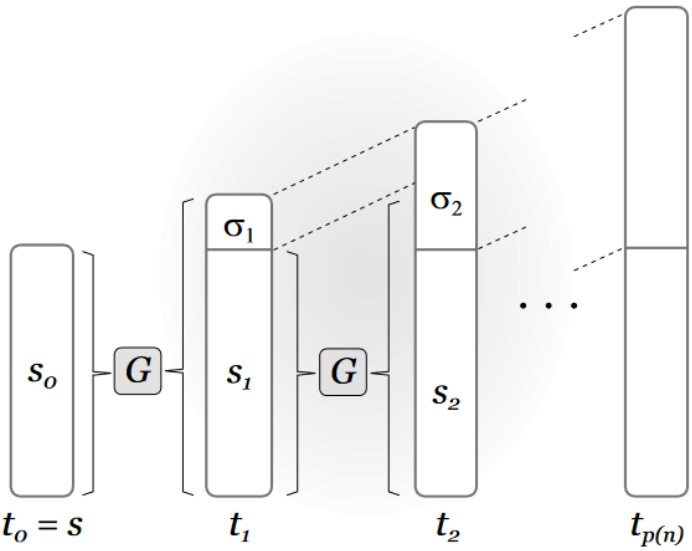
$$\begin{aligned}
&\left| \Pr_{s \leftarrow \{0,1\}^n}[D(\hat{G}(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{n+2}}[D(r) = 1] \right| \qquad (7.6)\\
&\quad = \left| \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1] \right| \\
&\quad \leq \left| \Pr_{t_2 \leftarrow H_n^0}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] \right| \\
&\qquad + \left| \Pr_{t_2 \leftarrow H_n^1}[D(t_2) = 1] - \Pr_{t_2 \leftarrow H_n^2}[D(t_2) = 1] \right| \\
&\quad \leq \mathsf{negl}'(n) + \mathsf{negl}''(n),
\end{aligned}$$

menggunakan Persamaan (7.4) dan (7.5). Karena D adalah pembeda waktu polinomial sembarang, hal ini membuktikan bahwa $\hat{G}$ adalah generator pseudorandom.

Kasus umum. Ide yang sama seperti di atas dapat diterapkan secara berulang untuk menghasilkan bit pseudorandom sebanyak yang diinginkan. Secara formal, katakanlah kita ingin membuat generator pseudorandom $\hat{G}$ dengan faktor ekspansi $n + p(n)$, untuk beberapa polinomial p. Pada masukan $s \in \{0,1\}^n$, algoritma $\hat{G}$ melakukan (lihat Gambar 7.1):

1. Tetapkan $t_0 := s$. Untuk $i = 1, \ldots, p(n)$ lakukan:
   (a) Misalkan s_{i-1} adalah n bit pertama dari t_{i-1}, dan misalkan σ_{i-1} menyatakan $i - 1$ bit sisanya. (Jika $i = 1$, $s_0 = t_0$ dan σ_0 adalah string kosong.)
   (b) Himpunan $t_i := G(s'_{i-1}) \parallel \sigma_{i-1}$.
2. Keluaran $t_{p(n)}$.

Kami menunjukkan bahwa $\hat{G}$ adalah generator pseudorandom. Pembuktiannya menggunakan teknik umum yang dikenal sebagai argumen hibrid. (Sebenarnya, bahkan kasus $p(n) = 2$ di atas, menggunakan argumen hibrid sederhana.) Perbedaan utama dengan pembuktian sebelumnya adalah perbedaan teknis. Sebelumnya, kita dapat mendefinisikan dan secara eksplisit bekerja dengan tiga rangkaian distribusi $\{H_n^0\}$, $\{H_n^1\}$, dan $\{H_n^2\}$. Di sini hal itu tidak mungkin karena jumlah distribusi yang dipertimbangkan bertambah dengan n.



GAMBAR 7.1: Meningkatkan perluasan generator pseudorandom.

Untuk setiap n dan $0 \le j \le p(n)$, misalkan H_n^j adalah distribusi string dengan panjang $n + p(n)$ yang didefinisikan sebagai berikut: pilih seragam $t_j \in \{0,1\}^{n+j}$, lalu jalankan $\hat{G}$ mulai dari iterasi $j + 1$ dan keluaran $t_{p(n)}$. (Jika $j = p(n)$ ini berarti kita cukup memilih seragam $t_{p(n)} \in \{0,1\}^{n+p(n)}$ dan mengeluarkannya.) Pengamatan krusialnya adalah bahwa H_n^0 berkaitan dengan keluaran $\hat{G}(s)$ untuk seragam $s \in \{0,1\}^n$, sedangkan $H_n^{p(n)}$ berkaitan dengan keluaran string bit $(n + p(n))$ yang seragam. Memperbaiki pembeda waktu polinomial D, artinya

$$\left| \Pr_{s \leftarrow \{0,1\}^n}[D(\hat{G}(s)) = 1] - \Pr_{r \leftarrow \{0,1\}^{n+p(n)}}[D(r) = 1] \right|$$

$$= \left| \Pr_{t \leftarrow H_n^0}[D(t) = 1] - \Pr_{t \leftarrow H_n^{p(n)}}[D(t) = 1] \right|. \qquad (7.7)$$

1.

Kami membuktikan hal di atas dapat diabaikan, maka $\hat{G}$ adalah generator pseudorandom. Perbaiki D seperti di atas, dan pertimbangkan pembeda D' yang melakukan hal berikut ketika diberi string $w \in \{0,1\}^{n+1}$ sebagai input:

1. Pilih seragam $j \in \{1, \ldots, p(n)\}$.
2. Pilih seragam $\sigma'_j \in \{0,1\}^{j-1}$. (Jika $j = 1$ maka σ'_j adalah string kosong.)
3. Tetapkan $t_j := w \parallel \sigma'_j$. Kemudian jalankan $\hat{G}$ mulai dari iterasi $j + 1$ untuk menghitung $t_{p(n)} \in \{0,1\}^{n+p(n)}$. Keluaran $D(t_{p(n)})$.

Jelasnya D' berjalan dalam waktu polinomial. Menganalisis perilaku D' lebih rumit dari sebelumnya, meskipun gagasan dasarnya sama. Perbaiki n dan katakan D' memilih $j = j^*$. Jika w seragam, maka t_{j^*} seragam sehingga distribusi pada $t \stackrel{\text{def}}{=} t_{p(n)}$ sama persis dengan distribusi $H_n^{j^*}$. Itu adalah,

$$\Pr_{w \leftarrow \{0,1\}^{n+1}}[D'(w) = 1 \mid j = j^*] = \Pr_{t \leftarrow H_n^{j^*}}[D(t) = 1].$$

Karena itu nilai untuk j dipilih dengan probabilitas yang sama,

$$\begin{aligned}\Pr_{w\leftarrow\{0,1\}^{n+1}}[D'(w)=1] &= \frac{1}{p(n)}\cdot\sum_{j^*=1}^{p(n)}\Pr_{w\leftarrow\{0,1\}^{n+1}}[D'(w)=1 \mid j=j^*] \\ &= \frac{1}{p(n)}\cdot\sum_{j^*=1}^{p(n)}\Pr_{t\leftarrow H_n^{j^*}}[D(t)=1]. \qquad (7.8)\end{aligned}$$

Disisi lain, misalnya D' memilih $j=j^*$ dan $w=G(s)$ untuk seragam $s\in\{0,1\}^n$. Mendefinisikan $t_{j^*-1}=s\parallel\sigma'_{j^*}$, kita lihat t_{j^*-1} ini seragam dan eksperimen yang melibatkan D' adalah kesetaraan untuk $\hat{G}$ dari pengulangan j^* untuk menghitung $t_{p(n)}$. Karena itu, ditribusi dalam $t \stackrel{\text{def}}{=} t_{p(n)}$ yang sekarang distribusinya menjadi $H_n^{j^*-1}$, dan juga

$$\Pr_{s\leftarrow\{0,1\}^n}[D'(G(s))=1 \mid j=j^*]=\Pr_{t\leftarrow H_n^{j^*-1}}[D(t)=1].$$

Untuk itu,

$$\begin{aligned}\Pr_{s\leftarrow\{0,1\}^n}[D'(\hat{G}(s))=1] &= \frac{1}{p(n)}\cdot\sum_{j^*=1}^{p(n)}\Pr_{s\leftarrow\{0,1\}^n}[D'(G(s))=1 \mid j=j^*] \\ &= \frac{1}{p(n)}\cdot\sum_{j^*=1}^{p(n)}\Pr_{t\leftarrow H_n^{j^*-1}}[D(t)=1] \\ &= \frac{1}{p(n)}\cdot\sum_{j^*=0}^{p(n)-1}\Pr_{t\leftarrow H_n^{j^*}}[D(t)=1]. \qquad (7.9)\end{aligned}$$

Sekarang kita dapat menganalisis seberapa baik D' membedakan keluaran G dari acak:

$$\begin{aligned}&\left|\Pr_{s\leftarrow\{0,1\}^n}[D'(G(s))=1]-\Pr_{w\leftarrow\{0,1\}^{n+1}}[D'(w)=1]\right| \qquad (7.10)\\ &= \frac{1}{p(n)}\cdot\left|\sum_{j^*=0}^{p(n)-1}\Pr_{t\leftarrow H_n^{j^*}}[D(t)=1]-\sum_{j^*=1}^{p(n)}\Pr_{t\leftarrow H_n^{j^*}}[D(t)=1]\right| \\ &= \frac{1}{p(n)}\cdot\left|\Pr_{t\leftarrow H_n^0}[D(t)=1]-\Pr_{t\leftarrow H_n^{p(n)}}[D(t)=1]\right|,\end{aligned}$$

mengandalkan Persamaan (7.8) dan (7.9) untuk persamaan pertama. (Persamaan kedua berlaku karena suku-suku yang sama dimasukkan dalam setiap penjumlahan, kecuali suku pertama dari jumlah kiri dan suku terakhir dari jumlah kanan.) Karena G adalah generator pseudorandom, suku di sebelah kiri sisi Persamaan (7.10) dapat diabaikan; karena p adalah polinomial, ini menyiratkan bahwa Persamaan (7.7) dapat diabaikan, melengkapi bukti bahwa $\hat{G}$ adalah generator pseudorandom.

Menyatukan semuanya. Misalkan f merupakan permutasi satu arah. Mengambil generator pseudorandom dengan faktor ekspansi $n + 1$ dari Teorema 7.19, dan meningkatkan faktor ekspansi menjadi $n + \ell$ menggunakan pendekatan pembuktian Teorema 7.20, kita memperoleh generator pseudorandom $\hat{G}$ berikut:

$$\hat{G}(s) = f^{(\ell)}(s) \,\|\, \mathsf{hc}(f^{(\ell-1)}(s)) \,\|\, \cdots \,\|\, \mathsf{hc}(s),$$

dimana $f^{(i)}(s)$ mengacu pada i untuk penghitungan dari f. Perhatikan bahwa $\hat{G}$ menggunakan ℓ evaluasi f, dan menghasilkan satu bit semu per evaluasi menggunakan predikat hard-core hc.

Koneksi ke streaming cipher. Ingat dari Bagian 3.3.1 bahwa stream cipher (tanpa IV) ditentukan oleh algoritma (Init, GetBits), di mana Init mengambil seed $s \in \{0,1\}^n$ dan mengembalikan status awal st, dan GetBits mengambil input status saat ini st dan mengeluarkan sedikit σ dan status st′ yang diperbarui. Konstruksi $\hat{G}$ dari bukti sebelumnya cocok dengan paradigma ini: anggap Init sebagai algoritma sepele yang menghasilkan $\text{st} = s$, dan tentukan GetBits(st) untuk menghitung $G(\text{st})$, parsing hasilnya sebagai $\text{st}' \,\|\, \sigma$ dengan $|\text{st}'| = n$, dan mengeluarkan bit σ dan status st′ yang diperbarui. (Jika kita menggunakan stream cipher ini untuk menghasilkan bit keluaran $p(n)$ mulai dari seed s, maka kita mendapatkan bit $p(n)$ terakhir dari $\hat{G}(s)$ dengan urutan terbalik.) Bukti sebelumnya menunjukkan bahwa ini menghasilkan pseudorandom generator.

Argumen hibrida. Argumen hibrid adalah alat dasar untuk membuktikan ketidakmampuan membedakan ketika sebuah primitif dasar (atau beberapa primitif berbeda) diterapkan berkali-kali. Secara informal, teknik ini bekerja dengan mendefinisikan serangkaian "distribusi hibrid" perantara yang menjembatani dua "distribusi ekstrem" yang ingin kita buktikan tidak dapat dibedakan. (Dalam pembuktian di atas, distribusi ekstrem ini sesuai dengan keluaran $\hat{G}$ dan string acak.) Untuk menerapkan teknik pembuktian, tiga kondisi harus dipenuhi.

Pertama, distribusi ekstrem harus sesuai dengan kasus awal yang menjadi perhatian. (Dalam bukti di atas, H_n^0 sama dengan distribusi yang disebabkan oleh $\hat{G}$, sedangkan $H_n^{p(n)}$ adalah distribusi seragam.) Kedua, kemampuan membedakan distribusi hibrid yang berurutan harus dapat diterjemahkan ke dalam mematahkan beberapa asumsi yang mendasarinya. (Di atas, pada dasarnya kami menunjukkan bahwa membedakan H_n^i dari H_n^{i+1} sama dengan membedakan keluaran G dari acak.) Terakhir, jumlah distribusi hibrid haruslah polinomial. Lihat juga Teorema 7.32.

7.5 MEMBANGUN FUNGSI PSEUDORANDOM

Kami sekarang menunjukkan bagaimana membangun fungsi pseudorandom dari generator pseudorandom (penggandaan panjang) apa pun. Ingatlah bahwa fungsi pseudorandom adalah fungsi berkunci F yang dapat dihitung secara efisien dan tidak dapat dibedakan dari fungsi yang benar-benar acak seperti yang dijelaskan dalam Bagian 3.5. Untuk mempermudah, kami membatasi perhatian kita di sini pada kasus di mana F adalah kekekalan

panjang, yang berarti bahwa untuk $k \in \{0,1\}^n$ fungsi F_k memetakan masukan n-bit ke keluaran n-bit. Fungsi pseudorandom (yang mempertahankan panjang) dapat dilihat, secara informal, sebagai generator pseudorandom dengan faktor ekspansi $n \cdot 2^n$; dengan generator acak semu G, kita dapat mendefinisikan $F_k(i)$ (untuk $0 \leq i < 2^n$) sebagai blok n-bit ke-i dari $G(k)$. Alasan mengapa hal ini tidak berhasil adalah karena F harus dapat dihitung secara efisien; ada banyak blok secara eksponensial, dan kita memerlukan cara untuk menghitung blok ke-i tanpa harus menghitung semua blok lainnya.

Kita akan melakukan ini dengan menghitung "blok" keluaran dengan menelusuri pohon biner. Kami mencontohkan konstruksinya dengan terlebih dahulu menunjukkan fungsi pseudorandom yang mengambil input 2-bit. Misalkan G adalah generator pseudorandom dengan faktor ekspansi $2n$. Jika kita menggunakan G seperti pada pembuktian Teorema 7.20 kita dapat memperoleh generator pseudorandom $\hat{G}$ dengan faktor ekspansi $4n$ yang menggunakan tiga pemanggilan G. (Kita menghasilkan n bit pseudorandom tambahan setiap kali G diterapkan.) Jika kita mendefinisikan $F'_k(i)$ (di mana $0 \leq i < 4$ dan i dikodekan sebagai string biner 2-bit) menjadi blok ke-i dari $\hat{G}(k)$, maka komputasi $F'_k(3)$ akan memerlukan komputasi seluruh $\hat{G}$ dan karenanya tiga pemanggilan G. Kami menunjukkan cara membuat fungsi pseudorandom F hanya dengan menggunakan dua pemanggilan G pada input apa pun.

Misalkan G_0 dan G_1 merupakan fungsi yang menyatakan paruh pertama dan kedua keluaran G; yaitu, $G(k) = G_0(k) \parallel G_1(k)$ di mana $|G_0(k)| = |G_1(k)| = |k|$. Definisikan F sebagai berikut:

$$F_k(00) = G_0(G_0(k)) \qquad F_k(10) = G_0(G_1(k))$$
$$F_k(01) = G_1(G_0(k)) \qquad F_k(11) = G_1(G_1(k)).$$

Kami mengklaim bahwa keempat string di atas adalah pseudorandom meskipun dilihat bersama-sama. (Ini cukup untuk membuktikan bahwa F adalah pseudorandom.) Secara intuitif, ini karena $G_0(k) \parallel G_1(k) = G(k)$ adalah pseudorandom dan karenanya tidak dapat dibedakan dari string $2n$-bit seragam $k_0 \parallel k_1$. Tapi kemudian

$$G_0(G_0(k)) \parallel G_1(G_0(k)) \parallel G_0(G_1(k)) \parallel G_1(G_1(k))$$

tidak dapat dibedakan dari

$$G_0(k_0) \parallel G_1(k_0) \parallel G_0(k_1) \parallel G_1(k_1) = G(k_0) \parallel G(k_1).$$

Karena G adalah generator pseudorandom, generator di atas tidak dapat dibedakan dari string $4n$-bit yang seragam. Bukti formal menggunakan argumen hibrid.

Menggeneralisasikan ide ini, kita dapat memperoleh fungsi pseudorandom pada input n-bit dengan mendefinisikan

$$F_k(x) = G_{x_n}(\cdots G_{x_1}(k)\cdots),$$

dimana $x = x_1 \ldots x_n$; lihat Konstruksi 7.21. Intuisi mengapa fungsi ini bersifat pseudorandom sama dengan sebelumnya, namun pembuktian formalnya diperumit oleh fakta bahwa kini terdapat banyak masukan yang perlu dipertimbangkan secara eksponensial.

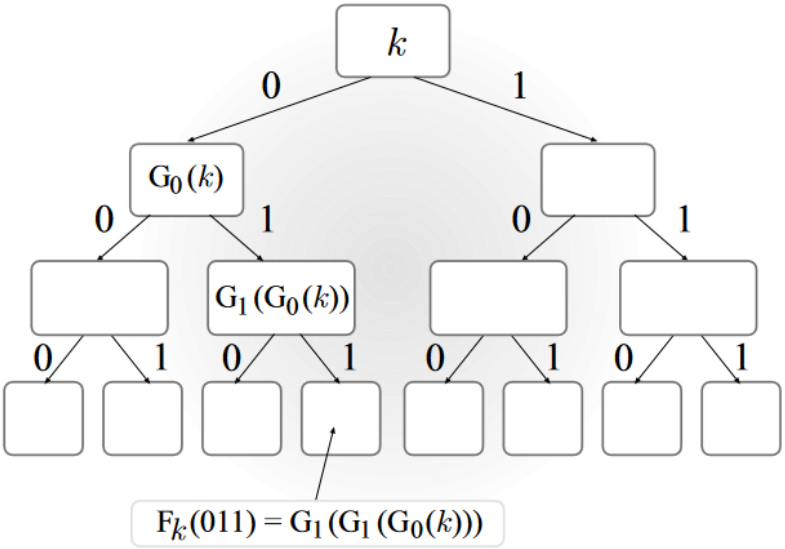
CONSTRUCTION 7.21

Let G be a pseudorandom generator with expansion factor $\ell(n) = 2n$, and define G_0, G_1 as in the text. For $k \in \{0,1\}^n$, define the function $F_k : \{0,1\}^n \to \{0,1\}^n$ as:

$$F_k(x_1 x_2 \cdots x_n) = G_{x_n}(\cdots(G_{x_2}(G_{x_1}(k)))\cdots).$$

Fungsi pseudorandom dari generator pseudorandom.

Hal ini berguna untuk melihat konstruksi ini sebagai pendefinisian, untuk setiap kunci $k \in \{0,1\}^n$, pohon biner lengkap dengan kedalaman n di mana setiap node berisi nilai n-bit. (Lihat Gambar 7.2, dimana $n = 3$.) Akar mempunyai nilai k, dan untuk setiap simpul internal yang bernilai k', anak kirinya bernilai $G_0(k')$ dan anak kanannya bernilai $G_1(k')$. Hasil $F_k(x)$ untuk $x = x_1 \ldots x_n$ kemudian didefinisikan sebagai nilai pada simpul daun yang dicapai dengan melintasi pohon sesuai dengan bit-bit x, dimana $x_i = 0$ berarti "ke kiri" dan $x_i = 1$ berarti "pergi" Kanan." (Fungsi ini hanya didefinisikan untuk masukan dengan panjang n, sehingga hanya nilai pada daun yang dikeluarkan.) Ukuran pohon adalah eksponensial dalam n. Namun demikian, untuk menghitung $F_k(x)$ seluruh pohon tidak perlu dibuat atau disimpan; hanya n evaluasi G yang diperlukan.



GAMBAR 7.2: Membangun fungsi pseudorandom.

TEOREMA 7.22 Jika G merupakan generator pseudorandom dengan faktor ekspansi $\ell(n) = 2n$, maka Konstruksi 7.21 merupakan fungsi pseudorandom.

BUKTI Pertama-tama kita tunjukkan bahwa untuk sembarang polinomial t tidak mungkin membedakan $t(n)$ string seragam $2n$-bit dari $t(n)$ string pseudorandom; yaitu, untuk polinomial apa pun t dan algoritma PPT apa pun A, hal berikut ini dapat diabaikan:

$$\left|\Pr\left[A\left(r_1\|\cdots\|r_{t(n)}\right)=1\right]-\Pr\left[A\left(G(s_1)\|\cdots\|G(s_{t(n)})\right)=1\right]\right|,$$

dimana probabilitas pertama melebihi pilihan seragam $r_1,\ldots,r_{t(n)} \in \{0,1\}^{2n}$, dan probabilitas kedua melebihi pilihan seragam $s_1,\ldots,s_{t(n)} \in \{0,1\}^n$.

Buktinya adalah dengan argumen hibrida. Perbaiki polinomial t dan algoritma PPT A, dan pertimbangkan algoritma A' berikut:

Pembeda A':
A' diberikan sebagai input string $w \in \{0,1\}^{2n}$.

1. Pilih seragam $j \in \{1,\ldots,\mathrm{t(n)}\}$.
2. Pilih nilai yang seragam dan independen $r_1,\ldots,r_{j-1} \in \{0,1\}^{2n}$ dan $s_{j+1},\ldots,s_{t(n)} \in \{0,1\}^n$.
3. Keluaran $A\ (r_1 \parallel \cdots \parallel r_{j-1} \parallel w \parallel G(s_{j+1}) \parallel \cdots \parallel G(s_{t(n)}))$.

Untuk sembarang n dan $0 \le i \le t(n)$, misalkan G_n^i menyatakan distribusi string dengan panjang $2n \cdot t(n)$ yang mana i "balok" pertama dengan panjang $2n$ adalah seragam dan sisa $t(n) - i$ adalah balok adalah pseudorandom. Perhatikan bahwa $G_n^{t(n)}$ berhubungan dengan distribusi di mana semua blok $t(n)$ seragam, sedangkan G_n^0 berhubungan dengan distribusi di mana semua blok $t(n)$ bersifat pseudorandom. Itu adalah,

$$\left|\Pr_{y\leftarrow G_n^{t(n)}}[A(y)=1]-\Pr_{y\leftarrow G_n^0}[A(y)=1]\right| \qquad (7.11)$$
$$=\left|\Pr\left[A\left(r_1\|\cdots\|r_{t(n)}\right)=1\right]-\Pr\left[A\left(G(s_1)\|\cdots\|G(s_{t(n)})\right)=1\right]\right|$$

Katakanlah A' memilih $j = j*$. Jika inputnya w adalah string $2n$-bit yang seragam, maka A dijalankan pada input yang didistribusikan menurut G_n^{j*}. Sebaliknya, jika $w = G(s)$ untuk s seragam, maka A dijalankan pada input yang terdistribusi menurut G_n^{j*-1}. Artinya

$$\Pr_{r\leftarrow\{0,1\}^{2n}}[A'(r)=1]=\frac{1}{t(n)}\cdot\sum_{j=1}^{t(n)}\Pr_{y\leftarrow G_n^j}[A(y)=1]$$

Dan

$$\Pr_{s\leftarrow\{0,1\}^n}[A'(G(s))=1]=\frac{1}{t(n)}\cdot\sum_{j=0}^{t(n)-1}\Pr_{y\leftarrow G_n^j}[A(y)=1].$$

Karena itu,

$$\left|\Pr_{r\leftarrow\{0,1\}^{2n}}[A'(r)=1]-\Pr_{s\leftarrow\{0,1\}^n}[A'(G(s))=1]\right| \qquad (7.12)$$
$$=\frac{1}{t(n)}\cdot\left|\Pr_{y\leftarrow G_n^{t(n)}}[A(y)=1]-\Pr_{y\leftarrow G_n^0}[A(y)=1]\right|.$$

Karena G adalah generator pseudorandom dan A' berjalan dalam waktu polinomial, kita tahu bahwa ruas kiri Persamaan (7.12) harus diabaikan; karena $t(n)$ adalah polinomial, ini berarti ruas kiri Persamaan (7.11) juga dapat diabaikan.

Beralih ke inti pembuktian, sekarang kami menunjukkan bahwa F seperti pada Konstruksi 7.21 adalah fungsi pseudorandom. Misalkan D adalah pembeda PPT sembarang yang diberikan 1^n sebagai masukan. Kami menunjukkan bahwa D tidak dapat membedakan antara kasus ketika ia diberi akses oracle ke fungsi yang sama dengan F_k untuk k yang seragam, atau fungsi yang dipilih secara seragam dari Func_n. (Lihat Bagian 3.5.) Untuk melakukannya, kami menggunakan argumen hibrid lainnya. Di sini, kita mendefinisikan urutan distribusi nilai pada daun pohon biner lengkap dengan kedalaman n. Dengan mengaitkan setiap daun dengan string dengan panjang n seperti pada Konstruksi 7.21, kita dapat melihatnya secara setara sebagai distribusi fungsi yang memetakan masukan n-bit ke keluaran n-bit. Untuk setiap n dan $0 \leq i \leq n$, misalkan H_n^i adalah distribusi berikut pada nilai-nilai pada daun pohon biner dengan kedalaman n: pertama-tama pilihlah nilai untuk node pada level i secara independen dan seragam dari $\{0,1\}^n$. Kemudian untuk setiap node pada level i atau dibawahnya yang bernilai k, anak kirinya diberi nilai $G_0(k)$ dan anak kanannya diberi nilai $G_1(k)$. Perhatikan bahwa H_n^n berhubungan dengan distribusi di mana semua nilai pada daun dipilih secara seragam dan independen, dan dengan demikian berhubungan dengan pemilihan fungsi seragam dari Func_n, sedangkan H_n^0 berhubungan dengan pemilihan kunci seragam k dalam Konstruksi 7.21 karena dalam kasus tersebut hanya root (pada level 0) dipilih secara seragam. Itu adalah,

$$\left|\Pr_{k\leftarrow\{0,1\}^n}[D^{F_k(\cdot)}(1^n)=1]-\Pr_{f\leftarrow\mathrm{Func}_n}[D^{f(\cdot)}(1^n)=1]\right|$$
$$=\left|\Pr_{f\leftarrow H_n^0}[D^{f(\cdot)}(1^n)=1]-\Pr_{f\leftarrow H_n^n}[D^{f(\cdot)}(1^n)=1]\right|. \qquad (7.13)$$

Kita tunjukkan bahwa Persamaan (7.13) dapat diabaikan, sehingga melengkapi pembuktiannya.

Misalkan $t = t(n)$ adalah batas atas polinomial dari jumlah kueri yang dibuat D ke oraclenya pada input 1^n. Tentukan pembeda A yang mencoba membedakan $t(n)$ string seragam $2n$-bit dari $t(n)$ string pseudorandom, sebagai berikut:

Pembeda A:
A diberikan sebagai input string $2n \cdot t(n)$-bit $w_1 \| \cdots \| w_{t(n)}$.

1. Pilih seragam $j \in \{0, \ldots, n-1\}$. Berikut ini, A (secara implisit) memelihara pohon biner dengan kedalaman n dengan nilai n-bit pada (subset dari) node internal pada kedalaman $j + 1$ dan di bawahnya.
2. Jalankan $D(1^n)$. Saat D membuat kueri Oracle $x = x_1 \ldots x_n$, lihat awalan $x_1 \ldots x_j$. Ada dua kasus:
   - Jika D belum pernah membuat query dengan awalan ini sebelumnya, maka gunakan $x_1 \ldots x_j$ untuk mencapai node v pada tingkat ke-j pohon. Ambil string $2n$-bit berikutnya yang tidak terpakai w dan atur nilai anak kiri dari simpul v ke separuh kiri w, dan nilai anak kanan v ke separuh kanan w.
   - Jika D telah membuat query dengan awalan $x_1 \ldots x_j$ sebelumnya, maka node $x_1 \ldots x_{j+1}$ telah diberi nilai.
   - Dengan menggunakan nilai pada simpul $x_1 \ldots x_{j+1}$, hitung nilai pada daun yang berhubungan dengan $x_1 \ldots x_n$ seperti pada Konstruksi 7.21, dan kembalikan nilai ini ke D.
3. Ketika eksekusi D selesai, keluarkan bit yang dikembalikan oleh D.
   A berjalan dalam waktu polinomial. Penting di sini bahwa A tidak perlu menyimpan seluruh pohon biner berukuran eksponensial. Sebaliknya, ia "mengisi" nilai paling banyak $2t(n)$ node di pohon. Katakanlah A memilih $j = j^*$. Perhatikan bahwa:
   1. Jika input A adalah string $2n \cdot t(n)$-bit yang seragam, maka jawaban yang diberikannya kepada D terdistribusi persis seolah-olah D berinteraksi dengan fungsi yang dipilih dari distribusi $H_n^{j^*+1}$. Hal ini berlaku karena nilai node pada level $j^* + 1$ pohon adalah seragam dan independen.
   2. Jika masukan A terdiri dari $t(n)$ string pseudorandom—yaitu, $w_i = G(s_i)$ untuk benih seragam s_i—maka jawaban yang diberikan kepada D terdistribusi tepat seolah-olah D berinteraksi dengan fungsi yang dipilih dari distribusi $H_n^{j^*}$.

Hal ini berlaku karena nilai node pada level j^* pohon (yaitu nilai s) seragam dan independen. (Nilai s ini tidak diketahui oleh A, namun hal ini tidak ada bedanya.)
Dengan melanjutkan seperti sebelumnya, kita dapat menunjukkan hal itu

$$\begin{aligned} &\left|\Pr\left[A\left(r_1\|\cdots\|r_{t(n)}\right)=1\right]-\Pr\left[A\left(G(s_1)\|\cdots\|G(s_{t(n)})\right)=1\right]\right| \qquad (7.14)\\ &\quad=\frac{1}{n}\cdot\left|\Pr_{f\leftarrow H_n^0}[D^{f(\cdot)}(1^n)=1]-\Pr_{f\leftarrow H_n^n}[D^{f(\cdot)}(1^n)=1]\right|. \end{aligned}$$

Telah kita tunjukkan sebelumnya bahwa Persamaan (7.14) harus diabaikan. Dengan demikian, persamaan di atas menyiratkan bahwa Persamaan (7.13) juga dapat diabaikan.

7.6 MEMBANGUN PERMUTASI PSEUDORANDOM (KUAT).

Kami selanjutnya menunjukkan bagaimana permutasi pseudorandom dan permutasi pseudorandom kuat dapat dibangun dari fungsi pseudorandom apa pun. Ingat dari Bagian 3.5 bahwa permutasi pseudorandom adalah fungsi pseudorandom yang juga dapat dibalik secara efisien, sementara permutasi pseudorandom yang kuat juga sulit dibedakan dari permutasi acak bahkan oleh musuh yang diberi akses oracle ke permutasi dan kebalikannya.

Jaringan Feistel ditinjau kembali. Jaringan Feistel, yang diperkenalkan pada Bagian 6.2, menyediakan cara untuk membangun fungsi yang dapat dibalik dari sekumpulan fungsi yang berubah-ubah. Jaringan Feistel beroperasi dalam serangkaian putaran. Masukan ke putaran ke-i adalah string dengan panjang $2n$, dibagi menjadi dua bagian n-bit L_{i-1} dan R_{i-1} ("separuh kiri" dan "setengah kanan"). Keluaran putaran ke-i adalah string $2n$-bit (L_i, R_i) dimana

$$L_i := R_{i-1} \quad \text{and} \quad R_i := L_{i-1} \oplus f_i(R_{i-1})$$

untuk beberapa fungsi yang dapat dihitung secara efisien (tetapi tidak harus dapat dibalik) f_i memetakan masukan n-bit ke keluaran n-bit. Kami menyatakan dengan $Feistel_{f_1}, \ldots, f_r$ jaringan Feistel putaran-r menggunakan fungsi $f_i, \ldots, f_r$. (Artinya, $Feistel_{f_1}, \ldots, f_r$ (L_0, R_0) mengeluarkan string $2n$-bit (L_r, R_r).) Kita melihat di Bagian 6.2.2 bahwa $Feistel_{f_1}, \ldots, f_r$ merupakan suatu bilangan yang dapat dibalik secara efisien permutasi terlepas dari $\{f_i\}$.

Kita dapat mendefinisikan permutasi berkunci dengan menggunakan jaringan Feistel di mana $\{f_i\}$ bergantung pada sebuah kunci. Misalnya, $F: \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ adalah fungsi pseudorandom, dan definisikan permutasi berkunci $F^{(1)}$ sebagai

$$F_k^{(1)}(x) \stackrel{\text{def}}{=} \mathsf{Feistel}_{F_k}(x).$$

(Perhatikan bahwa $F_k^{(1)}$ memiliki kunci n-bit dan memetakan masukan $2n$-bit ke keluaran $2n$-bit.) Apakah $F^{(1)}$ pseudorandom? Sedikit pemikiran menunjukkan bahwa itu jelas tidak benar. Untuk kunci apa pun $k \in \{0,1\}^n$, n bit pertama dari keluaran $F_k^{(1)}$ (yaitu, L_1) sama dengan n bit terakhir dari masukan (yaitu, R_0), sesuatu yang terjadi dengan hanya probabilitas yang dapat diabaikan untuk fungsi acak.

Coba lagi, tentukan $F^{(2)}: \{0,1\}^{2n} \times \{0,1\}^{2n} \to \{0,1\}^{2n}$sebagai berikut:

$$F_{k_1,k_2}^{(2)}(x) \stackrel{\text{def}}{=} \mathsf{Feistel}_{F_{k_1},F_{k_2}}(x). \qquad (7.15)$$

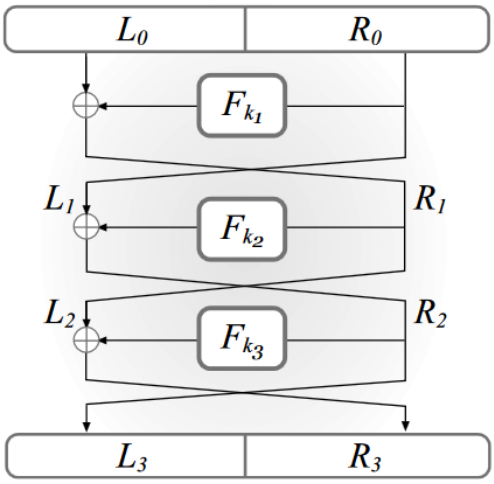
(Perhatikan bahwa k_1 dan k_2 merupakan kunci yang independen.) Sayangnya, $F^{(2)}$ juga bukan merupakan kunci acak semu (pseudorandom), seperti yang diminta untuk ditunjukkan pada Latihan 7.16.

Mengingat hal ini, mungkin agak mengejutkan bahwa jaringan Feistel tiga putaran bersifat pseudorandom. Tentukan permutasi kunci $F^{(3)}$, dengan mengambil kunci dengan panjang $3n$ dan memetakan masukan $2n$-bit ke keluaran $2n$-bit, sebagai berikut:

$$F^{(3)}_{k_1,k_2,k_3}(x) \stackrel{\text{def}}{=} \text{Feistel}_{F_{k_1},F_{k_2},F_{k_3}}(x) \qquad (7.16)$$

dimana, sekali lagi, k_1, k_2, dan k_3 saling bebas. Kita punya:

TEOREMA 7.23 Jika F merupakan fungsi pseudorandom, maka $F^{(3)}$ merupakan permutasi pseudorandom.



GAMBAR 7.3: Jaringan Feistel tiga putaran, seperti yang digunakan untuk membuat permutasi pseudorandom dari fungsi pseudorandom.

BUKTI Dengan cara standar, kita dapat mengganti fungsi pseudorandom yang digunakan dalam konstruksi $F^{(3)}$ dengan fungsi yang dipilih secara seragam dan acak. Keacakan semu dari F menyiratkan bahwa hal ini hanya memiliki efek yang dapat diabaikan pada keluaran pembeda waktu polinomial probabilistik yang berinteraksi dengan $F^{(3)}$ sebagai oracle. Kami meninggalkan detailnya sebagai latihan.

Biarkan D menjadi pembeda waktu polinomial probabilistik. Di sisa buktinya, kami menunjukkan bahwa hal berikut dapat diabaikan:

$$\left|\Pr[D^{\text{Feistel}_{f_1,f_2,f_3}(\cdot)}(1^n)=1]-\Pr[D^{\pi(\cdot)}(1^n)=1]\right|,$$

dimana probabilitas pertama diambil alih pilihan f_1, f_2, f_3 yang seragam dan independen dari Func_n, dan probabilitas kedua diambil alih pilihan seragam π dari Perm_{2n}. Tetapkan beberapa nilai untuk parameter keamanan n, dan biarkan $q = q(n)$ menunjukkan batas atas polinomial pada jumlah kueri Oracle yang dibuat oleh D. Kita berasumsi tanpa kehilangan keumuman bahwa D tidak pernah membuat kueri Oracle yang sama dua kali. Berfokus pada interaksi D dengan $\text{Feistel}_{f_1,f_2,f_3}(\cdot)$, misalkan (L_0^i, R_0^i) menunjukkan kueri ke-i yang dibuat D pada oraclenya, dan misalkan (L_1^i, R_1^i),(L_2^i, R_2^i), dan (L_3^i, R_3^i) menunjukkan nilai tengah setelah

putaran 1, 2, dan 3, masing-masing, hasil dari kueri itu. (Lihat Gambar 7.3.) Perhatikan bahwa D memilih (L_0^i, R_0^i) dan melihat hasilnya (L_3^i, R_3^i), tetapi tidak mengamati secara langsung (L_1^i, R_1^i) atau (L_2^i, R_2^i).

Dikatakan terjadi tumbukan di R_1 jika $R_1^i = R_1^j$ untuk beberapa i, j yang berbeda. Pertama-tama kita buktikan bahwa tumbukan di R_1 terjadi dengan probabilitas yang dapat diabaikan. Pertimbangkan i, j yang tetap dan berbeda. Jika $R_0^i = R_0^j$ maka $L_0^i \neq L_0^j$, tetapi kemudian

$$R_1^i = L_0^i \oplus f_1(R_0^i) \neq L_0^j \oplus f_1(R_0^j) = R_1^j.$$

Jika $R_0^i \neq R_0^j$ maka $f_i(R_0^i)$ dan $f_i(R_0^j)$ seragam dan bebas, jadi

$$\Pr\left[L_0^i \oplus f_1(R_0^i) = L_0^j \oplus f_1(R_0^j)\right] = \Pr\left[f_1(R_0^j) = L_0^i \oplus f_1(R_0^i) \oplus L_0^j\right] = 2^{-n}.$$

Mengambil ikatan gabungan pada semua i, j yang berbeda menunjukkan bahwa peluang tumbukan di R_1 paling banyak adalah $q^2/2^n$.

Katakanlah ada tumbukan di R_2 jika $R_2^i = R_2^j$ untuk beberapa i, j yang berbeda. Kita buktikan bahwa jika tidak terjadi tumbukan di R_1, maka peluang terjadinya tumbukan di R_2 dapat diabaikan. Analisanya seperti di atas: misalkan sembarang i, j tetap, dan perhatikan jika tidak ada tumbukan di R_1 maka $R_1^i \neq R_1^j$. Jadi $f_2(R_1^i)$ dan $f_2(R_1^j)$ adalah seragam dan independen, dan oleh karena itu

$$\Pr\left[L_1^i \oplus f_2(R_1^i) = L_1^j \oplus f_2(R_1^j) \mid \text{no collision at } R_1\right] = 2^{-n}.$$

(Perhatikan bahwa f_2 tidak bergantung pada f_1, membuat penghitungan di atas menjadi mudah.) Dengan mengambil ikatan secara keseluruhan yang berbeda i, j menghasilkan

$$\Pr[\text{collision at } R_2 \mid \text{no collision at } R_1] \leq q^2/2^n.$$

Perhatikan bahwa $L_3^i = R_2^i = L_1^i \oplus f_2(R_1^i)$; jadi, dengan syarat tidak ada tumbukan di R_1, nilai $L_3^1, \ldots, L_3^q$ semuanya independen dan terdistribusi seragam di $\{0,1\}^n$. Jika kita juga mengkondisikan jika tidak ada tumbukan di R_2, maka nilainya $L_3^1, \ldots, L_3^q$ terdistribusi secara merata di antara semua rangkaian q nilai yang berbeda dalam $\{0,1\}^n$. Demikian pula, $R_3^i = L_2^i \oplus f_3(R_2^i)$; dengan demikian, dengan syarat tidak ada tumbukan di R_2, nilai $R_3^1, \ldots, R_3^q$ semuanya terdistribusi secara merata di $\{0,1\}^n$, tidak tergantung satu sama lain begitu juga dengan $L_3^1, \ldots, L_3^q$.

Ringkasnya: saat menanyakan $F^{(3)}$ (dengan fungsi bulat seragam) pada serangkaian q masukan berbeda, kecuali dengan probabilitas yang dapat diabaikan nilai keluaran $(L_3^1, R_3^1), \ldots, (L_3^q, R_3^q)$ didistribusikan sedemikian rupa sehingga $\{L_3^i\}$ adalah nilai n-bit yang seragam dan independen, namun berbeda, dan $\{R_3^i\}$ adalah nilai n-bit yang seragam dan independen. Sebaliknya, ketika menanyakan permutasi acak pada serangkaian q masukan berbeda, nilai keluaran $(L_3^1, R_3^1), \ldots, (L_3^q, R_3^q)$ adalah nilai $2n$-bit yang seragam dan independen, namun berbeda. Oleh karena itu, serangan pembeda terbaik untuk D adalah menebak bahwa ia berinteraksi dengan permutasi acak jika $L_3^i = L_3^j$ untuk beberapa i, j yang berbeda. Namun kejadian tersebut terjadi dengan probabilitas yang dapat diabaikan bahkan dalam kasus tersebut. Ini bisa dijadikan bukti formal.
$F^{(3)}$ ini bukanlah pseudorandom permutasi, saat anda diminta untuk mendemonstrasikannya di Soal 7.17. Untungnya, menambahkan keempat hasil pseudorandom permutasi yang kuat. Detailnya adalah sebagai Konstruksi 7.24.

THEOREM 7.25 Jika F adalah fungsi pseudorandom, maka konstruksi 7.24 adalah Pseudorandom permutasi yang kuatyang memetakan $2n$-bit dimasukkan ke $2n$-bit keluaran (dan menggunakan sebuah $4n$-bit kunci)

CONSTRUCTION 7.24

Let F be a keyed, length-preserving function. Define the keyed permutation $F^{(4)}$ as follows:

- **Inputs:** A key $k = (k_1, k_2, k_3, k_4)$ with $|k_i| = n$, and an input $x \in \{0,1\}^{2n}$ parsed as (L_0, R_0) with $|L_0| = |R_0| = n$.
- **Computation:**
  1. Compute $L_1 := R_0$ and $R_1 := L_0 \oplus F_{k_1}(R_0)$.
  2. Compute $L_2 := R_1$ and $R_2 := L_1 \oplus F_{k_2}(R_1)$.
  3. Compute $L_3 := R_2$ and $R_3 := L_2 \oplus F_{k_3}(R_2)$.
  4. Compute $L_4 := R_3$ and $R_4 := L_3 \oplus F_{k_4}(R_3)$.
  5. Output (L_4, R_4).

Permutasi pseudorandom yang kuat dari fungsi pseudorandom apa pun.

7.7 ASUMSI UNTUK KRIPTOGRAFI KUNCI PRIBADI

Kita telah menunjukkan bahwa (1) jika terdapat permutasi satu arah, maka terdapat generator pseudorandom; (2) jika terdapat generator pseudorandom, maka terdapat fungsi pseudorandom; dan (3) jika terdapat fungsi pseudorandom, maka terdapat permutasi pseudorandom (kuat). Meskipun kami tidak membuktikannya di sini, generator pseudorandom dapat dibuat dari fungsi satu arah. Dengan demikian kami memiliki teorema dasar berikut:

TEOREMA 7.26 Jika ada fungsi satu arah, maka generator pseudorandom, fungsi pseudorandom, dan permutasi pseudorandom kuat juga ada.

Semua skema kunci privat yang telah kita pelajari di Bab 3 dan 4 dapat dibangun dari generator/fungsi pseudorandom. Oleh karena itu kami memiliki:

TEOREMA 7.27 Jika ada fungsi satu arah, maka skema enkripsi kunci pribadi yang aman dengan CCA dan kode autentikasi pesan aman juga ada.

Artinya, fungsi satu arah sudah cukup untuk semua kriptografi kunci privat.
Di sini, kami menunjukkan bahwa fungsi satu arah juga diperlukan.

Pseudorandomness menyiratkan fungsi satu arah. Kita mulai dengan menunjukkan bahwa generator pseudorandom menyiratkan adanya fungsi satu arah:

PROPOSISI 7.28 Jika ada generator pseudorandom, maka fungsi satu arah juga ada.

BUKTI Misalkan G adalah generator pseudorandom dengan faktor muai $\ell(n) = 2n$. (Berdasarkan Teorema 7.20, kita mengetahui bahwa keberadaan generator pseudorandom mengimplikasikan keberadaan generator dengan faktor ekspansi ini.) Kita menunjukkan bahwa G sendiri adalah satu arah. Komputabilitas yang efisien sangatlah mudah (karena G dapat dihitung dalam waktu polinomial). Kami menunjukkan bahwa kemampuan untuk membalikkan G dapat diterjemahkan menjadi kemampuan untuk membedakan keluaran G dari seragam. Secara intuitif, hal ini berlaku karena kemampuan untuk membalikkan G menyiratkan kemampuan untuk menemukan benih yang digunakan oleh generator.

Misalkan $\mathcal{A}$ adalah algoritma waktu polinomial probabilistik sembarang. Kami menunjukkan bahwa $\Pr[\mathrm{Invert}_{\mathcal{A},G}(n) = 1]$ dapat diabaikan (lih. Definisi 7.1). Untuk melihatnya, perhatikan pembeda PPT berikut D: pada input string $w \in \{0,1\}^{2n}$, jalankan $\mathcal{A}(w)$ untuk mendapatkan output s. Jika $G(s) = w$ maka keluaran 1; jika tidak, keluaran 0.

Sekarang kita menganalisis perilaku D. Pertama-tama pertimbangkan probabilitas bahwa D menghasilkan 1 ketika string masukannya w seragam. Karena terdapat paling banyak 2^n nilai dalam rentang G (yaitu, nilai $\{G(s)\}_{s\in\{0,1\}^n}$), probabilitas bahwa w berada dalam rentang G paling banyak adalah $2^n/2^{2n} = 2^{-n}$. Jika w tidak berada dalam kisaran G, mustahil untuk $\mathcal{A}$ menghitung invers dari w sehingga D tidak mungkin menghasilkan keluaran 1. Kita simpulkan bahwa

$$\Pr_{w\leftarrow\{0,1\}^{2n}}[D(w) = 1] \leq 2^{-n}.$$

Di sisi lain, jika $w = G(s)$ untuk sebuah seed $s \in \{0,1\}^n$ dipilih secara acak secara seragam, maka, menurut definisi, $\mathcal{A}$ menghitung invers yang benar (sehingga D menghasilkan 1) dengan probabilitas yang sama persis dengan $\Pr[\mathrm{Invert}_{\mathcal{A},G}(n) = 1]$. Dengan demikian,

$$\left|\Pr_{w\leftarrow\{0,1\}^{2n}}[D(w) = 1] - \Pr_{s\leftarrow\{0,1\}^{n}}[D(G(s)) = 1]\right| \geq \Pr[\mathsf{Invert}_{\mathcal{A},G}(n) = 1] - 2^{-n}.$$

Karena G adalah generator pseudorandom, maka hal di atas harus diabaikan. Karena 2^{-n} dapat diabaikan, hal ini berarti $\Pr[\mathrm{Invert}_{\mathcal{A},G}(n) = 1]$ juga dapat diabaikan sehingga G bersifat satu arah.

Enkripsi kunci privat non-sepele menyiratkan fungsi satu arah. Proposisi 7.28 tidak berarti bahwa fungsi satu arah diperlukan untuk membangun skema enkripsi kunci pribadi yang aman, karena skema enkripsi kunci pribadi dapat dibangun tanpa bergantung pada generator pseudorandom. Selain itu, dimungkinkan untuk membuat skema enkripsi yang sangat rahasia (lihat Bab 2), selama teks biasa tidak lebih panjang dari kuncinya. Dengan demikian, bukti bahwa enkripsi kunci pribadi yang aman menyiratkan fungsi satu arah memerlukan perhatian lebih.

PROPOSISI 7.29 Jika terdapat skema enkripsi kunci privat aman EAV yang mengenkripsi pesan dua kali lebih panjang dari kuncinya, maka terdapat fungsi satu arah.

BUKTI Misalkan $\Pi = (\mathrm{Enc}, \mathrm{Dec})$ adalah skema enkripsi kunci pribadi yang mempunyai enkripsi yang tidak dapat dibedakan jika ada penyadap dan mengenkripsi pesan dengan panjang $2n$ ketika kunci mempunyai panjang n. (Untuk kesederhanaan, kita asumsikan bahwa kunci dipilih secara seragam.) Katakanlah ketika kunci n-bit digunakan, Enc menggunakan paling banyak $\ell(n)$ bit keacakan. Nyatakan enkripsi pesan m menggunakan kunci k dan keacakan r oleh $\mathrm{Enc}_k(m; r)$.

Tentukan fungsi berikut f :

$$f(k, m, r) \stackrel{\mathrm{def}}{=} \mathrm{Enc}_k(m; r) \parallel m,$$

dimana $|k| = n, |m| - 2n$, dan $|r| = \ell(n)$. Kita menyatakan bahwa f adalah fungsi satu arah. Jelas hal ini dapat dihitung secara efisien; kami menunjukkan bahwa sulit untuk membalikkannya. Membiarkan $\mathcal{A}$ menjadi algoritma PPT berubah-ubah, kami menunjukkan bahwa $\Pr[Invert_{\mathcal{A},f}(n) = 1]$ dapat diabaikan (lih. Definisi 7.1).

Pertimbangkan musuh waktu polinomial probabilistik berikut $\mathcal{A}'$ yang menyerang skema enkripsi kunci pribadi Π (yaitu, dalam eksperimen $\mathrm{PrivK}^{\mathrm{eav}}_{\Pi,\mathcal{A}'}(n)$):

Musuh $\mathcal{A}'(\mathbf{1^n})$

1. Pilih seragam $m_0, m_1 \leftarrow \{0,1\}^{2n}$ dan keluarkan. Menerima kembali tantangan ciphertext c.
2. Jalankan $\mathcal{A}(c \parallel m_0)$ untuk memperoleh (k', m', r'). Jika $f(k', m', r') = c \parallel m_0$, keluaran 0; jika tidak, keluaran 1.

Kami sekarang menganalisis perilaku $\mathcal{A}'$. Jika c merupakan enkripsi dari m_0, maka $c \parallel m_0$ terdistribusi persis seperti $f(k, m_0, r)$ untuk k, m_0, dan r yang seragam. Oleh karena itu, $\mathcal{A}$ menghasilkan invers valid dari $c \parallel m_0$(dan karenanya $\mathcal{A}'$ menghasilkan 0) dengan probabilitas yang sama persis dengan $\Pr[\mathrm{Invert}_{\mathcal{A},f}(n) = 1]$.

Sebaliknya, jika c merupakan enkripsi dari m_1 maka c tidak bergantung pada m_0. Untuk setiap nilai tetap dari tantangan ciphertext c, terdapat paling banyak 2^n pesan yang mungkin (satu untuk setiap kunci yang mungkin) yang dapat dikorespondensi oleh c. Karena m_0 adalah string $2n$-bit yang seragam, ini berarti kemungkinan terdapat beberapa kunci k yang $\mathrm{Dec}_k(c) = m_0$ paling banyak adalah $2^n/2^{2n} = 2^{-n}$. Hal ini memberikan batas atas probabilitas dimana $\mathcal{A}$ dapat menghasilkan invers valid dari $c \parallel m_0$ pada f , dan karenanya merupakan batas atas probabilitas $\mathcal{A}'$ menghasilkan 0 dalam kasus tersebut.
Dengan menggabungkan semua hal di atas, kita memiliki:

$$\begin{aligned}
&\Pr\left[\mathsf{PrivK}^{\mathsf{eav}}_{\Pi,\mathcal{A}'}(n) = 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[\mathcal{A}' \text{ outputs } 0 \mid b = 0\right] + \frac{1}{2} \cdot \Pr\left[\mathcal{A}' \text{ outputs } 1 \mid b = 1\right] \\
&\geq \frac{1}{2} \cdot \Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] + \frac{1}{2} \cdot \left(1 - 2^{-n}\right) \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[\mathsf{Invert}_{\mathcal{A},f}(n) = 1] - 2^{-n}\right) .
\end{aligned}$$

Keamanan Π berarti $\Pr\left[\mathrm{PrivK}^{\mathrm{eav}}_{\Pi,\mathcal{A}'}(n) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$ untuk beberapa fungsi yang dapat diabaikan negl. Hal ini, pada gilirannya, menyiratkan bahwa $\Pr\left[\mathrm{Invert}_{\mathcal{A},f}(n) = 1\right]$ dapat diabaikan, melengkapi bukti bahwa f adalah satu arah.

Kode otentikasi pesan menyiratkan fungsi satu arah. Benar juga bahwa kode otentikasi pesan yang memenuhi Definisi 4.2 menyiratkan adanya fungsi satu arah. Seperti dalam kasus enkripsi kunci pribadi, bukti dari fakta ini agak halus karena kode otentikasi pesan tanpa syarat memang ada ketika ada batasan apriori pada jumlah pesan yang akan diautentikasi. (Lihat Bagian 4.6.) Dengan demikian, pembuktian bergantung pada fakta bahwa Definisi 4.2 memerlukan keamanan bahkan ketika musuh melihat tag otentikasi dari sejumlah pesan yang berubah-ubah (polinomial). Buktinya agak rumit, jadi kami tidak memberikannya di sini.
Diskusi. Kami menyimpulkan bahwa keberadaan fungsi satu arah diperlukan dan cukup untuk semua kriptografi kunci privat (yang tidak sepele). Dengan kata lain, fungsi satu arah merupakan asumsi minimal jika menyangkut kriptografi kunci privat. Menariknya, hal ini tampaknya tidak terjadi pada fungsi hash dan enkripsi kunci publik, dimana fungsi satu arah diketahui diperlukan namun tidak diketahui (atau diyakini) cukup.

7.8 KETIDAKMAMPUAN UNTUK MEMBEDAKAN KOMPUTASI

Gagasan tentang ketidakmampuan komputasi untuk membedakan adalah inti dari teori kriptografi, dan hal ini mendasari banyak hal yang telah kita lihat di Bab 3 dan bab ini. Secara informal, dua distribusi probabilitas secara komputasi tidak dapat dibedakan jika tidak ada algoritma efisien yang dapat membedakannya (atau membedakannya). Secara lebih rinci, pertimbangkan dua distribusi X dan Y pada string dengan panjang tertentu ℓ; yaitu, X dan Y masing-masing memberikan beberapa probabilitas untuk setiap string dalam $\{0,1\}^{\ell}$. Ketika kita mengatakan bahwa beberapa algoritma D tidak dapat membedakan kedua distribusi ini,

yang kami maksud adalah bahwa D tidak dapat menentukan apakah ia diberikan sampel string berdasarkan distribusi X atau apakah diberi sampel string sesuai distribusi Y. Dengan kata lain, jika kita membayangkan D mengeluarkan "0" ketika ia yakin bahwa masukannya diambil sampelnya berdasarkan X dan mengeluarkan "1" jika ia mengira masukannya diambil berdasarkan Y, maka probabilitas bahwa D mengeluarkan "1" kira-kira sama dengan sama terlepas dari apakah D diberikan sampel dari X atau dari Y. Dengan kata lain, kita ingin

$$\left| \Pr_{s \leftarrow X}[D(s) = 1] - \Pr_{s \leftarrow Y}[D(s) = 1] \right|$$

menjadi kecil.

Hal ini mengingatkan kita pada cara kita mendefinisikan generator pseudorandom dan, tentu saja, kita akan segera mendefinisikan ulang secara formal gagasan generator pseudorandom menggunakan terminologi ini.

Definisi formal dari ketidakmampuan komputasi mengacu pada ansambel probabilitas, yang merupakan rangkaian distribusi probabilitas yang tak terbatas. (Formalisme ini diperlukan untuk pendekatan asimtotik yang bermakna.) Meskipun gagasan ini dapat digeneralisasikan, untuk tujuan kami, kami mempertimbangkan ansambel probabilitas di mana distribusi yang mendasarinya diindeks oleh bilangan asli. Jika untuk setiap bilangan asli n kita mempunyai distribusi X_n, maka $\mathcal{X} = \{\mathrm{X}_n\}_{n\in\mathbb{N}}$ adalah ansambel probabilitas. Seringkali $X_n = Y_{t(n)}$ untuk beberapa fungsi t, dalam hal ini kita menulis $\{Y_{t(n)}\}_{n\in\mathbb{N}}$ sebagai pengganti $\{\mathrm{X}_n\}_{n\in\mathbb{N}}$.

Kami hanya akan tertarik pada ansambel probabilitas yang dapat dijadikan sampel secara efisien. Ansambel $\mathrm{X} = \{\mathrm{X}_n\}_{n\in\mathbb{N}}$ dapat dijadikan sampel secara efisien jika terdapat algoritma waktu polinomial probabilistik S sehingga variabel acak S(1n) dan Xn terdistribusi secara identik. Artinya, algoritma S adalah cara yang efisien untuk mengambil sampel $\mathcal{X}$. Sekarang kita dapat mendefinisikan secara formal apa yang dimaksud dengan dua ansambel yang tidak dapat dibedakan secara komputasi.

DEFINISI 7.30 Dua ansambel probabilitas $\mathrm{X} = \{\mathrm{X}_n\}_{n\in\mathbb{N}}$ dan $\mathrm{Y} = \{Y_n\}_{n\in\mathbb{N}}$ tidak dapat dibedakan secara komputasi, dilambangkan dengan $\mathrm{X} \overset{c}{\equiv} \mathrm{Y}$, jika untuk setiap pembeda waktu polinomial probabilistik D terdapat fungsi negl yang dapat diabaikan sehingga:

$$\left| \Pr_{x \leftarrow X_n}[D(1^n, x) = 1] - \Pr_{y \leftarrow Y_n}[D(1^n, y) = 1] \right| \leq \mathsf{negl}(n).$$

Dalam definisinya, D diberikan input unary 1^n sehingga dapat berjalan dalam polinomial waktu dalam n. Hal ini penting bila keluaran X_n dan Y_n mungkin mempunyai panjang kurang dari n. Sebagai singkatan dalam ekspresi probabilitas, terkadang kita akan menulis X sebagai pengganti sampel acak dari distribusi X. Artinya, kita akan menulis $\Pr[D(1^n, X_n) = 1]$ sebagai pengganti $Pr_{x \leftarrow x_n}[D(1^n, x) = 1]$.

Hubungan ketidakmampuan komputasi bersifat transitif: jika $\mathrm{X} \stackrel{c}{\equiv} \mathrm{Y}$ dan $\mathrm{Y} \stackrel{c}{\equiv} \mathrm{Z}$, maka $\mathrm{X} \stackrel{c}{\equiv} \mathrm{Z}$.

Generator keacakan semu dan keacakan semu. Keacakan semu hanyalah kasus khusus ketidakmampuan komputasi untuk membedakannya. Untuk bilangan bulat ℓ apa pun, misalkan U_ℓ menyatakan distribusi seragam pada $\{0,1\}^\ell$. Kita dapat mendefinisikan generator pseudorandom sebagai berikut:

DEFINISI 7.31 Misalkan $\ell(\cdot)$ adalah suatu polinomial dan misalkan G adalah algoritma waktu polinomial (deterministik) yang mana untuk semua s dinyatakan bahwa $|G(s)| = \ell(|s|)$. Kita mengatakan bahwa G adalah generator pseudorandom jika dua kondisi berikut dipenuhi:

1. (Ekspansi :) Untuk setiap n dinyatakan bahwa $\ell(n) > n$.
2. (Keacakan semu :) Ansambel $\{G(U_n)\}_{n\in\mathbb{N}}$ secara komputasi tidak dapat dibedakan dari ansambel $\{U_{\ell(n)}\}_{n\in\mathbb{N}}$.

Banyak definisi dan asumsi lain dalam buku ini juga dapat dijadikan sebagai kasus khusus atau varian dari ketidakmampuan komputasi untuk membedakannya.

Beberapa sampel. Sebuah teorema penting mengenai ketidakterbedaan komputasi adalah bahwa banyak sampel secara polinomial dari ansambel yang tidak dapat dibedakan secara komputasi (yang dapat diambil sampelnya secara efisien) juga tidak dapat dibedakan secara komputasi.

TEOREMA 7.32 Misalkan X dan Y menjadi kumpulan probabilitas yang dapat dijadikan sampel secara efisien dan tidak dapat dibedakan secara komputasi. Kemudian, untuk setiap polinomial p, ansambel $\overline{\mathrm{X}} = \{X_n^{(1)}, \ldots, X_n^{(p(n))})\}_{n\in\mathbb{N}}$ secara komputasi tidak dapat dibedakan dari ansambel $\overline{\mathrm{Y}} = \{Y_n^{(1)}, \ldots, Y_n^{(p(n))})\}_{n\in\mathbb{N}}$.

Misalnya, G adalah generator pseudorandom dengan faktor ekspansi $2n$, sehingga ansambel $\{G(U_n)\}_{n\in\mathbb{N}}$ dan $\{U_{2n}\}_{n\in\mathbb{N}}$ secara komputasi tidak dapat dibedakan. Dalam pembuktian Teorema 7.22 kami menunjukkan bahwa untuk setiap polinomial t adalah ansambel

$$\{(\underbrace{G(U_n), \ldots, G(U_n)}_{t(n)})\}_{n\in\mathbb{N}} \quad \text{and} \quad \{(\underbrace{U_{2n}, \ldots, U_{2n}}_{t(n)})\}_{n\in\mathbb{N}}$$

juga tidak dapat dibedakan secara komputasi. Teorema 7.32 dibuktikan dengan argumen hibrid dengan cara yang persis sama.

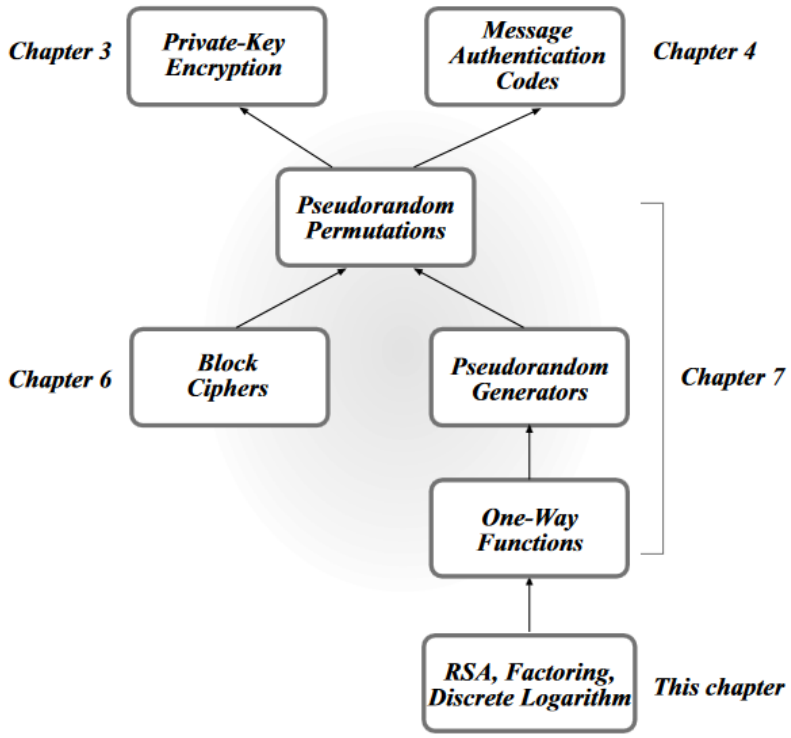
BAB 8
TEORI BILANGAN DAN ASUMSI KRIPTOGRAFI

Kriptosistem modern selalu didasarkan pada asumsi bahwa suatu masalah adalah hal yang sulit. Dalam Bab 3 dan 4, misalnya, kita melihat bahwa kriptografi kunci privat—baik skema enkripsi maupun kode autentikasi pesan—dapat didasarkan pada asumsi adanya permutasi pseudorandom (alias cipher blok). Ingat, secara kasar, ini berarti terdapat beberapa permutasi berkunci F yang sulit dibedakan dalam waktu polinomial antara interaksi dengan F_k (untuk kunci k yang seragam dan tidak diketahui) dan interaksi dengan permutasi yang benar-benar acak.

Sepintas lalu, asumsi adanya permutasi pseudorandom tampaknya cukup kuat dan tidak wajar, dan masuk akal untuk mempertanyakan apakah asumsi ini benar atau apakah ada bukti yang mendukungnya. Pada Bab 6 kita mengeksplorasi bagaimana block cipher dibangun dalam praktiknya. Fakta bahwa konstruksi yang ada tahan terhadap serangan merupakan indikasi bahwa keberadaan permutasi pseudorandom adalah masuk akal. Namun, mungkin sulit untuk percaya bahwa tidak ada serangan pembeda yang efisien terhadap cipher blok yang ada. Terlebih lagi, keadaan teori kita saat ini sedemikian rupa sehingga kita tidak tahu bagaimana membuktikan keacakan semu dari konstruksi praktis yang ada dibandingkan dengan asumsi yang "lebih sederhana" atau "lebih masuk akal". Secara keseluruhan, kondisi ini tidak sepenuhnya memuaskan.

Sebaliknya, sebagaimana disebutkan dalam Bab 3 (dan diselidiki secara rinci dalam Bab 7) adalah mungkin untuk membuktikan bahwa permutasi pseudorandom ada berdasarkan asumsi yang lebih ringan yaitu adanya fungsi satu arah. (Secara informal, suatu fungsi bersifat satu arah jika mudah dihitung tetapi sulit dibalik; lihat Bagian 8.4.) Namun, selain pembahasan singkat di Bagian 7.1, kita belum melihat contoh fungsi yang konkrit diyakini bersifat satu arah.

Salah satu tujuan bab ini adalah untuk memperkenalkan berbagai permasalahan yang diyakini "sulit", dan menyajikan dugaan fungsi satu arah berdasarkan permasalahan tersebut.1 Dengan demikian, bab ini dapat dipandang sebagai puncak dari pendekatan "top down" untuk mengatasi permasalahan tersebut. kriptografi kunci pribadi. (Lihat Gambar 8.1.) Artinya, pada Bab 3 dan 4 kita telah menunjukkan bahwa kriptografi kunci privat dapat didasarkan pada fungsi dan permutasi pseudorandom. Kita kemudian telah melihat bahwa yang terakhir ini dapat dipakai dalam praktek menggunakan cipher blok, seperti yang dibahas dalam Bab 6, atau dapat dibangun dengan cara yang ketat dari fungsi satu arah, seperti yang ditunjukkan dalam Bab 7. Di sini, kita mengambil satu langkah ini lebih jauh dan menunjukkan bagaimana fungsi satu arah dapat didasarkan pada permasalahan matematika sulit tertentu.



GAMBAR 8.1: Kriptografi kunci privat: pendekatan top-down.

Contoh-contoh yang kami jelajahi bersifat teori bilangan, dan oleh karena itu kami memulai dengan pengenalan singkat tentang teori bilangan dan teori grup. Karena kami juga tertarik pada masalah yang dapat diselesaikan secara efisien (bahkan fungsi satu arah harus mudah dihitung dalam satu arah, dan skema kriptografi harus mengakui algoritma yang efisien untuk pihak yang jujur), kami juga memulai studi algoritma. teori bilangan. Bahkan pembaca yang familiar dengan teori bilangan atau teori grup dianjurkan untuk membaca bab ini, karena aspek algoritmik biasanya diabaikan dalam pembahasan matematis murni pada topik ini.

Tujuan kedua dari bab ini adalah untuk mengembangkan materi yang dibutuhkan untuk kriptografi kunci publik, yang studinya akan kita mulai pada Bab 10. Menariknya, meskipun dalam pengaturan kunci privat terdapat konstruksi efisien dari primitif yang diperlukan (block cipher dan fungsi hash) tanpa menggunakan teori bilangan apa pun, dalam pengaturan kunci publik, semua konstruksi yang diketahui bergantung pada permasalahan teori bilangan yang sulit. Materi dalam bab ini berfungsi sebagai puncak dari apa yang telah kita pelajari sejauh ini sehubungan dengan kriptografi kunci privat, dan juga sebagai landasan bagi kriptografi kunci publik.

8.1 PENDAHULUAN DAN DASAR TEORI GRUP

Kita mulai dengan tinjauan bilangan prima dan aritmatika modular dasar. Bahkan pembaca yang telah melihat topik-topik ini sebelumnya harus membaca sekilas dua bagian berikutnya karena beberapa materi mungkin baru dan kami menyertakan bukti untuk sebagian besar hasil yang disebutkan.

Bilangan Prima dan Dapat Dibagi

Himpunan bilangan bulat dilambangkan dengan $\mathbb{Z}$. Untuk $a, b \in \mathbb{Z}$, kita katakan a membagi b, ditulis $a \mid b$, jika terdapat bilangan bulat c sehingga $ac = b$. Jika a tidak membagi b, kita tuliskan $a \nmid b$. (Kami terutama tertarik pada kasus di mana a, b, dan c semuanya positif, meskipun definisinya tetap masuk akal bahkan ketika satu atau lebih negatif atau nol.) Pengamatan sederhana adalah jika $a \mid b$ dan $a \mid c$ lalu $a \mid (X_b + Y_c)$ untuk sembarang $X, Y \in \mathbb{Z}$.

Jika $a \mid b$ dan a positif, kita sebut a sebagai pembagi b. Jika penjumlahan $a \notin \{1, b\}$ maka a disebut pembagi nontrivial, atau faktor, dari b. Bilangan bulat positif $p > 1$ adalah bilangan prima jika tidak mempunyai faktor; yaitu, ia hanya mempunyai dua pembagi: 1 dan dirinya sendiri. Bilangan bulat positif yang lebih besar dari 1 dan bukan bilangan prima disebut bilangan bulat komposit. Berdasarkan konvensi, bilangan 1 bukanlah bilangan prima atau komposit.

Teorema dasar aritmatika adalah bahwa setiap bilangan bulat yang lebih besar dari 1 dapat dinyatakan secara unik (hingga urutan) sebagai hasil kali bilangan prima. Artinya, bilangan bulat positif apa pun $N > 1$ dapat ditulis sebagai $N = \Pi_i p_i^{e_i}$, dengan $\{p_i\}$ adalah bilangan prima berbeda dan $e_i \geq 1$ untuk semua i; selanjutnya, $\{p_i\}$ (dan $\{e_i\}$) ditentukan secara unik hingga pemesanan.

Proses pembagian dengan sisa sudah kita kenal sejak sekolah dasar. Proposisi berikut memformalkan gagasan ini.

PROPOSISI 8.1 Misalkan a bilangan bulat dan b bilangan bulat positif. Lalu terdapat bilangan bulat unik q, r yang mana $a = qb + r$ dan $0 \leq r < b$.

Selanjutnya, bilangan bulat a dan b seperti pada proposisi memungkinkan untuk menghitung q dan r dalam waktu polinomial; (Waktu berjalan suatu algoritma diukur sebagai fungsi dari panjang masukannya. Poin penting dalam konteks teori bilangan algoritmik adalah bahwa masukan bilangan bulat selalu diasumsikan direpresentasikan dalam biner. Waktu berjalan dari algoritma yang mengambil bilangan bulat N sebagai masukan diukur dalam bentuk $\| N \|$, panjang representasi biner dari N. Perhatikan bahwa $\| N \| = \lfloor \log N \rfloor + 1.$) Pembagi persekutuan terbesar dari dua bilangan bulat a, b, ditulis $\gcd(a, b)$, adalah bilangan bulat terbesar c sehingga $c \mid a$ dan $c \mid b$. (Kita membiarkan gcd(0, 0) tidak terdefinisi.) Gagasan tentang pembagi persekutuan terbesar masuk akal jika salah satu atau kedua a, b bernilai negatif tetapi biasanya kita mempunyai $a, b \geq 1$; lagi pula, $\gcd(a, b) = \gcd(|a|, |b|)$.

Perhatikan bahwa $\gcd(b, 0) = \gcd(0, b) = b$; juga, jika p adalah bilangan prima maka $\gcd(a, p)$ sama dengan 1 atau p. Jika $\gcd(a, b) = 1$ dikatakan a dan b relatif prima.
Berikut ini adalah hasil yang bermanfaat:

PROPOSISI 8.2 Misalkan a, b bilangan bulat positif. Lalu terdapat bilangan bulat X, Y sehingga $X\ a + Y\ b = \gcd(a, b)$. Selanjutnya $\gcd(a, b)$ adalah bilangan bulat positif terkecil yang dapat dinyatakan dengan cara ini.

BUKTI Perhatikan himpunan $I \stackrel{\text{def}}{=} \{\hat{X}\, a + \hat{Y}\, b \mid \hat{X}, \hat{Y} \in \mathbb{Z}\}$. Perhatikan bahwa $a, b \in I$, dan I tentunya mengandung beberapa bilangan bulat positif. Misalkan d adalah bilangan bulat positif terkecil pada I. Kita tunjukkan bahwa $d = \gcd(a, b)$; karena d dapat ditulis sebagai $d = X\, a + Y\, b$ untuk beberapa $X, Y \in \mathbb{Z}$ (karena $d \in I$), ini membuktikan teorema tersebut.

Untuk menunjukkan hal ini, kita harus membuktikan bahwa $d \mid a$ dan $d \mid b$, dan bahwa d adalah bilangan bulat terbesar dengan sifat ini. Faktanya, kita dapat menunjukkan bahwa d membagi setiap elemen dalam I. Untuk melihatnya, ambil sembarang $c \in I$ dan tuliskan $c = X'a + Y'b$ dengan $X', Y' \in \mathbb{Z}$. Menggunakan pembagian dengan sisa (Proposisi 8.1) kita mendapatkan $c = qd + r$ dengan q, r bilangan bulat dan $0 \leq r < d$. Kemudian

$$r = c - qd = X'a + Y'b - q(Xa + Yb) = (X' - qX)a + (Y' - qY)b \in I.$$

Jika $r \neq 0$, ini bertentangan dengan pilihan kita untuk d sebagai bilangan bulat positif terkecil dalam I (karena $r < d$). Jadi, $r = 0$ dan karenanya $d \mid c$. Hal ini menunjukkan bahwa d membagi setiap elemen I.

Karena $a \in I$ dan $b \in I$, persamaan di atas menunjukkan bahwa $d \mid a$ dan $d \mid b$ sehingga d adalah pembagi persekutuan dari a dan b. Tetap menunjukkan bahwa ini adalah pembagi persekutuan terbesar. Asumsikan ada bilangan bulat $d' > d$ sehingga $d' \mid a$ dan $d' \mid b$. Kemudian berdasarkan pengamatan yang dilakukan sebelumnya, $d' \mid X\, a + Y\, b$. Karena yang terakhir sama dengan d, ini berarti $d' \mid d$. Namun hal ini tidak mungkin terjadi jika d' lebih besar dari d. Kita menyimpulkan bahwa d adalah bilangan bulat terbesar yang membagi a dan b, sehingga $d = \gcd(a, b)$.

Diberikan a dan b, algoritma Euclidean dapat digunakan untuk menghitung $\gcd(a, b)$ dalam waktu polinomial. Algoritma Euclidean yang diperluas juga dapat digunakan untuk menghitung X, Y (seperti pada proposisi di atas) dalam waktu polinomial. Lihat Lampiran B.1.2 untuk rinciannya.

PROPOSISI 8.3 Jika $c \mid ab$ dan $\gcd(a, c) = 1$, maka $c \mid b$. Jadi, jika p adalah bilangan prima dan $p \mid ab$ lalu $p \mid a$ atau $p \mid b$.

BUKTI Karena $c \mid ab$ kita mempunyai $\gamma c = ab$ untuk suatu bilangan bulat γ. Jika $\gcd(a, c) = 1$ maka berdasarkan proposisi sebelumnya diketahui terdapat bilangan bulat X, Y sehingga $1 = X\, a + Y\, c$. Mengalikan kedua ruas dengan b, kita peroleh

$$b = Xab + Ycb = X\gamma c + Ycb = c \cdot (X\gamma + Yb).$$

Karena $(X\gamma + Yb)$ bilangan bulat, maka $c \mid b$.
Bagian kedua dari proposisi ini mengikuti fakta bahwa jika $p \nmid a$ maka $\gcd(a, p) = 1$.

PROPOSISI 8.4 Jika $a \mid N$, $b \mid N$, dan $\gcd(a, b) = 1$, maka $ab \mid N$.

BUKTI Tulislah $ac = N, bd = N$, dan (menggunakan Proposisi 8.2) $1 = X\,a + Y\,b$, dimana c, d, X, Y semuanya bilangan bulat. Mengalikan kedua ruas persamaan terakhir dengan N kita peroleh

$$N = XaN + YbN = Xabd + Ybac = ab(Xd + Yc),$$

menunjukkan bahwa $ab \mid N$.

Aritmatika Modular

Misal $a, b, N \in \mathbb{Z}$ dengan $N > 1$. Kita menggunakan notasi $[a \bmod N]$ untuk menyatakan sisa a setelah pembagian dengan N. Secara lebih rinci: berdasarkan Proposisi 8.1 terdapat q, r unik dengan $a = qN + r$ dan $0 \leq r < N$, dan kita definisikan $[a \bmod N\,]$ sama dengan r ini. Oleh karena itu, perhatikan bahwa $0 \leq [a\ mod\ N\,] < N$. Kami menyebut proses pemetaan a ke $[a \bmod N\,]$ sebagai modulo reduksi N.

Kita katakan bahwa a dan b adalah modulo kongruen N, ditulis $a = b \bmod N$, jika $[a \bmod N\,] = [b \bmod N\,]$, yaitu jika sisa a dibagi N sama dengan sisa b dibagi dengan N. Perhatikan bahwa $a = b \bmod N$ jika dan hanya jika $N \mid (a - b)$. Melalui notasi, dalam ekspresi seperti

$$a = b = c = \cdots = z \bmod N,$$

pemahamannya adalah bahwa setiap tanda sama dengan dalam barisan ini (dan bukan hanya yang terakhir) mengacu pada modulo kongruensi N.

Perhatikan bahwa $a = [b \bmod N]$ menyiratkan $a = b\ mod\ N$, tetapi tidak sebaliknya. Misalnya, 36 = 21 mod 15 tetapi $36 \neq [21 \bmod 15] = 6$. (Sebaliknya, $[a\ mod\ N] = [b\ mod\ N]$ jika dan hanya jika $a = b\ mod\ N$.)

Modulo kongruensi N merupakan relasi ekivalensi: yakni bersifat refleksif ($a = a\ mod\ N$ untuk semua a), simetris ($a = b\ mod\ N$ berarti $b = a\ mod\ N$), dan transitif (jika $a = b\ mod\ N$ dan $b = c\ mod\ N$, lalu $a = c\ mod\ N$). Modulo Kongruensi N juga mematuhi aturan standar aritmatika sehubungan dengan penjumlahan, pengurangan, dan perkalian; jadi, misalnya, jika $a = a'\ mod\ N$ dan $b = b'\ mod\ N$ maka $(a + b) = (a' + b') \bmod N$ dan $ab = a'b'\ mod\ N$. Konsekuensinya adalah bahwa kita dapat "mengurangi lalu menambah/mengalikan" daripada harus "menambah/mengalikan lalu mengurangi", yang seringkali dapat menyederhanakan penghitungan.

Contoh 8.5

Mari kita hitung $[1093028 \cdot 190301 \bmod 100]$. Karena 1093028 = 28 mod 100 dan 190301 = 1 mod 100, kita punya

$$\begin{aligned} 1093028 \cdot 190301 &= [1093028 \bmod 100] \cdot [190301 \bmod 100] \bmod 100 \\ &= 28 \cdot 1 = 28 \bmod 100. \end{aligned}$$

Cara alternatif untuk menghitung jawabannya $1093028 \cdot 190301$ lalu dikurangi hasilnya modulo 100) kurang efisien. ♦

Modulo kongruensi N (secara umum) tidak menghormati pembagian. Artinya, jika $a = a' \bmod N$ dan $b = b' \bmod N$ maka belum tentu benar bahwa $a/b = a'/b' \bmod N$; pada kenyataannya, ungkapan "$a/b \bmod N$" tidak selalu terdefinisi dengan baik. Sebagai contoh spesifik yang sering menimbulkan kebingungan, $ab = cb \bmod N$ tidak serta merta berarti bahwa $a = c \bmod N$.

Contoh 8.6

Ambil $N = 24$. Maka $3 \cdot 2 = 6 = 15 \cdot 2 \bmod 24$, tetapi $3 \neq 15 \bmod 24$. ♦

Namun, dalam kasus tertentu, kita dapat mendefinisikan pengertian pembagian yang bermakna. Jika untuk suatu bilangan bulat b terdapat bilangan bulat c sehingga $bc = 1\ mod\ N$, kita katakan bahwa b adalah modulo N yang dapat dibalik dan memanggil c *a* (perkalian) invers dari b modulo N. Jelasnya, 0 tidak pernah bisa dibalik. Juga tidak sulit untuk menunjukkan bahwa jika c adalah invers perkalian dari b modulo N maka $[c \bmod N]$ juga demikian. Selanjutnya, jika c' merupakan invers perkalian lain dari b maka $[c \bmod N] = [c' \bmod N]$. Jika b dapat dibalik, maka kita dapat dengan mudah membiarkan b^{-1} menunjukkan invers perkalian unik dari b yang terletak pada rentang $\{1, \ldots, N-1\}$.

Ketika b adalah modulo N yang dapat dibalik, kita mendefinisikan pembagian dengan b modulo N sebagai perkalian dengan b^{-1} (yaitu, kita mendefinisikan $[a/b \bmod N] \stackrel{\text{def}}{=} [ab^{-1} \bmod N]$). Kami menekankan bahwa pembagian dengan b hanya didefinisikan jika b dapat dibalik. Jika $ab = cb \bmod N$ dan b dapat dibalik, maka kita dapat membagi setiap ruas persamaan dengan b (atau, kalikan setiap ruas dengan b^{-1}) untuk mendapatkan

$$(ab) \cdot b^{-1} = (cb) \cdot b^{-1} \bmod N \quad \Rightarrow \quad a = c \bmod N.$$

Kami melihat bahwa dalam kasus ini, pembagian berjalan "seperti yang diharapkan". Oleh karena itu, bilangan bulat yang dapat dibalik modulo N "lebih bagus" untuk digunakan, dalam beberapa hal.

Pertanyaan wajarnya adalah: bilangan bulat manakah yang modulonya dapat dibalikkan pada modulus N tertentu? Kita dapat menjawab pertanyaan ini sepenuhnya dengan menggunakan Proposisi 8.2:

PROPOSISI 8.7 Misalkan b, N bilangan bulat, dengan $b \geq 1$ dan $N > 1$. Maka b adalah modulo N yang dapat dibalik jika dan hanya jika $\gcd(b, N) = 1$.

BUKTI Asumsikan b adalah modulo N yang dapat dibalik, dan misalkan c menyatakan kebalikannya. Karena $bc = 1 \bmod N$, hal ini menyiratkan bahwa $bc - 1 = \gamma N$ untuk beberapa $\gamma \in \mathbb{Z}$. Sama halnya, $bc - \gamma N = 1$. Karena, berdasarkan Proposisi 8.2, $\gcd(b, N)$ adalah bilangan bulat positif terkecil yang dapat dinyatakan dalam dengan cara ini, dan tidak ada bilangan bulat positif yang lebih kecil dari 1, ini berarti $\gcd(b, N) = 1$.

Sebaliknya, jika $\gcd(b, N) = 1$ maka berdasarkan Proposisi 8.2 terdapat bilangan bulat X, Y sehingga $X\,b + Y\,N = 1$. Mengurangi setiap ruas persamaan ini modulo N menghasilkan $X\,b = 1 \bmod N$, dan kita melihat bahwa X adalah kebalikan perkalian dari b. (Faktanya, ini memberikan algoritma yang efisien untuk menghitung invers.)

Contoh 8.8

Misal $b = 11$ dan $N = 17$. Maka $(-3) \cdot 11 + 2 \cdot 17 = 1$, sehingga $14 = [-3 \bmod 17]$ adalah kebalikan dari 11. Dapat dibuktikan bahwa $14 \cdot 11 = 1 \bmod 17$.
Penjumlahan, pengurangan, perkalian, dan penghitungan invers (jika ada) modulo N semuanya dapat dilakukan dalam waktu polinomial; lihat Lampiran B.2. Eksponensial (yaitu, menghitung $[a^b \; mod \; N]$ untuk $b > 0$ bilangan bulat) juga dapat dihitung dalam waktu polinomial; lihat Lampiran B.2.3.

Grup

Misalkan $\mathbb{G}$ adalah himpunan. Operasi biner $\circ$ pada $\mathbb{G}$ hanyalah sebuah fungsi $\circ(\cdot,\cdot)$ yang mengambil dua elemen dari $\mathbb{G}$ sebagai masukan. Jika $g, h \in \mathbb{G}$ maka alih-alih menggunakan notasi rumit $\circ(g, h)$, kita menulis $g \circ h$.
Kami sekarang memperkenalkan gagasan penting tentang kelompok.

DEFINISI 8.9 Grup adalah himpunan $\mathbb{G}$ dengan operasi biner $\circ$ yang memenuhi kondisi berikut:

- (Penutupan :) Untuk semua $g, h \in \mathbb{G}, g \circ h \in \mathbb{G}$.
- (Keberadaan identitas :) Terdapat identitas $e \in \mathbb{G}$ sehingga untuk semua $g \in \mathbb{G}, e \circ g = g = g \circ e$.
- (Keberadaan invers :) Untuk semua $g \in \mathbb{G}$ terdapat elemen $h \in \mathbb{G}$
- sehingga $g \circ h = e = h \circ g$. h yang demikian disebut invers dari g.
- (Asosiasi :) Untuk semua $g_1, g_2, g_3 \in \mathbb{G}, (g_1 \circ g_2) \circ g_3 = g_1 \circ (g_2 \circ g_3)$.
  Jika $\mathbb{G}$ mempunyai jumlah elemen berhingga, kita katakan $\mathbb{G}$ berhingga dan misalkan $|\mathbb{G}|$ menyatakan orde golongannya (yaitu, jumlah elemen dalam $\mathbb{G}$).
- Grup $\mathbb{G}$ dengan operasi $\circ$ disebut abelian jika memenuhi:
- (Komutatifitas :) Untuk semua $g, h \in \mathbb{G}, g \circ h = h \circ g$.

Ketika operasi biner dipahami, kita cukup menyebut himpunan $\mathbb{G}$ sebagai grup.
Kita akan selalu berurusan dengan grup abelian yang terbatas. Namun, kami akan berhati-hati dalam menentukan kapan suatu hasil memerlukan asumsi tersebut.

Asosiatif menyiratkan bahwa kita tidak perlu menyertakan tanda kurung ketika menulis ekspresi panjang; yaitu, notasi $g_1 \circ g_2 \circ \ldots \circ g_n$ tidak ambigu karena tidak masalah dalam urutan apa kita mengevaluasi operasi tersebut $\circ$.

Kita dapat menunjukkan bahwa elemen identitas dalam suatu grup $\mathbb{G}$ adalah unik, sehingga kita dapat mengacu pada identitas suatu grup. Dapat juga ditunjukkan bahwa setiap elemen g suatu grup mempunyai invers yang unik. Lihat Latihan 8.1.

Jika $\mathbb{G}$ adalah suatu grup, maka himpunan $\mathbb{H} \subseteq \mathbb{G}$ adalah subgrup dari $\mathbb{G}$ jika $\mathbb{H}$ sendiri membentuk grup melalui operasi yang sama dengan $\mathbb{G}$. Untuk memeriksa apakah $\mathbb{H}$ adalah subgrup, kita perlu memverifikasi penutupan, keberadaan identitas dan invers, serta asosiatif.

sesuai Definisi 8.9. (Faktanya, asosiatif—serta komutatifitas jika $\mathbb{G}$ adalah abelian—diwarisi secara otomatis dari $\mathbb{G}$.) Setiap grup $\mathbb{G}$ selalu memiliki subgrup sepele $\mathbb{G}$ dan {1}. Kita menyebut $\mathbb{H}$ sebagai subgrup ketat dari $\mathbb{G}$ jika $\mathbb{H} \neq \mathbb{G}$.

Secara umum, kami tidak akan menggunakan notasi $\circ$ untuk menunjukkan operasi grup. Sebagai gantinya, kita akan menggunakan notasi tambahan atau notasi perkalian tergantung pada kelompok yang didiskusikan. Ini tidak berarti bahwa operasi grup berhubungan dengan penjumlahan atau perkalian bilangan bulat; itu hanyalah notasi yang berguna. Saat menggunakan notasi aditif, operasi grup diterapkan pada dua elemen g, h dilambangkan $g + h$; identitas dilambangkan dengan 0; kebalikan dari suatu elemen g dilambangkan dengan g; dan kita menulis $h - g$ sebagai pengganti $h + (-g)$. Saat menggunakan notasi perkalian, operasi grup yang diterapkan pada g, h dilambangkan dengan $g \cdot h$ atau sederhananya gh; identitas dilambangkan dengan 1; kebalikan dari suatu elemen g dilambangkan dengan g^{-1}; dan terkadang kami menulis h/g sebagai pengganti hg^{-1}.
Pada titik ini, mungkin bermanfaat untuk melihat beberapa contoh.

Contoh 8.10

Suatu himpunan dapat berupa suatu grup dalam suatu operasi, tetapi bukan grup lainnya. Misalnya, himpunan bilangan bulat $\mathbb{Z}$ adalah grup abelian yang dijumlahkan: identitasnya adalah elemen 0, dan setiap bilangan bulat g memiliki invers $-g$. Di sisi lain, ini bukan grup dalam perkalian karena, misalnya, bilangan bulat 2 tidak memiliki invers perkalian pada bilangan bulat tersebut. ♦

Contoh 8.11

Himpunan bilangan real $\mathbb{R}$ bukan merupakan grup perkalian, karena 0 tidak mempunyai invers perkalian. Himpunan bilangan real bukan nol merupakan grup abelian yang dikalikan dengan identitas 1. ♦

Contoh berikut memperkenalkan grup $\mathbb{Z}_N$ yang akan sering kita gunakan.

Contoh 8.12

Misalkan $N > 1$ bilangan bulat. Himpunan $\{0, \ldots, N — 1\}$ terhadap modulo penjumlahan N (yaitu, di mana $a + b \stackrel{\text{def}}{=} [a + b \bmod N]$) adalah grup abelian berorde N. Penutupan sudah jelas; asosiatif dan komutatifitas mengikuti fakta bahwa bilangan bulat memenuhi sifat-sifat ini; identitasnya adalah 0; dan, karena $a + (N - a) = 0 \bmod N$, maka invers dari setiap elemen a adalah $[(N - a) \bmod N]$. Kami menyatakan grup ini dengan $\mathbb{Z}_N$. (Terkadang kita juga menggunakan $\mathbb{Z}_N$ untuk menyatakan himpunan $\{0, \ldots, N — 1\}$ tanpa memperhatikan operasi grup tertentu.) ♦

Kami mengakhiri bagian ini dengan lemma mudah yang meresmikan “undang-undang pembatalan” untuk kelompok.

LEMMA 8.13 Misalkan $\mathbb{G}$ suatu grup dan $a, b, c \in \mathbb{G}$. Jika $ac = bc$, maka $a = b$. Khususnya, jika $ac = c$ maka a adalah identitas di $\mathbb{G}$.

BUKTI Kita mengetahui $ac = bc$. Mengalikan kedua ruas dengan invers unik c^{-1} dari c, kita memperoleh $a = b$. Secara terperinci:

$$ac = bc \;\Rightarrow\; (ac)c^{-1} = (bc)\cdot c^{-1} \;\Rightarrow\; a(cc^{-1}) = b(cc^{-1}) \;\Rightarrow\; a\cdot 1 = b\cdot 1 \;\Rightarrow a = b.$$

Bandingkan bukti di atas dengan pembahasan (Proposisi 8.7 sebelumnya) mengenai hukum pembatalan pembagian modulo N. Seperti yang ditunjukkan oleh kesamaannya, elemen-elemen yang dapat dibalik modulo N membentuk grup di bawah modulo perkalian N. Kami akan segera kembali ke contoh ini secara lebih rinci.

Eksponen Grup

Seringkali berguna untuk mendeskripsikan operasi grup yang diterapkan m kali pada elemen tetap g, di mana m adalah bilangan bulat positif. Saat menggunakan notasi aditif, kami menyatakannya sebagai $m \cdot g$ atau mg; itu adalah,

$$mg = m \cdot g \stackrel{\text{def}}{=} \underbrace{g + \cdots + g}_{m \text{ times}}.$$

Perhatikan bahwa m adalah bilangan bulat, sedangkan g adalah elemen grup. Jadi mg tidak mewakili operasi grup yang diterapkan pada m dan g (sebenarnya, kita bekerja dalam grup yang operasi grupnya ditulis secara aditif). Untungnya, notasi tersebut "berperilaku sebagaimana mestinya"; jadi, misalnya, jika $g \in \mathbb{G}$ dan m, m' adalah bilangan bulat maka $(mg) + (m'g) = (m + m')g, m(m'g) = (mm')g,$ dan $1 \cdot g = g$. Dalam grup abelian $\mathbb{G}$ dengan $g, h \in \mathbb{G}, (mg) + (mh) = m(g + h)$.

Saat menggunakan notasi perkalian, kami menyatakan penerapan operasi grup m kali pada elemen g kali g^m. Itu adalah,

$$g^m \stackrel{\text{def}}{=} \underbrace{g \cdots g}_{m \text{ times}}.$$

Aturan eksponensial yang lazim berlaku: $g^m \cdot g^{m'} = g^{m+m'}, (g^m)^{m'} = g^{mm'}$, dan $g^1 = g$. Juga, jika $\mathbb{G}$ adalah grup abelian dan $g, h \in \mathbb{G}$ maka $g^m \cdot h^m = (gh)^m$.

Semua ini hanyalah "penerjemahan" hasil dari paragraf sebelumnya ke dalam susunan kelompok yang ditulis secara perkalian, bukan penjumlahan.

Notasi di atas diperluas secara alami untuk kasus ketika m adalah nol atau bilangan bulat negatif. Saat menggunakan notasi aditif, kita mendefinisikan $0 \cdot g \stackrel{\text{def}}{=} 0$ dan $(-m) \cdot g \stackrel{\text{def}}{=} m \cdot (-g)$ untuk m bilangan bulat positif. (Perhatikan bahwa dalam persamaan $'0 \cdot g = 0'$, angka 0 di ruas kiri adalah bilangan bulat 0 sedangkan 0 di ruas kanan adalah elemen identitas dalam grup.) Perhatikan bahwa $-g$ adalah kebalikannya dari g dan, seperti yang diharapkan, $(-m) \cdot g = -(mg)$. Saat menggunakan notasi perkalian, $g^0 \stackrel{\text{def}}{=} 1$ dan $g^{-m} \stackrel{\text{def}}{=} (g^{-1})^m$. Sekali lagi, g^{-1} adalah kebalikan dari g, dan kita mendapatkan $g^{-m} = (g^m)^{-1}$.

Misalkan $g \in \mathbb{G}$ dan $b \geq 0$ bilangan bulat. Kemudian eksponensial g^b dapat dihitung menggunakan bilangan polinomial dari operasi grup yang mendasarinya di $\mathbb{G}$. Jadi, jika operasi grup dapat dihitung dalam waktu polinomial maka eksponensial juga dapat dihitung.

Kami sekarang memiliki cukup pengetahuan untuk membuktikan hasil luar biasa berikut ini:

TEOREMA 8.14 Misalkan $\mathbb{G}$ suatu grup berhingga dengan $m = |\mathbb{G}|$, orde grup tersebut. Maka untuk setiap elemen $g \in \mathbb{G}, g^m = 1$.

BUKTI Kita membuktikan teorema hanya jika $\mathbb{G}$ adalah abelian (walaupun berlaku untuk grup berhingga mana pun). Perbaiki sembarang $g \in \mathbb{G}$, dan biarkan $g_1, \ldots, g_m$ menjadi elemen $\mathbb{G}$. Kami mengklaim itu

$$g_1 \cdot g_2 \cdots g_m = (gg_1) \cdot (gg_2) \cdots (gg_m).$$

Untuk melihat hal ini, perhatikan bahwa $gg_i = gg_j$ menyiratkan $g_i = g_j$ oleh Lemma 8.13. Jadi masing-masing elemen m dalam tanda kurung di sebelah kanan berbeda. Karena terdapat tepat m elemen di $\mathbb{G}$, maka m elemen yang dikalikan di ruas kanan adalah seluruh elemen $\mathbb{G}$ dalam urutan permutasi tertentu. Karena $\mathbb{G}$ adalah abelian, urutan perkalian elemen tidak menjadi masalah, sehingga ruas kanan sama dengan ruas kiri.

Sekali lagi dengan menggunakan fakta bahwa $\mathbb{G}$ adalah abelian, kita dapat "menarik" semua kemunculan g dan memperolehnya

$$g_1 \cdot g_2 \cdots g_m = (gg_1) \cdot (gg_2) \cdots (gg_m) = g^m \cdot (g_1 \cdot g_2 \cdots g_m).$$

Sekali lagi mengacu pada Lemma 8.13, ini berarti $g^m = 1$.

Akibat wajar yang penting dari hal di atas adalah kita dapat mengerjakan "modulo urutan grup" dalam eksponen:

KORELARI 8.15 Misalkan $\mathbb{G}$ suatu grup berhingga dengan $m = |\mathbb{G}| > 1$. Maka untuk sembarang $g \in \mathbb{G}$ dan sembarang bilangan bulat x, kita mempunyai $g^x = g^{[x \bmod m]}$.

BUKTI Katakanlah $x = qm + r$, dimana q, r adalah bilangan bulat dan $r = [x \bmod m]$. Kemudian

$$g^x = g^{qm+r} = g^{qm} \cdot g^r = (g^m)^q \cdot g^r = 1^q \cdot g^r = g^r$$

(menggunakan Teorema 8.14), seperti yang diklaim.

Contoh 8.16

Ditulis secara aditif, akibat wajar di atas menyatakan bahwa jika g adalah suatu unsur dalam kelompok berorde m, maka $x \cdot g = [x \bmod m] \cdot g$. Sebagai contoh, perhatikan grup $\mathbb{Z}_{15}$ berorde $m = 15$, dan ambil $g = 11$. Akibat wajarnya adalah

$$152 \cdot 11 = [152 \bmod 15] \cdot 11 = 2 \cdot 11 = 11 + 11 = 22 = 7 \bmod 15.$$

Pernyataan di atas sesuai dengan fakta (lih. Contoh 8.5) bahwa kita dapat "mengurangi lalu mengalikan" daripada harus "menggandakan lalu mengurangi". ♦
Akibat wajar lainnya yang akan sangat berguna untuk aplikasi kriptografi adalah sebagai berikut:

KORELARI 8.17 Misalkan $\mathbb{G}$ suatu grup berhingga dengan $m = |\mathbb{G}| > 1$. Misalkan $e > 0$ bilangan bulat, dan definisikan fungsinya $f_e: \mathbb{G} \to \mathbb{G}$ dengan $f_e(g) = g^e$. Jika $\gcd(e, m) = 1$, maka f_e adalah permutasi (yaitu bijeksi). Apalagi jika $d = e^{-1} \bmod m$ maka f_d adalah kebalikan dari f_e. (Catatan dari Proposisi 8.7, $\gcd(e, m) = 1$ menyiratkan e adalah modulo m yang dapat dibalik.)

BUKTI Karena $\mathbb{G}$ terbatas, bagian kedua dari klaim tersebut menyiratkan bagian pertama; jadi, kita hanya perlu menunjukkan bahwa f_d adalah kebalikan dari f_e. Hal ini benar karena untuk setiap $g \in \mathbb{G}$ yang kita miliki

$$f_d\left(f_e(g)\right) = f_d(g^e) = (g^e)^d = g^{ed} = g^{[ed \bmod m]} = g^1 = g,$$

dimana persamaan keempat mengikuti Akibat Akibat 8.15.

Golongan $\mathbb{Z}*_N$

Seperti yang dibahas pada Contoh 8.12, himpunan $\mathbb{Z}_N = \{0, \ldots, N-1\}$ adalah grup di bawah modul penjumlahan N. Bisakah kita mendefinisikan grup sehubungan dengan modulo perkalian N? Dengan melakukan hal ini, kita harus menghilangkan elemen-elemen di $\mathbb{Z}_N$ yang tidak dapat dibalik; misalnya, kita harus menghilangkan 0 karena tidak memiliki invers perkalian. Unsur-unsur yang bukan nol mungkin juga gagal untuk dapat dibalik (lih. Proposisi 8.7). Unsur manakah $b \in \{1, \ldots, N-1\}$ adalah modulo N yang dapat dibalik? Proposisi 8.7 mengatakan bahwa ini adalah elemen b yang $\gcd(b, N) = 1$. Kita punya juga terlihat di Bagian 8.1 bahwa setiap kali b dapat dibalik, ia mempunyai invers yang terletak pada rentang $\{1, \ldots, N — 1\}$. Hal ini mengarahkan kita untuk mendefinisikan, untuk sembarang $N > 1$, himpunan tersebut

$$\mathbb{Z}_N^* \stackrel{\text{def}}{=} \{b \in \{1, \ldots, N-1\} \mid \gcd(b, N) = 1\};$$

yaitu, $\mathbb{Z}*_N$ terdiri dari bilangan bulat dalam himpunan $\{1, \ldots, N-1\}$ yang relatif prima terhadap N. Operasi grupnya adalah perkalian modulo N; yaitu, $ab \stackrel{\text{def}}{=} [ab \bmod N]$. Kami mengklaim bahwa $\mathbb{Z}*_N$ adalah grup abelian sehubungan dengan operasi ini. Karena 1 selalu dalam $\mathbb{Z}*_N$, himpunan tersebut jelas mengandung elemen identitas. Pembahasan di atas menunjukkan bahwa setiap elemen pada $\mathbb{Z}*_N$ mempunyai invers perkalian pada himpunan yang sama. Komutatifitas dan asosiatif mengikuti fakta bahwa sifat-sifat ini berlaku pada bilangan bulat. Untuk menunjukkan bahwa penutupan berlaku, misalkan $a, b \in \mathbb{Z}*_N$; maka

$[ab \bmod N]$ memiliki invers $[b^{-1}a^{-1} - 1 \bmod N]$, yang berarti $\gcd([ab \bmod N], N) = 1$ dan seterusnya $ab \in \mathbb{Z}*_N$. Meringkas:

PROPOSISI 8.18 Misalkan $N > 1$ bilangan bulat. Maka $\mathbb{Z}*_N$ adalah grup abelian pada modul perkalian N. Tentukan $\emptyset(N) \stackrel{\text{def}}{=} |\mathbb{Z}*_N|$, orde grup $\mathbb{Z}*_N$. ($\emptyset$ disebut fifungsi Euler.) Berapa nilai $\emptyset(N)$? Pertama-tama perhatikan kasus ketika $N = p$ adalah bilangan prima. Lalu semua elemen di $\{1, \ldots, p-1\}$ relatif prima terhadap p, sehingga $\emptyset(p) = |\mathbb{Z}*_p| = p - 1$. Selanjutnya perhatikan kasus $N = pq$, dimana p, q adalah bilangan prima berbeda. Jika bilangan bulat $a \in \{1, \ldots, N-1\}$ tidak relatif prima terhadap N, maka $p \mid a$ atau $q \mid a$ (a tidak habis dibagi p dan q karena ini berarti $pq \mid a$ tetapi $a < N = pq$). Elemen dalam $\{1, \ldots, N-1\}$ habis dibagi p adalah $(q-1)$ unsur-unsur $p, 2p, 3p, \ldots, (q-1)p$, dan unsur-unsur yang habis dibagi q sama persis dengan $(p-1)$ unsur $p, 2p, \ldots, (q-1)q$. Oleh karena itu, jumlah elemen yang tersisa (yaitu elemen yang tidak habis dibagi p atau q) diberikan oleh

$$(N-1) - (q-1) - (p-1) = pq - p - q + 1 = (p-1)(q-1).$$

Kita telah membuktikan bahwa $\emptyset(N) = (p-1)(q-1)$ ketika N adalah hasil kali dua bilangan prima berbeda p dan q. Anda diminta untuk membuktikan hasil umum berikut (jarang digunakan di sisa buku ini) pada Latihan 8.4:

TEOREMA 8.19 Misalkan $N = \Pi_i p_i^{e_i}$, dimana $\{p_i\}$ adalah bilangan prima berbeda dan $e_i \geq 1$. Maka $\emptyset(N) = \Pi_i p_i^{e_i - 1}(p_i - 1)$.

Contoh 8.20
Ambil $N = 15 = 5 \cdot 3$. Maka $\mathbb{Z}*_{15} = \{1,2,4,7,8,11,13.14\}$ dan $|\mathbb{Z}*_{15}| = 8 = 4 \cdot 2 = \emptyset(15)$. Kebalikan dari 8 pada $\mathbb{Z}*_{15}$ adalah 2, karena $8 \cdot 2 = 16 = 1 \bmod 15$.

Kita telah menunjukkan bahwa $\mathbb{Z}_N^*$ adalah grup berorde $\phi(N)$. Berikut ini adalah akibat wajar dari Teorema 8.14 dan Akibat Akibat 8.17:

KORELARI 8.21 Ambil bilangan bulat sembarang $N > 1$ dan $a \in \mathbb{Z}_N^*$. Kemudian

$$a^{\phi(N)} = 1 \bmod N.$$

Untuk kasus tertentu dimana $N = p$ adalah bilangan prima dan $a \in \{1, \ldots, p—1\}$, sudah

$$a^{p-1} = 1 \bmod p.$$

KORELARI 8.22 Perbaiki $N > 1$. Untuk bilangan bulat $e > 0$ tentukan $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$

dengan $f_e(x) = [x^e \bmod N]$. Jika e relatif prima terhadap $\phi(N)$ maka f_e merupakan permutasi. Selain itu, jika $d = e^{-1} \bmod \phi(N)$ maka f_d adalah kebalikan dari f_e.

Isomorfisme dan Teorema Sisa Cina

Dua kelompok disebut isomorfik jika mereka mempunyai struktur dasar yang sama. Dari sudut pandang matematika, isomorfisme grup $\mathbb{G}$ memberikan cara berpikir alternatif namun setara tentang $\mathbb{G}$. Dari perspektif komputasi, isomorfisme memberikan cara berbeda untuk merepresentasikan elemen dalam $\mathbb{G}$, yang sering kali dapat berdampak signifikan pada efisiensi algoritmik.

DEFINISI 8.23 Misalkan $\mathbb{G}, \mathbb{H}$ adalah grup terhadap operasinya $\circ_{\mathbb{G}}, \circ_{\mathbb{H}}$, masing-masing. Suatu fungsi $f : \mathbb{G} \to \mathbb{H}$ merupakan isomorfisme dari $\mathbb{G}$ ke $\mathbb{H}$ jika:

1. f adalah bijeksi, dan
2. Untuk semua $g_1, g_2 \in \mathbb{G}$ kita mempunyai $f(g_1 \circ_{\mathbb{G}} g_2) = f(g_1) \circ_{\mathbb{H}} f(g_2)$.

Jika terdapat isomorfisme dari $\mathbb{G}$ ke $\mathbb{H}$ maka kita katakan bahwa kelompok ini isomorfik dan tuliskan $\mathbb{G} \simeq \mathbb{H}$.

Intinya, isomorfisme dari $\mathbb{G}$ ke $\mathbb{H}$ hanyalah penggantian nama elemen $\mathbb{G}$ menjadi elemen $\mathbb{H}$. Perhatikan bahwa jika $\mathbb{G}$ berhingga dan $\mathbb{G} \simeq \mathbb{H}$, maka $\mathbb{H}$ harus berhingga dan berukuran sama dengan $\mathbb{G}$. Selain itu, jika terdapat isomorfisme f dari $\mathbb{G}$ ke $\mathbb{H}$ maka f^{-1} adalah isomorfisme dari $\mathbb{H}$ ke $\mathbb{G}$. Namun ada kemungkinan bahwa f dapat dihitung secara efisien sedangkan f^{-1} tidak (atau sebaliknya).

Tujuan dari bagian ini adalah menggunakan bahasa isomorfisme untuk lebih memahami struktur grup Z_N dan $\mathbb{Z}_N^*$ ketika $N = pq$ adalah hasil kali dua bilangan prima yang berbeda. Pertama-tama kita perlu memperkenalkan gagasan tentang produk langsung dari kelompok. Mengingat grup $\mathbb{G}, \mathbb{H}$ dengan operasi grup $\circ_{\mathbb{G}}, \circ_{\mathbb{H}}$, masing-masing, kami mendefinisikan grup baru $\mathbb{G} \times \mathbb{H}$ (produk langsung dari $\mathbb{G}$ dan $\mathbb{H}$) sebagai berikut. Unsur-unsur $\mathbb{G} \times \mathbb{H}$ merupakan pasangan terurut (g, h) dengan $g \in \mathbb{G}$ dan $h \in \mathbb{H}$; jadi, jika $\mathbb{G}$ mempunyai n elemen dan $\mathbb{H}$ memiliki n' elemen, $\mathbb{G} \times \mathbb{H}$ memiliki $n \cdot n'$ elemen. Operasi grup $\circ$ pada $\mathbb{G} \times \mathbb{H}$ diterapkan secara komponen; itu adalah:

$$(g, h) \circ (g', h') \stackrel{\text{def}}{=} (g \circ_{\mathbb{G}} g',\ h \circ_{\mathbb{H}} h').$$

Notasi di atas dapat diperluas untuk produk langsung lebih dari dua kelompok dengan cara alami, meskipun kita tidak memerlukannya untuk selanjutnya. Sekarang kita dapat menyatakan dan membuktikan teorema sisa Cina.

TEOREMA 8.24 (Teorema sisa Cina) Misalkan $N = pq$ dimana $p, q > 1$ relatif prima. Kemudian

$$\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad and \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*.$$

Selain itu, misalkan ƒ adalah elemen pemetaan fungsi $x \in \{0, \ldots, N— 1\}$ untuk memasangkan (x_p, x_q) dengan $x_p \in \{0, \ldots, p— 1\}$ dan $x_q \in \{0, \ldots, q— 1\}$ditentukan oleh

$$f(x) \stackrel{\text{def}}{=} ([x \bmod p], [x \bmod q]).$$

Maka ƒ adalah isomorfisme dari Z_N ke $Z_p \times Z_p$, dan pembatasan ƒ ke $\mathbb{Z}_N^*$ adalah isomorfisme dari $\mathbb{Z}_N^*$ ke $\mathbb{Z}_p^* \times \mathbb{Z}_p^*$.

BUKTI Untuk setiap $x \in Z_N$ keluarannya $f(x)$ adalah sepasang elemen (x_p, x_q) dengan $(x_p \in \mathbb{Z}_p)$ dan $(x_q \in \mathbb{Z}_q)$. Kita menyatakan bahwa jika $x \in \mathbb{Z}_N^*$, maka $(x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$.

Memang benar, jika $x_p \notin \mathbb{Z}_p^*$ maka ini berarti $\gcd([x \bmod p], p) \neq 1$. Tapi kemudian $\gcd(x, p) \neq 1$. Artinya $\gcd(x, N) \neq 1$, bertentangan dengan asumsi bahwa $x \in \mathbb{Z}_N^*$. (Argumen analog berlaku jika $x_q \notin \mathbb{Z}_p^*$.)

Kami sekarang menunjukkan bahwa ƒ adalah isomorfisme dari Z_N ke $\mathbb{Z}_p \times \mathbb{Z}_q$. (Bukti bahwa ini adalah isomorfisme dari $\mathbb{Z}_N^*$ ke $\mathbb{Z}_p^* \times \mathbb{Z}_p^*$ serupa.) Mari kita mulai dengan membuktikan bahwa ƒ adalah satu-ke-satu. Katakanlah $f(x) = (x_p, x_q) = f(x')$. Maka $x = x_p = x' \bmod p$ dan $x = x_q = x' \bmod q$. Hal ini berarti $(x - x')$ habis dibagi p dan q. Karena $\gcd(p, q) = 1$, Proposisi 8.4 menyatakan bahwa $pq = N$ membagi $(x - x')$. Tapi kemudian $x = x' \bmod N$. Untuk $x, x' \in \mathbb{Z}_N$, ini berarti $x = x'$ sehingga ƒ memang satu-satu. Karena $|\mathbb{Z}_N| = N = p \cdot q = |\mathbb{Z}_p| \cdot |\mathbb{Z}_q|$, maka ukuran $\mathbb{Z}_N$ dan $\mathbb{Z}_p \times \mathbb{Z}_q$ adalah sama. Hal ini dikombinasikan dengan fakta bahwa ƒ adalah satu-ke-satu menyiratkan bahwa ƒ bersifat bijektif.

Dalam paragraf berikut, misalkan $+_N$ menyatakan penjumlahan modulo N, dan misalkan $\boxplus$ menyatakan operasi grup di $\mathbb{Z}_p \times \mathbb{Z}_q$ (yaitu, penjumlahan modulo p pada komponen pertama dan penjumlahan modulo q pada komponen kedua). Untuk menyimpulkan bukti bahwa ƒ adalah isomorfisme dari $\mathbb{Z}_N$ ke $\mathbb{Z}_p \times \mathbb{Z}_q$, kita perlu menunjukkan bahwa untuk semua $a, b \in \mathbb{Z}_N$ dinyatakan bahwa $f(a + N^b) = f(a) \boxplus f(b)$.

Untuk melihat bahwa ini benar, perhatikan itu

$$\begin{aligned} f(a +_N b) &= \Big([(a +_N b) \bmod p], [(a +_N b) \bmod q]\Big) \\ &= \Big([(a + b) \bmod p], [(a + b) \bmod q]\Big) \\ &= \Big([a \bmod p], [a \bmod q]\Big) \boxplus \Big([b \bmod p], [b \bmod q]\Big) = f(a) \boxplus f(b). \end{aligned}$$

(Untuk persamaan kedua di atas, kita menggunakan fakta bahwa $[[X \bmod N] \bmod p] = [[X \bmod p] \bmod p]$ ketika $p|N$;)

Perpanjangan teorema sisa Cina menyatakan bahwa jika $p_1, p_2, \ldots, p_\ell$ berpasangan relatif prima (yaitu, $\gcd(p_i, p_j) = 1$ untuk semua $i \neq j$) dan $N \stackrel{\text{def}}{=} \Pi_{i=1}^{\ell} p_i$, maka

$$\mathbb{Z}_N \simeq \mathbb{Z}_{p_1} \times \cdots \times \mathbb{Z}_{p_\ell} \quad \text{and} \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_{p_1}^* \times \cdots \times \mathbb{Z}_{p_\ell}^*.$$

Isomorfisme dalam setiap kasus diperoleh dengan perluasan alami dari isomorfisme yang digunakan dalam teorema di atas.

Secara notasi, dengan N dipahami dan $x \in \{0,1,\ldots,N-1\}$ kita tulis $x \leftrightarrow (x_p, x_q)$ untuk $x_p = [x \bmod p]$ dan $x_p = [x \bmod p]$. Artinya, $x \leftrightarrow (x_p, x_q)$ jika dan hanya jika $f(x) = (x_p, x_q)$, dimana f seperti pada teorema di atas. Salah satu cara untuk memikirkan notasi ini adalah bahwa ini berarti "x (dalam $\mathbb{Z}_N$) sama dengan (x_p, x_q) (dalam $\mathbb{Z}_p \times \mathbb{Z}_q$)." Notasi yang sama digunakan ketika berhadapan dengan $x \in \mathbb{Z}_N^*$.

Contoh 8.25

Ambil $15 = 5 \cdot 3$, dan anggap $\mathbb{Z}_{15}^* = \{1,2,4,7,8,11,13,14\}$. Teorema sisa Cina menyatakan bahwa grup ini isomorfik terhadap $\mathbb{Z}_5^* \times \mathbb{Z}_3^*$. Kita bisa menghitung

$$\begin{array}{llll} 1 \leftrightarrow (1,1) & 2 \leftrightarrow (2,2) & 4 \leftrightarrow (4,1) & 7 \leftrightarrow (2,1) \\ 8 \leftrightarrow (3,2) & 11 \leftrightarrow (1,2) & 13 \leftrightarrow (3,1) & 14 \leftrightarrow (4,2)\text{,} \end{array}$$

dimana setiap pasangan (a, b) dengan $a \in \mathbb{Z}_5^*$ dan $b \in \mathbb{Z}_5^*$ muncul tepat satu kali.

Menggunakan Teorema Sisa Cina

Jika dua kelompok bersifat isomorfik, maka keduanya berfungsi sebagai representasi dari "struktur aljabar" dasar yang sama. Namun demikian, pilihan representasi mana yang akan digunakan dapat mempengaruhi efisiensi komputasi operasi kelompok. Kami membahasnya secara abstrak, dan kemudian dalam konteks spesifik $\mathbb{Z}_N$ dan $\mathbb{Z}_N^*$.

Misalkan $\mathbb{G}, \mathbb{H}$ adalah grup dengan operasi $\circ_{\mathbb{G}}, \circ_{\mathbb{H}}$, masing-masing, dan katakanlah f adalah isomorfisme dari $\mathbb{G}$ ke $\mathbb{H}$ di mana f dan f^{-1} dapat dihitung secara efisien. Kemudian untuk $g_1, g_2 \in \mathbb{G}$ kita dapat menghitung $g = g_1 \circ_{\mathbb{G}} g_2$ dengan dua cara: baik dengan menghitung operasi grup secara langsung di $\mathbb{G}$, atau melalui langkah-langkah berikut:

1. Hitung $h_1 = f(g_1)$ dan $h_2 = f(g_2)$;
2. Hitung $h = h_1 \circ_{\mathbb{H}} h_2$ menggunakan operasi grup di $\mathbb{H}$;
3. Hitung $g = f^{-1}(h)$.

Hal di atas berlaku secara alami ketika kita ingin menghitung beberapa operasi grup di $\mathbb{G}$ (misalnya, menghitung g^x untuk beberapa bilangan bulat x). Metode mana yang lebih baik

bergantung pada efisiensi relatif penghitungan operasi grup di setiap grup, serta efisiensi penghitungan f dan f^{-1}.

Kita sekarang beralih ke kasus spesifik perhitungan modulo N, ketika $N = pq$ adalah hasil kali bilangan prima yang berbeda. Teorema sisa Cina menunjukkan bahwa penjumlahan, perkalian, atau eksponensial (yang merupakan perkalian berulang) modulo N dapat "ditransformasikan" menjadi operasi analog modulo p dan q. Berdasarkan Contoh 8.25, kami menunjukkan beberapa contoh sederhana dengan $N = 15$.

Contoh 8.26

Katakanlah kita ingin menghitung produk $14 \cdot 13$ modulo 15 (yaitu, dalam $\mathbb{Z}_{15}^*$). Contoh 8.25 menghasilkan $14 \leftrightarrow (4,2)$ dan $13 \leftrightarrow (3,1)$. Dalam $\mathbb{Z}_5^* \times \mathbb{Z}_3^*$, kita punya

$$(4,2) \cdot (3,1) = ([4 \cdot 3 \bmod 5],\ [2 \cdot 1 \bmod 3]) = (2,2).$$

Catatan $(2,2) \leftrightarrow 2$ yang merupakan jawaban yang benar karena $14 \cdot 13 = 2 \bmod 15$.

Contoh 8.27

Katakanlah kita ingin menghitung $11^{53} \bmod 15$. Contoh 8.25 menghasilkan $11 \leftrightarrow (1,2)$. Perhatikan bahwa $2 =\!\!-\ 1 \bmod 3$ dan seterusnya

$$(1,2)^{53} = ([1^{53} \bmod 5], [(-1)^{53} \bmod 3]) = (1, [-1 \bmod 3]) = (1,2).$$

Jadi, $11^{53} \bmod 15 = 11$.

Contoh 8.28

Katakanlah kita ingin menghitung $[29^{100} \bmod 35]$. Pertama-tama kita menghitung korespondensi $29 \leftrightarrow ([29 \bmod 5], [29 \bmod 7]) = ([-1 \bmod 5], 1)$. Dengan menggunakan teorema sisa Cina, kita punya

$$([-1 \bmod 5], 1)^{100} = ([(-1)^{100} \bmod 5],\ [1^{100} \bmod 7]) = (1,1),$$

dan langsung $(1,1) \leftrightarrow 1$. Kita simpulkan bahwa $[29^{100} \bmod 35] = 1$.

Contoh 8.29

Katakanlah kita ingin menghitung $[18^{25} \bmod 35]$. Kami memiliki $18 \leftrightarrow (3,4)$ dan seterusnya

$$18^{25} \bmod 35 \leftrightarrow (3,4)^{25} = ([3^{25} \bmod 5],\ [4^{25} \bmod 7]).$$

Karena $\mathbb{Z}_5^*$ adalah grup berorde 4, kita dapat "mengerjakan modulo 4 dalam eksponen" (lih. Akibat wajar 8.15) dan melihat bahwa

$$3^{25} = 3^{[25 \bmod 4]} = 3^1 = 3 \bmod 5.$$

Demikian pula,

$$4^{25} = 4^{[25 \bmod 6]} = 4^1 = 4 \bmod 7.$$

Jadi, $([3^{25} \bmod 5], [4^{25} \bmod 7]) = (3,4) \leftrightarrow 18$ dan seterusnya $[18^{25} \bmod 35] = 18$. ♦

Satu hal yang belum kita bahas adalah bagaimana mengkonversi bolak-balik antara representasi elemen modulo N dan representasi modulo p dan q. Konversi dapat dilakukan secara efisien asalkan faktorisasi N diketahui. Dengan asumsi p dan q diketahui, mudah untuk memetakan elemen x modulo N ke representasi terkait modulo p dan q: elemen x berhubungan dengan $([x \bmod p], [x \bmod q])$, dan keduanya merupakan reduksi modular dapat dilakukan secara efisien.
Untuk arah yang lain, kita menggunakan pengamatan berikut: elemen dengan representasi (x_p, x_q) dapat ditulis sebagai

$$(x_p, x_q) = x_p \cdot (1,0) + x_q \cdot (0,1).$$

Jadi, jika kita dapat menemukan elemen $1_p, 1_q \in \{0, \ldots, N-1\}$ sehingga $1_p \leftrightarrow (1,0)$ dan $1_q \leftrightarrow (0,1)$, maka (sesuai dengan teorema sisa Cina) kita mengetahui bahwa

$$(x_p, x_q) \leftrightarrow [(x_p \cdot 1_p + x_q \cdot 1_q) \bmod N].$$

Karena p, q adalah bilangan prima yang berbeda, $\gcd(p, q) = 1$. Kita dapat menggunakan algoritma Euclidean yang diperluas untuk mencari bilangan bulat X, Y sedemikian rupa sehingga

$$X_p + Y_q = 1$$

Perhatikan bahwa $Y_q = 0 \bmod q$ dan $Y_q = 1 - X_p = 1 \bmod p$. Artinya $[Y_q \bmod N] \leftrightarrow (1,0)$; yaitu, $[Y_q \bmod N] = 1_p$. Demikian pula, $[X_p \bmod N] = 1_q$.

Ringkasnya, kita dapat mengkonversi sebuah elemen yang direpresentasikan sebagai (x_p, x_q) ke representasi modulo N dengan cara berikut (dengan asumsi p dan q diketahui):

1. Hitung X, Y sehingga $X_p + Y_q = 1$.

2. Atur $1_p := [Y_q \bmod N]$ dan $1_q := [X_p \bmod N]$.
3. Hitung $x := [(X_p \cdot 1_p + X_q \cdot 1_q) \bmod N]$.

Jika banyak konversi yang akan dilakukan, maka $1_p, 1_q$ dapat dihitung sekali dan untuk semua dalam fase prapemrosesan.

Contoh 8.30

Ambil $p = 5, q = 7$, dan $N = 5 \cdot 7 = 35$. Katakanlah kita diberikan representasi $(4,3)$ dan ingin mengubahnya menjadi elemen $\mathbb{Z}_{35}$ yang sesuai. Dengan menggunakan algoritma Euclidean yang diperluas, kami menghitung

$$3 \cdot 5 - 2 \cdot 7 = 1.$$

Jadi, $1_p = [-2 \cdot 7 \bmod 35] = 21$ dan $1_q = [3 \cdot 5 \bmod 35] = 15$. (Kita dapat memeriksa apakah ini benar: misalnya, untuk $1_p = 21$ kita dapat memverifikasi bahwa $[21 \bmod 5] = 1$ dan $[21 \bmod 7] = 0$.) Dengan menggunakan nilai-nilai ini, kita kemudian dapat menghitung

$$\begin{aligned}(4,3) &= 4 \cdot (1,0) + 3 \cdot (0,1) \\ &\leftrightarrow [4 \cdot 1_p + 3 \cdot 1_q \bmod 35] \\ &= [4 \cdot 21 + 3 \cdot 15 \bmod 35] = 24.\end{aligned}$$

Karena 24 = 4 mod 5 dan 24 = 3 mod 7, ini memang hasil yang benar.

8.2 BILANGAN PRIMA, FAKTORISASI, DAN RSA

Di bagian ini, kami menunjukkan contoh pertama soal teori bilangan yang dianggap "sulit". Kita mulai dengan pembahasan salah satu masalah tertua: faktorisasi bilangan bulat atau pemfaktoran saja.

Diberikan bilangan bulat komposit N, soal pemfaktoran adalah mencari bilangan bulat $p, q > 1$ sedemikian rupa sehingga $pq = N$. Pemfaktoran adalah contoh klasik dari permasalahan yang sulit, karena sangat sederhana untuk dijelaskan dan karena telah dikenal sebagai masalah komputasi yang sulit sejak lama (bahkan sebelum digunakan dalam kriptografi). Soal tersebut dapat diselesaikan dalam waktu eksponensial $\mathcal{O}(\sqrt{N} \cdot \text{polilog}(N))$ menggunakan pembagian percobaan: yaitu, dengan √memeriksa secara mendalam apakah p membagi N untuk $p = 2, \ldots, \lfloor\sqrt{N}\rfloor$. (Metode ini memerlukan $\sqrt{N}$pembagian, masing-masing mengambil polilog$(N) = \| N \|^c$ waktu untuk suatu konstanta c.) Cara ini selalu berhasil karena meskipun faktor prima terbesar dari N mungkin sebesar $N/2$, faktor prima terkecil dari N paling banyak bisa $\lfloor\sqrt{N}\rfloor$. Meskipun algoritma dengan waktu berjalan yang lebih baik telah diketahui (lihat Bab 9), tidak ada algoritma waktu polinomial untuk pemfaktoran yang telah dibuktikan meskipun telah dilakukan upaya bertahun-tahun.

Pertimbangkan percobaan berikut untuk algoritma tertentu $\mathcal{A}$ dan parameter n:
Eksperimen pemfaktoran lemah $\text{w} - \text{Factor}_{\mathcal{A}}(n)$:

1. Pilih dua bilangan bulat n-bit seragam x_1, x_2.
2. Hitung $N := x_1, x_2$.
3. $\mathcal{A}$ diberikan N, dan keluaran $x'_1, x'_2 > 1$.
4. Keluaran percobaan ditetapkan 1 jika $x'_1, x'_2 = N$, dan 0 jika tidak.

Kami baru saja mengatakan bahwa masalah anjak piutang diyakini sulit. Apakah ini berarti demikian

$$\Pr[\text{w-Factor}_{\mathcal{A}}(n) = 1] \leq \text{negl}(n)$$

dapat diabaikan untuk setiap algoritma PPT $\mathcal{A}$? Sama sekali tidak. Sebagai permulaan, bilangan N pada percobaan di atas adalah bilangan genap dengan probabilitas $3/4$ (ini terjadi jika x_1 atau x_2 genap); tentu saja mudah bagi $\mathcal{A}$ untuk memfaktorkan N dalam kasus ini. Meskipun kita dapat mempersulit pekerjaan $\mathcal{A}$ dengan mengharuskan $\mathcal{A}$ mengeluarkan bilangan bulat x'_1, x'_2 dengan panjang n, x_1 atau x_2 (dan karenanya N) mungkin memiliki faktor prima kecil yang masih dapat ditemukan dengan mudah. Untuk aplikasi kriptografi, kita perlu mencegah hal ini.

Sebagaimana ditunjukkan dalam diskusi ini, bilangan-bilangan yang "paling sulit" difaktorkan adalah bilangan-bilangan yang hanya memiliki faktor prima yang besar. Hal ini menyarankan untuk mendefinisikan ulang eksperimen di atas sehingga x_1, x_2 adalah bilangan prima n-bit acak dan bukan bilangan bulat n-bit acak, dan pada kenyataannya eksperimen seperti itu akan digunakan saat kita mendefinisikan asumsi pemfaktoran secara formal di Bagian 8.2. Namun, agar eksperimen ini berguna dalam pengaturan kriptografi, diperlukan kemampuan untuk menghasilkan bilangan prima n-bit acak secara efisien. Ini adalah topik dari dua bagian berikutnya.

Menghasilkan Bilangan Prima Acak

Pendekatan alami untuk menghasilkan bilangan prima n-bit acak adalah dengan berulang kali memilih bilangan bulat n-bit acak hingga kita menemukan bilangan bulat bilangan prima; kami mengulanginya paling banyak sebanyak t kali atau sampai kami berhasil. Lihat Algoritma 8.31 untuk deskripsi proses tingkat tinggi.

ALGORITHM 8.31
Generating a random prime – high-level outline

Input: Length n; parameter t
Output: A uniform n-bit prime

for $i = 1$ to t:
  $p' \leftarrow \{0,1\}^{n-1}$
  $p := 1 \| p'$
  if p is prime **return** p
return fail

Perhatikan bahwa algoritme memaksa output menjadi bilangan bulat dengan panjang tepat n (bukan paling panjang n) dengan menetapkan bit orde tinggi p ke "1". Konvensi kami di seluruh buku ini adalah bahwa "bilangan bulat dengan panjang n" berarti bilangan bulat yang representasi binernya dengan bit paling signifikan sama dengan 1 panjangnya tepat n bit.

Jika diberikan cara untuk menentukan apakah suatu bilangan bulat p adalah bilangan prima atau tidak, algoritma di atas menghasilkan n-bit bilangan prima seragam yang dikondisikan jika keluarannya tidak gagal. Probabilitas kegagalan keluaran algoritma bergantung pada t, dan untuk tujuan kita, kita ingin mengatur t sehingga memperoleh probabilitas kegagalan yang dapat diabaikan dalam n. Untuk menunjukkan bahwa Algoritma 8.31 menghasilkan algoritma yang efisien (yaitu, waktu polinomial dalam n) untuk menghasilkan bilangan prima, kita memerlukan pemahaman yang lebih baik tentang dua masalah: (1) probabilitas bahwa bilangan bulat n-bit yang seragam adalah bilangan prima dan (2) cara menguji secara efisien apakah bilangan bulat tertentu p adalah bilangan prima. Kini kita membahas isu-isu ini secara singkat, dan menunda eksplorasi topik kedua yang lebih mendalam ke bagian berikutnya.

Distribusi bilangan prima. Teorema bilangan prima, sebuah hasil penting dalam matematika, memberikan batasan yang cukup tepat pada pecahan bilangan bulat dengan panjang tertentu yang merupakan bilangan prima. Untuk tujuan kita, kita hanya memerlukan versi yang lemah dan sepihak dari hasil tersebut yang tidak kita buktikan di sini:

TEOREMA 8.32 (postulat Bertrand) Untuk sembarang $n > 1$, pecahan bilangan bulat n-bit bilangan prima paling sedikit adalah $1/3n$.

Kembali ke pendekatan untuk menghasilkan bilangan prima yang dijelaskan di atas, hal ini menyiratkan bahwa jika kita menetapkan $t = 3n^2$ maka probabilitas bahwa bilangan prima tidak terpilih dalam semua t iterasi algoritma adalah yang paling besar.

$$\left(1 - \frac{1}{3n}\right)^t = \left(\left(1 - \frac{1}{3n}\right)^{3n}\right)^n \leq \left(e^{-1}\right)^n = e^{-n}$$

yang dapat diabaikan di n. Jadi, dengan menggunakan iterasi poli(n) kita memperoleh algoritma yang kemungkinan kegagalan keluarannya dapat diabaikan dalam n. (Hasil yang diketahui lebih ketat daripada Teorema 8.32, sehingga dalam praktiknya diperlukan lebih sedikit iterasi.)

Menguji keutamaan. Masalah dalam menentukan secara efisien apakah suatu bilangan prima mempunyai sejarah yang panjang. Pada tahun 1970-an, algoritma efisien pertama untuk menguji primalitas dikembangkan. Algoritme ini bersifat probabilistik dan memiliki jaminan sebagai berikut: jika masukan p adalah bilangan prima, algoritme akan selalu menghasilkan keluaran "prima". Di sisi lain, jika p adalah gabungan, maka algoritme hampir selalu menghasilkan keluaran "gabungan", namun mungkin menghasilkan jawaban yang salah ("prima") dengan probabilitas yang dapat diabaikan sepanjang p. Dengan kata lain, jika algoritma menghasilkan keluaran "komposit" maka p sudah pasti merupakan komposit, namun jika keluarannya adalah "prima" maka kemungkinan besar p adalah bilangan prima tetapi mungkin juga telah terjadi kesalahan (dan p benar-benar komposit).

Saat menggunakan uji primalitas acak semacam ini dalam Algoritma 8.31 (algoritme pembangkitan prima yang ditunjukkan sebelumnya), keluaran algoritme adalah bilangan prima seragam dengan panjang yang diinginkan selama algoritme tidak menghasilkan keluaran yang gagal dan uji primalitas acak berhasil tidak salah selama eksekusi algoritma. Ini berarti bahwa sumber kesalahan tambahan (selain kemungkinan keluaran gagal) muncul, dan algoritme kini dapat mengeluarkan bilangan komposit secara tidak sengaja. Karena kita dapat memastikan bahwa hal ini terjadi dengan kemungkinan yang dapat diabaikan, kemungkinan kecil ini tidak menjadi perhatian praktis dan kita dapat mengabaikannya dengan aman.

Algoritme waktu polinomial deterministik untuk menguji primalitas ditunjukkan dalam hasil terobosan pada tahun 2002. Algoritme tersebut, meskipun berjalan dalam waktu polinomial, lebih lambat dibandingkan pengujian probabilistik yang disebutkan di atas. Oleh karena itu, uji primalitas probabilistik masih digunakan secara eksklusif dalam praktik untuk menghasilkan bilangan prima yang besar.

Di Bagian 8.2.2 kami menjelaskan dan menganalisis salah satu uji primalitas probabilistik yang paling umum digunakan: algoritma Miller – Rabin. Algoritma ini mengambil dua masukan: bilangan bulat p dan parameter t (dalam bentuk unary) yang menentukan probabilitas kesalahan. Algoritme Miller – Rabin berjalan dalam polinomial waktu di p dan t, dan memenuhi:

TEOREMA 8.33 Jika p adalah bilangan prima, maka uji Miller–Rabin selalu menghasilkan keluaran "prima." Jika p adalah komposit, algoritma mengeluarkan "komposit" kecuali dengan probabilitas paling banyak 2^{-t}.

Menyatukan semuanya. Mengingat diskusi sebelumnya, kita sekarang dapat menggambarkan algoritma pembangkitan prima waktu polinomial yang, pada input n, menghasilkan n-bit prima kecuali dengan probabilitas yang dapat diabaikan dalam n; terlebih

lagi, karena keluaran p menjadi bilangan prima, p adalah n-bit bilangan prima yang terdistribusi secara merata. Prosedur selengkapnya dijelaskan pada Algoritma 8.34.

ALGORITHM 8.34
Generating a random prime

Input: Length n
Output: A uniform n-bit prime

for $i = 1$ to $3n^2$:
  $p' \leftarrow \{0,1\}^{n-1}$
  $p := 1 \| p'$
  run the Miller–Rabin test on input p and parameter 1^n
  if the output is "prime," **return** p
return fail

Menghasilkan bilangan prima dari bentuk tertentu. Kadang-kadang diinginkan untuk menghasilkan n-bit prima p acak dari bentuk tertentu, misalnya, memenuhi $p = 3 \bmod 4$ atau sedemikian rupa sehingga $p = 2q + 1$ di mana q juga bilangan prima (p dari tipe terakhir disebut bilangan prima kuat). Dalam hal ini, modifikasi yang sesuai dari algoritma generasi prima yang ditunjukkan di atas dapat digunakan. (Misalnya, untuk mendapatkan bilangan prima berbentuk $p = 2q + 1$, modifikasi algoritme untuk menghasilkan bilangan prima acak q, hitung $p := 2q + 1$, lalu keluarkan p jika bilangan tersebut juga bilangan prima.) Sementara ini algoritma yang dimodifikasi bekerja dengan baik dalam praktiknya, bukti yang kuat bahwa algoritma tersebut berjalan dalam waktu polinomial dan gagal dengan probabilitas yang dapat diabaikan lebih kompleks (dan, dalam beberapa kasus, bergantung pada dugaan teori bilangan yang belum terbukti mengenai kepadatan bilangan prima dari bentuk tertentu). Eksplorasi mendetail mengenai isu-isu ini berada di luar cakupan buku ini, dan kami hanya akan mengasumsikan keberadaan algoritma generasi prima yang tepat bila diperlukan.

Pengujian Primalitas

Kami sekarang menjelaskan uji primalitas Miller – Rabin dan membuktikan Teorema 8.33. (Kami mengandalkan materi yang disajikan di Bagian 8.1) Materi ini tidak digunakan secara langsung di bagian lain buku ini.

Kunci dari algoritma Miller – Rabin adalah menemukan properti yang membedakan bilangan prima dan komposit. Misalkan N menunjukkan nomor masukan yang akan diuji. Kita mulai dengan pengamatan berikut: jika N bilangan prima maka $|\mathbb{Z}_N^*| = N - 1$, dan untuk sembarang $a \in \{1, \ldots, N-1\}$ kita mempunyai $a^{N-1} = 1 \bmod N$ dengan Teorema 8.14. Hal ini menyarankan pengujian apakah N bilangan prima dengan memilih elemen seragam a dan memeriksa apakah $a^{N-1} \stackrel{?}{=} 1 \bmod N$. Jika $a^{N-1} \neq 1 \bmod N$, maka N tidak bisa menjadi bilangan prima. Sebaliknya, kita mungkin berharap bahwa jika N bukan bilangan prima maka ada kemungkinan yang masuk akal bahwa kita akan memilih a dengan $a^{N-1} \neq 1 \bmod N$, sehingga dengan mengulangi pengujian ini berkali-kali kita dapat menentukan apakah N bilangan prima atau tidak dengan bilangan prima. kepercayaan diri. Pendekatan di atas

ditampilkan sebagai Algoritma 8.35. (Ingat bahwa modulo eksponensial N dan penghitungan pembagi persekutuan terbesar dapat dilakukan dalam waktu polinomial. Pemilihan elemen seragam $\{1,..,N— 1\}$ juga dapat dilakukan dalam waktu polinomial

ALGORITHM 8.35
Primality testing – first attempt

Input: Integer N and parameter 1^t
Output: A decision as to whether N is prime or composite

for $i = 1$ to t:
  $a \leftarrow \{1, \ldots, N-1\}$
  if $a^{N-1} \neq 1 \bmod N$ **return** "composite"
return "prime"

Jika N adalah bilangan prima, algoritme selalu menghasilkan keluaran "prima". Jika N adalah komposit, algoritma mengeluarkan "komposit" jika dalam setiap iterasi ia menemukan $a \in \{1,\ldots,N— 1\}$ sedemikian rupa sehingga $a^{N-1} \neq 1 \bmod N$. Perhatikan bahwa jika $a \notin \mathbb{Z}_N^*$ maka $a^{N-1} \neq 1 \bmod N$.

(Jika $\gcd(a,N) \neq 1$ maka $\gcd(a^{N-1},N) \neq 1$ sehingga $[a^{N-1} \bmod N]$ tidak bisa sama dengan 1.) Oleh karena itu, untuk saat ini, kami membatasi perhatian kami pada $a \notin \mathbb{Z}_N^*$. Kita mengacu pada a dengan $a^{N-1} \neq 1 \bmod N]$ sebagai saksi bahwa N adalah komposit, atau sekadar saksi. Kita mungkin berharap bahwa ketika N adalah komposit maka terdapat banyak saksi, dan dengan demikian algoritma menemukan saksi tersebut dengan probabilitas "tinggi". Intuisi ini benar asalkan setidaknya ada satu saksi. Sebelum membuktikan hal ini, kita memerlukan dua lemma teori grup.

PROPOSISI 8.36 Misalkan $\mathbb{G}$ adalah grup berhingga, dan $\mathbb{H} \subseteq \mathbb{G}$. Asumsikan $\mathbb{H}$ tidak kosong, dan untuk semua $a, b \in \mathbb{H}$ kita mempunyai $ab \in \mathbb{H}$. Maka $\mathbb{H}$ adalah subgrup dari $\mathbb{G}$.

BUKTI Kita perlu memverifikasi bahwa $\mathbb{H}$ memenuhi semua kondisi Definisi 8.9. Dengan asumsi, $\mathbb{H}$ ditutup pada operasi grup. Asosiatifitas dalam $\mathbb{H}$ diwariskan secara otomatis dari $\mathbb{G}$. Misal $m = |\mathbb{G}|$ (di sinilah kita menggunakan fakta bahwa $\mathbb{G}$ berhingga), dan pertimbangkan elemen arbitrer $a \in \mathbb{H}$. Penutupan $\mathbb{H}$ berarti $\mathbb{H}$ mengandung $a^{m-1} = a^{-1}$ sebagai serta $a^m = 1$. Jadi, H mengandung invers dari setiap elemennya, serta identitasnya.

LEMMA 8.37 Misalkan $\mathbb{H}$ adalah subgrup ketat dari grup berhingga $\mathbb{G}$ (yaitu, $\mathbb{H} \neq \mathbb{G}$). Kemudian $|\mathbb{H}| \leq |\mathbb{G}|/2$.

BUKTI Misalkan $\bar{h}$ merupakan unsur $\mathbb{G}$ yang tidak ada pada $\mathbb{H}$; karena $\mathbb{H} \neq \mathbb{G}$, kita mengetahui bahwa $\bar{h}$ tersebut ada. Misalkan himpunan $\bar{\mathbb{H}} \stackrel{\text{def}}{=} \{\bar{h}h \mid h \in \mathbb{H}\}$. Kita tunjukkan bahwa (1) $|\bar{\mathbb{H}}| = |\mathbb{H}|$, dan (2) setiap elemen $\bar{\mathbb{H}}$ terletak di luar $\mathbb{H}$; yaitu perpotongan $\mathbb{H}$ dan $\bar{\mathbb{H}}$ kosong. Karena

$\mathbb{H}$ dan $\overline{\mathbb{H}}$ adalah himpunan bagian dari $\mathbb{G}$, maka ini berarti $|\mathbb{G}| \geq |\mathbb{H}| + |\mathbb{H}| = 2|\mathbb{H}|$, membuktikan lemmanya.

Untuk setiap $h_1, h_2 \in \mathbb{H}$, jika $\bar{h}h_1 = \bar{h}h_2$ maka, kalikan dengan $\bar{h}^{-1}$ di setiap sisi, kita mendapatkan $h_1 = h_2$. Hal ini menunjukkan bahwa setiap elemen yang berbeda $h \in \mathbb{H}$ berhubungan dengan elemen yang berbeda $\bar{h}h \in \overline{\mathbb{H}}$, membuktikan (1).

Asumsikan terhadap kontradiksi bahwa $\bar{h}h \in \mathbb{H}$ untuk beberapa h. Artinya $\bar{h}h = h'$ untuk beberapa $h' \in \mathbb{H}$, sehingga $\bar{h} = h'h^{-1}$. Sekarang, $h'h^{-1} \in \mathbb{H}$ karena $\mathbb{H}$ adalah subgrup dan $h', h^{-1} \in \mathbb{H}$. Namun ini berarti bahwa $\bar{h} \in \mathbb{H}$, bertentangan dengan cara $\bar{h}$ dipilih. Ini membuktikan dan melengkapi bukti lemma. Teorema berikut akan memungkinkan kita menganalisis algoritma yang diberikan sebelumnya.

TEOREMA 8.38 Perbaiki N. Katakanlah terdapat saksi bahwa N adalah komposit.

Maka paling sedikit separuh unsur $\mathbb{Z}_N^*$ menjadi saksi bahwa N bersifat komposit.

BUKTI Misalkan Bad adalah himpunan elemen dalam $\mathbb{Z}_N^*$ yang bukan merupakan saksi; yaitu, $a \in \text{Bad}$ berarti $a^{N-1} = 1 \bmod N$. Jelas, $1 \in \text{Bad}$. Jika $a, b \in \text{Bad}$, maka $(ab)^{N-1} = a^{N-1} \cdot b^{N-1} = 1 \cdot 1 = 1 \bmod N$ dan karenanya $ab \in \text{Bad}$. Berdasarkan Lemma 8.36, kita menyimpulkan bahwa Bad adalah subgrup dari $\mathbb{Z}_N^*$. Karena (dengan asumsi) setidaknya ada satu saksi, Bad adalah subgrup ketat dari $\mathbb{Z}_N^*$. Lemma 8.37 kemudian menunjukkan bahwa $|\text{Bad}| \leq |\mathbb{Z}_N^*|/2$, menunjukkan bahwa setidaknya setengah elemen $\mathbb{Z}_N^*$ tidak berada di Bad (dan karenanya merupakan saksi).

Misalkan N adalah komposit. Jika ada saksi yang N komposit, maka paling sedikit ada $|\mathbb{Z}_N^*|/2$ saksi. Probabilitas bahwa kita menemukan salah satu saksi atau elemen yang tidak ada di $\mathbb{Z}_N^*$ pada setiap iterasi algoritma tertentu adalah paling sedikit $1/2$, sehingga probabilitas bahwa algoritma tidak menemukan saksi pada iterasi mana pun (dan oleh karena itu kemungkinan algoritma tersebut salah mengeluarkan "prime") paling banyak adalah 2^{-t}.

Sayangnya, penyelesaian di atas tidak memberikan penyelesaian yang lengkap karena terdapat banyak sekali bilangan komposit N yang tidak mempunyai bukti bahwa bilangan tersebut komposit! Nilai N tersebut dikenal sebagai bilangan Carmichael; pembahasan mendetail berada di luar cakupan buku ini.

Untungnya, penyempurnaan tes di atas terbukti berhasil untuk semua N. Misal $N - 1 = 2^r u$, dimana u ganjil dan $r \geq 1$. (Mudah untuk menghitung r dan u jika diberikan N. Selain itu, membatasi ke $r \geq 1$ berarti N ganjil, namun pengujian primalitas mudah dilakukan jika N genap!) algoritma yang ditunjukkan sebelumnya hanya menguji apakah $a^{N-1} = a^{2^r u} = 1 \bmod N$. Algoritme yang lebih halus melihat urutan nilai $r + 1$ $a^u, a^{2u}, \ldots, a^{2^r u}$ (semua modulo N). Setiap suku dalam barisan ini adalah kuadrat dari suku sebelumnya; jadi, jika suatu nilai sama dengan 1 maka semua nilai berikutnya akan sama dengan 1.

Katakanlah $a \in \mathbb{Z}_N^*$ adalah saksi kuat bahwa N adalah gabungan (atau sekadar saksi kuat) jika (1) $a^u \neq \pm 1 \bmod N$ dan (2) $a^{2^i u} \neq -1 \bmod N$ untuk semua $i \in \{1, \ldots, r-1\}$. Perhatikan bahwa jika suatu elemen a bukan merupakan saksi kuat maka barisan $(a^u, a^{2u}, \ldots, a^{2^r u})$ (semua diambil modulo N) mengambil salah satu bentuk berikut:

$$(\pm 1, 1, \ldots, 1) \quad \text{or} \quad (\star, \ldots, \star, -1, 1, \ldots, 1),$$

Di mana? adalah istilah yang sewenang-wenang. Jika a bukan saksi kuat maka kita mempunyai $a^{2^{r-1}u} = \pm 1 \bmod N$ dan

$$a^{N-1} = a^{2^r u} = \left(a^{2^{r-1}u}\right)^2 = 1 \bmod N,$$

jadi a juga bukan merupakan bukti bahwa N adalah komposit. Dengan kata lain, jika a adalah seorang saksi, maka ia juga merupakan saksi yang kuat sehingga kemungkinan besar akan ada lebih banyak saksi yang kuat daripada saksi.

Pertama-tama kita tunjukkan bahwa jika N bilangan prima maka tidak ada bukti kuat bahwa N bilangan komposit. Dalam melakukannya, kami mengandalkan lemma mudah berikut (yang merupakan kasus khusus dari Proposisi 13.16 yang dibuktikan kemudian di Bab 13):

LEMMA 8.39 Katakanlah $x \in \mathbb{Z}_N^*$ adalah akar kuadrat dari 1 modulo N jika $x^2 = 1 \bmod N$. Jika N adalah bilangan prima ganjil maka satu-satunya akar kuadrat dari 1 modulo N adalah $[\pm 1 \bmod N]$.

BUKTI Katakanlah $x^2 = 1 \bmod N$ dengan $x \in \{1, \ldots, N-1\}$. Maka $0 = x^2 - 1 = (x+1)(x-1) \bmod N$, menyiratkan bahwa $N|(x+1)$ atau $N|(x-1)$ berdasarkan Proposisi 8.3. Hal ini hanya mungkin terjadi jika $x = [\pm 1 \bmod N]$.

Misalkan N adalah bilangan prima ganjil dan perbaiki $a \in \mathbb{Z}_N^*$ sembarang. Misalkan $i \geq 0$ adalah nilai minimum dimana $a^{2^r u} = 1 \bmod N$; karena $a^{2^r u} = a^{N-1} = 1 \bmod N$ kita tahu bahwa beberapa $i \leq r$ tersebut ada. Jika $i = 0$ maka $a^u = 1 \bmod N$ dan a bukanlah saksi yang kuat. Jika tidak,

$$\left(a^{2^{i-1}u}\right)^2 = a^{2^i u} = 1 \bmod N$$

dan $a^{2^{i-1}u}$ adalah akar kuadrat dari 1. Jika N adalah bilangan prima ganjil, akar kuadrat dari 1 hanyalah ± 1; dengan pilihan i, bagaimanapun, $a^{2^{i-1}u} \neq 1 \bmod N$. Jadi $a^{2^{i-1}u} = -1 \bmod N$,

dan a bukanlah saksi yang kuat. Kita menyimpulkan bahwa jika N adalah bilangan prima ganjil maka tidak ada bukti kuat bahwa N adalah bilangan komposit.

Suatu bilangan bulat komposit N adalah pangkat prima jika $N = p^r$ untuk beberapa bilangan prima p dan bilangan bulat $r \geq 1$. Sekarang kita tunjukkan bahwa setiap bilangan ganjil, gabungan N yang bukan pangkat prima mempunyai banyak saksi yang kuat.

TEOREMA 8.40 Misalkan N suatu bilangan ganjil yang bukan merupakan pangkat prima.

Maka setidaknya separuh elemen $\mathbb{Z}_N^*$ merupakan bukti kuat bahwa N adalah komposit.

BUKTI Misalkan $\text{Bad} \subseteq \mathbb{Z}_N^*$ menyatakan himpunan unsur yang bukan merupakan saksi kuat. Kita mendefinisikan himpunan Bad' dan menunjukkan bahwa: (1) Bad adalah subset dari Bad', dan (2) Bad' adalah subgrup ketat dari $\mathbb{Z}_N^*$. Ini cukup karena dengan menggabungkan (2) dan Lemma 8.37 kita mendapatkan $|\text{Bad}'| \leq |\mathbb{Z}_N^*|/2$. Selanjutnya, dengan (1) dinyatakan bahwa $\text{Bad} \subseteq \text{Bad}'$, dan juga $|\text{Bad}| \leq |\text{Bad}'| \leq |\mathbb{Z}_N^*|/2$ seperti pada Teorema 8.38.

Jadi, setidaknya separuh elemen $\mathbb{Z}_N^*$ merupakan saksi kuat. (Kami menekankan bahwa kami tidak mengklaim bahwa Bad adalah subgrup dari $\mathbb{Z}_N^*$.)

Perhatikan dulu bahwa $-1 \in \text{Bad}$ karena $(-1)^u = -1 \bmod N$ (ingat u ganjil). Misalkan $i \in \{0, \ldots, r-1\}$ adalah bilangan bulat terbesar yang memiliki $a \in \text{Bad}$ dengan $a^{2^i u} = -1 \bmod N$; alternatifnya, i adalah bilangan bulat terbesar yang memiliki $a \in \text{Bad}$ dengan

$$(a^u, a^{2u}, \ldots, a^{2^r u}) = (\underbrace{\star, \ldots, \star, -1}_{i+1 \text{ terms}}, 1, \ldots, 1).$$

Karena $-1 \in \text{Bad}$ dan $(-1)^{2^0 u} = -1 \bmod N$, beberapa i ada.
Perbaiki i seperti di atas, dan definisikan

$$\text{Bad}' \stackrel{\text{def}}{=} \{a \mid a^{2^i u} = \pm 1 \bmod N\}.$$

Kami sekarang membuktikan apa yang kami klaim di atas.

KLAIM 8.41 $\text{Bad} \subseteq \text{Bad}'$.

Biarkan $\in$ Buruk. Maka $a^u = 1 \bmod N$ atau $a^{2^j u} = -1 \bmod N$ untuk beberapa $j \in \{0, \ldots, r-1\}$. Dalam kasus pertama, $a^{2^i u} = (a^u)^{2^i} = 1 \bmod N$ sehingga $a \in \text{Bad}'$. Dalam kasus kedua, kita mempunyai $j \leq i$ dengan pilihan i. Jika $j = i$ maka jelas merupakan $a \in \text{Bad}'$.

Jika $j < i$ maka $a^{2^i u} = (a^{2^j u})^{2^{i-j}} = 1 \bmod N$ dan $a \in$ Bad'. Karena a sewenang-wenang, ini menunjukkan Buruk $\subseteq$ Buruk'.

KLAIM 8.42 Bad ' adalah subgrup dari $\mathbb{Z}_N^*$.
Jelas $1 \in$ Bad '. Selanjutnya jika $a, b \in$ Bad' maka

$$(ab)^{2^i u} = a^{2^i u} b^{2^i u} = (\pm 1)(\pm 1) = \pm 1 \bmod N$$

dan seterusnya ∈ Bad'. Menurut Lemma 8.36, Bad' adalah sebuah subgrup.

KLAIM 8.43 Bad' adalah subgrup ketat dari $\mathbb{Z}_N^*$.

Jika N adalah bilangan bulat ganjil yang bukan pangkat prima, maka N dapat ditulis sebagai $N = N_1 N_2$ dengan $N_1, N_2 > 1$ ganjil dan $\gcd(N_1, N_2) = 1$. Berdasarkan teorema sisa Cina, misalkan $a \leftrightarrow (a_1, a_2)$ menunjukkan representasi $a \in \mathbb{Z}_N^*$ sebagai elemen $\mathbb{Z}_{N_1}^* \times \mathbb{Z}_{N_2}^*$; yaitu, $a_1 = [a \bmod N_1]$ dan $a_2 = [a \bmod N_2]$. Ambil $a \in$ Bad' sehingga $a^{2^i u} = -1 \bmod N$ (a harus ada berdasarkan definisi i), dan ucapkan $a \leftrightarrow (a_1, a_2)$. Karena $-1 \leftrightarrow (-1, -1)$ kita punya

$$(a_1, a_2)^{2^i u} = (a_1^{2^i u}, a_2^{2^i u}) = (-1, -1),$$

dan sebagainya

$$a_1^{2^i u} = -1 \bmod N_1 \quad \text{and} \quad a_2^{2^i u} = -1 \bmod N_2.$$

Perhatikan elemen $b \in \mathbb{Z}_N^*$ dengan $b \leftrightarrow (a_1, 1)$. Kemudian

$$b^{2^i u} \leftrightarrow (a_1, 1)^{2^i u} = ([a_1^{2^i u} \bmod N_1], 1) = (-1, 1) \not\leftrightarrow \pm 1.$$

Artinya, $b^{2^i u} \neq \pm 1 \bmod N$ dan kita telah menemukan elemen $b \notin$ Bad'. Hal ini membuktikan bahwa Bad' adalah subgrup ketat dari $\mathbb{Z}_N^*$ sehingga, berdasarkan Lemma 8.37, ukuran Bad' (dan dengan demikian ukuran Bad) paling banyak setengah dari ukuran $\mathbb{Z}_N^*$.

Suatu bilangan bulat N adalah pangkat sempurna jika $N = \hat{N}^e$ untuk bilangan bulat $\hat{N}$ dan $e \geq 2$ (di sini $\hat{N}$ tidak diharuskan menjadi bilangan prima, meskipun tentu saja pangkat prima apa pun juga merupakan pangkat sempurna). Algoritma 8.44 memberikan uji primalitas Miller – Rabin. Mengingat hasil ini, algoritma jelas berjalan dalam polinomial waktu dalam $\| N \|$ dan t. Sekarang kita dapat melengkapi pembuktian Teorema 8.33:

BUKTI Jika N adalah bilangan prima ganjil, tidak ada saksi yang kuat sehingga algoritma Miller – Rabin selalu menghasilkan bilangan prima. Jika N genap atau pangkat prima, algoritme selalu mengeluarkan keluaran "komposit". Kasus yang menarik adalah ketika N adalah bilangan bulat ganjil komposit yang bukan merupakan pangkat prima. Pertimbangkan setiap iterasi dari loop dalam. Perhatikan terlebih dahulu jika $a \notin \mathbb{Z}_N^*$ maka $a^u \neq \pm 1 \bmod N$ dan $a^{2^i u} \neq -1 \bmod N$ untuk $i \in \{1, \ldots, r-1\}$. Peluang menemukan salah satu saksi kuat atau unsur yang tidak ada di $\mathbb{Z}_N^*$ paling sedikit $1/2$ (menggunakan Teorema 8.40).

Jadi, probabilitas bahwa algoritme tidak akan pernah mengeluarkan "komposit" pada t iterasi mana pun t adalah paling banyak 2^{-t}.

Asumsi Anjak Piutang

Misalkan GenModulus adalah algoritma waktu polinomial yang, pada masukan 1^n, menghasilkan keluaran (N, p, q) dengan $N = pq$, dan p dan q adalah n-bit bilangan prima kecuali dengan probabilitas yang dapat diabaikan dalam n. (Cara alami untuk melakukan ini adalah dengan menghasilkan dua bilangan prima n-bit yang seragam, seperti yang dibahas sebelumnya, dan kemudian mengalikannya untuk mendapatkan N.) Kemudian pertimbangkan eksperimen berikut untuk algoritma tertentu $\mathcal{A}$ dan parameter n:

Eksperimen pemfaktoran $\mathrm{FACT}_{\mathcal{A},GenModulus}(n)$:

1. Jalankan GenModulus(1^n) untuk mendapatkan (N, p, q).
2. $\mathcal{A}$ diberikan N, dan keluaran $p', q' > 1$.
3. Keluaran percobaan ditetapkan 1 jika $p' q' = N$, dan 0 jika tidak.

Perhatikan bahwa jika keluaran percobaan adalah 1 maka $\{p', q'\} = \{p, q\}$, kecuali p atau q adalah komposit (yang terjadi dengan probabilitas yang dapat diabaikan).

Kami sekarang secara formal mendefinisikan asumsi anjak piutang:

DEFINISI 8.45 Pemfaktoran relatif sulit terhadap GenModulus jika untuk semua algoritme waktu polinomial $\mathcal{A}$ probabilistik terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{Factor}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n).$$

Asumsi pemfaktoran adalah asumsi bahwa terdapat GenModulus yang membuat pemfaktoran sulit dilakukan.

Asumsi RSA

Masalah pemfaktoran telah dipelajari selama ratusan tahun tanpa ditemukan algoritma yang efisien. Meskipun asumsi pemfaktoran memberikan fungsi satu arah (lihat Bagian 8.4), sayangnya asumsi ini tidak secara langsung menghasilkan kriptosistem praktis. (Namun, di Bagian 13.5, kami menunjukkan bagaimana membangun kriptosistem yang efisien berdasarkan permasalahan yang tingkat kesulitannya setara dengan pemfaktoran.) Hal ini telah memotivasi pencarian masalah lain yang kesulitannya terkait dengan tingkat kesulitan pemfaktoran. Masalah yang paling terkenal adalah masalah yang diperkenalkan pada tahun 1978 oleh Rivest, Shamir, dan Adleman dan sekarang disebut masalah RSA untuk menghormati mereka.

Mengingat modulus N dan bilangan bulat $e > 2$ yang relatif prima terhadap $\phi(N)$, Akibat wajar 8.22 menunjukkan bahwa eksponen terhadap modulo pangkat eth N adalah permutasi. Oleh karena itu kita dapat mendefinisikan $[y^{1/e} \bmod N]$ (untuk setiap $y \in \mathbb{Z}_N^*$) sebagai elemen unik dari $\mathbb{Z}_N^*$ yang menghasilkan y ketika dinaikkan ke modulo pangkat eth N; yaitu $x = y^{1/e} \bmod N$ jika dan hanya jika $x^e = y \bmod N$. Masalah RSA, secara informal, adalah menghitung $[y^{1/e} \bmod N]$ untuk modulus N faktorisasi yang tidak diketahui.

Secara formal, biarkan GenRSA menjadi algoritma waktu polinomial probabilistik yang, pada input 1^n, menghasilkan modulus N yang merupakan produk dari dua bilangan prima n-bit, serta bilangan bulat $e, d > 0$ dengan $\gcd(e, \phi(N)) = 1$ dan $ed = 1 \bmod \phi(N)$. (D seperti itu ada karena e adalah modulo yang dapat dibalik $\phi(N)$. Tujuan dari d akan menjadi jelas nanti.) Algoritme mungkin gagal dengan probabilitas yang dapat diabaikan dalam n. Pertimbangkan percobaan berikut untuk algoritma tertentu $\mathcal{A}$ dan parameter n:

Eksperimen RSA RSA-in$v_{\mathcal{A},GenRSA}(n)$:

1. Jalankan GenRSA(1^n) untuk mendapatkan (N, e, d).
2. Pilih seragam $y \in \mathbb{Z}_N^*$.
3. $\mathcal{A}$ diberikan N, e, y, dan keluaran $x \in \mathbb{Z}_N^*$.
4. Keluaran percobaan ditetapkan 1 jika $x^e = y \bmod N$, dan 0 jika tidak.

DEFINISI 8.46 Masalah RSA relatif sulit dibandingkan dengan GenRSA jika untuk semua algoritma waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga Pr[RSA-in$v_{\mathcal{A},GenRSA}(n) = 1] \leq \text{negl}(n)$.

Asumsi RSA adalah terdapat algoritma GenRSA yang relatif sulit mengatasi masalah RSA. Algoritma GenRSA yang sesuai dapat dibangun dari algoritma GenModulus apa pun yang menghasilkan modulus komposit beserta faktorisasinya. Garis besar tingkat tinggi disediakan sebagai Algoritma 8.47, di mana satu-satunya hal yang tidak ditentukan adalah bagaimana tepatnya e dipilih. Faktanya, permasalahan RSA diyakini sulit untuk setiap e yang relatif prima terhadap $\phi(N)$. Kami membahas beberapa pilihan khas e di bawah ini.

ALGORITHM 8.47
GenRSA – high-level outline

Input: Security parameter 1^n
Output: N, e, d as described in the text

$(N, p, q) \leftarrow$ GenModulus(1^n)
$\phi(N) := (p-1)(q-1)$
choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return N, e, d

iyah

Menghubungkan Asumsi RSA dan Anjak Piutang

Katakanlah GenRSA dibangun seperti pada Algoritma 8.47. Jika N dapat difaktorkan, maka kita dapat menghitung $\phi(N)$ dan menggunakannya untuk menghitung $d := [e^{-1} \bmod \phi(N)]$ untuk setiap e (menggunakan Algoritma B.11). Jadi agar masalah RSA menjadi sulit dibandingkan dengan GenRSA, maka masalah pemfaktoran harus sulit dibandingkan dengan GenModulus. Dengan kata lain, permasalahan RSA sangat sulit dibandingkan dengan pemfaktoran; kekerasan pemfaktoran (relatif terhadap GenModulus) berpotensi menjadi asumsi yang lebih lemah dibandingkan kekerasan masalah RSA (relatif terhadap GenRSA).

Bagaimana dengan arah lainnya? Yaitu, apakah sulitnya permasalahan RSA tersirat oleh sulitnya pemfaktoran? Ini masih merupakan pertanyaan terbuka. Hal terbaik yang dapat kami tunjukkan adalah bahwa menghitung d dari N dan e sama sulitnya dengan memfaktorkan.

TEOREMA 8.50 Ada algoritma waktu polinomial probabilistik yang, dengan memasukkan bilangan bulat komposit N dan bilangan bulat e, d dengan $ed = 1 \bmod \phi(N)$, menghasilkan faktor N kecuali dengan probabilitas yang dapat diabaikan dalam $\| N \|$.

BUKTI Teorema ini berlaku untuk semua N, namun untuk kesederhanaan—dan karena ini adalah kasus yang paling relevan dengan kriptografi—di sini kita fokus pada kasus di mana N adalah hasil kali dua bilangan prima (ganjil) yang berbeda. Kami mengandalkan Proposisi 8.36 dan Lemma 8.37 serta fakta-fakta berikut:

Jika N adalah hasil kali dua bilangan prima ganjil yang berbeda, maka 1 mempunyai tepat empat akar kuadrat modulo N. Dua di antaranya adalah akar kuadrat "sepele" ± 1, dan dua di antaranya adalah akar kuadrat "nontrivial".

Akar kuadrat nontrivial apa pun dari 1 dapat digunakan untuk (secara efisien) menghitung faktor N. Hal ini berdasarkan fakta bahwa $y^2 = 1 \bmod N$ menyiratkan

$$0 = y^2 - 1 = (y-1)(y+1) \bmod N,$$

dan jadi $N \mid (y-1)(y+1)$. Namun, $N \nmid (y-1)$ dan $N \nmid (y+1)$ karena $y \neq \pm 1 \bmod N$. Jadi, $\gcd(y-1, N)$ harus sama dengan salah satu faktor prima dari N.

Misalkan $k = ed - 1$ dan perhatikan bahwa $\phi(N) \mid k$. Dengan menggunakan Akibat wajar 8.21, kita mempunyai $x^k = 1 \bmod N$ untuk semua $x \in \mathbb{Z}_N^*$. Misalkan $k = 2^r u$ untuk u bilangan bulat ganjil; perhatikan bahwa $r \geq 1$ karena $\phi(N)$ (dan karenanya k) genap. Strategi kita untuk memfaktorkan N adalah dengan berulang kali memilih $x \in \mathbb{Z}_N^*$ yang seragam dan menghitung barisannya

$$x^u,\ x^{2u},\ \ldots,\ x^{2^r u},$$

semua modulo N. Setiap suku dalam barisan ini adalah kuadrat dari suku sebelumnya dan, seperti yang baru saja kita catat, suku terakhir barisan tersebut adalah 1. Ambil i terbesar (jika ada) yang mana $y \stackrel{\text{def}}{=} [x^{2^i u} \text{mod } N] \neq 1$. Berdasarkan pilihan kita pada i, kita mempunyai $y^2 = 1 \text{ mod } N$. Jika $y \neq -1$ kita telah menemukan akar kuadrat nontrivial dari N , dan kemudian dapat memfaktorkan N seperti yang dibahas di atas.

Semua langkah di atas dapat dilakukan dalam waktu polinomial, sehingga satu-satunya pertanyaan adalah menentukan probabilitas, dibandingkan pilihan x, bahwa y adalah akar kuadrat nontrivial dari N . Misalkan $i \in \{0, \ldots, r - 1\}$ adalah nilai terbesar dari i yang memiliki $x \in \mathbb{Z}_N^*$ sehingga $x^{2^i u} \neq 1 \text{ mod } N$. (Karena u ganjil $(-1)^u = -1 \neq 1 \text{ mod } N$, sehingga definisinya tidak hampa.) Maka untuk semua $x \in \mathbb{Z}_N^*$, kita mempunyai $x^{2^{i+1} u} = 1 \text{ mod } N$ dan seterusnya $[x^{2^i u} \text{ mod } N]$ adalah akar kuadrat dari 1. Definisikan

$$\text{Bad} \stackrel{\text{def}}{=} \{x \mid x^{2^i u} = \pm 1 \bmod N\}$$

dan amati bahwa jika algoritma kita memilih x Bad maka algoritma tersebut menemukan akar kuadrat nontrivial sebesar 1. Kita menunjukkan bahwa Bad adalah subgrup ketat dari $\mathbb{Z}_N^*$; oleh Lemma 8.37, ini menyiratkan bahwa $|\text{Bad}| \leq |\mathbb{Z}_N^*|/2$. Ini berarti $x \notin$ Buruk (dan algoritma menemukan akar kuadrat nontrivial sebesar 1) dengan probabilitas paling sedikit 1/2 pada setiap iterasi. Menggunakan cukup banyak iterasi akan memberikan hasil teorema.

Kami sekarang membuktikan bahwa Bad adalah subgrup ketat dari $\mathbb{Z}_N^*$. Perhatikan terlebih dahulu bahwa Buruk tidak kosong, karena $1 \in$ Buruk. Selanjutnya, jika $x, x' \in$ Buruk maka

$$(xx')^{2^i u} = x^{2^i u}(x')^{2^i u} = (\pm 1) \cdot (\pm 1) = \pm 1 \bmod N,$$

dan $xx' \in \text{Bad}$ and Bad adalah sebuah subgrup. Untuk melihat bahwa Bad adalah subgrup ketat, saya misalkan $x \in \mathbb{Z}_N^*$ sedemikian rupa sehingga $x^{2^i u} \neq 1 \text{ mod } N$ (x tersebut harus ada menurut definisi kita tentang i). Jika $x^{2^i u} \neq -1 \text{ mod } N$, maka $x \notin$ Buruk dan kita selesai. Jika tidak, misalkan $N = pq$ dengan p, q prima, dan misalkan $x \leftrightarrow (x_p, x_q)$ adalah representasi sisa x dalam bahasa Mandarin. Karena $x^{2^i u} = -1 \text{ mod } N$, kita mengetahuinya

$$(x_p, x_q)^{2^i u} = (x_p^{2^i u}, x_q^{2^i u}) = (-1, -1) \leftrightarrow -1.$$

Tapi kemudian $(x_p, 1)$ (atau lebih tepatnya, elemen yang terkait dengannya) tidak ada di Bad sejak itu

$$(x_p, 1)^{2^i u} = ([x_p^{2^i u} \bmod p], 1) = (-1, 1) \not\leftrightarrow \pm 1.$$

Ini melengkapi buktinya.

Dengan asumsi pemfaktoran itu sulit, hasil di atas mengesampingkan kemungkinan penyelesaian masalah RSA secara efisien dengan terlebih dahulu menghitung d dari N dan e. Namun, tidak menutup kemungkinan bahwa mungkin ada beberapa cara yang berbeda untuk mengatasi masalah RSA yang tidak melibatkan (atau menyiratkan) pemfaktoran N. Jadi, berdasarkan pengetahuan kita saat ini, asumsi RSA lebih kuat daripada asumsi pemfaktoran—yaitu, masalah RSA bisa saja diselesaikan dalam waktu polinomial meskipun pemfaktoran tidak bisa. Namun demikian, ketika GenRSA dibangun berdasarkan GenModulus seperti pada Algoritma 8.47, dugaan yang umum adalah bahwa masalah RSA lebih sulit dibandingkan dengan GenRSA setiap kali pemfaktoran relatif sulit terhadap GenModulus.

8.3 ASUMSI KRIPTOGRAFI DALAM GRUP SIKLIK

Pada bagian ini kami memperkenalkan kelas asumsi kekerasan kriptografi dalam kelompok siklik. Kita mulai dengan diskusi umum tentang kelompok siklik, diikuti dengan definisi abstrak dari asumsi yang relevan. Kami kemudian melihat dua contoh kelompok siklik yang konkrit dan banyak digunakan di mana asumsi-asumsi ini diyakini berlaku.

Grup Siklik dan Generator

Misalkan $\mathbb{G}$ adalah grup berorde m yang terbatas. Untuk sembarang $g \in \mathbb{G}$, pertimbangkan himpunannya

$$\langle g \rangle \stackrel{\text{def}}{=} \{g^0, g^1, \ldots\}.$$

(Kami memperingatkan pembaca bahwa jika $\mathbb{G}$ adalah grup tak hingga, $\langle g \rangle$ didefinisikan secara berbeda.) Berdasarkan Teorema 8.14, kita mendapatkan $g^m = 1$. Misal $i \leq m$ adalah bilangan bulat positif terkecil yang $g^i = 1$. Kemudian barisan di atas berulang setelah suku i (yaitu, $g^i = g^0, g^{i+1} = g^1$, dst.), dan seterusnya

$$\langle g \rangle = \{g^0, \ldots, g^{i-1}\}.$$

Kita melihat bahwa $\langle g \rangle$ mengandung paling banyak elemen i. Faktanya, ia mengandung elemen i karena jika $g^j = g^k$ dengan $0 \leq j < k < i$ maka $g^{k-j} = 1$ dan $0 < k - j < i$, bertentangan dengan pilihan kita terhadap i sebagai bilangan bulat positif terkecil yang gi = 1. Tidak sulit untuk memverifikasi bahwa $\langle g \rangle$ adalah subgrup dari $\mathbb{G}$ untuk setiap g; kami menyebut $\langle g \rangle$ sebagai subgrup yang dihasilkan oleh g. Jika ordo subgrup g adalah i, maka i disebut ordo g; itu adalah:

DEFINISI 8.51 Misalkan $\mathbb{G}$ adalah grup berhingga dan $g \in \mathbb{G}$. Orde g adalah bilangan bulat positif terkecil i dengan $g^i = 1$.

Berikut ini adalah analogi yang berguna dari Corollary 8.15 (buktinya sama):

PROPOSISI 8.52 Misalkan $\mathbb{G}$ adalah grup berhingga, dan $g \in \mathbb{G}$ merupakan elemen berorde i. Kemudian untuk bilangan bulat x apa pun, kita memiliki $g^x = g^{[x \bmod i]}$.

Kita bisa membuktikan sesuatu yang lebih kuat:

PROPOSISI 8.53 Misalkan $\mathbb{G}$ adalah grup berhingga, dan $g \in \mathbb{G}$ merupakan elemen berorde i. Maka $g^x = g^y$ jika dan hanya jika $x = y \bmod i$.

BUKTI Jika $x = y \bmod i$ maka $[x \bmod i] = [y \bmod i]$ dan proposisi sebelumnya berbunyi demikian

$$g^x = g^{[x \bmod i]} = g^{[y \bmod i]} = g^y.$$

Untuk arah yang lebih menarik, ucapkan $g^x = g^y$. Maka $1 = g^{x-y} = g^{[x-y \bmod i]}$ (menggunakan proposisi sebelumnya). Karena $[x - y \bmod i] < i$, tetapi i adalah bilangan bulat positif terkecil dengan $g^i = 1$, kita harus mempunyai $[x - y \bmod i] = 0$.

Elemen identitas grup mana pun $\mathbb{G}$ adalah satu-satunya elemen berorde 1, dan menghasilkan grup $\langle 1 \rangle = \{1\}$. Sebaliknya, jika ada elemen $g \in \mathbb{G}$ yang berorde m (m adalah orde dari $\mathbb{G}$), maka $\langle g \rangle = \mathbb{G}$. Dalam kasus ini, kita menyebut $\mathbb{G}$ sebagai grup siklik dan menyatakan bahwa g adalah generator dari $\mathbb{G}$. (Grup siklik mungkin mempunyai banyak generator, sehingga kita tidak dapat membicarakan generatornya.) Jika g adalah generator dari $\mathbb{G}$ maka, menurut definisi, setiap elemen $h \in \mathbb{G}$ sama dengan g^x untuk beberapa $x \in \{0, \ldots, m - 1\}$, titik yang akan kita bahas kembali di bagian berikutnya. Unsur-unsur yang berbeda dalam satu golongan mungkin $\mathbb{G}$ mempunyai ordo yang berbeda. Namun, kami dapat membatasi kemungkinan pesanan ini.

PROPOSISI 8.54 Misalkan $\mathbb{G}$ adalah grup berorde berhingga m, dan katakanlah $g \in \mathbb{G}$ memiliki pesanan i. Lalu saya | m.

BUKTI Berdasarkan Teorema 8.14 kita mengetahui bahwa $g^m = 1 = g^0$. Proposisi 8.53 menyiratkan bahwa $m = 0 \bmod i$.

Akibat wajar berikutnya menggambarkan kekuatan dari hasil ini:

KORELARI 8.55 Jika $\mathbb{G}$ adalah grup berorde prima p, maka $\mathbb{G}$ bersifat siklik. Selanjutnya, seluruh elemen $\mathbb{G}$ kecuali identitas merupakan generator dari $\mathbb{G}$.

BUKTI Berdasarkan Proposisi 8.54, satu-satunya orde elemen yang mungkin dalam $\mathbb{G}$ adalah 1 dan p. Hanya identitas yang berorde 1, sehingga semua elemen lainnya berorde p dan menghasilkan $\mathbb{G}$.

Grup orde prima membentuk satu kelas grup siklik. Grup aditif $\mathbb{Z}_n$, untuk $N > 1$, memberikan contoh lain dari grup siklik (elemen 1 selalu merupakan generator). Teorema berikutnya memberikan tambahan kelas grup siklik yang penting; pembuktian berada di luar cakupan buku ini, namun dapat ditemukan dalam teks aljabar abstrak standar apa pun.

TEOREMA 8.56 Jika p bilangan prima maka $\mathbb{Z}_p^*$ merupakan grup siklik berorde p — 1.

Untuk $p > 3$ bilangan prima, $\mathbb{Z}_p^*$ tidak mempunyai orde prima sehingga hal di atas tidak mengikuti akibat wajar sebelumnya.

Contoh 8.57

Pertimbangkan grup (aditif) $\mathbb{Z}_{15}$. Seperti yang telah kita catat, $\mathbb{Z}_{15}$ bersifat siklik dan elemen 1 adalah generator karena $15 \,.\, 1 = 0$ mod 15 dan $i \,.\, 1 = i \neq 0$ mod 15 untuk sembarang $0 < i < 15$ (ingat bahwa dalam grup ini identitasnya adalah 0).

$\mathbb{Z}_{15}$ memiliki generator lain. Misalnya, $\langle 2 \rangle = \{0, 2, 4, \ldots, 14, 1, 3, \ldots, 13\}$ dan 2 juga merupakan generator.

Tidak semua elemen menghasilkan $\mathbb{Z}_{15}$. Misalnya, elemen 3 mempunyai orde 5 karena $5 \cdot 3 = 0$ mod 15, sehingga 3 tidak menghasilkan $\mathbb{Z}_{15}$. Subgrup $\langle 3 \rangle$ terdiri dari 5 elemen $\{0, 3, 6, 9, 12\}$, dan ini memang merupakan subgrup dengan modul penjumlahan 15. Elemen 10 berorde 3 karena $3 \cdot 10 = 0$ mod 15, dan subgrup $\langle 10 \rangle$ terdiri dari 3 elemen $\{0, 5, 10\}$. Urutan subkelompok (yaitu, 5 dan 3) membagi $|\mathbb{Z}_{15}| = 15$ sebagaimana disyaratkan oleh Proposisi 8.54.

Contoh 8.58

Misalkan grup (perkalian) $\mathbb{Z}_{15}^*$ berorde $(5 — 1)(3 — 1) = 8$. Kita mempunyai $\langle 2 \rangle = \{1, 2, 4, 8\}$, sehingga orde dari 2 adalah 4. Sesuai kebutuhan dengan Proposisi 8.54, 4 membagi 8. 0

Contoh 8.59

Pertimbangkan grup (aditif) $\mathbb{Z}_p$ orde prima p. Kita tahu bahwa grup ini bersifat siklik, namun Akibat wajar 8.55 memberi tahu kita lebih banyak: yaitu, setiap elemen kecuali 0 adalah generator. Memang, untuk setiap $h \in \{1, \ldots, p-1\}$ dan bilangan bulat $i > 0$ kita mempunyai $ih = 0$ mod p jika dan hanya jika p ih. Tapi kemudian Proposisi 8.3 mengatakan bahwa antara $p \mid h$ atau $p \mid i$. Yang pertama tidak dapat muncul (karena $h < p$), dan bilangan bulat positif terkecil yang dapat muncul adalah $i = p$. Kita telah menunjukkan bahwa setiap elemen bukan nol h mempunyai orde p (sehingga menghasilkan $\mathbb{Z}_p$), sesuai dengan Korelari 8.55.

Contoh 8.60

Perhatikan grup (perkalian) $\mathbb{Z}_7^*$, yang merupakan siklik berdasarkan Teorema 8.56. Kita mempunyai $\langle 2 \rangle = \{1, 2, 4\}$, sehingga 2 bukanlah generator. Namun,

$$\langle 3 \rangle = \{1, 3, 2, 6, 4, 5\} = \mathbb{Z}_7^*,$$

dan 3 adalah generator dari $\mathbb{Z}_7^*$.

Contoh berikut bergantung pada materi Bagian 8.1.5.

Contoh 8.61

Misalkan $\mathbb{G}$ adalah grup siklik berorde n, dan misalkan g adalah generator dari $\mathbb{G}$. Maka pemetaan $f : \mathbb{Z}_n \to \mathbb{G}$ yang diberikan oleh $f(a) = g^a$ adalah isomorfisme antara $\mathbb{Z}_n$ dan $\mathbb{G}$. Memang benar, untuk $a, a' \in \mathbb{Z}_n$ yang kita punya

$$f(a + a') = g^{[a+a' \bmod n]} = g^{a+a'} = g^a \cdot g^{a'} = f(a) \cdot f(a').$$

Oebjektivitas f dapat dibuktikan dengan fakta bahwa n adalah orde g. Contoh sebelumnya menunjukkan bahwa semua grup siklik dengan orde yang sama adalah isomorfik dan karenanya sama dari sudut pandang aljabar. Kami menekankan bahwa hal ini tidak benar dalam pengertian komputasi, dan khususnya isomorfisme $f^{-1} : \mathbb{Z}_n \to \mathbb{G}$ (yang kita tahu pasti ada) tidak perlu dihitung secara efisien. Poin ini akan menjadi lebih jelas dari pembahasan pada bagian di bawah ini dan juga Bab 9.

Asumsi Logaritma Diskrit/Diffie–Hellman

Kami sekarang memperkenalkan beberapa masalah komputasi yang dapat didefinisikan untuk setiap kelas grup siklik. Kami akan menjaga diskusi di bagian ini tetap abstrak, dan mempertimbangkan contoh-contoh spesifik dari kelompok-kelompok yang permasalahannya diyakini sulit di Bagian 8 3.

Kami membiarkan $\mathcal{G}$ algoritma generasi grup generik, waktu polinomial. Ini adalah algoritma yang, pada input 1^n, mengeluarkan deskripsi grup siklik $\mathbb{G}$, ordenya q (dengan $\| q \| = n$), dan generator $g \in \mathbb{G}$. Deskripsi grup siklik menentukan bagaimana elemen grup direpresentasikan sebagai sedikit- string; kita berasumsi bahwa setiap elemen grup diwakili oleh bit-string yang unik. Kita memerlukan algoritma yang efisien (yaitu, algoritma yang berjalan dalam polinom waktu n) untuk menghitung operasi grup di $\mathbb{G}$, serta untuk menguji apakah bit-string tertentu mewakili elemen $\mathbb{G}$. Perhitungan operasi grup yang efisien berarti algoritma yang efisien untuk eksponensial dalam $\mathbb{G}$ dan untuk pengambilan sampel elemen seragam $h \in \mathbb{G}$ (cukup pilih seragam $x \in \mathbb{Z}_q$ dan atur $h := g^x$).

Seperti yang dibahas di akhir bagian sebelumnya, meskipun semua grup siklik dengan orde tertentu bersifat isomorfik, representasi grup menentukan kompleksitas komputasi operasi matematika dalam grup tersebut.

Jika $\mathbb{G}$ adalah grup siklik berorde q dengan generator g, maka $\{g^0, g^1, \ldots, g^{q-1}\}$ semuanya adalah $\mathbb{G}$. Ekuivalennya, untuk setiap $h \in \mathbb{G}$ terdapat $x \in \mathbb{Z}_q$ unik sehingga $g^x = h$. Ketika grup yang mendasari $\mathbb{G}$ dipahami dari konteksnya, kita menyebutnya x logaritma diskrit dari h terhadap g dan menulis $x = \log_g h$. (Logaritma dalam hal ini disebut "diskrit" karena mengambil nilai dalam rentang berhingga, berbeda dengan logaritma "standar" dari kalkulus yang nilainya berkisar pada himpunan bilangan real tak terhingga.) Perhatikan bahwa jika $g^{x'} = h$ untuk suatu bilangan bulat sembarang x', maka $[x' \bmod q] = \log_g h$.

Logaritma diskrit mematuhi banyak aturan yang sama dengan logaritma "standar". Misalnya $\log_g 1 = 0$ (di mana 1 adalah identitas $\mathbb{G}$); untuk bilangan bulat r apa pun, kita mempunyai $\log_g h^r = [r \cdot \log_g h \bmod q]$; dan $\log_g (h_1 h_2) = [(\log_g h_1 + \log_g h_2) \bmod q]$. Masalah logaritma diskrit dalam grup siklik $\mathbb{G}$ dengan generator g adalah menghitung $\log_g h$ untuk elemen seragam $h \in \mathbb{G}$. Perhatikan eksperimen berikut untuk algoritma pembangkitan grup $\mathcal{G}$, algoritma $\mathcal{A}$, dan parameter n:

Eksperimen logaritma diskrit $\mathrm{Dlog}_{\mathcal{A},\mathcal{G}}(n)$:

1. Jalankan $\mathcal{G}(1^n)$ untuk mendapatkan $(\mathbb{G}, q, g)$, dimana $\mathbb{G}$ adalah grup siklik berorde q (dengan $\| q \| = n$), dan g adalah generator dari $\mathbb{G}$.
2. Pilih seragam $h \in \mathbb{G}$.
3. A diberikan $\mathbb{G}, q, g, h$, dan keluaran $x \in \mathbb{Z}_q$.
4. Keluaran percobaan ditetapkan 1 jika $g^x = h$, dan 0 jika tidak.

DEFINISI 8.62 Kita mengatakan bahwa permasalahan logaritma diskrit relatif sulit terhadap G jika untuk semua algoritma waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi negl yang dapat diabaikan sehingga $\Pr[\mathrm{Dlog}_{\mathcal{A},\mathcal{G}}(n) = 1] \leq \mathrm{negl}(n)$.

Asumsi logaritma diskrit hanyalah asumsi bahwa terdapat suatu permasalahan yang sulit diselesaikan dengan logaritma diskrit. Dua bagian berikut membahas beberapa kandidat algoritma $\mathcal{G}$ pembangkitan kelompok yang diyakini menjadi kasusnya.

Masalah Diffie – Hellman. Masalah yang disebut Diffie – Hellman terkait, namun tidak diketahui setara, dengan masalah komputasi logaritma diskrit. Ada dua varian penting: masalah komputasi Diffie–Hellman (CDH) dan masalah keputusan Diffie–Hellman (DDH).

Perbaiki grup siklik $\mathbb{G}$ dan generator $g \in \mathbb{G}$. Elemen yang diketahui $h_1, h_2 \in \mathbb{G}$, tentukan $DH_g(h_1, h_2) \stackrel{\text{def}}{=} g^{\log_g h_1 \cdot \log_g h_2}$. Artinya, jika $h_1 = g^{x_1}$ dan h2 = gx2 maka

$$\mathrm{DH}_g(h_1, h_2) = g^{x_1 \cdot x_2} = h_1^{x_2} = h_2^{x_1}.$$

Masalah CDH adalah menghitung $DH_g(h_1, h_2)$ untuk h_1 dan h_2 yang seragam. Kerasnya masalah ini dapat diformalkan melalui eksperimen alami; kami meninggalkan detailnya sebagai latihan.

Jika soal logaritma diskrit relatif terhadap beberapa G mudah dilakukan, maka soal CDH juga mudah: jika diberikan h_1 dan h_2, hitung dulu $x_1 := \log_g h_1$lalu keluarkan jawabannya $h_2^{x_1}$. Sebaliknya, tidak jelas apakah masalah kekerasan diskritlogaritma berarti bahwa masalah CDH juga sulit.

Masalah DDH, secara kasar, adalah membedakan $DH_g(h_1, h_2)$ dari elemen grup seragam ketika h_1, h_2 seragam. Artinya, jika diberikan h_1, h_2 dan elemen golongan ketiga h' yang seragam, masalahnya adalah menentukan apakah $h' = DH_g(h_1, h_2)$ atau apakah h' dipilih secara seragam dari $\mathbb{G}$. Secara formal:

DEFINISI 8.63 Kita mengatakan bahwa masalah DDH relatif sulit jika $\mathcal{G}$ untuk semua algoritma waktu polinomial probabilistik $\mathcal{A}$terdapat fungsi yang dapat diabaikan sehingga

$$\left| \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] \right| \leq \text{negl}(n),$$

di mana dalam setiap kasus probabilitas diambil alih eksperimen di mana $\mathcal{G}(1^n)$ keluaran $(\mathbb{G}, q, g)$, dan kemudian seragam $x, y, z \in \mathbb{Z}_q$ dipilih. (Perhatikan bahwa jika z seragam di $\mathbb{Z}_q$, maka g^z terdistribusi seragam di $\mathbb{G}$.)

Kita telah melihat bahwa jika permasalahan logaritma diskrit relatif mudah, maka permasalahan CDH juga mudah. Demikian pula, jika masalah CDH relatif mudah, maka masalah DDH juga mudah; Anda diminta untuk menunjukkannya pada Latihan 8.15. Namun hal yang sebaliknya tampaknya tidak benar, dan ada contoh kelompok di mana soal logaritma diskrit dan CDH diyakini sulit meskipun soal DDH mudah.

Menggunakan Grup Prime-Order

Ada berbagai (kelas) grup siklik di mana masalah logaritma diskrit dan Diffie–Hellman diyakini sulit. Namun, ada preferensi untuk kelompok siklik dengan orde prima, karena alasan yang akan kami jelaskan sekarang.

Salah satu alasan untuk memilih kelompok dengan orde prima adalah karena, dalam arti tertentu, permasalahan logaritma diskrit paling sulit dilakukan pada kelompok tersebut. Hal ini merupakan konsekuensi dari algoritma Pohlig–Hellman yang dijelaskan pada Bab 9, yang menunjukkan bahwa permasalahan logaritma diskrit pada kelompok orde q menjadi lebih mudah jika q memiliki faktor prima (kecil). Hal ini tidak berarti bahwa permasalahan logaritma diskrit mudah dilakukan pada kelompok orde nonprima; itu hanya berarti masalahnya menjadi lebih mudah.

Terkait dengan hal di atas adalah kenyataan bahwa permasalahan DDH mudah jika orde grup q memiliki faktor prima yang kecil. Motivasi kedua dalam menggunakan grup orde prima adalah karena menemukan generator dalam grup tersebut adalah hal yang mudah. Hal ini mengikuti Corollary 8.55, yang mengatakan bahwa setiap elemen grup orde prima (kecuali identitas) adalah generator. Sebaliknya, menemukan generator grup siklik sembarang secara efisien memerlukan faktorisasi orde grup agar diketahui.

Bukti keamanan untuk beberapa konstruksi kriptografi memerlukan komputasi invers perkalian eksponen tertentu (kita akan melihat contohnya di Bagian 8.4). Jika orde grupnya prima, eksponen apa pun yang bukan nol akan dapat dibalik, sehingga penghitungan ini dapat dilakukan.

Alasan terakhir untuk bekerja dengan kelompok orde prima berlaku dalam situasi ketika masalah pengambilan keputusan Diffie–Hellman seharusnya sulit. Memperbaiki grup $\mathbb{G}$ dengan generator g, masalah DDH bermuara pada membedakan antara tupel berbentuk $(h_1, h_2, DH_g(h_1, h_2))$ untuk seragam h_1, h_2, dan tupel berbentuk (h_1, h_2, y) , untuk seragam h_1, h_2, y. Syarat yang diperlukan agar soal DDH menjadi sulit adalah $DH_g(h_1, h_2)$ dengan sendirinya tidak dapat dibedakan dari elemen golongan seragam. Kita dapat menunjukkan bahwa $DH_g(h_1, h_2)$ "mendekati" seragam (dalam pengertian yang tidak kita definisikan di sini) ketika orde grup q adalah prima, sesuatu yang sebaliknya tidak benar.

Bekerja di (Subkelompok) $\mathbb{Z}_{\mathrm{P}}^*$

Grup dengan bentuk $\mathbb{Z}_{\mathrm{P}}^*$, untuk p prima, memberikan satu kelas grup siklik yang permasalahan logaritma diskritnya diyakini sulit. Secara konkret, misalkan suatu algoritma yang, pada masukan 1^n, memilih n-bit prima p yang seragam, dan mengeluarkan keluaran p dan orde grup $q = p - 1$ bersama dengan generator g dari $\mathbb{Z}_{\mathbf{P}}^*$. (Bagian 8.2.1 membahas algoritma yang efisien untuk memilih bilangan prima acak, dan menunjukkan cara mencari generator $\mathbb{Z}_{\mathrm{P}}^*$ dengan faktorisasi $p - 1$ secara efisien.) Representasi $\mathbb{Z}_{\mathbf{P}}^*$ di sini adalah representasi sepele di mana elemen-elemennya adalah direpresentasikan sebagai bilangan bulat antara 1 dan p — 1. Diperkirakan bahwa permasalahan logaritma diskrit relatif sulit terhadap $\mathcal{G}$ jenis ini.

Namun, grup siklik $\mathbb{Z}_{\mathrm{P}}^*$ (untuk $p > 3$ prima) tidak mempunyai orde prima. (Preferensi terhadap kelompok orde prima telah dibahas pada bagian sebelumnya.) Yang lebih problematis, permasalahan pengambilan keputusan Diffie–Hellman, secara umum, tidak sulit untuk dilakukan pada kelompok seperti itu, dan oleh karena itu permasalahan tersebut tidak dapat diterima. untuk aplikasi kriptografi berdasarkan asumsi DDH yang akan kita bahas di bab selanjutnya.

Masalah ini dapat diatasi dengan menggunakan subgrup orde utama $\mathbb{Z}_{\mathrm{P}}^*$. Misalkan $p = rq + 1$ dimana p dan q keduanya bilangan prima. Kita buktikan bahwa $\mathbb{Z}_{\mathrm{P}}^*$ memiliki subgrup $\mathbb{G}$ berorde q yang diberikan oleh himpunan residu ke-r modulo p, yaitu himpunan elemen $\{[h^r \bmod p] \mid h \in \mathbb{Z}_{\mathrm{P}}^*\}$ yang sama dengan pangkat ke-r dari beberapa h ∈ $\mathbb{Z}_{\mathrm{P}}^*$.

TEOREMA 8.64 Misalkan $p = rq + 1$ dengan p, q prima. Kemudian

$$\mathbb{G} \stackrel{\text{def}}{=} \{[h^r \bmod p] \mid h \in \mathbb{Z}_p^*\}$$

adalah subgrup dari $\mathbb{Z}_{\mathrm{P}}^*$ orde q.

BUKTI Pembuktian bahwa $\mathbb{G}$ adalah subgrup sangatlah jelas dan dihilangkan. Kita buktikan bahwa $\mathbb{G}$ mempunyai orde q dengan menunjukkan bahwa fungsi $f_r: \mathbb{Z}_P^* \rightarrow \mathbb{G}$ yang didefinisikan oleh $f_r(g) = [g^r \bmod p]$ adalah fungsi r-ke-1. (Karena $|\mathbb{Z}_P^*| = p - 1$, ini menunjukkan bahwa $|\mathbb{G}| = (p-1)/r = q$.) Untuk melihatnya, misalkan g adalah generator dari $\mathbb{Z}_P^*$ sehingga $g^0, \ldots, g^{p-2}$ adalah semua elemen dari $\mathbb{Z}_P^*$. Dengan Proposisi 8.53 kita mempunyai $(g^i)^r = (g^j)^r$ jika dan hanya jika $ir = jr \bmod (p-1)$ atau, setara dengan, $p-1|(i-j)r$.

Karena $p - 1 = rq$, ini setara dengan $q|(i-j)$. Untuk sembarang $j \in \{0, \ldots, p-2\}$, ini berarti himpunan nilai $i \in \{0, \ldots, p-2\}$ yang mana $(g^i)^r = (g^j)^r$ merupakan himpunan r nilai yang berbeda

$$\{j,\ j+q,\ j+2q,\ \ldots,\ j+(r-1)q\},$$

semua modulo tereduksi p — 1. (Perhatikan bahwa $j + rq = j \bmod (p$ — $1).$) Ini membuktikan bahwa f_r adalah fungsi r-ke-1.

Selain menunjukkan keberadaan subkelompok yang sesuai, teorema ini juga menyiratkan bahwa mudah untuk menghasilkan elemen seragam dari $\mathbb{G}$ dan menguji apakah elemen tertentu dari $\mathbb{Z}_P^*$ terletak di $\mathbb{G}$. Secara khusus, pemilihan elemen seragam dari $\mathbb{G}$ dapat dilakukan dengan memilih seragam $h \in \mathbb{Z}_P^*$ dan komputasi $[h^r \ mod\ p]$. Karena $\mathbb{G}$ mempunyai orde prima, setiap elemen di $\mathbb{G}$ kecuali identitasnya merupakan generator dari $\mathbb{G}$. Terakhir, kita dapat menentukan apakah $h \in \mathbb{Z}_P^*$ juga ada di $\mathbb{G}$ dengan memeriksa apakah $h^q \overset{m}{=} 1 \bmod i$. Untuk memastikan hal ini berhasil, misalkan $h = g^i$ untuk g generator $\mathbb{Z}_P^*$ dan $i \in \{0, \ldots, i$ — 2}. Kemudian

$$\begin{aligned} h^q = 1 \bmod p &\iff g^{iq} = 1 \bmod p \\ &\iff iq = 0 \bmod (p-1) \iff rq \,|\, iq \iff r \,|\, i, \end{aligned}$$

menggunakan Proposisi 8.53. Jadi $h = g^i = g^{cr} = (g^c)^r$ untuk beberapa c, dan $h \in G$.

Algoritma 8.65 merangkum pembahasan di atas. Dalam algoritme, kita misalkan n menyatakan panjang q, orde grup, dan misalkan ` menunjukkan panjang p, modulus yang digunakan. Hubungan antara parameter-parameter ini dibahas di bawah.

ALGORITHM 8.65
A group-generation algorithm $\mathcal{G}$

Input: Security parameter 1^n, parameter $\ell = \ell(n)$
Output: Cyclic group $\mathbb{G}$, its (prime) order q, and a generator g

generate a uniform n-bit prime q
generate an ℓ-bit prime p such that $q \,|\, (p-1)$
  // we omit the details of how this is done
choose a uniform $h \in \mathbb{Z}_p^*$ with $h \neq 1$
set $g := [h^{(p-1)/q} \bmod p]$
return p, q, g // $\mathbb{G}$ is the order-q subgroup of $\mathbb{Z}_p^*$

Memilih ℓ. Misalkan $n = \|q\|$ dan $\ell = \|p\|$. Dua jenis algoritma adalah dikenal karena menghitung logaritma diskrit dalam subgrup orde-q dari $\mathbb{Z}_{\mathrm{P}}^*$ (lihat Bagian 9.2): subgrup yang berjalan dalam waktu $\mathcal{O}\left(\sqrt{q}\right) = \mathcal{O}(2^{n/2})$ dan subgrup yang berjalan dalam waktu $2^{\mathcal{O}\left((\log p)^{1/3}\cdot(\log\log p)^{2/3}\right)} = 2^{\mathcal{O}\left(\ell^{1/3}\cdot(\log \ell)^{2/3}\right)}$Memperbaiki beberapa parameter keamanan yang diinginkan n, parameter ℓ harus dipilih untuk menyeimbangkan waktu ini. (Jika ℓ lebih kecil, keamanan berkurang; jika ℓ lebih besar, operasi di $\mathbb{G}$ akan menjadi kurang efisien tanpa peningkatan keamanan apa pun.) Lihat juga Bagian 9.3.

Dalam praktiknya, nilai-nilai standar (misalnya, direkomendasikan oleh NIST) untuk p, q, dan generator g digunakan, dan tidak perlu membuat parameter sendiri.

Contoh 8.66

Perhatikan grup $\mathbb{Z}_{11}^*$ berorde 10. Mari kita coba mencari generator grup ini. Pertimbangkan untuk mencoba 2:

Powers of 2:	2^0	2^1	2^2	2^3	2^4	2^5	2^6	2^7	2^8	2^9
Values:	1	2	4	8	5	10	9	7	3	6

(Semua nilai di atas dihitung modulo 11.) Kita beruntung pertama kali—angka 2 adalah generator! Mari kita coba 3:

Powers of 3:	3^0	3^1	3^2	3^3	3^4	3^5	3^6	3^7	3^8	3^9
Values:	1	3	9	5	4	1	3	9	5	4

Kita melihat bahwa 3 bukanlah generator dari keseluruhan grup. Sebaliknya, ini menghasilkan subgrup $\mathrm{G} = \{1, 3, 4, 5, 9\}$ berorde 5. Sekarang, mari kita lihat apa yang terjadi dengan 10:

Powers of 10:	10^0	10^1	10^2	10^3	10^4	10^5	10^6	10^7	10^8	10^9
Values:	1	10	1	10	1	10	1	10	1	10

Dalam hal ini kami menghasilkan subgrup berorde 2.

Untuk tujuan kriptografi kami ingin bekerja dalam grup orde utama. Karena 11 = 2 5 + 1 kita dapat menerapkan Teorema 8.64 dengan q = 5 dan r = 2, atau dengan q = 2 dan r = 5. Dalam kasus pertama, teorema tersebut menyatakan bahwa kuadrat semua elemen Z∗ harus memberikan subkelompok urutan 5. Hal ini dapat dengan mudah diverifikasi:

Element:	1	2	3	4	5	6	7	8	9	10
Square:	1	4	9	5	3	3	5	9	4	1

Kita telah melihat di atas bahwa 3 adalah generator dari subgrup ini. (Faktanya, karena subgrupnya prima, setiap elemen subgrup selain 1 adalah generator subgrup tersebut.) Dengan mengambil $q = 2$ dan $r = 5$, Teorema 8.64 memberitahu kita bahwa mengambil pangkat 5 akan menghasilkan subgrup berorde 2. Kita dapat memeriksa bahwa ini memberikan subgrup order-2 yang dihasilkan oleh 10 yang kita temui sebelumnya.

Subkelompok bidang terbatas. Masalah logaritma diskrit juga diyakini sulit dilakukan dalam kelompok perkalian bidang berhingga dengan karakteristik besar ketika representasi polinomial digunakan. Ingatlah bahwa untuk setiap bilangan prima p dan bilangan bulat $k \geq 1$ terdapat sebuah medan (unik) ${F_p}^k$ berorde p^k; grup perkalian ${\mathrm{F}^*}_{p^k}$ dari bidang tersebut adalah grup siklik berorde $p^k - 1$. Jika q adalah faktor prima besar dari $p^k - 1$ maka Teorema 8.64 menunjukkan bahwa ${\mathrm{F}^*}_{p^k}$ mempunyai subgrup siklik berorde q. (Satu-satunya sifat Z_p^* yang kami gunakan dalam pembuktian teorema tersebut adalah bahwa Z_p^* bersifat siklik.) Hal ini menawarkan pilihan lain dari grup orde prima di mana permasalahan logaritma diskrit dan Diffie–Hellman diyakini sulit. Perlakuan kami terhadap Z_p^* di bagian ini sesuai dengan kasus khusus $k = 1$.

Kurva Elips

Kelompok-kelompok yang telah kita konsentrasikan sejauh ini semuanya didasarkan langsung pada aritmatika modular. Kelas grup lain yang penting untuk kriptografi diberikan oleh grup yang terdiri dari titik-titik pada kurva elips. Kelompok seperti ini sangat menarik dari perspektif kriptografi karena, berbeda dengan Z_p^* atau kelompok perkalian dari bidang berhingga, saat ini tidak ada algoritma waktu sub-eksponensial yang diketahui untuk memecahkan masalah logaritma diskrit dalam kelompok kurva elips jika dipilih dengan tepat. . (Lihat Bagian 9.3 untuk pembahasan lebih lanjut.) Untuk kriptosistem yang didasarkan pada logaritma diskrit atau asumsi Diffie-Hellman, hal ini berarti bahwa implementasi yang didasarkan pada kelompok kurva elips akan lebih efisien dibandingkan implementasi yang didasarkan pada subkelompok orde prima. Z_p^* pada tingkat keamanan tertentu. Pada bagian ini kami hanya memberikan pengenalan singkat tentang bidang ini. Pemahaman yang lebih dalam mengenai isu-isu yang dibahas di sini memerlukan matematika yang lebih canggih daripada yang dapat kita asumsikan oleh pembaca. Mereka yang tertarik untuk mendalami topik ini lebih jauh disarankan untuk membaca referensi di akhir bab ini.

Misalkan $p \geq 5$ adalah bilangan prima.[3] Perhatikan persamaan E dalam variabel x dan y dalam bentuk:

$$y^2 = x^3 + Ax + B \bmod p, \qquad (8.1)$$

dimana $A, B\ Z_p$ adalah konstanta dengan $4A^3 + 27B^2 \neq 0\ mod\ p$. (Hal ini memastikan bahwa persamaan $x^3 + Ax + B = 0$ mod p tidak memiliki akar yang berulang.) Misal $E(Z_p)$ menyatakan himpunan pasangan $(x, y) \in Z_p \times Z_p$ yang memenuhi persamaan di atas dengan nilai khusus O yang tujuannya akan kita bahas sebentar lagi; itu adalah,

$$E(\mathbb{Z}_p) \stackrel{\text{def}}{=} \{(x, y) \mid x, y \in \mathbb{Z}_p \text{ and } y^2 = x^3 + Ax + B \bmod p\} \cup \{\mathcal{O}\}.$$

Elemen $E(Z_p)$ disebut titik pada kurva elips E yang didefinisikan oleh Persamaan (8.1), dan O disebut titik tak terhingga.

TEOREMA 8.70 (Terikat Hasse) Misalkan p adalah bilangan prima, dan misalkan E adalah kurva elips di atas Z_p. Maka $p + 1 - 2\sqrt{p} \leq |E(Z_p)| \leq p + 1 + 2\sqrt{p}$.

Batasan ini menyiratkan bahwa selalu mudah untuk menemukan titik pada kurva elips tertentu $y^2 = f(x) \bmod p$: cukup pilih seragam $x \in Z_p$, periksa apakah $f(x)$ adalah 0 atau residu kuadrat, dan jika demikian misalkan y adalah akar kuadrat dari $f(x)$. (Algoritma untuk menentukan residuositas kuadrat dan menghitung akar kuadrat modulo a prime dibahas di Bab 13.) Karena titik pada kurva elips banyak sekali, kita tidak perlu mencoba terlalu banyak nilai x sebelum menemukan titik. Batas Hasse hanya memberikan rentang ukuran grup kurva elips.

Namun, untuk bilangan prima tetap, urutan kurva elips acak di atas Zp (yaitu, kurva yang ditentukan oleh Persamaan (8.1) di mana A, B dipilih secara seragam di Zp dengan tunduk pada batasan 4A3 + 27B2 = 0 mod p) secara heuristik ditemukan "mendekati" terdistribusi seragam dalam interval Hasse. Terdapat juga algoritma yang efisien yang deskripsi dan analisisnya berada di luar cakupan buku ini untuk menghitung jumlah titik pada kurva elips. Hal ini menyarankan pendekatan terhadap pembuatan parameter kurva elips seperti pada Algoritma 8.71.

ALGORITHM 8.71
Elliptic-curve group-generation algorithm $\mathcal{G}$

Input: Security parameter 1^n
Output: Cyclic group $\mathbb{G}$, its (prime) order q, and a generator g

generate a uniform n-bit prime p
until q is an n-bit prime **do**:
  choose $A, B \leftarrow \mathbb{Z}_p$ with $4A^3 + 27B^2 \neq 0 \bmod p$,
    defining elliptic curve E as in Equation (8.1)
  let q be the number of points on $E(\mathbb{Z}_p)$
choose $g \in E(\mathbb{Z}_p) \setminus \{\mathcal{O}\}$
return (A, B, p), q, g // $\mathbb{G}$ is the elliptic-curve group of E

Kelas kurva tertentu dianggap lemah secara kriptografis dan harus dihindari. Ini termasuk grup kurva elips di atas Z_p yang ordenya sama dengan p (kurva anomali) atau $p + 1$ (kurva supersingular), atau yang ordonya membagi $p^k - 1$ *untuk* k "*kecil*". Pembahasan lengkap berada di luar cakupan buku ini. Dalam praktiknya, kurva standar (seperti yang direkomendasikan oleh NIST) digunakan, dan tidak disarankan untuk membuat kurva sendiri.

Pertimbangan Efisiensi

Kami menyimpulkan bagian ini dengan diskusi singkat tentang beberapa peningkatan efisiensi standar saat menggunakan kurva elips.

Kompresi titik. Pengamatan yang bermanfaat adalah bahwa jumlah bit yang diperlukan untuk mewakili suatu titik pada kurva elips dapat dikurangi hampir setengahnya. Untuk melihatnya, perhatikan bahwa untuk setiap titik (x, y) pada kurva elips $E : y^2 = f(x) \bmod p$ paling banyak terdapat dua titik pada kurva yang mempunyai koordinat x: $yaitu, (x, y)$ dan $(x, -y)$. (Ada kemungkinan bahwa $y = 0$ dalam hal ini adalah titik yang sama.) Jadi, kita dapat menentukan titik mana pun $P = (x, y)$ berdasarkan koordinat x dan sedikit b yang membedakan antara (paling banyak) dua titik tersebut. kemungkinan nilai koordinat y-nya. Salah satu cara mudah untuk melakukan ini adalah dengan menetapkan $b = 0$ jika $y < p/2$ dan $b = 1$ sebaliknya. Mengingat x dan b kita dapat memulihkan P dengan menghitung dua akar kuadrat y_1, y_2 dari persamaan $y^2 = f(x) \ mod \ p$; karena $y_1 = — y_2 \ mod \ p \ dan \ y_1 = p — y_2$, tepat salah satu dari y_1, y_2, akan lebih kecil dari $p/2$.

Koordinat proyektif. Merepresentasikan titik-titik kurva elips seperti yang selama ini kita lakukan di mana titik P pada kurva elips digambarkan oleh sepasang elemen bidang (x, y) disebut menggunakan koordinat affine. Ada cara alternatif untuk merepresentasikan titik, menggunakan koordinat proyektif, yang dapat menawarkan peningkatan efisiensi. Meskipun representasi alternatif ini dapat dimotivasi secara matematis, kami memperlakukannya hanya sebagai alat bantu komputasi yang berguna.

Titik-titik dalam koordinat proyektif direpresentasikan menggunakan tiga elemen Z_p. Fitur yang menarik adalah suatu titik memiliki kelipatan representasi. Saat menggunakan koordinat proyektif standar, titik $P \neq \mathcal{O}$ dengan representasi (x, y) dalam koordinat affine diwakili oleh tupel apa pun $(X, Y, Z) \in Z_p^3$ dengan X/Z = x mod p dan Y/Z = y mod p. Titik tersebut diwakili oleh tupel mana pun $(0, Y, 0)$ dengan $Y \neq 0$, dan ini adalah satu-satunya titik (X, Y, Z) dengan $Z = 0$. Kita dapat dengan mudah menerjemahkan antar sistem koordinat: (x, y) dalam koordinat affine dapat dipetakan ke $(x, y, 1)$ dalam koordinat proyektif, dan (X, Y, Z) $(dengan \ Z \neq 0)$ dalam koordinat proyektif dipetakan ke representasi $([X/Z \ mod \ p \ ], [Y/Z \ mod \ p \ ])$ dalam koordinat affine.

Keuntungan utama menggunakan koordinat proyektif adalah kita dapat menjumlahkan titik tanpa harus menghitung invers modulo p. (Menambahkan titik dalam koordinat affine memerlukan komputasi invers; lihat Proposisi 8.68.) Kita mencapai hal ini dengan mengeksploitasi fakta bahwa titik memiliki banyak representasi. Untuk memahaminya, mari kita kerjakan hukum penjumlahan untuk dua titik $P_1 = (X_1, Y_1, Z_1)$ dan $P_2 = (X_2, Y_2, Z_2$ dengan P_1, P_2/= & (jadi Z_1, Z_2 /= 0) dan P_1/= ±P_2 (jadi X_1/Z_1 /= X_2/Z_2 mod p). (Jika P_1 atau P_2 sama dengan &, penjumlahan adalah hal sepele. Kasus P_1 = ±P_2juga dapat ditangani, namun kita menghilangkan detailnya di sini.) Kita dapat menyatakan P_1 dan P_2 sebagai (X_1/Z_1, Y_1/Z_1) dan (X_2/Z_2, Y_2/Z_2) dalam koordinat affine, jadi

$$P_3 \stackrel{\text{def}}{=} P_1 + P_2 = \big(m^2 - X_1/Z_1 - X_2/Z_2, \\ m \cdot (X_1/Z_1 - m^2 + X_1/Z_1 + X_2/Z_2) - Y_1/Z_1, \ 1\big),$$

Dimana

$$m = (Y_2/Z_2 - Y_1/Z_1)(X_2/Z_2 - X_1/Z_1)^{-1} = (Y_2Z_1 - Y_1Z_2)(X_2Z_1 - X_1Z_2)^{-1}$$

dan semua perhitungan dilakukan modulo p. Perhatikan bahwa kita menggunakan koordinat proyektif untuk mewakili P_3, dengan menetapkan $Z_3 = 1$ di atas. Namun menggunakan koordinat proyektif berarti kita tidak terbatas pada $Z_3 = 1$. Mengalikan setiap koordinat dengan $Z_1Z_2(X_2Z_1 — X_1Z_2)^3 \neq 0 \, mod \, p$, kita menemukan bahwa P_3 juga dapat direpresentasikan sebagai

$$P_3 = \left(vw, \;\; u(v^2X_1Z_2 - w) - v^3Y_1Z_2, \;\; Z_1Z_2v^3\right) \qquad (8.3)$$

Dimana

$$u = Y_2Z_1 - Y_1Z_2, \qquad v = X_2Z_1 - X_1Z_2,$$
$$w = u^2Z_1Z_2 - v^3 - 2v^2X_1Z_2. \qquad (8.4)$$

Hal yang perlu diperhatikan adalah bahwa perhitungan pada Persamaan (8.3) dan (8.4) dapat dilakukan tanpa harus melakukan inversi modular. Justru karena titik-titik mempunyai banyak representasi dalam koordinat proyektif, beberapa kehalusan dapat muncul ketika koordinat proyektif digunakan. (Kami secara eksplisit berasumsi sampai sekarang bahwa elemen grup memiliki representasi unik sebagai bit-string.) Secara khusus, sebuah titik yang dinyatakan dalam koordinat proyektif dapat mengungkapkan beberapa informasi tentang bagaimana titik tersebut diperoleh, yang mungkin bergantung pada beberapa informasi rahasia. Untuk mengatasi hal ini—dan juga untuk alasan efisiensi—koordinat affine harus digunakan untuk mentransmisikan dan menyimpan titik, dengan koordinat proyektif hanya digunakan sebagai representasi perantara selama komputasi (dengan titik dikonversi ke/dari koordinat proyektif pada saat awal/akhir perhitungan).

8.4 APLIKASI KRIPTOGRAFI

Kami telah menghabiskan cukup banyak waktu untuk mendiskusikan teori bilangan dan teori grup, dan memperkenalkan asumsi kekerasan komputasi yang diyakini secara luas berlaku. Penerapan asumsi-asumsi ini akan menyibukkan kita sepanjang sisa buku ini, namun kami memberikan beberapa contoh singkat di sini.

Fungsi dan Permutasi Satu Arah

Fungsi satu arah adalah kriptografi primitif minimal, dan keduanya diperlukan dan cukup untuk enkripsi kunci privat dan kode autentikasi pesan. Pembahasan yang lebih lengkap mengenai peran fungsi satu arah dalam kriptografi muncul pada Bab 7; di sini kami hanya memberikan definisi fungsi satu arah dan menunjukkan bahwa keberadaannya mengikuti asumsi kekerasan teori bilangan yang telah kita lihat dalam bab ini.

Secara informal, suatu fungsi ƒ bersifat satu arah jika mudah dihitung tetapi sulit dibalik. Eksperimen dan definisi berikut, yang merupakan pernyataan kembali Definisi 7.1, meresmikan hal ini.

Eksperimen pembalik $Invert_{A,f}(n)$:
1. Pilih seragam $x \in \{0,1\}^n$ dan hitung $y := f(x)$.
2. A diberikan 1^n dan y sebagai masukan, dan keluaran x'.
3. Hasil percobaan adalah 1 jika dan hanya jika $f(x') = y$.

DEFINISI 8.72 Suatu fungsi $f : \{0,1\}^* \rightarrow \{0,1\}^*$ bersifat satu arah jika memenuhi dua kondisi berikut:

1. (Mudah dihitung :) Ada algoritma waktu polinomial yang pada input x menghasilkan ƒ (x).
2. (Sulit untuk dibalik :) Untuk semua algoritma PPT A terdapat fungsi negl yang dapat diabaikan sehingga Pr[InvertA,f (n) = 1] ≤ negl(n).

Sekarang kita tunjukkan secara formal bahwa asumsi pemfaktoran mengimplikasikan adanya fungsi satu arah. Misalkan Gen adalah algoritma waktu polinomial yang, pada masukan 1^n, menghasilkan keluaran (N,p,q) dengan $N = pq\ dan\ p$ dan q adalah $n-bit$ bilangan prima kecuali dengan probabilitas yang dapat diabaikan dalam n. (Kami menggunakan Gen daripada GenModulus di sini semata-mata untuk kemudahan notasi.) Karena Gen berjalan dalam waktu polinomial, terdapat batas atas polinomial pada jumlah bit acak yang digunakan algoritme. Untuk mempermudah, dan untuk menyampaikan gagasan utama, kami berasumsi Gen selalu menggunakan paling banyak n bit acak pada input 1^n. Dalam Algoritma 8.73 kita mendefinisikan fungsi f_{Gen} yang menggunakan masukannya sebagai bit acak untuk menjalankan Gen. Jadi, f_{Gen} adalah fungsi deterministik sesuai kebutuhan.

ALGORITHM 8.73
Algorithm computing f_{Gen}

Input: String x of length n
Output: Integer N

compute $(N,p,q) := \mathsf{Gen}(1^n; x)$
// i.e., run $\mathsf{Gen}(1^n)$ using x as the random tape
return N

Jika permasalahan pemfaktoran relatif sulit terhadap Gen maka, secara intuitif, ƒGen merupakan fungsi satu arah. Tentu saja $fGen$ mudah dihitung. Mengenai sulitnya membalikkan fungsi ini, perhatikan bahwa distribusi berikut ini identik:

1. Modulus N keluaran oleh $f_{Gen}(x), ketika\ x \in \{0,1\}^n$ dipilih secara seragam.
2. Output modulus N oleh (algoritma acak) $Gen(1^n)$.

Jika moduli N yang dihasilkan menurut distribusi kedua sulit untuk difaktorkan, maka hal yang sama berlaku untuk moduli N yang dihasilkan menurut distribusi pertama. Selain itu,

mengingat setiap preimage x' dari N sehubungan dengan f_{Gen} (yaitu, sebuah x' yang mana $f_{Gen}(x') = N$; perhatikan bahwa kita tidak memerlukan $x' = x$), mudah untuk memulihkan faktor N dengan menjalankan $Gen(1n; x')$ untuk mendapatkan (N, p, q) dan mengeluarkan faktor p dan q. Jadi, menemukan gambaran awal N terhadapf_{Gen} sama sulitnya dengan memfaktorkan N . Seseorang dapat dengan mudah mengubahnya menjadi bukti formal berikut ini:

TEOREMA 8.74 Jika permasalahan pemfaktoran relatif sulit terhadap Gen, maka ƒGen merupakan fungsi satu arah.

Permutasi Satu Arah

Kita juga dapat menggunakan asumsi teori bilangan untuk menyusun rangkaian permutasi satu arah. Kita mulai dengan pernyataan kembali Definisi 7.2 dan 7.3, khusus untuk kasus permutasi:

DEFINISI 8.75 $\mathcal{A}$ Triple $\Pi = (\mathrm{Gen}, \mathrm{Samp}, \mathrm{f})$ dari algoritma waktu polinomial probabilistik adalah suatu kelompok permutasi jika hal berikut berlaku:

1. Algoritma pembangkitan parameter Gen, pada input 1^n, mengeluarkan parameter I dengan |I| ≥ n. Setiap nilai I mendefinisikan himpunan DI yang membentuk domain dan rentang permutasi (yaitu bijeksi) $f_I : D_I \rightarrow D_I$.
2. Algoritma pengambilan sampel Samp, pada masukan I, mengeluarkan elemen D_I yang terdistribusi secara merata.
3. Algoritma evaluasi deterministik f, pada masukan I dan x ∈ DI, mengeluarkan elemen $y \in D_I$. Kami menulis ini sebagai $y := f_I(x)$.

Diberikan sekumpulan $fungsi\ \Pi$, pertimbangkan eksperimen berikut untuk algoritma apa pun A dan parameter n:
Eksperimen pembalik $Invert_{A,\Pi}(n)$:

1. $Gen(1^n)$ dijalankan untuk mendapatkan I, kemudian Samp(I) dijalankan untuk memilih seragam $x \in D_I$. Akhirnya, $y := f_I(x)\ dihitung$.
2. A diberikan I dan y sebagai masukan, dan keluaran x'.
3. Hasil percobaan adalah 1 jika dan hanya jika $f_I(x') = y$.

DEFINISI 8.76 Kelompok permutasi $\Pi = (Gen, Samp, f)$ adalah satu arah jika untuk semua algoritma waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{Invert}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Mengingat GenRSA seperti pada Bagian 8.2.4, Konstruksi 8.77 mendefinisikan kelompok permutasi. Dapat disimpulkan bahwa jika permasalahan RSA lebih sulit dibandingkan dengan

GenRSA, maka permasalahan ini bersifat satu arah. Hal serupa juga dapat ditunjukkan bahwa kekerasan masalah logaritma diskrit di Z_p^*, dengan p prima, menyiratkan adanya keluarga permutasi satu arah.

CONSTRUCTION 8.77

Let GenRSA be as before. Define a family of permutations as follows:

- Gen: on input 1^n, run GenRSA(1^n) to obtain (N, e, d) and output $I = \langle N, e \rangle$. Set $\mathcal{D}_I = \mathbb{Z}_N^*$.
- Samp: on input $I = \langle N, e \rangle$, choose a uniform element of $\mathbb{Z}_N^*$.
- f: on input $I = \langle N, e \rangle$ and $x \in \mathbb{Z}_N^*$, output $[x^e \bmod N]$.

Keluarga permutasi berdasarkan masalah RSA.

Membangun Fungsi Hash Tahan Tabrakan

Fungsi hash tahan benturan diperkenalkan di Bagian 5.1. Meskipun kita telah membahas konstruksi fungsi hash tahan benturan yang digunakan dalam praktik di Bagian 6.3, kita belum melihat konstruksi yang dapat didasarkan pada asumsi yang lebih sederhana. Di sini kami menunjukkan konstruksi berdasarkan asumsi logaritma diskrit dalam kelompok orde prima. (Konstruksi berdasarkan masalah RSA dijelaskan dalam Latihan 8.20.) Meskipun konstruksi ini kurang efisien dibandingkan fungsi hash yang digunakan dalam praktik, konstruksi ini penting karena menggambarkan kelayakan mencapai ketahanan benturan berdasarkan standar dan telah dipelajari dengan baik. asumsi teori bilangan.

Misalkan $\mathcal{G}$ algoritma waktu polinomial yang, pada input 1^n, menghasilkan output a (deskripsi a) grup siklik $\mathbb{G}$, ordenya q $(dengan\, q = n)$, dan generator g. Di sini kita juga mensyaratkan bahwa q adalah bilangan prima kecuali mungkin dengan probabilitas yang dapat diabaikan. Fungsi hash dengan panjang tetap berdasarkan $\mathcal{G}$ diberikan dalam Konstruksi 8.78.

CONSTRUCTION 8.78

Let $\mathcal{G}$ be as described in the text. Define a fixed-length hash function (Gen, H) as follows:

- Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select a uniform $h \in \mathbb{G}$. Output $s := \langle \mathbb{G}, q, g, h \rangle$ as the key.
- H: given a key $s = \langle \mathbb{G}, q, g, h \rangle$ and input $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$, output $H^s(x_1, x_2) := g^{x_1} h^{x_2} \in \mathbb{G}$.

Fungsi hash dengan panjang tetap.

Perhatikan bahwa Gen dan H dapat dihitung dalam waktu polinomial. Sebelum melanjutkan dengan analisis konstruksi, kami membuat beberapa catatan teknis:

- Untuk $s = \langle G, q, g, h \rangle$ tertentu dengan $n = \| q \|$, fungsi H^s digambarkan dengan mengambil elemen $\mathbb{Z}_q$ $\mathbb{Z}_q$ sebagai input. Namun, H^s dapat dilihat sebagai mengambil bit-string dengan panjang $2 \cdot (n-1)$ sebagai input jika kita mengurai input $x \in \{0,1\}^{2(n-1)}$ sebagai dua string x_1, x_2, masing-masing dengan panjang $n-1$, dan kemudian melihat x_1, x_2sebagai elemen $\mathbb{Z}_q$ dengan cara alami.
- Output dari H^s juga dispesifikasikan sebagai elemen dari $\mathbb{G}$, namun kita dapat melihatnya sebagai bit-string jika kita memperbaiki beberapa representasi dari $\mathbb{G}$. Untuk memenuhi persyaratan Definisi 5.2 (yang mengharuskan panjang output ditetapkan sebagai fungsi n) kita dapat mengisi output sesuai kebutuhan.
- Mengingat hal di atas, konstruksi hanya memampatkan inputnya untuk grup $\mathbb{G}$ di mana elemen $\mathbb{G}$ dapat direpresentasikan menggunakan kurang dari $2n-2$ bit. Hal ini berlaku baik untuk keluaran grup berdasarkan Algoritma 8.65 (dengan asumsi nl, yang biasanya merupakan kasus), maupun untuk grup kurva elips ketika kompresi titik digunakan. Generalisasi Konstruksi 8.78 dapat digunakan untuk memperoleh kompresi dari mana pun yang masalah diskritlogaritmanya sulit, berapa pun jumlah bit yang diperlukan untuk merepresentasikan elemen grup; lihat Latihan 8.21.

TEOREMA 8.79 Jika permasalahan logaritma diskrit relatif sulit, maka Konstruksi 8.78 adalah fungsi hash tahan tumbukan dengan panjang tetap (sesuai dengan pembahasan mengenai kompresi di atas).

BUKTI Misalkan $\Pi = (\mathrm{Gen}, \mathrm{H})$ seperti pada Konstruksi 8.78, dan misalkan algoritma waktu polinomial probabilistik dengan

$$\varepsilon(n) \stackrel{\text{def}}{=} \Pr[\text{Hash-coll}_{\mathcal{A},\Pi}(n) = 1]$$

(lih. Definisi 5.2). Kami menunjukkan bagaimana A dapat digunakan oleh algoritma A′ untuk menyelesaikan masalah logaritma diskrit dengan probabilitas keberhasilan $\varepsilon(n)$:
Algoritma $\mathcal{A}'$
Algoritma diberikan $\mathbb{G}, \mathrm{q}, \mathrm{g}, \mathrm{h}$ sebagai input.

1. Misalkan $\mathrm{s} := \langle \mathbb{G}, \mathrm{q}, \mathrm{g}, \mathrm{h} \rangle$. Jalankan $\mathcal{A}'(s)$ dan dapatkan keluaran x dan x'.
2. Jika $\mathrm{x} \neq \mathrm{x}'$ dan $\mathrm{H}^s(\mathrm{x}) = \mathrm{H}^s(\mathrm{x}')$ maka:

- Jika $\mathrm{h} = 1$ menghasilkan 0.
  - Jika tidak $(\mathrm{h} \neq 1)$, uraikan x sebagai $(\mathrm{x1}, \mathrm{x2})$ dan uraikan x' sebagai (x'_1, x'_2), di mana $x1, x2, x'1, x'2 \in \mathrm{Zq}$, dan kembalikan hasilnya$[(x_1, x'_1) \cdot (x'_2, x_2)^{-1} \bmod \mathrm{q}]$.

Jelasnya, $\mathcal{A}'$ berjalan dalam waktu polinomial. Selanjutnya, masukan s yang diberikan kepada $\mathcal{A}'$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}''$ didistribusikan persis seperti dalam percobaan $Hash - coll_{A,\Pi}$ untuk nilai parameter keamanan n yang sama. (Masukan ke $\mathcal{A}'$ dihasilkan dengan menjalankan $\mathcal{G}(1^n)$ untuk mendapatkan $\mathbb{G}, q, g$ dan kemudian memilih h G secara

seragam secara acak. Ini persis bagaimana s dihasilkan oleh $Gen(1^n)$.) Jadi, dengan probabilitas tepat $\varepsilon(n)$ terjadi tabrakan; yaitu $x \neq x' \, dan \, H^s(x) = H^s(x')$.

Kita mengklaim bahwa setiap kali terjadi tabrakan, $\mathcal{A}'$ mengembalikan log jawaban yang benar h. Jika h = 1 maka ini jelas benar (karena logg h = 0 dalam kasus ini). Dengan asumsi $h \neq 1$, maka adanya tumbukan berarti demikian

$$\begin{aligned} H^s(x_1, x_2) = H^s(x_1', x_2') &\Rightarrow g^{x_1} h^{x_2} = g^{x_1'} h^{x_2'} \\ &\Rightarrow g^{x_1 - x_1'} = h^{x_2' - x_2}. \end{aligned} \qquad (8.5)$$

Perhatikan bahwa $x'_1 - x_2 \neq 0$ mod q; jika tidak, kita akan mendapatkan $x_1 = x'_1$ mod q tetapi kemudian $x = x'$ dan kita tidak akan mengalami tumbukan. Karena q adalah bilangan prima, maka $invers \, \Delta \stackrel{\text{def}}{=} [(x'_2 - x_2)^{-1} \text{mod q}]$ ada. Menaikkan setiap sisi Persamaan (8.5) ke pangkat ini menghasilkan:

$$g^{(x_1 - x_1') \cdot \Delta} = \left(h^{x_2' - x_2}\right)^{\Delta} = h^1 = h,$$

dan output yang dikembalikan oleh A' adalah

$$\log_g h = [(x_1 - x_1') \cdot \Delta \bmod q] = \left[(x_1 - x_1') \cdot (x_2' - x_2)^{-1} \bmod q\right].$$

Kita melihat bahwa $\mathcal{A}$' menyelesaikan permasalahan logaritma diskrit dengan tepat dengan probabilitas tepat $\varepsilon(n)$. Karena, dengan asumsi, permasalahan logaritma diskrit relatif sulit terhadap $\mathcal{G}$, kita menyimpulkan bahwa $\varepsilon(n)$ dapat diabaikan. Dengan menggunakan Latihan 8.21 yang dikombinasikan dengan transformasi Merkle–Damg˚ard (lihat Bagian 5.2) kita memperoleh:

TEOREMA 8.80 Jika permasalahan logaritma diskrit sulit, maka terdapat fungsi hash yang tahan benturan.

BAB 9
ALGORITMA FAKTORISASI DAN KOMPUTASI LOGARITMA DISKRIT

Pada bab terakhir, kami memperkenalkan beberapa soal teori bilangan yang paling menonjol, memfaktorkan hasil kali dua bilangan prima besar dan menghitung logaritma diskrit dalam kelompok tertentu yang secara luas diyakini sulit. Sebagaimana didefinisikan di sana, ini berarti tidak ada algoritma waktu polinomial untuk permasalahan ini. Namun, gagasan kekerasan yang asimtotik ini tidak banyak memberi tahu kita tentang cara menetapkan parameter keamanan terkadang disebut panjang kunci, meskipun istilahnya tidak dapat dipertukarkan untuk mencapai tingkat keamanan konkret yang diinginkan dalam praktiknya. Pemahaman yang tepat tentang masalah ini sangat penting untuk penerapan sistem kriptografi di dunia nyata berdasarkan masalah ini. Menetapkan parameter keamanan terlalu rendah berarti sistem kriptografi mungkin rentan terhadap serangan yang lebih efisien daripada yang diantisipasi; bersikap terlalu konservatif dan menetapkan parameter keamanan terlalu tinggi akan memberikan keamanan yang baik, namun mengorbankan efisiensi bagi pengguna yang jujur. Kesulitan relatif dari berbagai permasalahan teori bilangan juga dapat berperan dalam menentukan permasalahan mana yang akan digunakan sebagai dasar untuk membangun kriptosistem.

Masalah mendasarnya, tentu saja, adalah bahwa pencarian brute force mungkin bukan algoritma terbaik untuk memecahkan masalah tertentu; dengan demikian, menggunakan panjang kunci n secara umum tidak memberikan keamanan terhadap penyerang yang berjalan selama 2^n kali. Hal ini berbeda dengan pengaturan kunci pribadi di mana serangan terbaik terhadap cipher blok yang ada memiliki kompleksitas pencarian brute force (mengabaikan pra-perhitungan). Sebagai konsekuensinya, panjang kunci yang digunakan dalam pengaturan kunci publik cenderung jauh lebih besar dibandingkan dengan panjang kunci yang digunakan dalam pengaturan kunci privat.

Untuk lebih memahami poin ini, dalam bab ini kita akan mengeksplorasi beberapa algoritma waktu nonpolinomial untuk memfaktorkan dan menghitung logaritma diskrit yang jauh lebih baik daripada pencarian brute-force. Tujuannya hanyalah untuk memberikan gambaran algoritma yang ada untuk permasalahan ini, serta memberikan beberapa panduan dasar untuk menetapkan parameter dalam praktiknya. Fokus kami adalah pada ide-ide tingkat tinggi, dan kami secara sadar tidak membahas banyak detail penting pada tingkat implementasi yang penting untuk ditangani jika algoritma ini akan digunakan dalam praktik. Kami juga berkonsentrasi secara eksklusif pada algoritma klasik, dan merujuk pembaca ke tempat lain untuk diskusi tentang algoritma kuantum polinomial-waktu (!) yang diketahui untuk memfaktorkan dan menghitung logaritma diskrit. (Ini adalah keputusan sadar lainnya, karena kami tidak ingin mengasumsikan latar belakang mekanika kuantum di pihak pembaca, dan karena komputer kuantum tampaknya tidak mungkin terjadi dalam waktu dekat.)

Pembaca mungkin juga memperhatikan bahwa kami hanya menjelaskan algoritma untuk memfaktorkan dan menghitung logaritma diskrit, dan bukan algoritma untuk, misalnya,

memecahkan masalah RSA atau keputusan Diffie–Hellman. Pilihan kami dibenarkan oleh fakta bahwa algoritma yang paling terkenal untuk menyelesaikan RSA memerlukan pemfaktoran modulus, dan (dalam kelompok yang dibahas di Bagian 8.3.3 dan 8.3.4) pendekatan yang paling terkenal untuk memecahkan masalah keputusan Diffie-Hellman melibatkan komputasi logaritma diskrit.

9.1 ALGORITMA PEMFAKTORAN

Secara keseluruhan, kita berasumsi bahwa N = pq adalah hasil kali dua bilangan prima berbeda dengan p < q. Kita akan sangat tertarik pada kasus ketika p dan q masing-masing memiliki panjang n yang sama (yang diketahui), sehingga n = Θ(log N).

Kita akan sering menggunakan teorema sisa Cina bersama dengan notasi yang dikembangkan. Teorema sisa Tiongkok menyatakan bahwa

$$\mathbb{Z}_N \simeq \mathbb{Z}_p \times \mathbb{Z}_q \quad \text{and} \quad \mathbb{Z}_N^* \simeq \mathbb{Z}_p^* \times \mathbb{Z}_q^*,$$

dengan isomorfisme yang diberikan oleh $f(x) \stackrel{\text{def}}{=} ([x \bmod p], [x \bmod q])$. Fakta bahwa f adalah isomorfisme berarti, khususnya, ia memberikan bijeksi antara elemen $x \in \mathbb{Z}_n$ dan pasangan$(x_p, x_q) \in \mathbb{Z}_p \times \mathbb{Z}_q$. *Kita menulis* $x \leftrightarrow (x_p, x_q)$, *dengan* $x_p = [x \bmod p]$ *dan* $x_q = [x \bmod q]$, untuk menyatakan bijeksi ini.

Ingatlah dari Bagian 8.2 bahwa pembagian percobaan metode pemfaktoran brute force yang sepele menemukan faktor dari bilangan tertentu N dalam waktu $\mathcal{O}(N^{1/2}\ polylog(N))$. (Ini adalah algoritma waktu eksponensial*, karena ukuran inputnya adalah panjang representasi biner dari* N *, yaitu,* $\|N\| = \mathcal{O}(log\ N)$.[1]) Kita membahas tiga algoritma pemfaktoran dengan kinerja yang lebih baik:

- Metode $p - 1$ Pollard efektif jika $p - 1$ hanya mempunyai faktor prima yang “kecil”.
- Metode rho Pollard berlaku untuk N sembarang. (Oleh karena itu, ini disebut algoritma pemfaktoran tujuan umum.) Waktu berjalannya untuk N bentuk yang dibahas di awal bagian ini adalah $\mathcal{O}(N^{1/4}\ polilog(N))$. Perhatikan ini masih eksponensial dalam n, panjang N.
- Algoritme saringan kuadrat adalah algoritma pemfaktoran tujuan umum yang berjalan dalam waktu sub-eksponensial dalam panjang N. Kami memberikan ikhtisar tingkat tinggi tentang cara kerja algoritme ini, namun detailnya agak rumit dan berada di luar cakupan buku ini.
- Algoritme pemfaktoran tujuan umum tercepat yang diketahui adalah saringan bidang bilangan umum. Secara heuristik, algoritme ini memfaktorkan masukannya N dalam waktu yang diharapkan $2^{\mathcal{O}}((log\ N)^{1/3} \cdot (log\ log\ N)^{2/3})$, yang merupakan sub-eksponensial dalam panjang N .

Algoritma p − 1 Pollard

Jika $N = pq$ dan $p - 1$ hanya memiliki faktor prima "kecil", algoritma $p - 1$ Pollard dapat digunakan untuk memfaktorkan N secara efisien. Ide dasarnya sederhana. Misalkan B adalah bilangan bulat yang $(p - 1) \mid B \; dan \; (q - 1) \not\mid B$; kami tunduk pada rincian di bawah ini tentang bagaimana B tersebut ditemukan. Katakanlah $B = \gamma \cdot (p - 1)$ untuk suatu bilangan bulat γ. Pilih seragam $x \in Z_N^*$ dan hitung y := [xB − 1 mod N]. (Perhatikan bahwa y dapat dihitung menggunakan algoritma eksponensial efisien dari Lamp*iran B.2.3.) Karena* $1 \leftrightarrow (1, 1)$, kita mempunyai

$$\begin{aligned} y = [x^B - 1 \bmod N] \leftrightarrow (x_p, x_q)^B - (1,1) \\ = (x_p^B - 1 \bmod p,\; x_q^B - 1 \bmod q) \\ = ((x_p^{p-1})^\gamma - 1 \bmod p,\; x_q^B - 1 \bmod q) \\ = (0, [x_q^B - 1 \bmod q]) \end{aligned}$$

menggunakan Teorema 8.14 dan fakta bahwa orde Z_N^* adalah p — 1. Di bawah ini kita tunjukkan bahwa, dengan probabilitas tinggi, $x_q^B \neq 1 \; mod \; q$. Dengan asumsi demikian, kita telah memperoleh bilangan bulat y yang mana

$$y = 0 \bmod p \quad \text{but} \quad y \neq 0 \bmod q;$$

yaitu, $p \; y \; tapi \; q \; \lambda \; y$. Hal ini, pada gilirannya, menyiratkan bahwa $gcd(y, N) = p$. Jadi, perhitungan gcd sederhana (yang dapat dilakukan secara efisien seperti dijelaskan dalam Lampiran B.1.2) menghasilkan faktor prima sebesar N.

ALGORITHM 9.1
Pollard's $p - 1$ algorithm for factoring

Input: Integer N
Output: A non-trivial factor of N

$x \leftarrow \mathbb{Z}_N^*$
$y := [x^B - 1 \bmod N]$
// B is as in the text
$p := \gcd(y, N)$
if $p \notin \{1, N\}$ **return** p

Pertama-tama mari kita berargumen bahwa algoritme tersebut berhasil (dengan probabilitas tinggi). Asumsikan $(p$ — $1) \mid B \; tapi \; (q$ — $1) \; B$. Dalam hal ini, selama $xq \overset{\text{def}}{=} [x \; mod \; q]$ merupakan generator dari Z_q^*, kita mempunyai $x_q^B \neq 1 \; mod \; q$. (Ini mengikuti Proposisi 8.52.) Masih menganalisis probabilitas bahwa x_q adalah generator. Di sini kami mengandalkan

beberapa hasil yang dibuktikan pada Lampiran B.3.1. Karena q adalah bilangan prima, Z_q^* adalah grup berorde siklik $q - 1$ yang memiliki generator tepat $\varphi(q - 1)$ (lih. Teorema B.16). Jika x dipilih secara seragam dari Z_N^* , maka xq terdistribusi secara merata di Z_q^* . (Ini adalah konsekuensi dari fakta bahwa teorema sisa Cina memberikan bijeksi antara Z_N^* dan $Z_p^* \times Z_q^*$.) Jadi, probabilitas xq adalah generator adalah $\frac{\varphi(q-1)}{q-1} = \Omega(1/\log q) = \Omega(1/n)$. Beberapa nilai x bisa dipilih untuk meningkatkan kemungkinan keberhas*ilan.*

Kita dihadapkan pada masalah mencari B sehingga $(p-1) \mid B$ *tapi* $(q-1)6 \mid B$.

Salah satu kemungkinannya adalah memilih $B = \prod_{i=1}^{k} p_i^{n/\log p_i}$ untuk beberapa k, dengan pi melambangkan bilangan prima ke-i (yakni, $p_1 = 2, p_2 = 3, p_3 = 5, \ldots$) dan n adalah panjangnya dari hal. (Perhatikan bahwa pbn/ log pic i adalah pangkat terbesar dari pi yang mungkin dapat membagi $p - 1$.) Jika p − 1 dapat ditulis sebagai $\prod_{i=1}^{k} p_i^{ei}$ $dengan\ ei \geq 0$ (yaitu, jika faktor prima terbesar dari p − 1 lebih kecil dari p_k), kita akan mendapatkan (p − 1) | $B. Sebaliknya, jika\ q - 1$ mempunyai faktor prima yang lebih besar dari pk, maka (q − 1) /| B. Memilih nilai yang lebih besar untuk k akan meningkatkan B sehingga meningkatkan waktu berjalan algoritma (yang melakukan eksponen modular ke pangkat B).

Nilai k yang lebih besar juga membuat kemungkinan (p − 1) | B, namun pada saat yang sama memperkecil kemungkinan bahwa (q − 1)/| B. Tentu saja dimungkinkan untuk menjalankan algoritme berulang kali menggunakan pilihan ganda untuk k. Algoritma p − 1 Pollard digagalkan jika p − 1 dan q − 1 memiliki faktor prima yang besar. (Lebih tepatnya, algoritma ini masih berfungsi tetapi hanya untuk B yang sangat besar sehingga algoritma menjadi tidak praktis.) Jika p dan q adalah n-bit bilangan prima yang seragam, maka kecil kemungkinannya bahwa $p - 1$ atau $q - 1$ hanya mempunyai bilangan prima kecil faktor. Namun demikian, ketika menghasilkan modulus $N = pq$ untuk aplikasi kriptografi, p dan q terkadang dipilih sebagai bilangan prima kuat. (Ingatlah bahwa p adalah bilangan prima kuat jika $(p - 1)/2$ juga merupakan bilangan prima.) Memilih p dan q dengan cara ini jauh lebih tidak efisien dibandingkan hanya memilih p dan q sebagai bilangan prima sembarang (acak). Karena algoritma pemfaktoran yang lebih baik sudah tersedia (seperti yang akan kita lihat di bawah), dan berdasarkan pengamatan di atas, konsensus saat ini adalah bahwa tambahan biaya komputasi untuk menghasilkan p dan q sebagai bilangan prima kuat tidak menghasilkan keuntungan keamanan yang berarti.

Algoritma Rho Pollard

Algoritme rho Pollard dapat digunakan untuk memfaktorkan bilangan bulat sembarang N = pq; dalam hal ini, ini adalah algoritma pemfaktoran untuk tujuan umum. Secara heuristik, algoritma memfaktorkan N dengan probabilitas konstan dalam waktu O $(N^{1/4}\,\mathrm{polylog(N)})$; ini masih eksponensial, namun merupakan kemajuan besar dibandingkan divisi percobaan.

ALGORITHM 9.2
Pollard's rho algorithm for factoring

Input: Integer N, a product of two n-bit primes
Output: A non-trivial factor of N

$x^{(0)} \leftarrow \mathbb{Z}_N^*, \quad x' := x := x^{(0)}$
for $i = 1$ to $2^{n/2}$:
  $x := F(x)$
  $x' := F(F(x'))$
  $p := \gcd(x - x', N)$
  if $p \notin \{1, N\}$ **return** p and **stop**

Bagaimana kita bisa menemukan pasangan yang baik? Katakanlah kita memilih nilai $\left(x_p^{(1)}, x_q^{(1)}\right), \ldots, \left(x_p^{(k)} x_q^{(k)}\right)$ seragam dari $\mathbb{Z}_N^*$, dimana k = $2^{n/2}$ = $\mathcal{O}(\sqrt{p}\,)$). Melihat ini dalam representasi sisa bahasa China sebagai $(x_p^{(1)}, x_q^{(1)}), \ldots, (x_p^{(k)}, x_q^{(k)})$, kita mendapatkan bahwa setiap $x_p^{(i)} \stackrel{\text{def}}{=} [x^{(i)} \bmod p]$ seragam dalamZ_P^* . (Ini mengikuti bijektivitas antara Z_N^* dan $Z_P^* \times Z_Q^*$.) Jadi, dengan menggunakan batasan ulang tahun Lemma A.16, kita melihat bahwa dengan probabilitas tinggi terdapat i, j yang berbeda dengan $x(i)p = x(j)p$ atau, setara dengan, $x^{(i)} = x^{(j)} \bmod p$. Selain itu, Lemma 9.15 menunjukkan bahwa $x_p^{(6)} = x^{(j)}$ kecuali dengan probabilitas yang dapat diabaikan. Jadi, dengan probabilitas tinggi kita memperoleh pasangan $x^{(i)} = x^{(j)}$ yang baik yang dapat digunakan untuk mencari faktor non-trivial dari N , seperti yang dibahas sebelumnya.

Kita dapat menghasilkan elemen seragam $k = \mathcal{O}(\sqrt{p}\,)$ dari$i$ $\mathbb{Z}_N^*$dalam waktu = $\mathcal{O}(\sqrt{p}\,)$ = $\mathcal{O}$ ($N^{1/4}$). Namun, menguji semua pasangan elemen untuk mengidentifikasi pasangan yang baik memerlukan waktu (k^2) = $\mathcal{O}$ (k^2) = $\mathcal{O}(p)$ = $\mathcal{O}$ ($N^{1/2}$)! (Perhatikan bahwa karena p tidak diketahui, kita tidak bisa begitu saja menghitung $x_p^{(1)}, \ldots, x_p^{(k)}$ secara eksplisit lalu mengurutkan $x_p^{(1)}$untuk menemukan pasangan yang baik. Sebaliknya, untuk semua pasangan berbeda i, j kita harus menghitung gcd($x^{(i)} - x^{(j)}$, N) untuk melihat apakah ini menghasilkan faktor non-trivial sebesar N .) Tanpa optimasi lebih lanjut, ini tidak akan lebih baik daripada pembagian percobaan.

Ide Pollard adalah menggunakan teknik yang telah kita lihat di Bagian 5.4.2 dalam konteks serangan ulang tahun di ruang kecil. Secara khusus, kami menghitung barisan $x^{(1)}, x^{(2)}, \ldots$ dengan membiarkan setiap nilai menjadi fungsi dari nilai sebelumnya; yaitu, kita memperbaiki beberapa $fungsi\ F : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$, pilih seragam $x^{(0)} \in \mathbb{Z}_N^*$, lalu $himpunanx(i) := F\,(x(i-1))$ untuk $i = 1, \ldots, k$. Kita memerlukan F untuk mempunyai sifat bahwa jika $x = x' \bmod p$, maka $F\,(x) = F\,(x') \bmod p$; ini memastikan bahwa ketika modulo ekivalensi p terjadi, modulo p tersebut tetap ada. (Pilihan standarnya adalah $F\,(x)$ = [x^2 +1 mod N], tetapi polinomial F mana pun akan memiliki sifat ini.) Jika kita memodelkan F sebagai fungsi acak (yang bekerja secara heuristik), maka dengan probabilitas tinggi terdapat fungsi yang baik. pasangkan k elemen pertama barisan ini. Dengan proses yang kira-kira seperti pada Algoritma 5.9 dari Bagian 5.4, kita dapat mendeteksi pasangan yang baik (jika

ada) hanya dengan menggunakan perhitungan O(k) gcd; lihat Algoritma 9.2. Selain meningkatkan waktu berjalan, ide Pollard juga secara drastis mengurangi jumlah memori yang dibutuhkan.

Algoritma Saringan Kuadrat

Algoritma rho Pollard lebih baik dibandingkan pembagian percobaan, namun tetap berjalan dalam waktu yang eksponensial. Algoritma saringan kuadratik berjalan dalam waktu sub-eksponensial. Ini adalah algoritma pemfaktoran tercepat yang diketahui hingga awal tahun 1990an dan tetap menjadi algoritma pemfaktoran pilihan untuk bilangan dengan panjang hingga sekitar 300 bit. Kami menjelaskan prinsip-prinsip umum algoritma tetapi memperingatkan pembaca bahwa beberapa rincian penting dihilangkan.

Suatu elemen $z \in \mathbb{Z}_N^*$ merupakan residu kuadrat modulo N jika terdapat $x \in \mathbb{Z}_N^*$ sehingga $x^2 = z \bmod N$; dalam hal ini, kita katakan bahwa x adalah akar kuadrat dari z. Pengamatan berikut ini menjadi titik awal kami:

Jika N adalah hasil kali dua bilangan prima ganjil yang berbeda, maka setiap modulo residu kuadrat N mempunyai tepat empat akar kuadrat. (Lihat Bagian 13.4)

- Diberikan x, y dengan $x^2 = y^2 \bmod N$ dan $x \neq \pm y \bmod N$, adalah mungkin untuk menghitung faktor nontrivial dari N dalam waktu polinomial. Hal ini berdasarkan fakta bahwa $x^2 = y^2 \bmod N$ menyiratkan

$$0 = x^2 - y^2 = (x - y)(x + y) \bmod N,$$

dan seterusnya $N|\ (x - y)(x + y)$. Namun, $N|\ (x - y)$ dan N $|\backslash\ (x + y)$ karena $x = y \bmod N$. Jadi, $gcd(x\ y, N)$ harus sama dengan salah satu faktor prima dari N . (Lihat juga Lemma 13.35.)

Algoritma saringan kuadrat mencoba menghasilkan x, y dengan $x^2 = y^2 \bmod N$ dan $x \neq \pm y \bmod N$. Cara naif untuk melakukan hal ini—yang menjadi dasar algoritma pemfaktoran lama karena Fermat—adalah dengan memilih $x \in \mathbb{Z}_N^*$, menghitung $q := [x^2 \bmod N]$, dan kemudian memeriksa apakah q adalah kuadrat dari bilangan bulat (yaitu, tanpa modulo reduksi N). Jika ya, maka $q = y^2$ untuk bilangan bulat y dan $x^2 = y^2 \bmod N$. Sayangnya, kemungkinan $[x^2 \bmod N]$ adalah persegi sangat rendah sehingga proses ini harus diulang berkali-kali secara eksponensial.

Peningkatan yang signifikan diperoleh dengan menghasilkan urutan nilai $q_1 := [x_1^2 \bmod N]$, . . . dan mengidentifikasi subset dari nilai-nilai yang hasil perkaliannya adalah kuadrat atas bilangan bulat. Dalam algoritma saringan kuadratik hal ini dilakukan dengan menggunakan dua langkah berikut:

Langkah 1. Perbaiki beberapa batasan B. Katakanlah suatu bilangan bulat mulus-B jika semua faktor primanya kurang dari atau sama dengan B. Pada fase pertama algoritme, kita mencari bilangan bulat berbentuk $q_1 := [x_1^2 \bmod N]$, yang B-halus dan faktorkan. (Meskipun memfaktorkan itu sulit, mencari dan memfaktorkan bilangan mulus B is dapat dilakukanvbila B cukup kecil.) $\{x_i\}$ ini dipilih dengan mencoba $x = \sqrt{N} + 1, \sqrt{N} + 2, \ldots$; ini memastikan

modulo reduksi nontrivial N (karena $x > \sqrt{N}$) dan memiliki keuntungan bahwa $q \stackrel{\text{def}}{=} [x^2 \bmod N] = x^2 - N$ adalah "kecil" sehingga q lebih cenderung menjadi B-smooth.

Misalkan $\{p_1, \ldots, p_k\}$ adalah himpunan bilangan prima yang kurang dari atau sama dengan B. Setelah kita menemukan dan memfaktorkan $\{q_i\}$ mulus-B seperti dijelaskan di atas, kita mempunyai himpunan persamaan dalam bentuk:

$$\begin{aligned} q_1 &= [x_1^2 \bmod N] = \prod_{i=1}^{k} p_i^{e_{1,i}} \\ &\vdots \\ q_\ell &= [x_\ell^2 \bmod N] = \prod_{i=1}^{k} p_i^{e_{\ell,i}}. \end{aligned} \tag{9.1}$$

Perhatikan bahwa persamaan di atas melebihi bilangan bulat.)

Langkah 2. Selanjutnya kita ingin mencari subset dari $\{q_i\}$ yang hasil perkaliannya adalah persegi. Jika kita mengalikan beberapa subset S dari $\{q_i\}$, kita akan melihat hasilnya

$$z = \prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}}$$

adalah persegi jika dan hanya jika eksponen setiap p_i prima adalah genap. Hal ini menunjukkan bahwa kita hanya memperhatikan eksponen $\{e_{j,i}\}$ pada Persamaan (9.1) modulo 2; selain itu, kita dapat menggunakan aljabar linier untuk mencari subset $\{q_i\}$ yang "vektor eksponennya" berjumlah modulo 2 vektor-0.

Lebih detailnya: jika kita mengurangi eksponen pada Persamaan (9.1) modulo 2, kita memperoleh matriks 0/1 Γ yang diberikan oleh

$$\begin{pmatrix} \gamma_{1,1} & \gamma_{1,2} & \cdots & \gamma_{1,k} \\ \vdots & \vdots & \ddots & \vdots \\ \gamma_{\ell,1} & \gamma_{\ell,2} & \cdots & \gamma_{\ell,k} \end{pmatrix} \stackrel{\text{def}}{=} \begin{pmatrix} [e_{1,1} \bmod 2] & [e_{1,2} \bmod 2] & \cdots & [e_{1,k} \bmod 2] \\ \vdots & \vdots & \ddots & \vdots \\ [e_{\ell,1} \bmod 2] & [e_{\ell,2} \bmod 2] & \cdots & [e_{\ell,k} \bmod 2] \end{pmatrix}.$$

Jika $\ell = k + 1$, maka Γ memiliki lebih banyak baris daripada kolom dan harus ada subset S yang tidak kosong dari baris yang berjumlah modulo 2 vektor 0. Subset seperti itu dapat dicari secara efisien menggunakan aljabar linier. Kemudian:

$$z \stackrel{\text{def}}{=} \prod_{j \in S} q_j = \prod_{i=1}^{k} p_i^{\sum_{j \in S} e_{j,i}} = \left(\prod_{i=1}^{k} p_i^{(\sum_{j \in S} e_{j,i})/2} \right)^2,$$

menggunakan fakta bahwa semua

$$\left\{ \sum_{j \in S} e_{j,i} \right\}$$

genap. Sejak

$$z = \prod_{j \in S} q_j = \prod_{j \in S} x_j^2 = \left(\prod_{j \in S} x_j \right)^2 \bmod N,$$

kita telah memperoleh dua akar kuadrat (modulo N) dari z. Meskipun tidak ada jaminan bahwa akar kuadrat ini akan memungkinkan faktorisasi N (untuk alasan yang dibahas di awal bagian ini), secara heuristik hal tersebut dilakukan dengan probabilitas yang konstan. Dengan mengambil $\ell > k + 1$ kita dapat memperoleh beberapa himpunan bagian S dengan sifat yang diinginkan dan mencoba memfaktorkan N menggunakan setiap kemungkinan.

Contoh 9.3

Ambil N = 377753. Kita mempunyai 6647 = [6202 mod N], dan kita dapat memfaktorkan 6647 (di atas bilangan bulat, tanpa pengurangan modular apa pun) sebagai

$$\left[620^2 \bmod N\right] = 6647 = 17^2 \cdot 23.$$

Demikian pula,

$$\begin{aligned}
\left[621^2 \bmod N\right] &= 2^4 \cdot 17 \cdot 29 \\
\left[645^2 \bmod N\right] &= 2^7 \cdot 13 \cdot 23 \\
\left[655^2 \bmod N\right] &= 2^3 \cdot 13 \cdot 17 \cdot 29.
\end{aligned}$$

Jika subset S kita menyertakan keempat persamaan di atas, kita akan melihatnya

$$\begin{aligned}
620^2 \cdot 621^2 \cdot 645^2 \cdot 655^2 &= 2^{14} \cdot 13^2 \cdot 17^4 \cdot 23^2 \cdot 29^2 \bmod N \\
\Rightarrow \left[620 \cdot 621 \cdot 645 \cdot 655 \bmod N\right]^2 &= \left[2^7 \cdot 13 \cdot 17^2 \cdot 23 \cdot 29 \bmod N\right]^2 \bmod N \\
\Rightarrow 127194^2 &= 45335^2 \bmod N,
\end{aligned}$$

dengan $127194 \neq \pm 45335 \bmod N$. Menghitung gcd(127194 — 45335, 377753) = 751 menghasilkan faktor non-trivial sebesar N.

Durasi. Memilih nilai B yang lebih besar akan memperbesar kemungkinan bahwa nilai seragam $q = [x^2 \bmod N]$ adalah B-smooth; di sisi lain, ini berarti kita harus bekerja lebih keras untuk mengidentifikasi dan memfaktorkan bilangan mulus B, dan kita harus mencari lebih banyak bilangan tersebut (karena kita memerlukan $\ell > k$, dengan k adalah banyaknya bilangan prima yang kurang dari atau sama dengan B). Ini juga berarti bahwa matriks Γ akan lebih besar, sehingga langkah aljabar linier akan lebih lambat. Memilih nilai optimal B menghasilkan algoritma yang (setidaknya secara heuristik) memfaktorkan N dalam waktu $2^{O(\sqrt{\log N \log \log N})}$. (Faktanya, suku konstanta dalam eksponen dapat ditentukan dengan cukup tepat.) Poin penting untuk tujuan kita adalah bahwa bilangan ini merupakan sub-eksponensial pada panjang N.

9.2 ALGORITMA UNTUK MENGHITUNG LOGARITMA DISKRIT

Misalkan $\mathbb{G}$ adalah grup yang diketahui ordenya q. Contoh masalah logaritma diskrit di $\mathbb{G}$ menentukan basis $g \in \mathbb{G}$ (yang tidak harus berupa generator dari $\mathbb{G}$) dan elemen $h \in \langle g \rangle$, subgrup yang dihasilkan oleh h; tujuannya adalah mencari x sehingga $g^x = h$. (Lihat Bagian

8.3.2.) Solusi x disebut logaritma diskrit dari h terhadap g. Pencarian brute force yang sepele untuk x dapat dilakukan dalam waktu $|\langle g\rangle| \leq q$ (hanya dengan mencoba semua nilai yang mungkin), sehingga kita hanya akan tertarik pada algoritma yang waktu kerjanya lebih baik dari ini.

Algoritma untuk memecahkan masalah logaritma diskrit terbagi dalam dua kategori: algoritma yang bersifat generik dan berlaku untuk kelompok arbitrer, dan algoritma yang disesuaikan untuk bekerja pada kelas kelompok tertentu. Kita mulai dengan membahas tiga algoritma umum:

Ketika orde grup q bukan bilangan prima dan faktorisasi q diketahui, atau mudah ditentukan, algoritma Pohlig–Hellman mengurangi masalah pencarian logaritma diskrit di $\mathbb{G}$ menjadi masalah menemukan logaritma diskrit di subgrup orde prima dari $\mathbb{G}$. Secara kasar Artinya, kompleksitas penyelesaian logaritma diskrit pada kelompok berorde q tidak lebih besar daripada kompleksitas penyelesaian logaritma diskrit pada kelompok berorde q', dengan q' adalah pembagi bilangan prima terbesar q. Hal ini menjelaskan preferensi untuk menggunakan kelompok orde prima (lih. Bagian 8.3).

Metode langkah bayi/langkah raksasa, berkat Shanks, menghitung logaritma diskrit dalam grup orde q dalam waktu $\mathcal{O}(\sqrt{q} \cdot \text{polilog}\ (q))$ dan menyimpan elemen grup $\mathcal{O}(\sqrt{q})$. Algoritme rho Pollard juga memungkinkan penghitungan logaritma diskrit dalam waktu $\mathcal{O}(\sqrt{q}\ \text{polilog}\ (q))$, tetapi menggunakan memori konstan. Hal ini dapat dilihat sebagai eksploitasi hubungan antara masalah logaritma diskrit dan hashing anti-tabrakan seperti yang telah kita lihat di Bagian 8.4.2. Kami menjelaskan algoritma berbeda yang menggambarkan hubungan ini dengan lebih jelas.

Dapat ditunjukkan bahwa kompleksitas waktu dari dua algoritma terakhir adalah optimal sejauh menyangkut algoritma generik. Oleh karena itu, untuk memiliki harapan untuk melakukan yang lebih baik, kita harus melihat algoritma untuk kelompok tertentu yang mengeksploitasi representasi elemen kelompok dalam kelompok tersebut, dimana yang dimaksud dengan "representasi" adalah cara elemen kelompok dikodekan sebagai bit-string.

Poin ini memerlukan beberapa diskusi. Dari sudut pandang matematika, dua grup siklik dengan orde yang sama adalah isomorfik, yang berarti bahwa grup-grup tersebut identik hingga "penggantian nama" elemen grup. Namun, dari sudut pandang komputasi/algoritmik, "penggantian nama" ini dapat mempunyai dampak yang signifikan. Misalnya, perhatikan grup siklik $\mathbb{Z}_q$ dari bilangan bulat $\{0,\ldots,q^{-1}\}$ di bawah penambahan modulo q. Menghitung logaritma diskrit dalam grup ini sangatlah mudah. Katakanlah kita diberikan $g, h \in \mathbb{Z}_q$ dengan g sebuah generator (jadi $g \neq 0$), dan kita ingin mencari x sedemikian rupa sehingga $x \,.\, g = h \bmod q$. Karena $g \neq 0$ kita mempunyai $\gcd(g,q) = 1$ sehingga g mempunyai invers perkalian g^{-1} modulo q. Selain itu, g^{-1} dapat dihitung secara efisien, seperti dijelaskan dalam Lampiran B.2.2. Namun $x = h \,.\, g^{-1} \bmod q$ adalah solusi yang diinginkan. Perhatikan bahwa, secara formal, x di sini menyatakan bilangan bulat dan bukan elemen grup—bagaimanapun juga, operasi grup adalah penjumlahan, bukan perkalian. Namun demikian, dalam menyelesaikan masalah logaritma diskrit di $\mathbb{Z}_q$ kita dapat memanfaatkan fakta bahwa

operasi lain (yaitu perkalian) dapat didefinisikan pada elemen-elemen dalam golongan tersebut.

Beralih ke grup dengan signifikansi kriptografi, kami memfokuskan perhatian kami pada (subgrup dari) $\mathbb{Z}_p^*$ untuk p prime. (Lihat Bagian 8.3.3.) Kami memberikan gambaran tingkat tinggi algoritma kalkulus indeks untuk memecahkan masalah logaritma diskrit dalam kelompok tersebut dalam waktu sub-eksponensial. Saat ini, algoritma yang paling dikenal untuk kelas grup ini adalah saringan bidang bilangan,[2] yang berjalan secara heuristik dalam waktu $2^{O((\log p)^{1/3} \cdot (\log \log p)^{2/3})}$. Algoritma sub-eksponensial untuk menghitung logaritma diskrit dalam subkelompok perkalian bidang berhingga dengan karakteristik besar juga telah diketahui. Pada awal tahun 2013, kemajuan signifikan dicapai dalam algoritma untuk menghitung logaritma diskrit dalam subkelompok perkalian bidang berhingga dengan karakteristik kecil; tampaknya bijaksana untuk menghindari penggunaan kelompok tersebut untuk aplikasi kriptografi.

Yang penting, tidak ada algoritma sub-eksponensial yang dikenal untuk menghitung logaritma diskrit dalam kelompok kurva elips tertentu. Ini berarti bahwa untuk tingkat keamanan tertentu, kita dapat menggunakan kelompok dengan urutan yang lebih kecil ketika bekerja dalam kelompok kurva elips dibandingkan dengan, katakanlah, bekerja di $\mathbb{Z}_p^*$, sehingga menghasilkan skema kriptografi dengan efisiensi asimtotik yang lebih baik.

Algoritma Pohlig–Hellman

Algoritma Pohlig – Hellman dapat digunakan untuk mempercepat penghitungan logaritma diskrit dalam grup $\mathbb{G}$ ketika faktor non-trivial dari orde grup q diketahui. Ingatlah bahwa orde suatu elemen g, yang dilambangkan di sini dengan ord(g), adalah bilangan bulat positif terkecil i yang $g^i = 1$. Kita memerlukan lemma berikut:

LEMMA 9.4 Misalkan ord$(g) = q$, dan ucapkan $p \mid q$. Maka ord$(g^p) = q/p$.

BUKTI Karena $(g^p)^{q/p} = g^q = 1$, maka orde gp paling banyak adalah q/p. Misal $i > 0$ sehingga$(g^p)^i = 1$. Maka $g^{pi} = 1$ dan, karena q adalah orde dari g, kita harus mempunyai $pi \geq q$ atau ekuivalen $i \geq q/p$. Urutan gp dengan demikian tepat q/p.

Kita juga akan menggunakan generalisasi dari teorema sisa Cina: jika $q = \prod_{i=1}^{k} q_i$ dan gcd$(q_i, q_j) = 1$ untuk semua $i \neq j$ maka

$$\mathbb{Z}_q \simeq \mathbb{Z}_{q_1} \times \cdots \times \mathbb{Z}_{q_k} \quad \text{and} \quad \mathbb{Z}_q^* \simeq \mathbb{Z}_{q_1}^* \times \cdots \times \mathbb{Z}_{q_k}^*.$$

(Hal ini dapat dibuktikan dengan induksi pada k, menggunakan teorema sisa dasar bahasa Cina untuk $k = 2$.) Terlebih lagi, dengan perluasan algoritma pada Bagian 8.1.5 dimungkinkan untuk mengkonversi secara efisien antara representasi elemen sebagai elemen $\mathbb{Z}_q$ dan representasinya sebagai elemen $\mathbb{Z}_{q1} \times \cdots \times \mathbb{Z}_{qk}$.

Kami sekarang menjelaskan algoritma Pohlig – Hellman. Kita diberi generator g dan elemen h dan ingin mencari x sehingga $g^x = h$. Katakanlah faktorisasi $q = \prod_{i=1}^{k} q_i$ diketahui dengan $\{q_i\}$ berpasangan relatif prima. (Ini tidak harus berupa faktorisasi prima lengkap dari q.) Kita tahu itu

$$\left(g^{q/q_i}\right)^x = (g^x)^{q/q_i} = h^{q/q_i} \text{ for } i = 1, \ldots, k. \qquad (9.2)$$

Membiarkan $g^i \stackrel{\text{def}}{=} g^{q/qi}$ dan $h^i \stackrel{\text{def}}{=} h^{q/qi}$, maka kita mempunyai k contoh masalah diskretelogaritma dalam k kelompok yang lebih kecil. Secara khusus, setiap soal $g_i^x = h_i$ berada dalam subgrup berukuran ord$(g_i) = q_i$ (menurut Lemma 9.4). (Perhatikan bahwa setiap soal hanya menentukan $[x \bmod q_i]$; ini mengikuti Proposisi 8.53.)

Kita dapat menyelesaikan setiap k instance yang dihasilkan menggunakan algoritma apa pun untuk menyelesaikan masalah logaritma diskrit. Menyelesaikan kasus ini memberikan serangkaian jawaban $\{x_i\}_{i=1}^k$, dengan $x_i \in \mathbb{Z}_{qi}$, yang mana $g_i^{xi} = h_i = g_i^x$. Proposisi 8.53 menyiratkan bahwa $x = x_i$ mod qi untuk semua i. Dengan teorema sisa umum Tiongkok yang dibahas sebelumnya, kendalanya

$$\begin{aligned} x &= x_1 \bmod q_1 \\ &\vdots \\ x &= x_k \bmod q_k \end{aligned}$$

tentukan secara unik x modulo q, dan solusi yang diinginkan x dapat direkonstruksi secara efisien dari $\{x_i\}$.

Contoh 9.5

Pertimbangkan masalah penghitungan logaritma diskrit dalam $\mathbb{Z}_{31}^*$, sekelompok ordo $q = 30 = 5 \cdot 3 \cdot 2$. Katakanlah $g = 3$ dan $h = 26 = g^x$ dengan x tidak diketahui. Kita punya:

$$\begin{aligned} (g^{30/5})^x = h^{30/5} &\Rightarrow (3^6)^x = 26^6 \Rightarrow 16^x = 1 \\ (g^{30/3})^x = h^{30/3} &\Rightarrow (3^{10})^x = 26^{10} \Rightarrow 25^x = 5 \\ (g^{30/2})^x = h^{30/2} &\Rightarrow (3^{15})^x = 26^{15} \Rightarrow 30^x = 30. \end{aligned}$$

(Semua persamaan di atas adalah modulo 31.) Kita mempunyai ord(16) = 5, ord(25) = 3, dan ord(30) = 2. Menyelesaikan setiap persamaan, kita peroleh

$$x = 0 \bmod 5, \quad x = 2 \bmod 3, \quad \text{and} \quad x = 1 \bmod 2,$$

jadi $x = 5$ mod 30. Memang $3^5 = 26$ mod 31.

Jika q mempunyai faktorisasi prima (yang diketahui) $q = \prod_{i=}^k p_i^{ei}$ maka dengan menggunakan algoritma Pohlig–Hellman, waktu untuk menghitung logaritma diskrit pada suatu kelompok orde q didominasi oleh perhitungan logaritma diskrit pada subgrup yang berukuran max$_i$ $\{p_i^{ei}\}$. Hal ini selanjutnya dapat direduksi menjadi penghitungan logaritma diskrit dalam subkelompok berukuran max$_i\{p_i\}$;

Algoritma Langkah Bayi/Langkah Raksasa

Algoritma baby-step/giant-step menghitung logaritma diskrit dalam grup orde q menggunakan operasi grup $\mathcal{O}(\sqrt{q})$. Idenya sederhana. Diberikan generator $g \in \mathbb{G}$, kita dapat membayangkan pangkat g membentuk suatu siklus

$$1 = g^0, g^1, g^2, \ldots, g^{q-2}, g^{q-1}, g^q = 1.$$

Kita tahu bahwa h pasti terletak di suatu tempat dalam siklus ini. Menghitung semua titik dalam siklus ini akan memerlukan waktu $\Omega(q)$. Sebaliknya, kita "menandai" siklus pada interval ukuran $t \stackrel{\text{def}}{=} \lfloor\sqrt{q}\rfloor$; lebih tepatnya, kita menghitung dan menyimpan elemen $\lfloor q/t \rfloor + 1 = \mathcal{O}(\sqrt{q})$.

$$g^0, g^t, g^{2t}, \ldots, g^{\lfloor q/t \rfloor \cdot t}.$$

(Ini adalah "langkah raksasa.") Perhatikan bahwa jarak antara "tanda" yang berurutan paling banyak adalah t. Lebih lanjut, kita mengetahui bahwa $h = g^x$ terletak pada salah satu celah tersebut. Jadi, jika kita selanjutnya mengambil "langkah kecil" dan menghitung elemen t

$$h \cdot g^1, \ldots, h \cdot g^t,$$

yang masing-masing sesuai dengan "pergeseran" h, kita tahu bahwa salah satu nilai ini akan sama dengan salah satu titik yang telah kita tandai. Katakanlah kita menemukan $h \cdot g^i = g^{k.t}$. Kita kemudian dapat dengan mudah menghitung $\log_g h := [(kt - i) \bmod q]$. Pseudocode untuk algoritma ini mengikuti

ALGORITHM 9.6
The baby-step/giant-step algorithm

Input: Elements $g, h \in \mathbb{G}$; the order q of $\mathbb{G}$
Output: $\log_g h$

$t := \lfloor\sqrt{q}\rfloor$
for $i = 0$ to $\lfloor q/t \rfloor$:
  compute $g_i := g^{i \cdot t}$
sort the pairs (i, g_i) by their second component
for $i = 1$ to t:
  compute $h_i := h \cdot g^i$
  if $h_i = g_k$ for some k, **return** $[kt - i \bmod q]$

Algoritma ini membutuhkan $\mathcal{O}(\sqrt{q})$ eksponensial/perkalian dalam $\mathbb{G}$. (Faktanya, selain nilai pertama $g_1 = g^t$, setiap nilai g_i dapat dihitung menggunakan perkalian tunggal sebagai $g_i := g_{i-1} \cdot g_1$. Demikian pula, masing-masing h_i dapat dihitung sebagai $h_i := h_{i-1} \cdot g.$) Menyortir pasangan $\mathcal{O}(\sqrt{q})\{(i, g_i)\}$ membutuhkan waktu $\mathcal{O}(\sqrt{q} . \log q)$, dan kita kemudian dapat menggunakan pencarian biner untuk memeriksa apakah setiap hi sama dengan beberapa g_k dalam waktu $\mathcal{O}(\log q)$. Algoritme keseluruhan berjalan dalam waktu $\mathcal{O}(\sqrt{q} . poly \log(q))$,

Contoh 9.7

Kami menunjukkan penerapan algoritma dalam grup siklik $\mathbb{Z}_{29}^*$ berorde $q = 29 - 1 = 28$. Ambil $g = 2$ dan $h = 17$. Kita tentukan $t = 5$ dan hitung:

$$2^0 = 1,\ \ 2^5 = 3,\ \ 2^{10} = 9,\ \ 2^{15} = 27,\ \ 2^{20} = 23,\ \ 2^{25} = 11.$$

(Perlu dipahami bahwa semua operasi berada di $\mathbb{Z}_{29}^*$.) Kemudian hitung:

$$17 \cdot 2^1 = 5,\quad 17 \cdot 2^2 = 10,\quad 17 \cdot 2^3 = 20,\quad 17 \cdot 2^4 = 11,$$

dan perhatikan bahwa $17 \cdot 2^4 = 11 = 2^{25}$. Jadi kita mempunyai $\log_2 17 = 25 — 4 = 21$.

Logaritma Diskrit dari Tumbukan

Kelemahan dari algoritma baby-step/giant-step adalah algoritma ini menggunakan sejumlah besar memori, karena memerlukan penyimpanan poin $\mathcal{O}(\sqrt{q})$. Kita dapat memperoleh algoritme yang menggunakan memori konstan—dan memiliki waktu berjalan asimtotik yang sama—dengan mengeksploitasi hubungan antara masalah logaritma diskrit dan hashing anti-tabrakan seperti yang ditunjukkan pada Bagian 8.4, dan mengingat serangan ulang tahun berruang kecil untuk menemukan tabrakan dari Bagian 5.4.

Kami menggambarkan ide tingkat tinggi. Perbaiki basis $g \in \mathbb{G}$ dan beberapa elemen $h \in \langle g \rangle$. Ingat kembali hasil Bagian 8.4.2 bahwa jika kita mendefinisikan fungsi hash $H_{g,h} : \mathbb{Z}_q \times \mathbb{Z}_q \to \mathbb{G}$ oleh $H_{g,h}(x_1, x_2) = g^{x_1} h^{x_2}$, maka menemukan tumbukan di $H_{g,h}$ menyiratkan kemampuan untuk menghitung $\log_g h$. (Lihat bukti Teorema 8.79.) Dengan demikian, kita telah mengurangi masalah penghitungan $\log_g h$ menjadi masalah menemukan tumbukan dalam fungsi hash, sesuatu yang kita tahu bagaimana melakukannya dalam waktu $\mathcal{O}(\sqrt{|\mathbb{G}|}) = \mathcal{O}(\sqrt{q})$ menggunakan serangan ulang tahun! Terlebih lagi, serangan ulang tahun dengan ruang kecil akan memberikan tumbukan dalam waktu dan ruang yang konstan.

Tinggal membahas beberapa rincian teknis saja. Salah satunya adalah serangan ulang tahun dalam ruang kecil yang dijelaskan pada Bagian 5.4 mengasumsikan bahwa rentang fungsi hash adalah subset dari domainnya; bukan itu yang terjadi di sini, dan pada kenyataannya (tergantung pada representasi yang digunakan untuk elemen $\mathbb{G}$) bahkan bisa jadi $H_{g,h}$ tidak terkompresi. Masalah kedua adalah analisis pada Bagian 5.4 memperlakukan fungsi hash sebagai fungsi acak, sedangkan $H_{g,h}$ memiliki sejumlah besar struktur aljabar.

Algoritma rho Pollard memberikan salah satu cara untuk mengatasi masalah di atas. Kami menjelaskan algoritme berbeda yang dapat dipandang sebagai implementasi lebih langsung dari gagasan di atas. (Dalam praktiknya, algoritme Pollard akan lebih efisien, meskipun kedua algoritme hanya menggunakan operasi grup $\mathcal{O}(\sqrt{q})$.) Misalkan $F : \mathbb{G} \to \mathbb{Z}_q \times \mathbb{Z}_q$ menunjukkan fungsi hash kriptografi yang diperoleh, misalnya, modifikasi yang sesuai dari SHA-1 . Definisikan $H : \mathbb{G} \to \mathbb{G}$ dengan $H(k) \stackrel{\text{def}}{=} H_{g,h}(F(k))$.

Kita dapat menggunakan Algoritma 5.9, dengan modifikasi alami, untuk menemukan tumbukan di H menggunakan evaluasi $\mathcal{O}\left(\sqrt{|\mathbb{G}|}\right) = \mathcal{O}\left(\sqrt{q}\right)$ dari H dalam ekspektasi (dan memori konstan). Dengan probabilitas yang sangat besar, hal ini menghasilkan tumbukan dalam $H_{g,h}$. Anda diminta untuk menyempurnakan rinciannya dalam Latihan 9.6.

Menarik untuk diamati di sini bahwa bukti keamanan yang berdasarkan pada tingkat kekerasan masalah logaritma diskrit—yakni, bahwa hal ini menyiratkan fungsi hash yang tahan benturan—mengarah pada algoritma yang lebih baik untuk memecahkan masalah yang sama! Sedikit refleksi akan meyakinkan kita bahwa hal ini tidak mengherankan: pembuktian dengan reduksi menunjukkan bahwa serangan terhadap beberapa konstruksi (dalam hal ini, menemukan tabrakan dalam fungsi hash) secara langsung menghasilkan serangan terhadap asumsi yang mendasarinya (di sini, kekerasan dari masalah logaritma diskrit), yang merupakan properti yang dieksploitasi oleh algoritma di atas.

Algoritma Kalkulus Indeks

Kami menyimpulkan dengan sekilas algoritma kalkulus indeks (non-generik) untuk menghitung logaritma diskrit dalam grup siklik $\mathbb{Z}_p^*$ (untuk p prima). Berbeda dengan algoritma (generik) sebelumnya, pendekatan ini memiliki waktu berjalan sub-eksponensial. Algoritme ini memiliki beberapa kemiripan dengan algoritma saringan kuadrat yang diperkenalkan pada Bagian 9.1, dan kami berasumsi pembaca sudah familiar dengan diskusi di sana. Dalam hal ini, kami membahas gagasan utama metode kalkulus indeks namun meninggalkan analisis terperinci di luar cakupan pembahasan kami. Juga, beberapa penyederhanaan diperkenalkan untuk memperjelas presentasi.

Metode kalkulus indeks menggunakan proses dua langkah. Yang penting, langkah pertama hanya memerlukan pengetahuan tentang modulus p dan basis g sehingga dapat dijalankan sebagai langkah pra-pemrosesan sebelum h—nilai yang logaritma diskritnya ingin kita hitung—diketahui. Untuk alasan yang sama, cukup menjalankan langkah pertama sekali saja untuk menyelesaikan beberapa contoh masalah logaritma diskrit (selama semua contoh tersebut memiliki p dan g yang sama).

Langkah 1. Perbaiki beberapa batasan B, dan biarkan $\{p_1, \ldots, p_k\}$ adalah himpunan bilangan prima yang kurang dari atau sama dengan B. Pada langkah ini, kita mencari $\ell \geq k$ nilai berbeda $x_1, \ldots, x_\ell \in \mathbb{Z}_{p-1}$ yang mana $g_i \stackrel{\text{def}}{=} [g^{x_i} \bmod p]$ adalah B-halus. Hal ini dilakukan hanya dengan memilih seragam $\{x_i\}$ hingga nilai yang sesuai ditemukan.

Memfaktorkan bilangan halus B yang dihasilkan, kita mendapatkan persamaan ℓ:

$$g^{x_1} = \prod_{i=1}^{k} p_i^{e_{1,i}} \bmod p$$

$$\vdots$$

$$g^{x_\ell} = \prod_{i=1}^{k} p_i^{e_{\ell,i}} \bmod p\,.$$

Dengan mengambil logaritma diskrit, kita dapat mengubahnya menjadi persamaan linier

$$x_1 = \sum_{i=1}^{k} e_{1,i} \cdot \log_g p_i \mod (p-1) \qquad (9.3)$$

$$\vdots$$

$$x_\ell = \sum_{i=1}^{k} e_{\ell,i} \cdot \log_g p_i \mod (p-1). \qquad (9.4)$$

Perhatikan bahwa $\{x_i\}$ dan $\{e_{i,j}\}$ diketahui, sedangkan $\{\log_g p_i\}$ tidak diketahui.

Langkah 2. Sekarang kita diberikan elemen h dan ingin menghitung $\log_g h$. Di sini, kita menemukan nilai $x \in \mathbb{Z}_{p-1}$ yang $[g^x . h \bmod p]$ adalah B-smooth. (Sekali lagi, ini dilakukan hanya dengan memilih x secara seragam.) Katakanlah

$$g^x \cdot h = \prod_{i=1}^{k} p_i^{e_i} \bmod p$$

$$\Rightarrow x + \log_g h = \sum_{i=1}^{k} e_i \cdot \log_g p_i \mod (p-1),$$

dimana x dan $\{e_i\}$ diketahui. Jika digabungkan dengan Persamaan (9.3)–(9.4), kita mempunyai persamaan linier $\ell + 1 \geq k + 1$ pada $k + 1$ bilangan tak diketahui $\{\log_g p_i\}_{i=1}^{k}$ dan $\log_g h$. Dengan menggunakan metode aljabar linier3 (dan dengan asumsi sistem persamaan tidak terdefinisi di bawah), kita dapat menyelesaikan setiap hal yang tidak diketahui dan khususnya mendapatkan solusi yang diinginkan $\log_g h$.

Contoh 9.8

Misalkan $p = 101, g = 3$, dan $h = 87$. Kita mempunyai $[3^{10} \bmod 101] = 65 = 5\,13$. Demikian pula, $[3^{12} \bmod 101] = 80 = 2^4 . 5$ dan $[3^{14} \bmod 101] = 13$. Jadi kita mempunyai persamaan linier

$$10 = \log_3 5 + \log_3 13 \mod 100$$
$$12 = 4 \cdot \log_3 2 + \log_3 5 \mod 100$$
$$14 = \log_3 13 \mod 100.$$

Kami juga memiliki $3^5 \cdot 87 = 32 = 2^5 \bmod 101$, atau

$$5 + \log_3 87 = 5 \cdot \log_3 2 \mod 100. \qquad (9.5)$$

Dengan menjumlahkan persamaan kedua dan ketiga serta mengurangkan persamaan pertama, kita memperoleh $4 \cdot \log_3 2 = 16 \bmod 100$. Persamaan ini tidak menentukan $\log_3 2$ secara unik (karena 4 bukan modulo 100 yang dapat dibalik), namun hal ini memberi tahu kita bahwa $\log_3 2 = 4, 29, 54$, atau 79 (lih. Latihan 9.3). Mencoba semua kemungkinan

menghasilkan $\log_3 2 = 29$. Memasukkannya ke dalam Persamaan (9.5) menghasilkan $\log_3 87 = 40$.

Durasi. Memilih nilai B yang lebih besar akan memperbesar kemungkinan bahwa nilai seragam di $\mathbb{Z}_p^*$ adalah B-smooth; namun, ini berarti kita harus bekerja lebih keras untuk mengidentifikasi dan memfaktorkan bilangan mulus B, dan kita harus menemukan lebih banyak bilangan tersebut. Karena sistem persamaannya akan lebih besar, penyelesaian sistemnya akan memakan waktu lebih lama. Memilih nilai optimal B menghasilkan algoritma yang (setidaknya secara heuristik) menghitung logaritma diskrit dalam $\mathbb{Z}_p^*$ dalam waktu $2^{\mathcal{O}(\sqrt{\log p \log\log p})}$. (Faktanya, suku konstanta dalam eksponen dapat ditentukan dengan cukup tepat.) Hal yang penting untuk tujuan kita adalah bahwa bilangan ini merupakan sub-eksponensial pada panjang p.

9.3 PANJANG KUNCI YANG DIREKOMENDASIKAN

Pengetahuan tentang algoritma terbaik yang tersedia untuk memecahkan berbagai masalah kriptografi sangat penting untuk menentukan panjang kunci yang tepat untuk mencapai tingkat keamanan yang diinginkan. Tabel berikut merangkum panjang kunci yang saat ini direkomendasikan oleh Institut Standar dan Teknologi Nasional AS4 (NIST):

	RSA	Discrete Logarithm	
Effective Key Length	Modulus Length	Order-q Subgroup of $\mathbb{Z}_p^*$	Elliptic-Curve Group Order q
112	2048	p: 2048, q: 224	224
128	3072	p: 3072, q: 256	256
192	7680	p: 7680, q: 384	384
256	15360	p: 15360, q: 512	512

Semua angka dalam tabel diukur dalam bit. “Panjang kunci efektif” adalah nilai n sehingga algoritma yang paling dikenal untuk menyelesaikan suatu masalah membutuhkan waktu sekitar 2^n; yaitu, kesulitan komputasi dalam menyelesaikan suatu masalah kira-kira setara dengan melakukan pencarian brute force terhadap skema kunci simetris dengan kunci n-bit, atau waktu untuk menemukan tabrakan dalam fungsi hash dengan keluaran $2n$-bit panjang. NIST menganggap panjang kunci efektif 112-bit dapat diterima untuk keamanan hingga tahun 2030, namun merekomendasikan panjang kunci 128-bit atau lebih tinggi untuk aplikasi yang membutuhkan keamanan lebih dari itu.

Mengingat apa yang telah kita pelajari dalam bab ini, ada baiknya jika kita melihat lebih dekat beberapa angka dalam tabel. Hal pertama yang mungkin kita perhatikan adalah bahwa grup kurva elips dapat digunakan untuk mewujudkan tingkat keamanan tertentu dengan parameter yang lebih kecil dibandingkan RSA atau subgrup $\mathbb{Z}_p^*$. Hal ini disebabkan karena tidak ada algoritma sub-eksponensial yang dikenal untuk memecahkan masalah logaritma diskrit dalam kelompok tersebut (jika dipilih dengan tepat). Namun, untuk mencapai keamanan n-bit, diperlukan grup kurva elips yang pesanan q panjangnya $2n$-bit. Ini adalah konsekuensi dari algoritma umum yang telah kita lihat dalam bab ini, yang menyelesaikan masalah logaritma diskrit (dalam grup mana pun) dalam waktu $\mathcal{O}(\sqrt{q}) \approx 2^n$.

Beralih ke kasus $\mathbb{Z}_p^*$ kita melihat bahwa di sini juga, nilai q sebesar $2n$-bit diperlukan untuk keamanan n-bit. Namun, panjang p harus jauh lebih besar, karena saringan bidang bilangan dapat digunakan untuk menghitung logaritma diskrit dalam $\mathbb{Z}_p^*$ dalam waktu sub-eksponensial dalam panjang p. (Artinya, p dan q dipilih sedemikian rupa sehingga waktu berjalan dari saringan bidang bilangan, yang bergantung pada panjang p, dan waktu berjalan dari algoritma generik, yang bergantung pada panjang q, akan kira-kira sama.) Konsekuensi praktis dari hal ini adalah, untuk tingkat keamanan apa pun yang diinginkan, sistem kriptografi kurva elips dapat menggunakan parameter yang jauh lebih kecil, dengan operasi grup yang jauh lebih cepat untuk pihak yang jujur, dibandingkan sistem kriptografi yang didasarkan pada subgrup $\mathbb{Z}_p^*$.

BAB 10
MANAJEMEN KUNCI DAN REVOLUSI KUNCI PUBLIK

10.1 DISTRIBUSI KUNCI DAN MANAJEMEN KUNCI

Dalam Bab 1–7 kita telah melihat bagaimana kriptografi kunci pribadi dapat digunakan untuk memastikan kerahasiaan dan integritas dua pihak yang berkomunikasi melalui saluran yang tidak aman, dengan asumsi kedua pihak memiliki kunci rahasia bersama. Namun, pertanyaan yang kami tunda sejak Bab 1 adalah:

Bagaimana cara para pihak berbagi kunci rahasia?
Jelasnya, kunci tersebut tidak bisa begitu saja dikirimkan melalui saluran komunikasi publik karena musuh yang menguping akan dapat mengamatinya dalam perjalanan. Beberapa mekanisme lain harus digunakan sebagai gantinya.

Dalam beberapa situasi, para pihak mungkin memiliki akses ke saluran aman yang dapat mereka gunakan untuk berbagi kunci rahasia secara andal. Salah satu contoh umum adalah ketika kedua pihak berada di lokasi yang sama pada suatu waktu, di mana mereka dapat berbagi kunci. Alternatifnya, para pihak mungkin bisa menggunakan jasa kurir terpercaya sebagai jalur aman. Kami menekankan bahwa fakta bahwa para pihak dapat berbagi kunci—dan karenanya harus memiliki akses ke saluran aman pada suatu saat—tidak menjadikan kriptografi kunci privat tidak berguna: dalam contoh pertama, para pihak memiliki saluran aman pada satu titik dalam satu waktu. waktu tetapi tidak tanpa batas waktu; pada contoh kedua, penggunaan saluran aman mungkin lebih lambat dan lebih mahal dibandingkan berkomunikasi melalui saluran tidak aman.

Pendekatan di atas telah digunakan untuk berbagi kunci dalam konteks pemerintahan, diplomatik, dan militer. Misalnya, “telepon merah” yang menghubungkan Moskow dan Washington pada tahun 1960an dienkripsi menggunakan one-time pad, dengan kunci yang dibagikan oleh kurir yang terbang dari satu negara ke negara lain dengan membawa tas kerja yang penuh dengan hasil cetakan. Pendekatan tersebut juga dapat digunakan di perusahaan, misalnya, untuk menyiapkan kunci bersama antara database pusat dan karyawan baru pada hari pertama mereka bekerja. (Kami kembali ke contoh ini di bagian selanjutnya.) Namun, mengandalkan saluran aman untuk mendistribusikan kunci tidak akan berfungsi dengan baik di banyak situasi lainnya. Misalnya saja, bayangkan sebuah perusahaan multinasional yang besar dimana setiap pasangan karyawannya mungkin memerlukan kemampuan untuk berkomunikasi dengan aman, dan komunikasi mereka juga terlindungi dari karyawan lain.

Setidaknya akan merepotkan jika setiap pasangan karyawan bertemu sehingga mereka dapat berbagi kunci dengan aman; bagi karyawan yang bekerja di kota berbeda, hal ini bahkan mungkin mustahil. Meskipun kumpulan karyawan saat ini dapat berbagi kunci satu sama lain, tidak praktis bagi mereka untuk berbagi kunci dengan karyawan baru yang bergabung setelah pembagian awal ini selesai.

Dengan asumsi N karyawan ini entah bagaimana dapat berbagi kunci satu sama lain dengan aman, kelemahan signifikan lainnya adalah setiap karyawan harus mengelola dan menyimpan $N - 1$ kunci rahasia (satu untuk setiap karyawan lainnya). Faktanya, hal ini mungkin secara signifikan mengurangi jumlah kunci yang disimpan oleh setiap pengguna, karena karyawan mungkin juga memerlukan kunci untuk berkomunikasi secara aman dengan sumber daya jarak jauh seperti database, server, printer, dan sebagainya. Perkembangan begitu banyak kunci rahasia merupakan masalah logistik yang signifikan. Terlebih lagi, semua kunci ini harus disimpan dengan aman. Semakin banyak kunci yang ada, semakin sulit untuk melindunginya, dan semakin tinggi kemungkinan beberapa kunci dicuri oleh penyerang. Sistem komputer sering kali terinfeksi virus, worm, dan bentuk perangkat lunak berbahaya lainnya yang dapat mencuri kunci rahasia dan mengirimkannya secara diam-diam melalui jaringan ke penyerang. Oleh karena itu, menyimpan kunci di komputer pribadi karyawan tidak selalu merupakan solusi yang aman.

Jelasnya, potensi kebocoran kunci rahasia selalu menjadi perhatian, terlepas dari jumlah kunci yang dimiliki masing-masing pihak. Namun, ketika hanya beberapa kunci yang perlu disimpan, ada solusi bagus yang tersedia untuk mengatasi ancaman ini. Solusi umum saat ini adalah menyimpan kunci pada perangkat keras yang aman seperti kartu pintar. Kartu pintar dapat melakukan perhitungan kriptografi menggunakan kunci rahasia yang disimpan, memastikan bahwa kunci ini tidak pernah masuk ke komputer pribadi pengguna. Jika dirancang dengan benar, kartu pintar akan jauh lebih tahan terhadap serangan dibandingkan komputer pribadi—misalnya, kartu tersebut biasanya tidak dapat terinfeksi oleh malware—sehingga menawarkan cara yang baik untuk melindungi kunci rahasia pengguna. Sayangnya, kartu pintar biasanya memiliki memori yang terbatas sehingga tidak dapat menyimpan ratusan (atau ribuan) kunci.

Kekhawatiran yang diuraikan di atas semuanya dapat diatasi—pada prinsipnya, meskipun tidak dalam praktiknya—dalam organisasi “tertutup” yang terdiri dari populasi pengguna yang terdefinisi dengan baik, yang semuanya bersedia mengikuti kebijakan yang sama dalam mendistribusikan dan menyimpan kunci. Namun, sistem tersebut tidak berfungsi dalam “sistem terbuka” di mana pengguna melakukan interaksi sementara, tidak dapat mengatur pertemuan fisik, dan bahkan mungkin tidak menyadari keberadaan satu sama lain hingga mereka ingin berkomunikasi. Faktanya, ini adalah situasi yang lebih umum daripada yang mungkin disadari pada awalnya: pertimbangkan untuk menggunakan enkripsi untuk mengirim informasi kartu kredit ke pedagang Internet yang belum pernah Anda beli sebelumnya, atau mengirim email ke seseorang yang belum pernah Anda beli. bertemu secara langsung. Dalam kasus seperti ini, kriptografi kunci privat saja tidak memberikan solusi, dan kita harus mencari solusi yang memadai.

Ringkasnya, setidaknya ada tiga masalah berbeda terkait penggunaan kriptografi kunci privat. Yang pertama adalah distribusi kunci; yang kedua adalah menyimpan dan mengelola kunci rahasia dalam jumlah besar; yang ketiga adalah tidak dapat diterapkannya kriptografi kunci pribadi pada sistem terbuka.

10.2 SOLUSI PARSIAL: PUSAT DISTRIBUSI KUNCI

Salah satu cara untuk mengatasi beberapa kekhawatiran dari bagian sebelumnya adalah dengan menggunakan pusat distribusi kunci (KDC) untuk membuat kunci bersama. Coba perhatikan lagi kasus perusahaan besar yang seluruh pasangan karyawannya harus bisa berkomunikasi dengan aman. Dalam situasi seperti ini, kita dapat memanfaatkan fakta bahwa semua karyawan mungkin mempercayai suatu entitas—misalnya, administrator sistem—setidaknya dalam hal keamanan informasi terkait pekerjaan. Entitas tepercaya ini kemudian dapat bertindak sebagai KDC dan membantu semua karyawan berbagi kunci berpasangan.

Ketika seorang karyawan baru bergabung, KDC dapat membagikan kunci dengan karyawan tersebut (secara langsung, di lokasi yang aman) sebagai bagian dari hari pertama karyawan tersebut bekerja. Pada saat yang sama, KDC juga dapat mendistribusikan kunci bersama antara karyawan tersebut dan seluruh karyawan yang ada. Artinya, ketika karyawan ke-i bergabung, KDC dapat (selain berbagi kunci antara dirinya dan karyawan baru ini) menghasilkan $i - 1$ kunci $k_1, \ldots, k_{i-1}$, berikan kunci ini kepada karyawan baru, lalu kirimkan kunci k_j ke karyawan ke-j yang ada dengan mengenkripsinya menggunakan kunci yang telah dibagikan oleh karyawan tersebut dengan KDC. Setelah itu, karyawan baru tersebut berbagi kunci dengan setiap karyawan lainnya (dan juga dengan KDC).

Pendekatan yang lebih baik, yang menghindari keharusan karyawan untuk menyimpan dan mengelola banyak kunci, adalah dengan memanfaatkan KDC secara online untuk menghasilkan kunci “sesuai permintaan” setiap kali dua karyawan ingin berkomunikasi dengan aman. Seperti sebelumnya, KDC akan membagikan kunci (berbeda) kepada setiap karyawan, sesuatu yang dapat dilakukan dengan aman pada hari pertama karyawan bekerja. Katakanlah KDC berbagi kunci k_A dengan karyawan Alice, dan k_B dengan karyawan Bob. Di lain waktu, ketika Alice ingin berkomunikasi secara aman dengan Bob, dia cukup mengirimkan pesan ''Saya, Alice, ingin berbicara dengan Bob'' ke KDC. (Jika diinginkan, pesan ini dapat diautentikasi menggunakan kunci yang dibagikan oleh Alice dan KDC.) KDC kemudian memilih kunci acak baru—disebut kunci sesi—dan mengirimkan kunci ini k ke Alice yang dienkripsi menggunakan k_A, dan ke Bob yang dienkripsi menggunakan k_B. (Protokol ini terlalu sederhana untuk digunakan dalam praktik; lihat pembahasan lebih lanjut di bawah.) Setelah Alice dan Bob memulihkan kunci sesi ini, mereka dapat menggunakannya untuk berkomunikasi dengan aman. Ketika mereka selesai dengan percakapannya, mereka dapat (dan harus) menghapus kunci sesi karena mereka selalu dapat menghubungi KDC lagi jika mereka ingin berkomunikasi di lain waktu.

Pertimbangkan keuntungan dari pendekatan ini:

1. Setiap karyawan hanya perlu menyimpan satu kunci rahasia jangka panjang (yaitu kunci yang mereka bagikan dengan KDC). Karyawan masih perlu mengelola dan menyimpan kunci sesi, namun ini adalah kunci jangka pendek yang dihapus setelah sesi komunikasi selesai.

   KDC perlu menyimpan banyak kunci jangka panjang. Namun, KDC dapat disimpan di lokasi yang aman dan diberikan perlindungan setinggi mungkin terhadap serangan jaringan.

2. Ketika seorang karyawan bergabung dengan organisasi, yang harus dilakukan hanyalah menyiapkan kunci antara karyawan tersebut dan KDC. Tidak ada karyawan lain yang perlu memperbarui kumpulan kunci yang mereka pegang.

Dengan demikian, KDC dapat meringankan dua masalah yang telah kita lihat sehubungan dengan kriptografi kunci pribadi: KDC dapat menyederhanakan distribusi kunci (karena hanya satu kunci baru yang harus dibagikan ketika seorang karyawan bergabung, dan masuk akal untuk mengasumsikan adanya saluran aman antara KDC dan karyawan tersebut pada hari pertama mereka bekerja), dan dapat mengurangi kompleksitas penyimpanan kunci (karena setiap karyawan hanya perlu menyimpan satu kunci). KDC berupaya keras untuk menjadikan kriptografi kunci pribadi praktis dalam organisasi besar di mana terdapat satu entitas yang dipercaya oleh semua orang.

Namun, ada beberapa kelemahan dalam mengandalkan KDC:

1. Serangan yang berhasil pada KDC akan mengakibatkan kerusakan total pada sistem: penyerang dapat mengkompromikan semua kunci dan kemudian menguping semua lalu lintas jaringan. Hal ini menjadikan KDC sebagai target yang bernilai tinggi. Perhatikan bahwa meskipun KDC terlindungi dengan baik dari serangan eksternal, selalu ada kemungkinan serangan orang dalam oleh karyawan yang memiliki akses ke KDC (misalnya, manajer TI).
2. KDC adalah satu titik kegagalan: jika KDC mati, komunikasi yang aman untuk sementara tidak mungkin dilakukan. Jika karyawan terus-menerus menghubungi KDC dan meminta kunci sesi dibuat, beban pada KDC bisa sangat tinggi, sehingga meningkatkan kemungkinan kegagalan atau lambatnya respons.

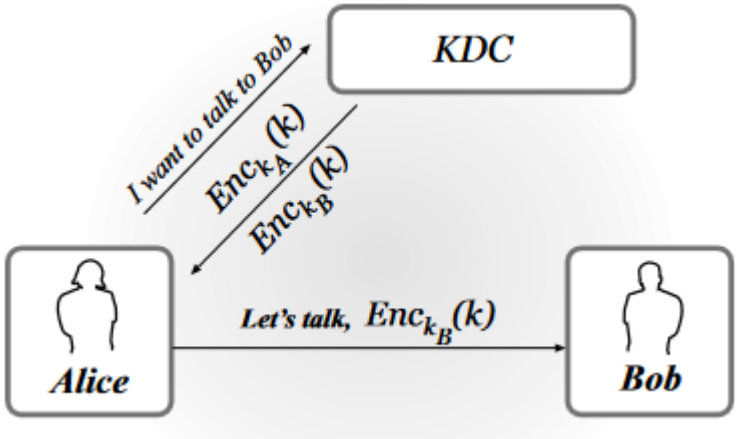
Solusi sederhana untuk masalah kedua adalah dengan mereplikasi KDC. Hal ini berhasil (dan dilakukan dalam praktiknya), namun juga berarti kini terdapat lebih banyak titik serangan pada sistem. Menambahkan lebih banyak KDC juga mempersulit penambahan karyawan baru, karena pembaruan harus disebarkan secara aman ke setiap KDC.

Protokol untuk distribusi kunci menggunakan KDC. Ada sejumlah protokol dalam literatur untuk distribusi kunci yang aman menggunakan KDC. Kami menyebutkan secara khusus protokol Needham-Schroeder, yang merupakan inti dari Kerberos, sebuah layanan penting dan banyak digunakan untuk melakukan otentikasi dan mendukung komunikasi yang aman. (Kerberos digunakan di banyak universitas dan perusahaan, dan merupakan mekanisme default untuk mendukung otentikasi dan komunikasi jaringan yang aman di Windows dan banyak sistem UNIX.) Kami hanya menyoroti satu fitur dari protokol ini. Ketika Alice menghubungi KDC dan meminta untuk berkomunikasi dengan Bob, KDC tidak mengirimkan kunci sesi terenkripsi ke Alice dan Bob seperti yang telah kami jelaskan sebelumnya. Sebaliknya, KDC mengirimkan kepada Alice kunci sesi yang dienkripsi dengan kunci Alice selain kunci sesi yang dienkripsi dengan kunci Bob. Alice kemudian meneruskan ciphertext kedua kepada Bob seperti pada Gambar 10.1. Cipherteks kedua terkadang disebut tiket, dan dapat dilihat sebagai kredensial yang memungkinkan Alice untuk berbicara dengan Bob (dan memungkinkan Bob yakin bahwa dia sedang berbicara dengan Alice). Memang benar, meskipun kami belum menekankan hal ini dalam diskusi kami, pendekatan berbasis

KDC juga dapat memberikan cara yang berguna untuk melakukan otentikasi. Perhatikan juga bahwa Alice dan Bob tidak harus keduanya menjadi pengguna; Alice mungkin adalah pengguna dan Bob mungkin adalah sumber daya seperti server, disk jarak jauh, atau printer.

Protokol dirancang sedemikian rupa untuk mengurangi beban pada KDC. Dalam protokol seperti yang dijelaskan, KDC tidak perlu memulai koneksi kedua ke Bob, dan tidak perlu khawatir apakah Bob sedang online ketika Alice memulai protokol. Terlebih lagi, jika Alice menyimpan tiketnya (dan salinan kunci sesinya), maka dia dapat memulai kembali komunikasi yang aman dengan Bob hanya dengan mengirimkan ulang tiket tersebut ke Bob, tanpa keterlibatan KDC sama sekali. (Dalam praktiknya, tiket akan habis masa berlakunya dan pada akhirnya perlu diperbarui. Namun sesi dapat dimulai kembali dalam jangka waktu tertentu yang dapat diterima.)

Kami menyimpulkan dengan mencatat bahwa dalam praktiknya, kunci yang dibagikan Alice dengan KDC mungkin berupa kata sandi yang pendek dan mudah diingat. Dalam hal ini, muncul banyak masalah keamanan tambahan yang harus ditangani. Kami juga secara implisit mengasumsikan penyerang yang hanya menguping secara pasif, bukan penyerang yang mungkin secara aktif mencoba mengganggu protokol. Kami merujuk pembaca yang tertarik pada referensi di akhir bab ini untuk informasi lebih lanjut tentang bagaimana permasalahan tersebut dapat diatasi.



GAMBAR 10.1: Templat umum untuk protokol distribusi kunci.

10.3 PERTUKARAN KUNCI DAN PROTOKOL DIFFIE–HELLMAN

KDC dan protokol seperti Kerberos biasanya digunakan dalam praktik. Namun pendekatan terhadap masalah distribusi kunci ini masih memerlukan, pada titik tertentu, saluran pribadi dan terautentikasi yang dapat digunakan untuk berbagi kunci. (Secara khusus, kami mengasumsikan adanya saluran seperti itu antara KDC dan karyawan pada hari pertama mereka.) Dengan demikian, mereka masih belum dapat menyelesaikan masalah distribusi kunci dalam sistem terbuka seperti Internet, yang mungkin tidak tersedia saluran pribadi. antara dua pengguna yang ingin berkomunikasi.

Untuk mencapai komunikasi pribadi tanpa harus berkomunikasi melalui saluran pribadi, diperlukan pendekatan yang sangat berbeda. Pada tahun 1976, Whitfield Diffie dan

Martin Hellman menerbitkan sebuah makalah dengan judul “New Directions in Cryptography.” Dalam penelitian tersebut mereka mengamati bahwa sering kali terdapat asimetri di dunia; khususnya, ada tindakan tertentu yang dapat dilakukan dengan mudah namun tidak mudah dibatalkan. Misalnya, gembok dapat dikunci tanpa kunci (yaitu dengan mudah), namun kemudian tidak dapat dibuka kembali. Yang lebih menakjubkan lagi, vas kaca mudah sekali dipecahkan, namun sangat sulit untuk disatukan kembali. Secara algoritmik (dan lebih sesuai dengan tujuan kita), mudah untuk mengalikan dua bilangan prima besar namun sulit untuk mendapatkan kembali bilangan prima tersebut dari perkaliannya. (Inilah masalah pemfaktoran yang dibahas dalam bab sebelumnya.) Diffie dan Hellman menyadari bahwa fenomena seperti itu dapat digunakan untuk mendapatkan protokol interaktif untuk pertukaran kunci aman yang memungkinkan dua pihak untuk berbagi kunci rahasia, melalui komunikasi melalui saluran publik, dengan memiliki para pihak melakukan operasi yang dapat mereka balikkan tetapi tidak dapat dilakukan oleh penyadap.

Keberadaan protokol pertukaran kunci yang aman sungguh menakjubkan. Artinya, Anda dan seorang teman dapat menyepakati suatu rahasia hanya dengan berteriak ke seberang ruangan (dan melakukan perhitungan lokal); rahasianya tidak akan diketahui orang lain, bahkan jika mereka mendengarkan semua yang dikatakan. Memang benar, hingga tahun 1976, secara umum diyakini bahwa komunikasi yang aman tidak dapat dicapai tanpa terlebih dahulu membagikan sejumlah informasi rahasia menggunakan saluran pribadi. Pengaruh makalah Diffie dan Hellman sangat besar. Selain memperkenalkan cara baru yang mendasar dalam memandang kriptografi, ini adalah salah satu langkah pertama menuju perpindahan kriptografi dari domain privat ke domain publik. Kami mengutip dua paragraf pertama makalah mereka:

> *Saat ini kita berada di ambang revolusi kriptografi. Perkembangan perangkat keras digital yang murah telah membebaskannya dari keterbatasan desain komputasi mekanis dan menurunkan biaya perangkat kriptografi tingkat tinggi sehingga dapat digunakan dalam aplikasi komersial seperti mesin ATM tunai jarak jauh dan terminal komputer.*

Pada gilirannya, aplikasi semacam itu menciptakan kebutuhan akan sistem kriptografi jenis baru yang meminimalkan kebutuhan saluran distribusi kunci yang aman. . . . Pada saat yang sama, perkembangan teoretis dalam teori informasi dan ilmu komputer menjanjikan penyediaan kriptosistem yang terbukti aman, mengubah seni kuno ini menjadi sebuah sains.

Diffie dan Hellman tidak melebih-lebihkan, dan revolusi yang mereka bicarakan sebagian besar disebabkan oleh kerja mereka.

Pada bagian ini kami menyajikan protokol pertukaran kunci Diffie–Hellman. Kami membuktikan keamanannya terhadap musuh yang melakukan penyadapan atau, setara, dengan asumsi bahwa pihak-pihak tersebut berkomunikasi melalui saluran publik namun terautentikasi (sehingga penyerang tidak dapat mengganggu komunikasi mereka). Keamanan terhadap musuh yang melakukan penyadapan merupakan jaminan yang relatif lemah, dan dalam praktiknya protokol pertukaran kunci harus memenuhi gagasan keamanan yang lebih kuat yang berada di luar cakupan kita saat ini. (Selain itu, di sini kami tertarik pada situasi di

mana pihak-pihak yang berkomunikasi tidak memiliki informasi yang dibagikan sebelumnya, dalam hal ini tidak ada yang dapat dilakukan untuk mencegah musuh menyamar sebagai salah satu pihak. Kami akan kembali ke poin ini nanti.)

Pengaturan dan definisi keamanan. Kami mempertimbangkan pengaturan dengan dua pihak—biasanya disebut Alice dan Bob—yang menjalankan protokol probabilistik $\prod$ untuk menghasilkan kunci rahasia bersama; $\prod$ dapat dilihat sebagai kumpulan instruksi untuk Alice dan Bob dalam protokol. Alice dan Bob memulai dengan memegang parameter keamanan 1^n; mereka kemudian menjalankan $\prod$ menggunakan bit acak (independen). Di akhir protokol, Alice dan Bob mengeluarkan kunci $k_A, k_B \in \{0,1\}^n$, masing-masing. Persyaratan kebenaran dasar adalah $k_A = k_B$. Karena kita hanya akan menangani protokol yang memenuhi persyaratan ini, kita hanya akan berbicara tentang kunci $k = k_A = k_B$ yang dihasilkan oleh eksekusi $\prod$ yang jujur.

Kita sekarang beralih ke definisi keamanan. Secara intuitif, protokol pertukaran kunci aman jika keluaran kunci oleh Alice dan Bob benar-benar tidak dapat ditebak oleh musuh yang menguping. Hal ini secara formal didefinisikan dengan mensyaratkan bahwa musuh yang telah menguping eksekusi protokol tidak boleh membedakan kunci k yang dihasilkan oleh eksekusi tersebut (dan sekarang digunakan bersama oleh Alice dan Bob) dari kunci seragam yang panjangnya n. Hal ini jauh lebih kuat dari sekedar mensyaratkan bahwa musuh tidak dapat menghitung k secara tepat, dan gagasan yang lebih kuat ini diperlukan jika pihak-pihak tersebut selanjutnya akan menggunakan k untuk beberapa aplikasi kriptografi (misalnya, sebagai kunci untuk skema enkripsi kunci pribadi). Dengan meresmikan hal di atas, misalkan $\prod$ adalah protokol pertukaran kunci, $\mathcal{A}$ adalah musuh, dan n adalah parameter keamanan. Kami memiliki percobaan berikut:

Eksperimen pertukaran kunci $KE^{eav}_{\mathcal{A},\prod}(n)$:

1. Dua pihak yang memegang 1^n menjalankan protokol $\prod$. Hal ini menghasilkan transkrip trans yang berisi semua pesan yang dikirim oleh para pihak, dan output kunci k oleh masing-masing pihak.
2. Bit seragam $b \in \{0,1\}$ dipilih. Jika $b = 0$ himpunan $\hat{k} := k$, dan jika $b = 1$ maka pilihlah $\hat{k} \in \{0,1\}^n$ secara seragam dan acak.
3. $\mathcal{A}$ diberi trans dan $\hat{k}$, dan menghasilkan sedikit b'.
4. Keluaran percobaan dinyatakan 1 jika $b' = b$, dan 0 sebaliknya. (Jika $KE^{eav}_{\mathcal{A},\prod}(n) = 1$, kita katakan $\mathcal{A}$ berhasil.)

$\mathcal{A}$ diberikan trans untuk menangkap fakta yang menguping seluruh pelaksanaan protokol dan dengan demikian melihat semua pesan yang dipertukarkan oleh para pihak. Di dunia nyata, $\mathcal{A}$ tidak akan diberikan kunci apa pun; dalam percobaan, musuh diberikan kˆ hanya sebagai cara untuk mendefinisikan apa artinya $\mathcal{A}$ "merusak" keamanan $\prod$.

Artinya, musuh berhasil "memecahkan" $\prod$ jika ia dapat menentukan dengan benar apakah kunci $\hat{k}$ adalah kunci sebenarnya yang sesuai dengan pelaksanaan protokol tertentu, atau apakah $\hat{k}$ adalah kunci seragam yang tidak bergantung pada transkrip.

Seperti yang diharapkan, kita mengatakan $\prod$ aman jika musuh berhasil dengan probabilitas paling besar dari 1/2. Itu adalah:

DEFINISI 10.1 Protokol pertukaran kunci $\prod$ aman di hadapan penyadap jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga

$$\Pr\left[\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n).$$

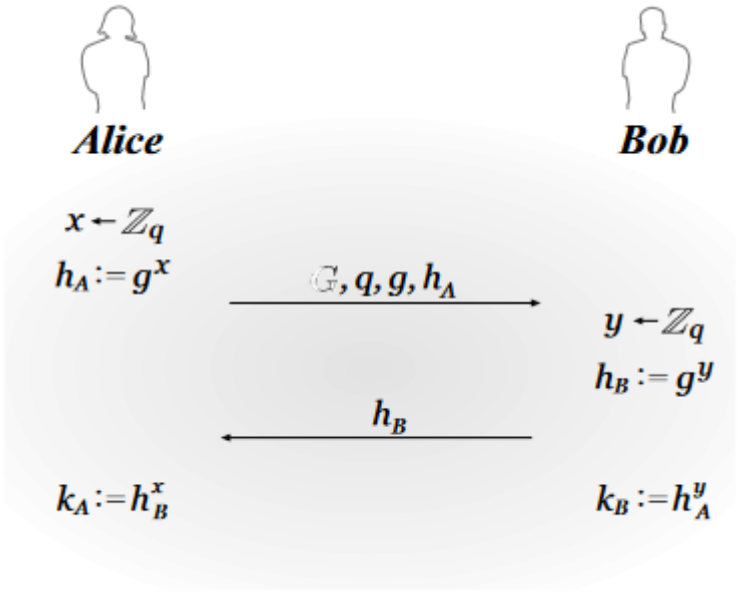
Tujuan dari protokol pertukaran kunci hampir selalu untuk menghasilkan kunci bersama k yang akan digunakan oleh para pihak untuk beberapa tujuan kriptografi lebih lanjut, misalnya untuk mengenkripsi dan mengautentikasi komunikasi selanjutnya menggunakan, katakanlah, skema enkripsi yang diautentikasi. Secara intuitif, menggunakan kunci bersama yang dihasilkan oleh protokol pertukaran kunci yang aman seharusnya "sama baiknya" dengan menggunakan kunci yang dibagikan melalui saluran pribadi. Hal ini dapat dibuktikan secara formal.

Protokol pertukaran kunci Diffie–Hellman. Kami sekarang menggambarkan protokol pertukaran kunci yang muncul dalam makalah asli oleh Diffie dan Hellman (walaupun protokol tersebut kurang formal dibandingkan yang akan kita bahas di sini). Misalkan $\mathcal{G}$ algoritma waktu polinomial probabilistik yang, pada input 1^n, menghasilkan deskripsi grup siklik $\mathbb{G}$, ordenya q (dengan $\| q \| = n$), dan generator $g \in \mathbb{G}$. (Lihat Bagian 8.3.2.) Protokol pertukaran kunci Diffie–Hellman dijelaskan secara formal sebagai Konstruksi 10.2 dan diilustrasikan pada Gambar 10.2.

CONSTRUCTION 10.2

- **Common input:** The security parameter 1^n
- **The protocol:**
  1. Alice runs $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$.
  2. Alice chooses a uniform $x \in \mathbb{Z}_q$, and computes $h_A := g^x$.
  3. Alice sends $(\mathbb{G}, q, g, h_A)$ to Bob.
  4. Bob receives $(\mathbb{G}, q, g, h_A)$. He chooses a uniform $y \in \mathbb{Z}_q$, and computes $h_B := g^y$. Bob sends h_B to Alice and outputs the key $k_B := h_A^y$.
  5. Alice receives h_B and outputs the key $k_A := h_B^x$.

Protokol pertukaran kunci Diffie–Hellman.



GAMBAR 10.2: Protokol pertukaran kunci Diffie–Hellman.

Dalam uraian kami, kami berasumsi bahwa Alice menghasilkan $(\mathbb{G}, q, g)$ dan mengirimkan parameter ini ke Bob sebagai bagian dari pesan pertamanya. Dalam praktiknya, parameter-parameter ini distandarisasi dan ditetapkan serta diketahui oleh kedua belah pihak sebelum protokol dimulai. Dalam hal ini Alice hanya perlu mengirim h_A, dan Bob tidak perlu menunggu untuk menerima pesan Alice sebelum menghitung dan mengirim h_B. Tidak sulit untuk melihat bahwa protokolnya benar: Bob menghitung kuncinya

$$k_B = h_A^y = (g^x)^y = g^{xy}$$

dan Alice menghitung kuncinya

$$k_A = h_B^x = (g^y)^x = g^{xy},$$

jadi $k_A = k_B$. (Pembaca yang jeli akan melihat bahwa kunci bersama adalah elemen grup, bukan bit-string. Kita akan kembali ke poin ini nanti.)

Diffie dan Hellman tidak membuktikan keamanan protokol mereka; memang, gagasan yang tepat (baik kerangka definisi maupun gagasan untuk merumuskan asumsi yang tepat) belum ada. Mari kita lihat asumsi seperti apa yang diperlukan agar protokol aman. Pengamatan pertama, yang dilakukan oleh Diffie dan Hellman, adalah bahwa persyaratan minimal untuk keamanan di sini adalah bahwa masalah logaritma diskrit relatif sulit terhadap $\mathcal{G}$. Jika tidak, maka musuh diberikan transkripnya (yang, khususnya, mencakup h_A) dapat menghitung nilai rahasia salah satu pihak (yaitu x) dan kemudian dengan mudah menghitung kunci bersama menggunakan nilai tersebut. Jadi, masalah kekerasan logaritma diskrit diperlukan agar protokol aman. Namun hal ini tidak cukup, karena ada kemungkinan bahwa

ada cara lain untuk menghitung kunci $k_A = k_B$ tanpa secara eksplisit menghitung x atau y. Asumsi komputasi Diffie–Hellman—yang hanya akan menjamin bahwa kunci g^{xy} sulit dihitung secara keseluruhan dari transkrip—juga tidak cukup. Apa yang disyaratkan oleh Definisi 10.1 adalah bahwa kunci bersama g^{xy} harus tidak dapat dibedakan dari kunci seragam untuk setiap musuh yang diberikan g, g^x, dan g^y. Hal ini persis dengan asumsi keputusan Diffie–Hellman yang diperkenalkan di Bagian 8.3.2. Seperti yang akan kita lihat, bukti keamanan protokol segera mengikuti asumsi keputusan Diffie–Hellman. Hal ini seharusnya tidak mengejutkan, karena asumsi Diffie–Hellman diperkenalkan—jauh setelah Diffie dan Hellman menerbitkan makalah mereka—sebagai cara untuk mengabstraksi properti yang mendasari (dugaan) keamanan protokol Diffie–Hellman. Mengingat hal ini, wajar jika kita mempertanyakan apakah ada manfaat yang diperoleh dengan mendefinisikan dan membuktikan keamanan di sini. Pada bagian buku ini, semoga Anda yakin bahwa jawabannya adalah ya. Mendefinisikan pertukaran kunci aman secara tepat akan memaksa kita memikirkan properti keamanan apa yang kita perlukan; menentukan asumsi yang tepat (yaitu, asumsi keputusan Diffie–Hellman) berarti kita dapat mempelajari asumsi ini secara independen dari aplikasi tertentu dan—setelah kita yakin akan kemungkinannya—membangun protokol lain berdasarkan asumsi tersebut; terakhir, pembuktian keamanan menunjukkan bahwa asumsi tersebut memang cukup bagi protokol untuk memenuhi gagasan keamanan yang kita inginkan.

Dalam pembuktian keamanan kami, kami menggunakan versi Definisi 10.1 yang dimodifikasi yang mengharuskan kunci bersama tidak dapat dibedakan dari elemen seragam $\mathbb{G}$, bukan dari string n-bit yang seragam. Perbedaan ini perlu diatasi sebelum protokol dapat digunakan dalam praktik—bagaimanapun juga, elemen grup biasanya tidak berguna sebagai kunci kriptografi, dan representasi elemen grup yang seragam, secara umum, tidak akan berupa bit-string yang seragam— dan kami membahas secara singkat satu cara standar untuk melakukannya berikut buktinya. Untuk saat ini, kita misalkan $\widehat{KE}^{eav}_{\mathcal{A},\prod}(n)$ menunjukkan eksperimen yang dimodifikasi di mana jika $b = 1$ maka lawan diberikan $\hat{k}$ yang dipilih secara seragam dari $\mathbb{G}$, bukan string n-bit yang seragam.

TEOREMA 10.3 Jika masalah pengambilan keputusan Diffie–Hellman relatif sulit terhadap G, maka protokol pertukaran kunci Diffie–Hellman $\prod$ aman dengan adanya penyadap (sehubungan dengan eksperimen yang dimodifikasî $\widehat{KE}^{eav}_{\mathcal{A},\prod}(n)$. BUKTI Biarkan $\mathcal{A}$ menjadi musuh ppt. Karena $Pr[b = 0] = Pr[b = 1] = 1/2$, kita punya

$$\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] = \frac{1}{2}\cdot\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0\right] + \frac{1}{2}\cdot\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1\right].$$

Dalam eksperimeñ $\widehat{KE}^{eav}_{\mathcal{A},\prod}(n)$ musuh $\mathcal{A}$ menerima $(\mathbb{G}, q, g, h_A, h_B, \hat{k})$, di mana

$(\mathbb{G}, q, g, h_A, h_B)$ mewakili transkrip eksekusi protokol, dan $\hat{k}$ adalah baik kunci sebenarnya yang dihitung oleh para pihak (jika $b = 0$) atau elemen grup seragam (jika $b = 1$). Membedakan kedua kasus ini sama persis dengan menyelesaikan masalah pengambilan keputusan Diffie–Hellman. Itu adalah

$$\begin{aligned}
&\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \\
&= \frac{1}{2} \cdot \Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0\right] + \frac{1}{2} \cdot \Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1\right] \\
&= \frac{1}{2} \cdot \Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] \\
&= \frac{1}{2} \cdot \left(1 - \Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^{xy}) = 1]\right) + \frac{1}{2} \cdot \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] \\
&= \frac{1}{2} + \frac{1}{2} \cdot \left(\Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\right) \\
&\leq \frac{1}{2} + \frac{1}{2} \cdot \left|\Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\right|,
\end{aligned}$$

dimana semua probabilitas diambil alih $(\mathbb{G}, q, g)$ keluaran sebesar $\mathcal{G}(1^n)$, dan pilihan seragam $x, y, z \in \mathbb{Z}_q$. (Perhatikan bahwa karena g adalah generator, g^z adalah elemen seragam dari $\mathbb{G}$ ketika z terdistribusi seragam dalam $\mathbb{Z}_q$.) Jika asumsi keputusan Diffie– Hellman relatif sulit terhadap $\mathcal{G}$, itu berarti terdapat fungsi yang dapat diabaikan yang mana

$$\left|\Pr[\mathcal{A}(\mathbb{G}, g, q, g^x, g^y, g^z) = 1] - \Pr[\mathcal{A}(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\right| \leq \mathsf{negl}(n).$$

Kami menyimpulkan bahwa

$$\Pr\left[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \frac{1}{2} \cdot \mathsf{negl}(n),$$

melengkapi buktinya.

Elemen grup seragam vs. bit-string seragam. Teorema sebelumnya menunjukkan bahwa keluaran kunci oleh Alice dan Bob dalam protokol Diffie–Hellman tidak dapat dibedakan (untuk penyadap waktu polinomial) dari elemen grup yang seragam. Untuk menggunakan kunci untuk aplikasi kriptografi selanjutnya—dan juga untuk memenuhi Definisi 10.1—output kunci oleh para pihak harus tidak dapat dibedakan dari bit-string seragam dengan panjang yang sesuai. Protokol Diffie–Hellman dapat dimodifikasi untuk mencapai hal ini dengan meminta para pihak menerapkan fungsi derivasi kunci yang sesuai (lih. Bagian 5.6.4) ke elemen grup bersama g^{xy} yang mereka hitung masing-masing.

Musuh aktif. Sejauh ini kami hanya mempertimbangkan musuh yang menguping. Meskipun serangan penyadapan merupakan serangan yang paling umum (karena paling mudah dilakukan), serangan ini bukanlah satu-satunya serangan yang mungkin dilakukan. Serangan aktif, di mana musuh mengirimkan pesannya sendiri ke salah satu atau kedua pihak, juga menjadi perhatian, dan protokol apa pun yang digunakan dalam praktiknya juga harus tahan terhadap serangan tersebut. Ketika mempertimbangkan serangan aktif, akan berguna untuk membedakan, secara informal, antara serangan peniruan identitas di mana pihak yang berlawanan menyamar sebagai salah satu pihak saat berinteraksi dengan pihak lain, dan serangan man-in-the-middle di mana kedua pihak yang jujur menjalankan protokol dan pihak yang dituding menjalankan protokol. mencegat dan memodifikasi pesan yang dikirim dari satu pihak ke pihak lain. Kami tidak akan secara formal mendefinisikan keamanan terhadap kedua jenis serangan tersebut, karena definisi tersebut agak rumit dan tidak dapat dicapai tanpa pihak-pihak yang berbagi informasi terlebih dahulu. Namun demikian, perlu diperhatikan bahwa protokol Diffie–Hellman sepenuhnya tidak aman terhadap serangan man-in-the-middle. Faktanya, musuh man-in-the-middle dapat bertindak sedemikian rupa sehingga Alice dan Bob mengakhiri protokol dengan kunci k_A dan k_B yang berbeda yang keduanya diketahui oleh musuh, namun baik Alice maupun Bob tidak dapat mendeteksi bahwa ada serangan dilakukan. Kami membiarkan rincian serangan ini sebagai latihan.

Pertukaran kunci Diffie–Hellman dalam praktiknya. Protokol Diffie–Hellman dalam bentuk dasarnya biasanya tidak digunakan dalam praktik karena ketidakamanannya terhadap serangan man-in-the-middle, seperti yang dibahas di atas. Hal ini sama sekali tidak mengurangi pentingnya hal ini. Protokol Diffie–Hellman menjadi demonstrasi pertama bahwa teknik asimetris (dan masalah teori bilangan) dapat digunakan untuk meringankan masalah distribusi kunci dalam kriptografi. Selain itu, protokol Diffie–Hellman merupakan inti dari protokol pertukaran kunci terstandarisasi yang tahan terhadap serangan man-in-the-middle dan digunakan secara luas saat ini. Salah satu contoh penting adalah TLS; lihat Bagian 12.8.

10.4 REVOLUSI KUNCI PUBLIK

Selain pertukaran kunci, Diffie dan Hellman juga memperkenalkan gagasan kriptografi kunci publik (atau asimetris) dalam karya terobosan mereka. Dalam pengaturan kunci publik (berbeda dengan pengaturan kunci privat yang telah kita pelajari di Bab 1–7), pihak yang ingin berkomunikasi secara aman menghasilkan sepasang kunci: kunci publik yang disebarluaskan, dan kunci privat. yang dirahasiakannya. (Fakta bahwa sekarang ada dua kunci yang berbeda membuat skema menjadi asimetris.) Setelah membuat kunci-kunci ini, suatu pihak dapat menggunakannya untuk memastikan kerahasiaan pesan yang diterimanya menggunakan skema enkripsi kunci publik, atau integritas untuk pesan yang dikirim menggunakan skema tersebut. skema tanda tangan digital. (Lihat Gambar 10.3.) Kami memberikan gambaran singkat tentang primitif ini di sini, dan membahasnya secara rinci di Bab 11 dan 12.

Dalam skema enkripsi kunci publik, kunci publik yang dihasilkan oleh beberapa pihak berfungsi sebagai kunci enkripsi; siapa pun yang mengetahui kunci publik tersebut dapat menggunakannya untuk mengenkripsi pesan dan menghasilkan teks sandi yang sesuai. Kunci

pribadi berfungsi sebagai kunci dekripsi dan digunakan oleh pihak yang mengetahuinya untuk memulihkan pesan asli dari teks sandi apa pun yang dihasilkan menggunakan kunci publik yang cocok. Lebih jauh lagi—dan sungguh menakjubkan bahwa hal seperti ini ada!—kerahasiaan pesan terenkripsi tetap terjaga bahkan terhadap musuh yang mengetahui kunci enkripsi (tetapi bukan kunci dekripsi). Dengan kata lain, kunci enkripsi (publik) tidak berguna bagi penyerang yang mencoba mendekripsi teks sandi yang dienkripsi menggunakan kunci tersebut. Maka, untuk mengaktifkan komunikasi rahasia, penerima cukup mengirimkan kunci publiknya ke calon pengirim (tanpa harus khawatir tentang musuh yang menguping yang mengamati kunci publiknya), atau mempublikasikan kunci publiknya di halaman webnya atau di database pusat. Skema enkripsi kunci publik memungkinkan komunikasi pribadi tanpa bergantung pada saluran pribadi untuk distribusi kunci.

	Private-Key Setting	Public-Key Setting
Secrecy	Private-key encryption	Public-key encryption
Integrity	Message authentication codes	Digital signature schemes

GAMBAR 10.3: Kriptografi primitif dalam pengaturan kunci pribadi dan kunci publik.

Skema tanda tangan digital adalah analogi kunci publik dari kode otentikasi pesan (MAC). Di sini, kunci pribadi berfungsi sebagai "kunci autentikasi" (biasanya disebut kunci penandatanganan) yang memungkinkan pihak yang mengetahui kunci ini menghasilkan "tag autentikasi" (yaitu, tanda tangan) untuk pesan yang dikirimkannya. Kunci publik bertindak sebagai kunci verifikasi, memungkinkan siapa saja yang mengetahuinya untuk memverifikasi tanda tangan yang dikeluarkan oleh pengirim. Seperti halnya MAC, skema tanda tangan digital dapat digunakan untuk mencegah gangguan pesan yang tidak terdeteksi. Fakta bahwa verifikasi dapat dilakukan oleh siapa saja yang mengetahui kunci publik pengirimnya, ternyata mempunyai konsekuensi yang luas. Secara khusus, hal ini memungkinkan untuk mengambil dokumen yang ditandatangani oleh Alice dan menyerahkannya kepada pihak ketiga (misalnya hakim) sebagai bukti bahwa Alice memang menandatangani dokumen tersebut. Properti ini disebut non-penyangkalan dan memiliki penerapan luas dalam perdagangan elektronik. Misalnya, dimungkinkan untuk menandatangani kontrak secara digital, mengirimkan pesanan pembelian elektronik yang ditandatangani atau janji pembayaran, dan sebagainya. Tanda tangan digital juga digunakan untuk distribusi kunci publik yang aman sebagai bagian dari infrastruktur kunci publik, sebagaimana dibahas lebih rinci di Bagian 12.7. Dalam makalah mereka, Diffie dan Hellman mengemukakan gagasan kriptografi kunci publik tetapi tidak memberikan konstruksi kandidat apa pun. Setahun kemudian, Ron Rivest, Adi Shamir, dan Len Adleman mengusulkan masalah RSA dan mempresentasikan masalah publik pertama. -enkripsi kunci dan skema tanda tangan digital berdasarkan pada beratnya masalah ini. Varian skema mereka sekarang termasuk primitif kriptografi yang paling banyak digunakan saat ini. Pada tahun 1985, Taher El Gamal memperkenalkan skema enkripsi yang pada dasarnya merupakan perubahan kecil pada protokol pertukaran kunci Diffie–Hellman, dan sekarang

juga digunakan secara luas. Jadi, meskipun Diffie dan Hellman tidak berhasil membangun skema enkripsi kunci publik (non-interaktif), mereka hampir berhasil.

Kami menutup dengan merangkum bagaimana kriptografi kunci publik mengatasi keterbatasan pengaturan kunci privat yang dibahas di Bagian 10.1:

1. Enkripsi kunci publik memungkinkan distribusi kunci dilakukan melalui saluran publik (tetapi diautentikasi). Hal ini dapat menyederhanakan distribusi dan pemutakhiran materi utama.
2. Kriptografi kunci publik mengurangi kebutuhan pengguna untuk menyimpan banyak kunci rahasia. Pertimbangkan kembali situasi di sebuah perusahaan besar di mana setiap pasang karyawan memerlukan kemampuan untuk berkomunikasi dengan aman. Dengan menggunakan kriptografi kunci publik, setiap karyawan cukup menyimpan satu kunci pribadi (milik mereka) dan kunci publik semua karyawan lainnya. Yang penting, kunci terakhir ini tidak perlu disimpan secara rahasia; mereka bahkan dapat disimpan di beberapa repositori pusat (publik).
3. Terakhir, kriptografi kunci publik (lebih) cocok untuk lingkungan terbuka dimana pihak-pihak yang belum pernah berinteraksi sebelumnya menginginkan kemampuan untuk berkomunikasi dengan aman. Sebagai salah satu contoh umum, pedagang Internet dapat memposting kunci publik mereka secara online; pengguna yang melakukan pembelian kemudian dapat memperoleh kunci publik pedagang, sesuai kebutuhan, ketika mereka perlu mengenkripsi informasi kartu kredit mereka.

Penemuan enkripsi kunci publik merupakan sebuah revolusi dalam kriptografi. Bukan suatu kebetulan bahwa hingga akhir tahun 1970-an dan awal tahun 1980-an, enkripsi dan kriptografi secara umum berada dalam domain organisasi intelijen dan militer, dan hanya dengan munculnya teknik kunci publik barulah penggunaan kriptografi menyebar ke masyarakat luas.

BAB 11
ENKRIPSI KUNCI PUBLIK

11.1 ENKRIPSI KUNCI PUBLIK

Pengenalan enkripsi kunci publik menandai sebuah revolusi dalam kriptografi. Sampai saat itu, para kriptografer hanya mengandalkan kunci rahasia bersama untuk mencapai komunikasi pribadi. Sebaliknya, teknik kunci publik memungkinkan para pihak untuk berkomunikasi secara pribadi tanpa harus menyetujui informasi rahasia terlebih dahulu. Seperti yang telah kami catat, sungguh menakjubkan dan berlawanan dengan intuisi bahwa hal ini mungkin terjadi: artinya dua orang yang berada di sisi berlawanan dari sebuah ruangan yang hanya dapat berkomunikasi dengan berteriak satu sama lain, dan tidak memiliki rahasia awal, dapat berbicara sedemikian rupa. bahwa tidak ada orang lain di ruangan itu yang mengetahui apa pun tentang apa yang mereka katakan!

Dalam pengaturan enkripsi kunci pribadi, dua pihak menyepakati kunci rahasia yang dapat digunakan, oleh salah satu pihak, untuk enkripsi dan dekripsi. Enkripsi kunci publik bersifat asimetris dalam kedua hal ini. Satu pihak (penerima) menghasilkan sepasang kunci (pk, sk), masing-masing disebut kunci publik dan kunci privat. Kunci publik digunakan oleh pengirim untuk mengenkripsi pesan; penerima menggunakan kunci pribadi untuk mendekripsi ciphertext yang dihasilkan.

Karena tujuannya adalah untuk menghindari perlunya dua pihak bertemu terlebih dahulu untuk menyepakati suatu informasi, bagaimana cara pengirim mengetahui pk? Pada tingkat abstrak, ada dua cara hal ini bisa terjadi. Hubungi penerima Alice dan pengirim Bob. Pada pendekatan pertama, ketika Alice mengetahui bahwa Bob ingin berkomunikasi dengannya, dia dapat membuat (pk, sk) (dengan asumsi dia belum melakukannya) dan kemudian mengirimkan pk ke Bob dengan jelas; Bob kemudian dapat menggunakan pk untuk mengenkripsi pesannya. Kami menekankan bahwa saluran antara Alice dan Bob mungkin bersifat publik, tetapi diasumsikan telah diautentikasi, yang berarti bahwa musuh tidak dapat mengubah kunci publik yang dikirimkan oleh Alice kepada Bob (dan, khususnya, tidak dapat menggantinya dengan kuncinya sendiri). Lihat Bagian 12.7 untuk diskusi tentang bagaimana kunci publik dapat didistribusikan melalui saluran yang tidak diautentikasi.

Pendekatan alternatifnya adalah Alice membuat kuncinya (pk, sk) terlebih dahulu, secara independen dari pengirim tertentu. (Faktanya, pada saat pembuatan kunci, Alice bahkan tidak perlu menyadari bahwa Bob ingin berbicara dengannya, atau bahkan bahwa Bob ada.) Alice dapat menyebarkan pk kunci publiknya secara luas dengan, misalnya, mempublikasikannya di halaman webnya, meletakkan itu di kartu namanya, atau menempatkannya di direktori publik. Sekarang, siapapun yang ingin berkomunikasi secara pribadi dengan Alice dapat mencari kunci publiknya dan melanjutkan seperti di atas. Perhatikan bahwa beberapa pengirim dapat berkomunikasi beberapa kali dengan Alice menggunakan pk kunci publik yang sama untuk mengenkripsi semua komunikasi mereka.

Perhatikan bahwa pk pada dasarnya bersifat publik—dan dengan demikian dapat dipelajari dengan mudah oleh penyerang—dalam salah satu skenario di atas. Dalam kasus pertama, musuh yang menguping komunikasi antara Alice dan Bob memperoleh pk secara langsung; dalam kasus kedua, musuh juga bisa mencari kunci publik Alice sendiri. Kita melihat bahwa keamanan enkripsi kunci publik tidak bisa bergantung pada kerahasiaan pk, namun harus bergantung pada kerahasiaan sk. Oleh karena itu, sangat penting bagi Alice untuk tidak mengungkapkan kunci pribadinya kepada siapa pun, termasuk Bob pengirimnya.

Perbandingan dengan Enkripsi Kunci Pribadi

Mungkin perbedaan yang paling jelas antara enkripsi kunci privat dan kunci publik adalah enkripsi kunci privat mengasumsikan kerahasiaan lengkap seluruh kunci kriptografi, sedangkan enkripsi kunci publik memerlukan kerahasiaan hanya untuk kunci sk privat saja. Meskipun hal ini tampak seperti perbedaan kecil, dampaknya sangat besar: dalam pengaturan kunci privat, pihak-pihak yang berkomunikasi harus dapat berbagi kunci rahasia tanpa membiarkan pihak ketiga mempelajarinya, sedangkan dalam pengaturan kunci publik kunci publik dapat dikirim dari satu pihak ke pihak lain melalui saluran publik tanpa mengorbankan keamanan. Bagi pihak-pihak yang berteriak di seberang ruangan atau, lebih realistisnya, berkomunikasi melalui jaringan publik seperti saluran telepon atau Internet, enkripsi kunci publik adalah satu-satunya pilihan.

Perbedaan penting lainnya adalah skema enkripsi kunci privat menggunakan kunci yang sama untuk enkripsi dan dekripsi, sedangkan skema enkripsi kunci publik menggunakan kunci yang berbeda untuk setiap operasi. Artinya, enkripsi kunci publik pada dasarnya bersifat asimetris. Asimetri dalam pengaturan kunci publik ini berarti bahwa peran pengirim dan penerima tidak dapat dipertukarkan seperti halnya dalam pengaturan kunci pribadi: satu pasangan kunci memungkinkan komunikasi dalam satu arah saja. (Komunikasi dua arah dapat dicapai dengan beberapa cara; intinya adalah bahwa satu pemanggilan skema enkripsi kunci publik memaksa pembedaan antara satu pengguna yang bertindak sebagai penerima dan pengguna lain yang bertindak sebagai pengirim.) Selain itu, a satu contoh skema enkripsi kunci publik memungkinkan banyak pengirim untuk berkomunikasi secara pribadi dengan satu penerima, berbeda dengan kasus kunci pribadi di mana kunci rahasia yang digunakan bersama antara dua pihak memungkinkan komunikasi pribadi hanya antara kedua pihak tersebut.

Meringkas dan menguraikan pembahasan sebelumnya, kita melihat bahwa enkripsi kunci publik memiliki keunggulan dibandingkan enkripsi kunci privat sebagai berikut:

> *Enkripsi kunci publik (sampai batas tertentu) mengatasi masalah distribusi kunci, karena pihak yang berkomunikasi tidak perlu berbagi kunci secara diam-diam sebelum komunikasi mereka. Dua pihak dapat berkomunikasi secara diam-diam meskipun semua komunikasi di antara mereka dipantau.*

Ketika satu penerima berkomunikasi dengan N pengirim (misalnya, pedagang online yang memproses pesanan kartu kredit dari beberapa pembeli), akan lebih mudah bagi penerima untuk menyimpan satu kunci pribadi sk daripada berbagi, menyimpan, dan mengelola N kunci rahasia yang berbeda (yaitu satu untuk setiap pengirim). Faktanya, ketika menggunakan enkripsi kunci publik, jumlah dan identitas pengirim potensial tidak perlu diketahui pada saat

pembuatan kunci. Hal ini memungkinkan fleksibilitas yang sangat besar dalam "sistem terbuka."

Fakta bahwa skema enkripsi kunci publik memungkinkan siapa pun bertindak sebagai pengirim dapat menjadi kelemahan ketika penerima hanya ingin menerima pesan dari satu individu tertentu. Dalam hal ini, skema enkripsi yang diautentikasi (kunci pribadi) akan menjadi pilihan yang lebih baik daripada enkripsi kunci publik.

Kelemahan utama enkripsi kunci publik adalah kecepatannya sekitar 2 hingga 3 kali lipat lebih lambat dibandingkan enkripsi kunci pribadi. Penerapan enkripsi kunci publik pada perangkat dengan sumber daya terbatas seperti kartu pintar atau frekuensi radio dapat menjadi sebuah tantangan. tag identifikasi (RFID). Bahkan ketika komputer desktop melakukan operasi kriptografi, melakukan ribuan operasi tersebut per detik (seperti dalam kasus pedagang online yang memproses transaksi kartu kredit) mungkin menjadi penghalang. Oleh karena itu, ketika enkripsi kunci pribadi merupakan suatu pilihan (yaitu, jika dua pihak dapat berbagi kunci dengan aman terlebih dahulu), maka enkripsi tersebut biasanya harus digunakan.

Faktanya, seperti yang akan kita lihat di Bagian 11.3, enkripsi kunci privat digunakan dalam pengaturan kunci publik untuk meningkatkan efisiensi enkripsi (kunci publik) pada pesan panjang. Oleh karena itu, pemahaman menyeluruh tentang enkripsi kunci pribadi sangat penting untuk memahami bagaimana enkripsi kunci publik diterapkan dalam praktiknya.

Distribusi Kunci Publik yang Aman

Dalam keseluruhan diskusi kita sejauh ini, kita secara implisit berasumsi bahwa musuh bersifat pasif; artinya, musuh hanya menguping komunikasi antara pengirim dan penerima tetapi tidak secara aktif mengganggu komunikasi tersebut. Jika musuh memiliki kemampuan untuk merusak semua komunikasi antara pihak-pihak yang jujur, dan pihak-pihak yang jujur ini tidak berbagi kunci sebelumnya, maka privasi tidak dapat dicapai. Misalnya, jika penerima Alice mengirimkan kunci publiknya pk kepada Bob tetapi musuh menggantinya dengan kunci pk' miliknya sendiri (yang mana ia mengetahui kunci privat yang cocok sk'), maka meskipun Bob mengenkripsi pesannya menggunakan pk' musuh akan dengan mudah dapat memulihkan pesan tersebut (menggunakan sk'). Serangan serupa berhasil jika musuh mampu mengubah nilai kunci publik Alice yang disimpan di beberapa direktori publik, atau jika musuh dapat merusak kunci publik saat kunci tersebut dikirimkan dari direktori publik ke Bob. Jika Alice dan Bob tidak membagikan informasi apa pun sebelumnya, dan tidak mau bergantung pada pihak ketiga yang saling percaya, Alice atau Bob tidak dapat melakukan apa pun untuk mencegah serangan aktif semacam ini, atau bahkan mengatakan bahwa serangan tersebut adalah serangan aktif. sedang berlangsung.

Yang penting, perlakuan kami terhadap enkripsi kunci publik dalam bab ini mengasumsikan bahwa pengirim dapat memperoleh salinan kunci publik penerima yang sah. (Hal ini tersirat dalam definisi keamanan yang kami berikan.) Artinya, kami mengasumsikan distribusi kunci aman. Asumsi ini dibuat bukan karena jenis serangan aktif yang dibahas di atas tidak menimbulkan kekhawatiran—bahkan, serangan tersebut merupakan ancaman serius

yang harus ditangani dalam sistem dunia nyata yang menggunakan enkripsi kunci publik. Sebaliknya, asumsi ini dibuat karena terdapat mekanisme lain untuk mencegah serangan aktif (lihat, misalnya, Bagian 12.7), dan oleh karena itu akan lebih mudah (dan berguna) untuk memisahkan studi tentang enkripsi kunci publik yang aman dari studi tentang enkripsi kunci publik yang aman. distribusi kunci publik yang aman.

11.2 DEFINISI ENKRIPSI KUNCI PUBLIK

Kita mulai dengan mendefinisikan sintaks enkripsi kunci publik. Definisi ini sangat mirip dengan Definisi 3.7, dengan pengecualian bahwa alih-alih bekerja hanya dengan satu kunci, kini kami memiliki kunci enkripsi dan dekripsi yang berbeda.

DEFINISI 11.1 Skema enkripsi kunci publik adalah tiga algoritma waktu polinomial probabilistik (Gen, Enc, Des) sehingga:

1. Algoritme pembangkitan kunci Gen mengambil parameter keamanan 1n sebagai masukan dan mengeluarkan sepasang kunci (pk, sk). Kami menyebut yang pertama sebagai kunci publik dan yang kedua sebagai kunci privat. Kita asumsikan agar lebih mudah bahwa pk dan sk masing-masing memiliki panjang paling sedikit n, dan n dapat ditentukan dari pk, sk.
2. Algoritma enkripsi Enc mengambil masukan kunci publik pk dan pesan m dari beberapa ruang pesan (yang mungkin bergantung pada pk). Ini menghasilkan ciphertext c, dan kami menulisnya sebagai $c \leftarrow \text{Enc}_{pk}(m)$. (Ke depannya, Enc harus bersifat probabilistik untuk mencapai keamanan yang berarti.)
3. Algoritme dekripsi deterministik Dec mengambil masukan kunci pribadi sk dan teks sandi c, dan mengeluarkan pesan m atau simbol khusus $\perp$ yang menunjukkan kegagalan. Kami menulis ini sebagai $m := \text{Dec}_{sk}(c)$.

Diperlukan bahwa, kecuali mungkin dengan probabilitas yang dapat diabaikan atas keluaran (pk, sk) oleh Gen(1^n), kita mempunyai $\text{Dec}_{sk}\ (\text{Enc}_{pk}(m)) = m$ untuk setiap pesan (legal) m.

Perbedaan penting dari pengaturan kunci privat adalah algoritma pembangkitan kunci Gen sekarang mengeluarkan dua kunci, bukan satu. Kunci publik pk digunakan untuk enkripsi, sedangkan kunci privat sk digunakan untuk dekripsi. Mengulangi diskusi kita sebelumnya, pk diasumsikan didistribusikan secara luas sehingga siapa pun dapat mengenkripsi pesan untuk pihak yang menghasilkan kunci ini, namun sk harus dijaga kerahasiaannya oleh penerima agar keamanan tetap terjaga. Kami mengizinkan probabilitas kesalahan dekripsi yang dapat diabaikan dan, tentu saja, beberapa skema yang kami sajikan akan memiliki probabilitas kesalahan yang dapat diabaikan (misalnya, jika bilangan prima perlu dipilih tetapi dengan probabilitas yang dapat diabaikan, maka diperoleh komposit). Meskipun demikian, secara umum kami akan mengabaikan masalah ini mulai saat ini.

Untuk penggunaan praktis enkripsi kunci publik, kita ingin ruang pesan menjadi $\{0,1\}^n$ atau $\{0,1\}^*$ (dan, khususnya, tidak bergantung pada kunci publik). Meskipun kadang-kadang kita akan menggambarkan skema enkripsi menggunakan beberapa ruang pesan M yang tidak

berisi semua bit-string dengan panjang tetap (dan itu mungkin juga bergantung pada kunci publik), dalam kasus seperti itu kita juga akan menentukan cara mengkodekan bit-string sebagai elemen dari M. Pengkodean ini harus dapat dihitung secara efisien dan dapat dibalik secara efisien, sehingga penerima dapat memulihkan bit-string yang telah dienkripsi.

Keamanan terhadap Serangan Teks Biasa Terpilih

Kami memulai perlakuan kami terhadap keamanan dengan memperkenalkan padanan "alami" dari Definisi 3.8 dalam pengaturan kunci publik. Karena motivasi yang luas untuk definisi ini (dan juga definisi lain yang akan kita lihat) telah diberikan di Bab 3, pembahasan di sini akan relatif singkat dan akan fokus terutama pada perbedaan antara pengaturan kunci privat dan kunci publik.

Diberikan skema enkripsi kunci publik $\prod$ = (Gen, Enc, Des) dan musuh $\mathcal{A}$, perhatikan percobaan berikut:

Eksperimen penyadapan ketidakmungkinan membedakan Pub$K^{eav}_{\mathcal{A},\prod}(n)$:

1. Gen(1^n) dijalankan untuk mendapatkan kunci (pk, sk).
2. Musuh $\mathcal{A}$ diberikan pk, dan mengeluarkan sepasang pesan dengan panjang yang sama m_0, m_1 di ruang pesan.
3. Bit seragam $b \in \{0,1\}$ dipilih, dan kemudian teks tersandi $c \leftarrow$ Enc$_{pk}(mb)$ dihitung dan diberikan kepada $\mathcal{A}$. Kami menyebut c sebagai tantangan ciphertext.
4. $\mathcal{A}$ mengeluarkan sedikit b'. Keluaran percobaan ini adalah 1 jika $b' = b$, dan 0 jika tidak. Jika b' = b kita katakan $\mathcal{A}$ berhasil.

DEFINISI 11.2 Skema enkripsi kunci publik $\prod$ = (Gen, Enc, Des) memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap jika untuk semua musuh $\mathcal{A}$ waktu polinomial probabilistik terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Perbedaan utama antara definisi di atas dan Definisi 3.8 adalah di sini $\mathcal{A}$ diberikan kunci publik pk. Selanjutnya, kami mengizinkan $\mathcal{A}$ untuk memilih pesannya m_0 dan m_1 berdasarkan kunci publik ini. Hal ini penting ketika mendefinisikan keamanan enkripsi kunci publik karena, seperti yang dibahas sebelumnya, kita berasumsi bahwa musuh mengetahui kunci publik penerima.

Modifikasi yang tampaknya "kecil" dalam memberikan pk kunci publik $\mathcal{A}$ yang digunakan untuk mengenkripsi pesan kepada musuh memiliki dampak yang luar biasa: secara efektif memberikan akses $\mathcal{A}$ ke oracle enkripsi secara gratis. (Konsep oracle enkripsi dijelaskan di Bagian 3.4.) Hal ini benar karena musuh, mengingat pk, dapat mengenkripsi pesan apa pun m sendiri hanya dengan menghitung Enc$_{pk}(m)$. (Seperti biasa, $\mathcal{A}$ diasumsikan mengetahui algoritme Enc.) Hasilnya adalah bahwa Definisi 11.2 setara dengan keamanan CPA (yaitu, keamanan terhadap serangan teks biasa yang dipilih), yang didefinisikan dengan cara yang

analog dengan Definisi 3.22, dengan satu-satunya Perbedaannya adalah penyerang diberikan kunci publik dalam eksperimen terkait. Dengan demikian kami memiliki:

PROPOSISI 11.3 Jika skema enkripsi kunci publik mempunyai enkripsi yang tidak dapat dibedakan jika ada penyadap, maka skema tersebut aman terhadap CPA.

Hal ini berbeda dengan pengaturan kunci privat, dimana terdapat skema yang memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap namun tidak aman jika terkena serangan teks biasa (lihat Proposisi 3.20). Perbedaan lebih lanjut dari pengaturan kunci pribadi yang segera menyusul sebagai konsekuensi dari hal di atas akan dibahas selanjutnya.

Ketidakmungkinan enkripsi kunci publik yang sangat rahasia. Enkripsi kunci publik yang sangat rahasia dapat didefinisikan secara analog dengan Definisi 2.3 dengan mengkondisikan seluruh tampilan penyadap (yaitu, termasuk kunci publik). Dengan cara yang sama, hal ini dapat didefinisikan dengan memperluas Definisi 11.2 yang mensyaratkan bahwa untuk semua musuh $\mathcal{A}$ (tidak hanya musuh yang efisien), kita mempunyai:

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}.$$

Berbeda dengan pengaturan kunci pribadi, enkripsi kunci publik yang sangat rahasia tidak mungkin dilakukan, terlepas dari berapa panjang kuncinya atau seberapa kecil ruang pesannya. Faktanya, musuh tak terbatas yang diberikan pk dan teks sandi c yang dihitung melalui $c \leftarrow \mathrm{Enc}_{pk}(m)$ dapat menentukan m dengan probabilitas 1. Ketidakamanan enkripsi kunci publik deterministik. Sebagaimana disebutkan dalam konteks enkripsi kunci pribadi, tidak ada skema enkripsi deterministik yang aman terhadap CPA. Hal yang sama juga berlaku di sini:

TEOREMA 11.4 Tidak ada skema enkripsi kunci publik deterministik yang aman terhadap CPA.

Karena Teorema 11.4 sangat penting, maka Teorema 11.4 perlu didiskusikan lebih lanjut. Teorema ini bukan merupakan "artefak" dari definisi keamanan kami, atau indikasi bahwa definisi kami terlalu kuat. Skema enkripsi kunci publik deterministik rentan terhadap serangan praktis dalam skenario realistis dan tidak boleh digunakan. Alasannya adalah bahwa skema deterministik tidak hanya memungkinkan musuh untuk menentukan kapan pesan yang sama dikirim dua kali (seperti dalam pengaturan kunci pribadi), namun juga memungkinkan musuh untuk memulihkan pesan, dengan probabilitas 1, jika serangkaian kemungkinan pesan yang dienkripsi kecil. Misalnya, seorang profesor mengenkripsi nilai siswanya. Di sini, seorang penyadap mengetahui bahwa nilai setiap siswa haruslah salah satu dari $\{A, B, C, D, F\}$. Jika profesor menggunakan skema enkripsi kunci publik deterministik, penyadap dapat dengan cepat menentukan nilai aktual siswa dengan mengenkripsi semua kemungkinan nilai dan membandingkan hasilnya dengan teks sandi yang diberikan.

Meskipun teorema di atas tampak sederhana, sejak lama banyak sistem dunia nyata dirancang menggunakan enkripsi kunci publik deterministik. Ketika enkripsi kunci publik diperkenalkan, dapat dikatakan bahwa pentingnya enkripsi probabilistik belum sepenuhnya disadari. Karya penting Goldwasser dan Micali, di mana (sesuatu yang setara dengan) Definisi 11.2 diusulkan dan Teorema 11.4 dinyatakan, menandai titik balik dalam bidang kriptografi. Pentingnya mendefinisikan intuisi seseorang dalam definisi formal dan melihat segala sesuatu dengan cara yang benar untuk pertama kalinya—walaupun tampak sederhana jika ditinjau kembali—tidak boleh diremehkan.

Beberapa Enkripsi

Seperti pada Bab 3, penting untuk memahami pengaruh penggunaan kunci yang sama (dalam hal ini, kunci publik yang sama) untuk mengenkripsi banyak pesan. Kita dapat memformulasikan keamanan dalam situasi seperti ini dengan meminta musuh mengeluarkan dua daftar teks biasa, seperti pada Definisi 3.19. Namun, untuk alasan yang dibahas di Bagian 3.4.2, kami memilih untuk menggunakan definisi di mana penyerang diberi akses ke oracle $LR_{pk,b}$"kiri-atau-kanan", yang, jika diinputkan, sepasang pesan yang panjangnya sama m_0, m_1, menghitung ciphertext $c \leftarrow \text{Enc}_{pk}(m_b)$ dan mengembalikan c. Penyerang diperbolehkan untuk menanyakan oracle ini sebanyak yang diinginkannya, dan definisi tersebut memodelkan keamanan ketika beberapa pesan (tidak dikenal) dienkripsi menggunakan kunci publik yang sama.

Secara formal, pertimbangkan percobaan berikut yang didefinisikan untuk skema enkripsi kunci publik $\prod$ = (Gen, Enc, Des) dan musuh $\mathcal{A}$:

Eksperimen LR-Oracle $PubK_{\mathcal{A},\prod}^{LR-cpa}(n)$:

1. Gen(1^n) dijalankan untuk mendapatkan kunci (pk, sk).
2. Bit seragam $b \in \{0,1\}$ dipilih.
3. Musuh $\mathcal{A}$ diberi masukan pk dan akses oracle ke $LR_{pk,b}(\cdot,\cdot)$.
4. Musuh $\mathcal{A}$ mengeluarkan sedikit b'.
5. Keluaran percobaan ditetapkan 1 jika $b' = b$, dan 0 jika tidak. Jika $PubK_{\mathcal{A},\prod}^{LR-cpa}(n) = 1$, kita katakan $\mathcal{A}$ berhasil.

DEFINISI 11.5 Skema enkripsi kunci publik $\prod$ = (Gen, Enc, Des) mempunyai banyak enkripsi yang tidak dapat dibedakan jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga:

$$\Pr[\mathsf{PubK}_{\mathcal{A},\Pi}^{\mathsf{LR\text{-}cpa}}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Kami akan menunjukkan bahwa setiap skema aman CPA secara otomatis memiliki banyak enkripsi yang tidak dapat dibedakan; yaitu, dalam pengaturan kunci publik, keamanan untuk enkripsi satu pesan berarti keamanan untuk enkripsi beberapa pesan. Ini berarti kita dapat membuktikan keamanan beberapa skema sehubungan dengan Definisi 11.2, yang lebih sederhana dan mudah untuk dikerjakan, dan menyimpulkan bahwa skema tersebut

memenuhi Definisi 11.5, definisi yang tampaknya lebih kuat yang memodelkan penggunaan enkripsi kunci publik di dunia nyata dengan lebih akurat. . Bukti teorema berikut diberikan di bawah ini.

TEOREMA 11.6 Jika skema enkripsi kunci publik $\prod$ aman terhadap CPA, maka skema tersebut juga mempunyai beberapa enkripsi yang tidak dapat dibedakan.

Hasil analogi dalam pengaturan kunci privat dinyatakan, namun tidak dibuktikan, seperti Teorema 3.24. Mengenkripsi pesan dengan panjang sewenang-wenang. Konsekuensi langsung dari Teorema 11.6 adalah bahwa skema enkripsi kunci publik yang aman dengan CPA untuk pesan dengan panjang tetap menyiratkan skema enkripsi kunci publik untuk pesan dengan panjang sembarang yang memenuhi gagasan keamanan yang sama. Kami menggambarkan hal ini dalam kasus ekstrim ketika skema asli hanya mengenkripsi pesan 1-bit. Katakanlah $\prod$ = (Gen, Enc, Dec) adalah skema enkripsi untuk pesan bit tunggal. Kita dapat membuat skema baru $\prod'$ = (Gen, Enc', Dec') yang memiliki ruang pesan $\{0,1\}^*$ dengan mendefinisikan Enc' sebagai berikut:

$$\mathsf{Enc}'_{pk}(m) = \mathsf{Enc}_{pk}(m_1), \ldots, \mathsf{Enc}_{pk}(m_\ell), \qquad (11.1)$$

dimana $m = m_1 \ldots m_\ell$. (Algoritma dekripsi Dec' dibuat dengan cara yang jelas.) Kita memiliki:

KLAIM 11.7 Misalkan $\prod$ dan $\prod'$ seperti di atas. Jika $\prod$ aman terhadap CPA, maka $\prod'$ juga aman.

Klaim tersebut mengikuti karena kita dapat melihat enkripsi pesan m menggunakan $\prod'$ sebagai enkripsi t pesan $(m_1, \ldots, m_t)$ menggunakan skema $\prod$. Catatan tentang terminologi. Kami telah memperkenalkan tiga definisi keamanan untuk skema enkripsi kunci publik—enkripsi yang tidak dapat dibedakan dengan adanya penyadap, keamanan CPA, dan beberapa enkripsi yang tidak dapat dibedakan—yang semuanya setara. Mengikuti konvensi umum dalam literatur kriptografi, kami hanya akan menggunakan istilah "keamanan CPA" untuk merujuk pada skema yang memenuhi gagasan keamanan ini.

Pembuktian Teorema 11.6

Pembuktian Teorema 11.6 agak rumit. Oleh karena itu kami memberikan beberapa intuisi sebelum beralih ke detailnya. Untuk diskusi intuitif ini, kami berasumsi untuk kesederhanaan bahwa $\mathcal{A}$ hanya melakukan dua panggilan ke oracle LR dalam eksperimen $PubK^{LR-cpa}_{\mathcal{A},\prod}(n)$. (Dalam bukti lengkapnya, jumlah panggilan berubah-ubah.)

Perbaiki musuh ppt sewenang-wenang $\mathcal{A}$ dan skema enkripsi kunci publik aman CPA $\prod$, dan pertimbangkan eksperimen $PubK^{LR-cpa2}_{\mathcal{A},\prod}(n)$ di mana $\mathcal{A}$ hanya dapat membuat dua kueri ke oracle LR. Nyatakan pertanyaan yang dibuat oleh $\mathcal{A}$ kepada Oracle dengan $(m_{1,0}, m_{1,1})$ dan $(m_{2,0}, m_{2,1})$; perhatikan bahwa pasangan pesan kedua mungkin bergantung pada teks sandi pertama yang diperoleh $\mathcal{A}$ dari oracle. Dalam percobaan, $\mathcal{A}$ menerima

sepasang teks tersandi $(Enc_{pk}(m_{1,0}), Enc_{pk}(m_{2,0}))$ (jika $b = 0$), atau sepasang teks tersandi $(Enc_{pk}(m_{1,1}), Enc_{pk}(m_{2,1}))$ (jika $b = 1$). Kita menulis $\mathcal{A}(pk, Enc_{pk}(m_{1,0}), Enc_{pk}(m_{2,0}))$ untuk menunjukkan keluaran $\mathcal{A}$ pada kasus pertama, dan analog dengan kasus kedua.

Misalkan $\vec{C}_0$ menunjukkan distribusi pasangan ciphertext pada kasus pertama, dan $\vec{C}_1$ menunjukkan distribusi pasangan ciphertext pada kasus kedua. Untuk menunjukkan bahwa Definisi 11.5 berlaku (untuk $PubK_{\mathcal{A},\prod}^{LR-cpa2}$), kita perlu membuktikan bahwa $\mathcal{A}$ tidak dapat membedakan antara diberikan sepasang teks sandi yang didistribusikan menurut $\vec{C}_0$, atau sepasang teks sandi yang didistribusikan menurut $\vec{C}_1$. Artinya, kita perlu membuktikan bahwa ada fungsi yang dapat diabaikan sehingga

$$\left| \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0})) = 1] - \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,1}), \mathsf{Enc}_{pk}(m_{2,1})) = 1] \right| \leq \mathsf{negl}(n). \quad (11.2)$$

(Ini setara dengan Definisi 11.5 karena alasan yang sama bahwa Definisi 3.9 setara dengan Definisi 3.8.) Untuk membuktikannya, kami akan menunjukkan bahwa

1. Keamanan CPA $\prod$ menyiratkan bahwa $\mathcal{A}$ tidak dapat membedakan antara kasus ketika diberikan sepasang teks tersandi yang didistribusikan menurut $\vec{C}_0$, atau sepasang teks tersandi $(Enc_{pk}(m_{1,0}), Enc_{pk}(m_{2,1}))$, yang berhubungan dengan mengenkripsi pesan pertama dalam kueri oracle pertama $\mathcal{A}$ dan pesan kedua dalam kueri oracle kedua $\mathcal{A}$. (Meskipun hal ini tidak dapat terjadi di $PubK_{\mathcal{A},\prod}^{LR-cpa2}(n)$, kita masih dapat menanyakan perilaku $\mathcal{A}$ jika diberi pasangan ciphertext seperti itu.) Misalkan $\vec{C}_{01}$ menunjukkan distribusi pasangan ciphertext dalam kasus terakhir ini.
2. Demikian pula, keamanan CPA dari $\prod$ menyiratkan bahwa $\mathcal{A}$ tidak dapat membedakan antara kasus ketika diberikan sepasang ciphertext yang didistribusikan menurut $\vec{C}_{01}$, atau sepasang ciphertext yang didistribusikan menurut $\vec{C}_1$.

Di atas mengatakan bahwa $\mathcal{A}$ tidak dapat membedakan antara distribusi $\vec{C}_0$ dan $\vec{C}_{01}$, maupun antara distribusi $\vec{C}_{01}$ dan $\vec{C}_1$. Kami menyimpulkan (menggunakan aljabar sederhana) bahwa $\mathcal{A}$ tidak dapat membedakan antara distribusi $\vec{C}_0$ dan $\vec{C}_1$. Inti dari pembuktiannya adalah menunjukkan bahwa $\mathcal{A}$ tidak dapat membedakan antara diberikan sepasang ciphertext yang didistribusikan menurut $\vec{C}_0$, atau sepasang ciphertext yang didistribusikan menurut $\vec{C}_{01}$. (Kasus lainnya mengikuti hal serupa.)

Artinya, kami ingin menunjukkan bahwa ada fungsi yang dapat diabaikan

$$\left| \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0})) = 1] - \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,1})) = 1] \right| \leq \mathsf{negl}(n). \quad (11.3)$$

Perhatikan bahwa satu-satunya perbedaan antara masukan musuh $\mathcal{A}$ dalam setiap kasus adalah pada elemen kedua. Secara intuitif, ketidakmampuan membedakan muncul dari kasus pesan tunggal karena $\mathcal{A}$ dapat menghasilkan $Enc_{pk}(m_{1,0})$ dengan sendirinya. Secara formal, pertimbangkan musuh ppt berikut $\mathcal{A}'$ yang dijalankan dalam eksperimen $PubK^{eav}_{\mathcal{A}',\prod}(n)$:

Musuh $\mathcal{A}'$:

1. Pada masukan pk, musuh $\mathcal{A}'$ menjalankan $\mathcal{A}(pk)$ sebagai subrutin.
2. Ketika A membuat query pertamanya $(m_{1,0}, m_{1,1})$ ke oracle LR, $\mathcal{A}'$ menghitung $c_1 \leftarrow Enc_{pk}(m_{1,0})$ dan mengembalikan c_1 ke $\mathcal{A}$ sebagai respon dari oracle.
3. Ketika $\mathcal{A}$ membuat query kedua $(m_{2,0}, m_{2,1})$ ke oracle LR, $\mathcal{A}'$ mengeluarkan $(m_{2,0}, m_{2,1})$ dan menerima kembali tantangan ciphertext c_2. Ini dikembalikan ke $\mathcal{A}$ sebagai respons dari oracle LR.
4. $\mathcal{A}'$ mengeluarkan bit b' yang dikeluarkan oleh $\mathcal{A}$.

Melihat percobaan $PubK^{eav}_{\mathcal{A}',\prod}(n)$, kita melihat bahwa ketika $b = 0$ maka tantangan ciphertext c_2 dihitung sebagai $Enc_{pk}(m_{2,0})$. Dengan demikian,

$$\Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,0})) = 1] = \Pr[\mathcal{A}(\mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,0})) = 1]. \quad (11.4)$$

(Kami menyembunyikan penyebutan pk secara eksplisit untuk menghemat ruang.) Sebaliknya, jika $b = 1$ dalam eksperimen $PubK^{eav}_{\mathcal{A}',\prod}(n)$, maka c_2 dihitung sebagai $Enc_{pk}(m_{2,1})$ dan seterusnya

$$|\Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,0})) = 1] - \Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,1})) = 1]| \leq \mathsf{negl}(n).$$

Keamanan CPA dari $\prod$ menyiratkan bahwa ada fungsi yang dapat diabaikan sehingga

$$|\Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,0})) = 1] - \Pr[\mathcal{A}'(\mathsf{Enc}_{pk}(m_{2,1})) = 1]| \leq \mathsf{negl}(n).$$

Ini, bersama dengan Persamaan (11.4) dan (11.5), menghasilkan Persamaan (11.3).
Dengan cara yang hampir sama, kita dapat membuktikan bahwa:

$$\big|\Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,0}), \mathsf{Enc}_{pk}(m_{2,1})) = 1] - \Pr[\mathcal{A}(pk, \mathsf{Enc}_{pk}(m_{1,1}), \mathsf{Enc}_{pk}(m_{2,1})) = 1]\big| \leq \mathsf{negl}(n). \quad (11.6)$$

Persamaan (11.2) dilanjutkan dengan menggabungkan Persamaan (11.3) dan (11.6).
Komplikasi utama yang muncul dalam kasus umum adalah bahwa jumlah kueri ke oracle LR tidak lagi tetap tetapi mungkin berupa polinomial n yang berubah-ubah. Dalam pembuktian

formal, hal ini ditangani dengan menggunakan argumen hibrid. (Argumen hibrid juga digunakan di Bab 7.)

ATAP (Teorema 11.6) Misalkan Π adalah skema enkripsi kunci publik yang aman dengan CPA dan A adalah musuh ppt sembarang dalam eksperimen $PubK^{LR-cpa}_{\mathcal{A},\prod}(n)$. Misalkan $t = t(n)$ adalah batas atas polinomial pada jumlah kueri yang dibuat oleh $\mathcal{A}$ ke oracle LR, dan asumsikan tanpa kehilangan keumuman bahwa $\mathcal{A}$ selalu menanyakan oracle sebanyak ini. Untuk kunci publik tertentu pk dan $0 \leq i \leq t$, misalkan LR^i_{pk} menunjukkan oracle yang pada input (m_0, m_1) mengembalikan $\text{Enc}_{pk}(m_0)$ untuk kueri i pertama yang diterimanya, dan mengembalikan $\text{Enc}_{pk}(m_1)$ untuk $t - i$ berikutnya − saya menanyakan apa yang diterimanya. (Artinya, untuk kueri i pertama, pesan pertama pada pasangan masukan dienkripsi, dan sebagai respons terhadap kueri yang tersisa, pesan kedua dalam pasangan masukan dienkripsi.) Kami menekankan bahwa setiap enkripsi dihitung menggunakan keacakan yang seragam dan independen. Dengan menggunakan notasi ini, kita punya

$$\Pr\left[\mathsf{PubK}^{\mathsf{LR\text{-}cpa}}_{\mathcal{A},\Pi}(n) = 1\right] = \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{LR}^t_{pk}}(pk) = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{LR}^0_{pk}}(pk) = 1]$$

karena, dari sudut pandang $\mathcal{A}$ (yang membuat tepat t kueri), Oracle LR^t_{pk} setara dengan $LR_{pk,0}$, dan Oracle LR^0_{pk} setara dengan $LR_{pk,1}$. Untuk membuktikan bahwa $\prod$ memenuhi Definisi 11.5, kita akan menunjukkan bahwa untuk setiap ppt $\mathcal{A}$ terdapat fungsi yang dapat diabaikan negl′ sehingga

$$\left|\Pr[\mathcal{A}^{\mathsf{LR}^t_{pk}}(pk) = 1] - \Pr[\mathcal{A}^{\mathsf{LR}^0_{pk}}(pk) = 1]\right| \leq \mathsf{negl}'(n). \qquad (11.7)$$

(Seperti sebelumnya, ini setara dengan Definisi 11.5 dengan alasan yang sama bahwa Definisi 3.9 setara dengan Definisi 3.8.) Pertimbangkan musuh ppt berikut $\mathcal{A}'$ yang menguping enkripsi satu pesan:

Musuh $\mathcal{A}'$:

1. $\mathcal{A}'$, jika diketahui pk, memilih indeks seragam $i \leftarrow \{1, \ldots, T\}$.
2. $\mathcal{A}'$ menjalankan $\mathcal{A}(pk)$, menjawab query oracle ke-j $(m_{j,o}, m_{j,1})$ sebagai berikut:
   (a) Untuk $j < i$, musuh $\mathcal{A}'$ menghitung $c_j \leftarrow Enc_{pk}(m_{j,0})$ dan mengembalikan c_j ke $\mathcal{A}$ sebagai respons dari oraclenya.
   (b) Untuk $j = i$, musuh $\mathcal{A}'$ mengeluarkan $(m_{j,o}, m_{j,1})$ dan menerima kembali teks sandi tantangan c_j. Ini dikembalikan ke $\mathcal{A}$ sebagai respon dari oraclenya.
   (c) Untuk $j > i$, musuh $\mathcal{A}'$ menghitung $c_j \leftarrow Enc_{pk}(m_{j,1})$ dan mengembalikan c_j ke $\mathcal{A}$ sebagai respon dari oraclenya.
3. $\mathcal{A}'$ mengeluarkan bit b' yang dikeluarkan oleh $\mathcal{A}$.

Pertimbangkan eksperimen $PubK^{eav}_{\mathcal{A}',\prod}(n)$. Memperbaiki beberapa pilihan $i = i^*$, perhatikan bahwa jika c_{i^*} merupakan enkripsi dari $m_{i^*,0}$ maka interaksi $\mathcal{A}$ dengan oraclenya identik dengan interaksi dengan oracle $LR^{i^*}_{pk}$. Dengan demikian,

$$\begin{aligned}\Pr[\mathcal{A}' \text{ outputs } 1 \mid b=0] &= \sum_{i^*=1}^{t} \Pr[i=i^*] \cdot \Pr[\mathcal{A}' \text{ outputs } 1 \mid b=0 \wedge i=i^*] \\ &= \sum_{i^*=1}^{t} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right].\end{aligned}$$

Sebaliknya, jika c_{i^*} merupakan enkripsi dari $m_{i^*,1}$ maka interaksi $\mathcal{A}$ dengan oraclenya identik dengan interaksi dengan oracle $LR^{i^*-1}_{pk}$, dan seterusnya

$$\begin{aligned}\Pr[\mathcal{A}' \text{ outputs } 1 \mid b=1] &= \sum_{i^*=1}^{t} \Pr[i=i^*] \cdot \Pr[\mathcal{A}' \text{ outputs } 1 \mid b=1 \wedge i=i^*] \\ &= \sum_{i^*=1}^{t} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*-1}_{pk}}(pk)=1\right] \\ &= \sum_{i^*=0}^{t-1} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right],\end{aligned}$$

dimana persamaan ketiga diperoleh hanya dengan menggeser indeks penjumlahan. Karena $\mathcal{A}'$ berjalan dalam waktu polinomial, asumsi bahwa $\prod$ aman terhadap CPA berarti terdapat fungsi yang dapat diabaikan sehingga

$$\left|\Pr[\mathcal{A}' \text{ outputs } 1 \mid b=0] - \Pr[\mathcal{A}' \text{ outputs } 1 \mid b=1]\right| \leq \mathsf{negl}(n).$$

Tapi ini berarti bahwa

$$\begin{aligned}\mathsf{negl}(n) &\geq \left|\sum_{i^*=1}^{t} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right] - \sum_{i^*=0}^{t-1} \frac{1}{t} \cdot \Pr\left[\mathcal{A}^{\mathsf{LR}^{i^*}_{pk}}(pk)=1\right]\right| \\ &= \frac{1}{t} \cdot \left|\Pr\left[\mathcal{A}^{\mathsf{LR}^{t}_{pk}}(pk)=1\right] - \Pr\left[\mathcal{A}^{\mathsf{LR}^{0}_{pk}}(pk)=1\right]\right|,\end{aligned}$$

karena semua kecuali satu suku dalam setiap penjumlahan dibatalkan. Kami menyimpulkan bahwa

$$\left|\Pr\left[\mathcal{A}^{\mathsf{LR}^{t}_{pk}}(pk)=1\right] - \Pr\left[\mathcal{A}^{\mathsf{LR}^{0}_{pk}}(pk)=1\right]\right| \leq t(n) \cdot \mathsf{negl}(n).$$

Karena t polinomial, fungsi $t \cdot \mathrm{negl}(n)$ dapat diabaikan. Karena adalah musuh ppt yang sewenang-wenang, ini menunjukkan bahwa Persamaan (11.7) berlaku dan melengkapi bukti bahwa $\prod$ memiliki banyak enkripsi yang tidak dapat dibedakan.

Keamanan terhadap Serangan Ciphertext Terpilih

Serangan teks sandi terpilih, di mana musuh dapat memperoleh dekripsi teks sandi sewenang-wenang pilihannya (dengan satu batasan teknis dijelaskan di bawah), merupakan masalah dalam pengaturan kunci publik seperti halnya dalam kunci privat. pengaturan. Faktanya, hal ini bisa dibilang lebih menjadi perhatian dalam pengaturan kunci publik karena

penerima mengharapkan untuk menerima teks sandi dari beberapa pengirim yang mungkin tidak diketahui sebelumnya, sedangkan penerima dalam pengaturan kunci pribadi bermaksud untuk berkomunikasi hanya dengan satu pengirim. , pengirim yang diketahui menggunakan kunci rahasia tertentu.

Asumsikan seorang penyadap $\mathcal{A}$ mengamati ciphertext c yang dikirim oleh pengirim S ke penerima $\mathcal{R}$. Secara umum, dalam pengaturan kunci publik ada dua kelas serangan ciphertext yang dipilih:

- $\mathcal{A}$ mungkin mengirim ciphertext c' yang dimodifikasi ke $\mathcal{R}$ atas nama S. (Misalnya, dalam konteks email terenkripsi, $\mathcal{A}$ mungkin membuat email terenkripsi c' dan memalsukan kolom "Dari" sehingga muncul email tersebut berasal dari S.) Dalam kasus ini, meskipun kecil kemungkinannya $\mathcal{A}$ dapat memperoleh seluruh dekripsi m' dari c', $\mathcal{A}$ mungkin dapat menyimpulkan beberapa informasi tentang m' berdasarkan perilaku selanjutnya dari $\mathcal{R}$. Berdasarkan informasi ini, $\mathcal{A}$ mungkin dapat mempelajari sesuatu tentang pesan asli m.
- $\mathcal{A}$ mungkin mengirimkan ciphertext c' yang dimodifikasi ke $\mathcal{R}$ atas namanya sendiri. Dalam hal ini, $\mathcal{A}$ mungkin memperoleh seluruh dekripsi m′ dari c' jika $\mathcal{R}$ merespons langsung ke $\mathcal{A}$. Bahkan jika $\mathcal{A}$ tidak mengetahui apa pun tentang m', pesan yang dimodifikasi ini mungkin memiliki hubungan yang diketahui dengan pesan asli m yang dapat dimanfaatkan oleh $\mathcal{A}$; lihat skenario ketiga di bawah sebagai contoh.

Serangan kelas kedua khusus untuk konteks enkripsi kunci publik, dan tidak memiliki analogi dalam pengaturan kunci privat. Tidak sulit untuk mengidentifikasi sejumlah skenario realistis yang menggambarkan jenis serangan di atas:

Skenario 1. Katakanlah pengguna S masuk ke rekening banknya dengan mengirimkan ke banknya enkripsi kata sandinya pw yang digabungkan dengan stempel waktu. Asumsikan lebih lanjut bahwa ada dua jenis pesan kesalahan yang dikirimkan bank: bank mengembalikan "kata sandi salah" jika kata sandi terenkripsi tidak cocok dengan kata sandi yang disimpan di S, dan "stempel waktu salah" jika kata sandi benar tetapi stempel waktu tidak.

Jika musuh memperoleh ciphertext c yang dikirim oleh S ke bank, maka musuh sekarang dapat melancarkan serangan ciphertext yang dipilih dengan mengirimkan ciphertext c' ke bank atas nama S dan mengamati pesan kesalahan yang dihasilkan. (Ini mirip dengan serangan padding-Oracle yang kita lihat di Bagian 3.7.2.) Dalam beberapa kasus, informasi ini mungkin cukup untuk memungkinkan musuh mengetahui seluruh kata sandi pengguna.

Skenario 2. Katakanlah S mengirimkan email terenkripsi c ke $\mathcal{R}$, dan email ini adalah diamati oleh $\mathcal{A}$. Jika $\mathcal{A}$ mengirim, atas namanya sendiri, email terenkripsi c' ke $\mathcal{R}$, maka $\mathcal{R}$ mungkin membalas email ini dan mengutip teks yang didekripsi m' yang sesuai dengan c'. Dalam hal ini, $\mathcal{R}$ pada dasarnya bertindak sebagai oracle dekripsi untuk $\mathcal{A}$ dan berpotensi mendekripsi ciphertext apa pun yang dikirimkan $\mathcal{A}$.

Skenario 3. Masalah yang berkaitan erat dengan keamanan teks tersandi yang dipilih adalah potensi kelenturan teks tersandi. Karena definisi formal cukup rumit, kami tidak memberikannya di sini namun hanya memberikan gagasan intuitif.

Suatu skema dapat ditempa jika memiliki properti berikut: diberi enkripsi c dari beberapa pesan yang tidak diketahui m, dimungkinkan untuk menghasilkan ciphertext c' yang merupakan enkripsi dari pesan m' yang terkait dengan cara yang diketahui ke m. Misalnya, jika diberi enkripsi m, maka dimungkinkan untuk membuat enkripsi $2m$.

Sekarang bayangkan $\mathcal{R}$ mengadakan lelang, di mana dua pihak S dan $\mathcal{A}$ mengajukan tawaran mereka dengan mengenkripsi mereka menggunakan kunci publik $\mathcal{R}$. Jika skema enkripsi yang dapat ditempa digunakan, musuh $\mathcal{A}$ mungkin selalu menempatkan tawaran tertinggi (tanpa menawar secara maksimal) dengan melakukan serangan berikut: tunggu sampai S mengirimkan ciphertext c sesuai dengan tawarannya m (yang tidak diketahui $\mathcal{A}$); kemudian kirimkan ciphertext c' sesuai dengan tawaran $m' = 2m$. Perhatikan bahwa m(dan m', dalam hal ini) tetap tidak diketahui oleh $\mathcal{A}$ sampai $\mathcal{R}$ mengumumkan hasilnya, sehingga kemungkinan serangan semacam itu tidak bertentangan dengan fakta bahwa skema enkripsinya aman terhadap CPA. Sebaliknya, skema yang aman dengan CCA dapat terbukti tidak mudah diubah, artinya skema tersebut tidak rentan terhadap serangan semacam itu.

Definisi. Keamanan terhadap serangan teks sandi yang dipilih ditentukan oleh modifikasi yang sesuai dari definisi analog dari pengaturan kunci pribadi (Definisi 3.33). Mengingat skema enkripsi kunci publik $\prod$ dan musuh $\mathcal{A}$, pertimbangkan eksperimen berikut:

Eksperimen CCA yang tidak dapat dibedakan Pub$K^{cca}_{\mathcal{A},\prod}(n)$:

1. Gen(1^n) dijalankan untuk mendapatkan kunci (pk, sk).
2. Musuh $\mathcal{A}$ diberikan pk dan akses ke oracle dekripsi Dec$_{sk}(\cdot)$. Ini menghasilkan sepasang pesan m_0, m_1 dengan panjang yang sama. (Pesan-pesan ini harus berada di ruang pesan yang terkait dengan pk.)
3. Bit seragam $b \in \{0,1\}$ dipilih, dan kemudian ciphertext $c \leftarrow$ Enc$_{pk}(m_b)$ dihitung dan diberikan kepada $\mathcal{A}$.
4. $\mathcal{A}$ terus berinteraksi dengan oracle dekripsi, tetapi tidak boleh meminta dekripsi c itu sendiri. Akhirnya, $\mathcal{A}$ mengeluarkan sedikit b'.
5. Keluaran percobaan ditetapkan 1 jika $b' = b$, dan 0 jika tidak.

DEFINISI 11.8 Skema enkripsi kunci publik $\prod$ = (Gen, Enc, Des) memiliki enkripsi yang tidak dapat dibedakan dalam serangan teks sandi yang dipilih (atau aman CCA) jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\text{PubK}^{\text{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \text{negl}(n).$$

Analog alami dari Teorema 11.6 juga berlaku untuk keamanan CCA. Artinya, jika suatu skema memiliki enkripsi yang tidak dapat dibedakan dalam serangan teks sandi yang dipilih, maka skema tersebut memiliki beberapa enkripsi yang tidak dapat dibedakan dalam serangan teks sandi yang dipilih, yang mana hal ini didefinisikan dengan tepat. Menariknya, analogi Klaim 11.7 tidak berlaku untuk keamanan CCA.

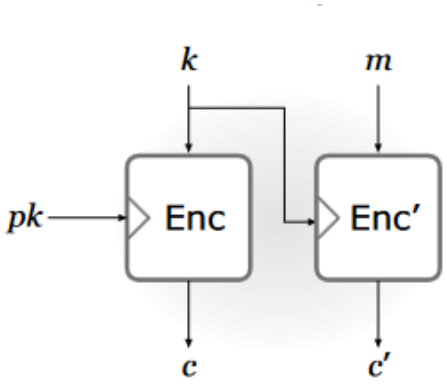
Seperti pada Definisi 3.33, kita harus mencegah penyerang mengirimkan tantangan ciphertext c ke oracle dekripsi agar definisi dapat dicapai. Namun pembatasan ini tidak membuat definisi tersebut menjadi tidak berarti dan, khususnya, untuk masing-masing dari tiga skenario motivasi yang diberikan sebelumnya, orang dapat berargumentasi bahwa pengaturan $c' = c$ tidak bermanfaat bagi penyerang:

Dalam skenario pertama yang melibatkan login berbasis kata sandi, penyerang tidak mengetahui apa pun tentang $S's$ kata sandi tersebut dengan memutar ulang c ke bank karena dalam kasus ini penyerang sudah mengetahui bahwa tidak ada pesan kesalahan yang akan dihasilkan.

Dalam skenario kedua yang melibatkan email terenkripsi, pengiriman $c' = c$ ke penerima kemungkinan besar akan membuat penerima curiga sehingga menolak merespons sama sekali.

Dalam skenario akhir yang melibatkan lelang, $\mathcal{R}$ kecurangan dapat dengan mudah dideteksi jika tawaran terenkripsi pihak lawan identik dengan tawaran terenkripsi pihak lain. Bagaimanapun, dalam hal ini, yang dicapai penyerang dengan mengulangi c adalah bahwa ia mengajukan tawaran yang sama dengan pihak yang jujur. Gambar 11.7 menunjukkan bahwa setiap skema enkripsi kunci publik aman-CPA untuk pesan ℓ'-bit dapat digunakan untuk mendapatkan skema enkripsi kunci publik aman-CPA untuk pesan-pesan dengan panjang sembarang. Mengenkripsi pesan l-bit menggunakan pendekatan ini memerlukan pemanggilan $\gamma \stackrel{\text{def}}{=} [\ell/\ell']$ dari skema enkripsi asli, yang berarti bahwa komputasi dan panjang ciphertext bertambah dengan faktor perkalian γ relatif terhadap skema yang mendasarinya.

Dimungkinkan untuk melakukan lebih baik dengan menggunakan enkripsi kunci pribadi bersama-sama dengan enkripsi kunci publik. Hal ini meningkatkan efisiensi karena enkripsi kunci privat secara signifikan lebih cepat dibandingkan enkripsi kunci publik, dan meningkatkan bandwidth karena skema kunci privat memiliki perluasan teks sandi yang lebih rendah. Kombinasi yang dihasilkan disebut enkripsi hibrid dan digunakan secara luas dalam praktik. Ide dasarnya adalah menggunakan enkripsi kunci publik untuk mendapatkan kunci bersama k, dan kemudian mengenkripsi pesan m menggunakan skema enkripsi kunci pribadi dan kunci k. Penerima menggunakan kunci privat jangka panjang (asimetris) untuk memperoleh k, dan kemudian menggunakan dekripsi kunci privat (dengan kunci k) untuk memulihkan pesan asli. Kami menekankan bahwa meskipun enkripsi kunci pribadi digunakan sebagai komponen, ini adalah skema enkripsi kunci publik yang lengkap berdasarkan fakta bahwa pengirim dan penerima tidak membagikan kunci rahasia apa pun sebelumnya.



GAMBAR 11.1: Enkripsi hibrid. Enc menunjukkan skema enkripsi kunci publik, sedangkan Enc' adalah skema enkripsi kunci pribadi.

Dalam implementasi langsung ide ini (lihat Gambar 11.1), pengirim akan membagi k dengan (1) memilih nilai k yang seragam dan kemudian (2) mengenkripsi k menggunakan skema enkripsi kunci publik. Pendekatan yang lebih langsung adalah dengan menggunakan kunci publik primitif yang disebut mekanisme enkapsulasi kunci (KEM) untuk mencapai kedua hal ini "dalam satu kesempatan." Hal ini menguntungkan baik dari sudut pandang konseptual maupun dari segi efisiensi, seperti yang akan kita lihat nanti.

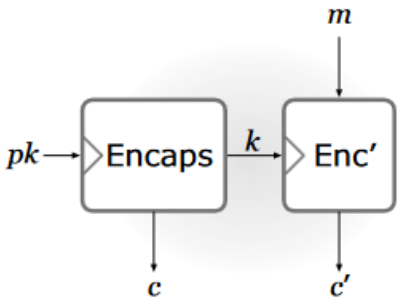
KEM memiliki tiga algoritma yang serupa dengan skema enkripsi kunci publik. Seperti sebelumnya, algoritma pembangkitan kunci Gen digunakan untuk menghasilkan sepasang kunci publik dan privat. Sebagai pengganti enkripsi, kita sekarang memiliki algoritma enkapsulasi Encaps yang hanya mengambil kunci publik sebagai masukan (dan tidak ada pesan), dan mengeluarkan teks sandi c bersama dengan kunci k. Algoritma dekapsulasi Decaps yang sesuai dijalankan oleh penerima untuk memulihkan k dari ciphertext c menggunakan kunci privat. Secara formal:

DEFINISI 11.9 Mekanisme enkapsulasi kunci (KEM) adalah rangkaian algoritma waktu polinomial probabilistik (Gen, Encaps, Decaps) sedemikian rupa sehingga:

1. Algoritme pembangkitan kunci Gen mengambil parameter keamanan 1^n sebagai masukan dan mengeluarkan pasangan kunci publik/pribadi (pk, sk). Kita asumsikan pk dan sk masing-masing memiliki panjang paling sedikit n, dan n dapat ditentukan dari pk.
2. Algoritma enkapsulasi Encaps mengambil input kunci publik pk dan parameter keamanan 1^n. Ini menghasilkan teks sandi c dan kunci $k \in \{0,1\}^{\ell(n)}$ dengan ℓ adalah panjang kunci. Kami menulis ini sebagai $(c, k) \rightarrow \text{Encaps}_{pk}(1^n)$.
3. Algoritma dekapsulasi deterministik Decaps mengambil input kunci privat sk dan ciphertext c, dan mengeluarkan kunci k atau simbol khusus $\perp$ yang menunjukkan kegagalan. Kami menulis ini sebagai $k := \text{Decaps}_{sk}(c)$.

Diperlukan bahwa dengan semua probabilitas yang dapat diabaikan atas (sk, pk) keluaran sebesar $\text{Gen}(1^n)$, jika $\text{Encaps}_{pk}(1^n)$ mengeluarkan (c, k) maka $\text{Decaps}_{sk}(c)$ mengeluarkan k.

Dalam definisi tersebut kita berasumsi untuk kesederhanaan bahwa Encaps selalu menghasilkan (teks sandi c dan) kunci dengan panjang tetap $\ell(n)$. Kita juga dapat mempertimbangkan definisi yang lebih umum di mana Encaps mengambil 1^{ℓ} sebagai masukan tambahan dan mengeluarkan kunci dengan panjang ℓ. Skema enkripsi kunci publik apa pun secara sederhana memberikan KEM dengan memilih kunci acak k dan mengenkripsinya. Namun, seperti yang akan kita lihat, konstruksi khusus KEMs bisa lebih efisien.



GAMBAR 11.2: Enkripsi hibrid menggunakan pendekatan KEM/DEM.

Dengan menggunakan KEM (dengan panjang kunci n), kita dapat menerapkan enkripsi hibrid seperti pada Gambar 11.2. Pengirim menjalankan Encaps$_{pk}(1^n)$ untuk mendapatkan c bersama dengan kunci k; ia kemudian menggunakan skema enkripsi kunci pribadi untuk mengenkripsi pesannya m, menggunakan k sebagai kuncinya. Dalam konteks ini, skema enkripsi kunci privat disebut mekanisme enkapsulasi data (DEM) karena alasan yang jelas. Teks sandi yang dikirim ke penerima mencakup c dan teks sandi c' dari skema kunci privat. Konstruksi 11.10 memberikan spesifikasi formal.

Berapa efisiensi skema enkripsi hibrid yang dihasilkan $\prod^{hy}$? Untuk beberapa nilai tetap n, misalkan a menunjukkan biaya enkapsulasi kunci n-bit menggunakan Encaps, dan misalkan β menunjukkan biaya (per bit teks biasa) enkripsi menggunakan Enc'. Asumsikan $|m| > n$, yang merupakan kasus menarik. Maka biaya, per bit teks biasa, untuk mengenkripsi pesan m menggunakan $\prod^{hy}$ adalah

$$\frac{\alpha + \beta \cdot |m|}{|m|} = \frac{\alpha}{|m|} + \beta, \qquad (11.8)$$

yang mendekati β untuk waktu yang cukup lama m. Maka, dalam batas pesan yang sangat panjang, biaya per bit yang dikeluarkan oleh skema enkripsi kunci publik $\prod^{hy}$ sama dengan biaya per bit skema kunci privat $\prod'$. Enkripsi hibrid memungkinkan kita mencapai fungsionalitas enkripsi kunci publik dengan efisiensi enkripsi kunci pribadi, setidaknya untuk pesan yang cukup panjang.

CONSTRUCTION 11.10

Let $\Pi = (\mathsf{Gen}, \mathsf{Encaps}, \mathsf{Decaps})$ be a KEM with key length n, and let $\Pi' = (\mathsf{Gen}', \mathsf{Enc}', \mathsf{Dec}')$ be a private-key encryption scheme. Construct a public-key encryption scheme $\Pi^{\mathsf{hy}} = (\mathsf{Gen}^{\mathsf{hy}}, \mathsf{Enc}^{\mathsf{hy}}, \mathsf{Dec}^{\mathsf{hy}})$ as follows:

- $\mathsf{Gen}^{\mathsf{hy}}$: on input 1^n run $\mathsf{Gen}(1^n)$ and use the public and private keys (pk, sk) that are output.
- $\mathsf{Enc}^{\mathsf{hy}}$: on input a public key pk and a message $m \in \{0,1\}^*$ do:
  1. Compute $(c, k) \leftarrow \mathsf{Encaps}_{pk}(1^n)$.
  2. Compute $c' \leftarrow \mathsf{Enc}'_k(m)$.
  3. Output the ciphertext $\langle c, c' \rangle$.
- $\mathsf{Dec}^{\mathsf{hy}}$: on input a private key sk and a ciphertext $\langle c, c' \rangle$ do:
  1. Compute $k := \mathsf{Decaps}_{sk}(c)$.
  2. Output the message $m := \mathsf{Dec}'_k(c')$.

Enkripsi hibrid menggunakan paradigma KEM/DEM.

Perhitungan serupa dapat digunakan untuk mengukur pengaruh enkripsi hibrid pada panjang ciphertext. Untuk beberapa nilai tetap n, misalkan L menunjukkan panjang keluaran teks tersandi dengan Encaps, dan katakanlah enkripsi kunci pribadi dari sebuah pesan m menggunakan Enc' menghasilkan teks tersandi dengan panjang $n + |m|$ (ini dapat dicapai dengan menggunakan salah satu dari mode enkripsi yang dibahas di Bagian 3.6; sebenarnya, teks tersandi dengan panjang $|m|$ pun dimungkinkan karena, seperti yang akan kita lihat, $\prod'$ tidak harus aman terhadap CPA). Maka total panjang ciphertext pada skema $\prod^{hy}$ adalah

$$L + n + |m|. \qquad (11.9)$$

Sebaliknya, ketika menggunakan enkripsi blok demi blok seperti pada Persamaan (11.1), dan dengan asumsi bahwa enkripsi kunci publik dari pesan n-bit menggunakan Enc menghasilkan ciphertext dengan panjang L, enkripsi pesan m akan menghasilkan a teks sandi dengan panjang $L \cdot \lceil |m| / n \rceil$. Panjang ciphertext yang dilaporkan pada Persamaan (11.9) merupakan peningkatan yang signifikan untuk m yang cukup panjang.

Kita dapat menggunakan beberapa perkiraan kasar untuk memahami arti dari hasil di atas dalam praktiknya. (Kami menekankan bahwa angka-angka ini hanya dimaksudkan untuk memberikan pembaca gambaran mengenai peningkatan; nilai realistis akan bergantung pada berbagai faktor.) Nilai umum untuk panjang kunci k mungkin adalah $n = 128$. Selanjutnya, " skema enkripsi kunci publik asli" mungkin menghasilkan teks tersandi 256-bit ketika mengenkripsi pesan 128-bit; asumsikan KEM memiliki ciphertext dengan panjang yang sama ketika mengenkapsulasi kunci 128-bit. Membiarkan α, seperti sebelumnya, menunjukkan biaya komputasi enkripsi/enkapsulasi kunci publik dari kunci 128-bit, kita melihat bahwa enkripsi blok demi blok seperti pada Persamaan (11.1) akan mengenkripsi kunci 1 MB ($= 10^6$-bit) pesan dengan biaya komputasi $\alpha \lceil 10^6/128 \rceil \approx 7800 \cdot \alpha$ dan teks tersandi akan berukuran

2 MB. Bandingkan ini dengan efisiensi enkripsi hibrid. Membiarkan β, seperti sebelumnya, menunjukkan biaya komputasi per-bit enkripsi kunci pribadi, perkiraan yang masuk akal adalah $\beta \approx \alpha/10^5$. Dengan menggunakan Persamaan (11.8), kita melihat bahwa keseluruhan biaya komputasi untuk enkripsi hibrid untuk pesan 1 Mb adalah

$$\alpha + 10^6 \cdot \frac{\alpha}{10^5} = 11 \cdot \alpha,$$

dan ciphertextnya hanya sedikit lebih panjang dari 1 MB. Dengan demikian, enkripsi hibrid meningkatkan efisiensi komputasi dalam hal ini sebesar 700 kali lipat, dan panjang ciphertext sebesar 2 kali lipat.

Masih menganalisis keamanan $\prod^{hy}$. Hal ini tentu saja bergantung pada keamanan komponen dasarnya $\prod$ dan $\prod'$. Pada bagian berikut ini kami mendefinisikan pengertian keamanan CPA dan keamanan CCA untuk KEMs, dan menunjukkan:

Jika $\prod$ adalah KEM yang aman terhadap CPA dan skema kunci privat $\prod'$ mempunyai enkripsi yang tidak dapat dibedakan jika ada penyadap, maka $\prod^{hy}$ merupakan skema enkripsi kunci publik yang aman terhadap CPA. Perhatikan bahwa $\prod'$ cukup memenuhi definisi keamanan yang lebih lemah—yang, perlu diingat, tidak menyiratkan keamanan CPA dalam pengaturan kunci privat—agar skema hibrid $\prod^{hy}$ menjadi aman CPA. Secara intuitif, alasannya adalah kunci k yang baru dan seragam dipilih setiap kali pesan baru dienkripsi. Karena setiap kunci k hanya digunakan sekali, enkripsi tunggal $\prod'$ yang tidak dapat dibedakan sudah cukup untuk keamanan skema hibrid $\prod^{hy}$. Ini berarti enkripsi kunci pribadi dasar menggunakan generator pseudorandom (atau stream cipher), seperti pada Konstruksi 3.17, sudah cukup.

Jika Π adalah KEM yang aman dengan CCA dan $\prod'$ adalah skema enkripsi kunci pribadi yang aman dengan CCA, maka $\prod^{hy}$ adalah skema enkripsi kunci publik yang aman dengan CCA.

11.3 KEAMANAN CPA

Untuk mempermudah, kita asumsikan pada bagian ini dan bagian selanjutnya terdapat KEM dengan panjang kunci n. Kami mendefinisikan gagasan keamanan CPA untuk KEM dengan analogi dengan Definisi 11.2. Seperti di sana, musuh di sini menguping satu teks tersandi c. Definisi 11.2 mengharuskan penyerang tidak dapat membedakan apakah c merupakan enkripsi beberapa pesan m_0 atau pesan lain m_1. Dengan KEM tidak ada pesan, dan sebagai gantinya kita mengharuskan kunci yang dienkapsulasi k tidak dapat dibedakan dari kunci seragam yang tidak bergantung pada ciphertext c.

Misalkan $\prod =$ (Gen, Encaps, Decaps) adalah KEM dan $\mathcal{A}$ adalah musuh yang sewenang-wenang.

Eksperimen ketidakmampuan CPA untuk membedakan $KEM^{cpa}_{\mathcal{A},\prod}(n)$:

1. Gen(1^n) dijalankan untuk mendapatkan kunci (pk, sk). Kemudian Encaps$_{pk}(1^n)$ dijalankan untuk menghasilkan (c, k) dengan $k \in \{0,1\}^n$.
2. Bit seragam $b \in \{0,1\}$ dipilih. Jika $b = 0$ himpunan $\hat{k} := k$. Jika $b = 1$ maka pilihlah seragam $\hat{k} \in \{0,1\}^n$.
3. Berikan $(pk, c, \hat{k})$ kepada $\mathcal{A}$, yang mengeluarkan sedikit b'. Keluaran percobaan didefinisikan sebagai 1 jika $b' = b$, dan 0 jika tidak.

Dalam percobaan tersebut, $\mathcal{A}$ diberikan ciphertext c dan kunci sebenarnya k yang berhubungan dengan c, atau kunci seragam yang independen. KEM aman dari CPA jika tidak ada musuh efisien yang dapat membedakan kemungkinan-kemungkinan ini.

DEFINISI 11.11 Mekanisme enkapsulasi kunci $\prod$ aman terhadap CPA jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n).$$

Di sisa bagian ini kami membuktikan teorema berikut:

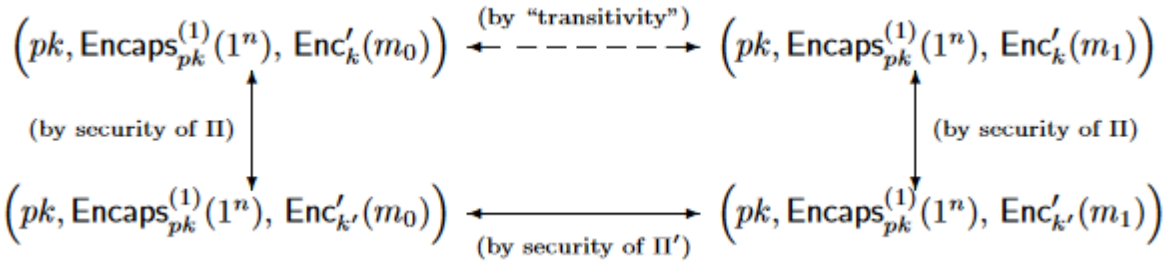
TEOREMA 11.12 Jika $\prod$ adalah KEM yang aman terhadap CPA dan $\prod'$ adalah skema enkripsi kunci pribadi yang enkripsinya tidak dapat dibedakan jika ada penyadap, maka $\prod^{hy}$ seperti pada Konstruksi 11.10 adalah skema enkripsi kunci publik yang aman terhadap CPA.

Sebelum membuktikan teorema secara formal, kami memberikan beberapa intuisi. Misalkan notasi "$X \overset{c}{\equiv} Y$" menunjukkan kejadian yang tidak dapat terjadi pada musuh dengan waktu polinomial membedakan antara dua distribusi X dan Y . (Konsep ini dibahas secara lebih formal di Bagian 7.8, meskipun kami tidak mengandalkan bagian tersebut di sini.)
Misalnya, Encaps$^{(1)}_{pk}(1^n)$ (resp., Encaps$^{(2)}_{pk}(1^n)$) menunjukkan keluaran ciphertext (resp., key) oleh Encaps. Fakta bahwa $\prod$ aman terhadap CPA berarti demikian

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Encaps}^{(2)}_{pk}(1^n)\right) \overset{c}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), k'\right),$$

dimana pk dihasilkan oleh Gen(1^n) dan k' dipilih secara independen dan seragam dari $\{0,1\}^n$. Demikian pula, fakta bahwa $\prod'$ memiliki enkripsi yang tidak dapat dibedakan dengan adanya penyadap berarti bahwa untuk setiap m_0, m_1 keluaran oleh $\mathcal{A}$ kita memiliki Enc$'_k(m_0) \overset{c}{\equiv}$ Enc$'_k(m_1)$ jika k dipilih secara acak dan seragam.
Untuk membuktikan keamanan CPA pada $\prod^{hy}$ kita perlu menunjukkannya

$$\left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0)\right) \overset{c}{\equiv} \left(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_1)\right) \quad (11.10)$$



GAMBAR 11.3: Struktur tingkat tinggi dari bukti Teorema 11.12 (tanda panah menunjukkan tidak dapat dibedakan).

untuk m_0, m_1 keluaran oleh musuh PPT $\mathcal{A}$. (Persamaan (11.10) cukup untuk menunjukkan bahwa $\prod^{hy}$ memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap, dan berdasarkan Proposisi 11.3 ini menyiratkan bahwa $\prod^{hy}$ aman terhadap CPA.) Pembuktiannya berlangsung dalam tiga langkah. (Lihat Gambar 11.3.) Pertama kita buktikan

$$\left(pk, \mathsf{Encaps}_{pk}^{(1)}(1^n), \mathsf{Enc}'_k(m_0)\right) \stackrel{c}{\equiv} \left(pk, \mathsf{Encaps}_{pk}^{(1)}(1^n), \mathsf{Enc}'_{k'}(m_0)\right), \quad (11.11)$$

dimana di sebelah kiri k dihasilkan oleh Encaps$_{pk}^{(2)}(1^n)$, dan di sebelah kanan k' adalah kunci yang independen dan seragam. Hal ini mengikuti pengurangan yang cukup jelas, karena keamanan CPA $\prod$ berarti bahwa Encaps$_{pk}^{(2)}(1^n)$ tidak dapat dibedakan dari kunci seragam k' bahkan jika diberikan pk dan Encaps$_{pk}^{(1)}(1^n)$. Selanjutnya kita buktikan hal tersebut

$$\left(pk, \mathsf{Encaps}_{pk}^{(1)}(1^n), \mathsf{Enc}'_{k'}(m_0)\right) \stackrel{c}{\equiv} \left(pk, \mathsf{Encaps}_{pk}^{(1)}(1^n), \mathsf{Enc}'_{k'}(m_1)\right). \quad (11.12)$$

Di sini perbedaannya adalah antara mengenkripsi m_0 atau m_1 menggunakan $\prod'$ dan kunci independen yang seragam, k'. Persamaan (11.12) selanjutnya menggunakan fakta bahwa $\prod'$ memiliki enkripsi yang tidak dapat dibedakan dengan adanya penyadap.
Persis seperti pada kasus Persamaan (11.11), kita juga dapat menunjukkan hal tersebut

$$\left(pk, \mathsf{Encaps}_{pk}^{(1)}(1^n), \mathsf{Enc}'_k(m_1)\right) \stackrel{c}{\equiv} \left(pk, \mathsf{Encaps}_{pk}^{(1)}(1^n), \mathsf{Enc}'_{k'}(m_1)\right), \quad (11.13)$$

dengan mengandalkan kembali keamanan CPA $\prod$. Persamaan (11.11)–(11.13) menyiratkan, berdasarkan transitivitas, hasil yang diinginkan dari Persamaan (11.10). (Transitivitas akan tersirat dalam bukti yang kami berikan di bawah.)
Kini kami hadirkan bukti lengkapnya.

BUKTI (Teorema 11.12) Kami membuktikan bahwa $\prod^{hy}$ mempunyai enkripsi yang tidak dapat dibedakan jika ada penyadap; berdasarkan Proposisi 11.3, ini berarti aman untuk CPA.

Perbaiki musuh ppt sewenang-wenang $\mathcal{A}^{hy}$, dan pertimbangkan eksperimen Pub$K^{eav}_{\mathcal{A}^{hy},\prod^{hy}}(n)$. Tujuan kami adalah untuk membuktikan bahwa ada fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}^{\mathsf{hy}},\Pi^{\mathsf{hy}}}(n)=1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Berdasarkan definisi percobaan, kita punya

$$\begin{aligned}&\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A}^{\mathsf{hy}},\Pi^{\mathsf{hy}}}(n)=1] \qquad (11.14)\\ &= \frac{1}{2}\cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0))=0]\\ &\quad + \frac{1}{2}\cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_1))=1],\end{aligned}$$

dimana dalam setiap kasus k sama dengan $\mathsf{Encaps}^{(2)}_{pk}(1^n)$. Perhatikan musuh ppt berikut $\mathcal{A}_1$ yang menyerang $\prod$.

Musuh $\mathcal{A}_1$:

1. Diberikan $\mathcal{A}_1$ $(pk, c, \hat{k})$.
2. $\mathcal{A}_1$ menjalankan $\mathcal{A}^{hy}(pk)$ untuk mendapatkan dua pesan m_0, m_1. Kemudian $\mathcal{A}_1$ menghitung $c' \leftarrow Enc'_{\hat{k}}(m_0)$, memberikan ciphertext $\langle c, c' \rangle$ ke $\mathcal{A}^{hy}$, dan mengeluarkan bit b' yang dihasilkan $\mathcal{A}^{hy}$.

Perhatikan perilaku $\mathcal{A}_1$ saat menyerang $\prod$ dalam percobaan $KEM^{cpa}_{\mathcal{A}_1,\prod}(n)$.
Jika $b = 0$ pada percobaan tersebut, maka $\mathcal{A}_1$ diberikan $(pk, c, \hat{k})$ dimana c dan $\hat{k}$ keduanya merupakan output dari $\mathsf{Encaps}_{pk}(1^n)$. Artinya $\mathcal{A}^{hy}$ diberikan ciphertext dengan bentuk $\langle c, c' \rangle = \langle c, Enc'_k(m_0) \rangle$, dimana k adalah kunci yang dienkapsulasi oleh c. Jadi,

$$\Pr[\mathcal{A}_1 \text{ outputs } 0 \mid b=0] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0))=0].$$

Sebaliknya, jika $b = 1$ pada percobaan $KEM^{cpa}_{\mathcal{A}_1,\prod}(n)$ maka $\mathcal{A}_1$ diberikan $(pk, c, \hat{k})$ dengan $\hat{k}$ seragam dan tidak bergantung pada c. Jika kunci tersebut dilambangkan dengan k', ini berarti $\mathcal{A}^{hy}$ diberikan teks sandi berbentuk $\langle c, Enc'_k(m_0)$, dan

$$\Pr[\mathcal{A}_1 \text{ outputs } 1 \mid b=1] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0))=1].$$

Karena $\prod$ adalah KEM aman-CPA, terdapat fungsi negI_1 yang dapat diabaikan sehingga

$$\begin{aligned}\frac{1}{2} + \mathsf{negl}_1(n) &\geq \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A}_1,\Pi}(n) = 1] \qquad (11.15)\\ &= \frac{1}{2} \cdot \Pr[\mathcal{A}_1 \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}_1 \text{ outputs } 1 \mid b = 1]\\ &= \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_k(m_0)) = 0]\\ &\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)) = 1]\end{aligned}$$

dimana k sama dengan $\mathsf{Encaps}^{(2)}_{pk}(1^n)$ dan k' adalah kunci yang seragam dan independen. Selanjutnya, pertimbangkan musuh ppt berikut $\mathcal{A}'$ yang menguping pesan yang dienkripsi menggunakan skema kunci pribadi $\prod'$.

Musuh $\mathcal{A}'$:

1. $\mathcal{A}(1^n)$ menjalankan $\mathsf{Gen}(1^n)$ sendiri untuk menghasilkan kunci (pk, sk). Ini juga menghitung $c \leftarrow \mathsf{Encaps}^{(1)}_{pk}(1^n)$.
2. A′ menjalankan $\mathcal{A}^{hy}(pk)$ untuk mendapatkan dua pesan m_0, m_1. Ini adalah output dari $\mathcal{A}'$, dan sebagai balasannya diberikan ciphertext c'.
3. $\mathcal{A}'$ memberikan ciphertext $\langle c, c' \rangle$ kepada $\mathcal{A}^{hy}$, dan mengeluarkan bit b' yang dihasilkan $\mathcal{A}^{hy}$.

Ketika $b = 0$ dalam percobaan $\mathsf{Priv}K^{eav}_{\mathcal{A}',\prod'}(n)$, musuh $\mathcal{A}'$ diberikan teks sandi c' yang merupakan enkripsi m_0 menggunakan kunci k' yang seragam dan tidak bergantung pada apa pun. Jadi $\mathcal{A}^{hy}$ diberikan ciphertext berbentuk $\langle c, Enc'_{k'}(m_1) \rangle$ dimana k' seragam dan tidak bergantung pada c, dan

$$\Pr[\mathcal{A}' \text{ outputs } 0 \mid b = 0] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)) = 0].$$

sebaliknya, ketika $b = 1$ dalam percobaan $\mathsf{Priv}K^{eav}_{\mathcal{A}',\prod'}(n)$, maka $\mathcal{A}'$ diberikan enkripsi m_1 menggunakan kunci independen dan seragam k'. Artinya $\mathcal{A}^{hy}$ diberikan ciphertext berbentuk $\langle c, Enc'_{k'}(m_1) \rangle$ dan sebagainya

$$\Pr[\mathcal{A}' \text{ outputs } 1 \mid b = 1] = \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)) = 1].$$

Karena $\prod'$ memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap, terdapat fungsi yang dapat diabaikan sehingga dapat diabaikan

$$\begin{aligned}\frac{1}{2} + \mathsf{negl}'(n) &\geq \Pr[\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A}',\Pi'}(n) = 1] \qquad (11.16)\\ &= \frac{1}{2} \cdot \Pr[\mathcal{A}' \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}' \text{ outputs } 1 \mid b = 1]\\ &= \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_0)) = 0]\\ &\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)) = 1].\end{aligned}$$

Dengan melanjutkan persis seperti yang kita lakukan untuk membuktikan Persamaan (11.15), kita dapat menunjukkan bahwa terdapat fungsi negI_2 yang dapat diabaikan sehingga

$$\begin{aligned}\frac{1}{2} + \mathsf{negl}_2(n) &\geq \Pr[\mathsf{KEM}^{\mathsf{cpa}}_{\mathcal{A}_2,\Pi}(n) = 1] \qquad (11.17)\\ &= \frac{1}{2} \cdot \Pr[\mathcal{A}_2 \text{ outputs } 0 \mid b = 0] + \frac{1}{2} \cdot \Pr[\mathcal{A}_2 \text{ outputs } 1 \mid b = 1]\\ &= \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k}(m_1)) = 1]\\ &\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}^{\mathsf{hy}}(pk, \mathsf{Encaps}^{(1)}_{pk}(1^n), \mathsf{Enc}'_{k'}(m_1)) = 0].\end{aligned}$$

Menjumlahkan Persamaan (11.15)–(11.17) dan menggunakan fakta bahwa jumlah tiga fungsi yang dapat diabaikan dapat diabaikan, kita melihat terdapat fungsi yang dapat diabaikan sehingga

$$\begin{aligned}&\frac{3}{2} + \mathsf{negl}(n) \geq\\ &\frac{1}{2} \cdot \Big(\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k}(m_0)) = 0] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 1]\\ &\quad + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 0] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 1]\\ &\quad + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k}(m_1)) = 1] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 0] \Big),\end{aligned}$$

di sini $c = \mathsf{Encaps}^{(1)}_{pk}(1^n)$ semua hal di atas. Perhatikan itu

$$\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 1] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_0)) = 0] = 1,$$

karena peluang kejadian-kejadian yang saling melengkapi selalu berjumlah 1. Demikian pula,

$$\Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 1] + \Pr[\mathcal{A}^{\mathsf{hy}}(pk, c, \mathsf{Enc}'_{k'}(m_1)) = 0] = 1.$$

Karena itu,

$$\begin{aligned}&\frac{1}{2}+\mathrm{negl}(n)\\&\geq\frac{1}{2}\cdot\left(\Pr[\mathcal{A}^{\mathrm{hy}}(pk,c,\mathrm{Enc}'_k(m_0))=0]+\Pr[\mathcal{A}^{\mathrm{hy}}(pk,c,\mathrm{Enc}'_k(m_1))=1]\right)\\&=\Pr[\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A}^{\mathrm{hy}},\Pi^{\mathrm{hy}}}(n)=1]\end{aligned}$$

(menggunakan Persamaan (11.14) untuk persamaan terakhir), membuktikan teorema.

CCA-Keamanan

Jika skema enkripsi kunci privat $\prod'$ tidak aman terhadap serangan teks sandi yang dipilih, maka (terlepas dari KEM yang digunakan) skema enkripsi hibrid yang dihasilkan juga tidak $\prod^{hy}$. Sebagai contoh sederhana dan ilustratif, katakanlah kita menggunakan Konstruksi 3.17 sebagai skema enkripsi kunci pribadi. Kemudian, membiarkan KEM tidak ditentukan, enkripsi pesan m oleh $\prod^{hy}$ dilakukan dengan menghitung $(c,k) \longleftarrow \mathrm{Encaps}_{pk}(1^n)$ dan kemudian mengeluarkan teks sandi

$$\langle c, G(k)\oplus m\rangle,$$

dimana G adalah generator pseudorandom. Dengan adanya ciphertext $\langle c,c'\rangle$, penyerang dapat dengan mudah membalik bit terakhir dari c' untuk mendapatkan ciphertext yang dimodifikasi yang merupakan enkripsi valid dari m dengan bit terakhirnya dibalik.

Cara alami untuk memperbaikinya adalah dengan menggunakan skema enkripsi kunci pribadi yang aman dengan CCA. Namun hal ini jelas tidak cukup jika KEM rentan terhadap serangan teks sandi terpilih. Karena kami belum mendefinisikan gagasan ini, kami mendefinisikannya sekarang.

Seperti dalam Definisi 11.11, kita mensyaratkan bahwa musuh yang diberi teks sandi c tidak dapat membedakan kunci k yang dirangkum oleh teks sandi tersebut dari kunci k' yang seragam dan independen. Namun sekarang, kami juga mengizinkan penyerang untuk meminta dekapsulasi ciphertext pilihannya (selama ciphertext tersebut berbeda dari tantangan ciphertext).

Secara formal, misalkan $\prod$ = (Gen, Encaps, Decaps) adalah KEM dengan panjang kunci n dan $\mathcal{A}$ sebagai musuh, dan pertimbangkan eksperimen berikut:

Eksperimen ketidakmampuan CCA untuk membedakan KE$M^{cca}_{\mathcal{A},\prod}(n)$:

1. Gen(1^n) dijalankan untuk mendapatkan kunci (pk, sk). Kemudian Encaps$_{pk}(1^n)$ dijalankan untuk menghasilkan (c,k) dengan $k \in \{0,1\}^n$.
2. Bit seragam $b \in \{0,1\}$ dipilih. Jika $b = 0$ himpunan $\hat{k} := k$. Jika $b = 1$ maka pilihlah seragam $\hat{k} \in \{0,1\}^n$.
3. $\mathcal{A}$ diberikan $(pk, c, \hat{k})$ dan akses ke oracle Decaps$_{sk}(\cdot)$, tetapi tidak boleh meminta dekapsulasi c itu sendiri.

4. $\mathcal{A}$ mengeluarkan sedikit b'. Keluaran percobaan didefinisikan sebagai 1 jika $b' = b$, dan 0 jika tidak.

DEFINISI 11.13 Mekanisme enkapsulasi kunci $\prod$ aman terhadap CCA jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{KEM}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Untungnya, kami dapat menunjukkan bahwa menggunakan KEM yang aman dengan CCA yang dikombinasikan dengan skema enkripsi kunci privat yang aman dengan CCA akan menghasilkan skema enkripsi kunci publik yang aman terhadap serangan teks sandi yang dipilih.

TEOREMA 11.14 Jika $\prod$ adalah KEM yang aman-CCA dan $\prod'$ adalah skema enkripsi kunci pribadi yang aman-CCA, maka $\prod^{hy}$ seperti pada Konstruksi 11.10 adalah skema enkripsi kunci publik yang aman-CCA. Bukti diperoleh dengan modifikasi yang sesuai terhadap bukti Teorema 11.12.

11.4 ENKRIPSI BERBASIS CDH/DDH

Sejauh ini kita telah membahas enkripsi kunci publik secara abstrak, namun belum melihat contoh nyata skema enkripsi kunci publik (atau KEM). Di sini kita mengeksplorasi beberapa konstruksi berdasarkan masalah Diffie–Hellman.

Enkripsi El Gamal

Pada tahun 1985, Taher El Gamal mengamati bahwa protokol pertukaran kunci Diffie–Hellman (lih. Bagian 10.3) dapat diadaptasi untuk menghasilkan skema enkripsi kunci publik. Ingatlah bahwa dalam protokol Diffie–Hellman, Alice mengirimkan pesan ke Bob dan kemudian Bob membalas dengan pesan ke Alice; berdasarkan pesan-pesan ini, Alice dan Bob dapat memperoleh nilai bersama k yang tidak dapat dibedakan (bagi penyadap) dari elemen seragam dari beberapa grup $\mathbb{G}$. Kita dapat membayangkan Bob menggunakan nilai bersama tersebut untuk mengenkripsi pesan $m \in \mathbb{G}$ hanya dengan mengirimkan $k \cdot m$ ke Alice; Alice dapat dengan jelas memulihkan m menggunakan pengetahuannya tentang k, dan di bawah ini kita akan berargumentasi bahwa seorang penyadap tidak mengetahui apa pun tentang m.

Dalam *skema enkripsi El Gamal* kita cukup mengubah perspektif kita terhadap interaksi di atas. Kita memandang pesan awal Alice sebagai kunci publiknya, dan balasan Bob (baik respons awalnya maupun $k \cdot m$) sebagai teks tersandi. Keamanan CPA berdasarkan asumsi keputusan Diffie–Hellman (DDH) mengikuti dengan cukup mudah keamanan protokol pertukaran kunci Diffie–Hellman (Teorema 10.3).

Dalam pembahasan formal kami, kami mulai dengan menyatakan dan membuktikan lemma sederhana yang mendasari skema enkripsi El Gamal. Misalkan $\mathbb{G}$ adalah grup berhingga, dan misalkan $m \in \mathbb{G}$ adalah elemen sembarang. Lemma menyatakan bahwa

mengalikan m dengan unsur golongan seragam k menghasilkan unsur golongan k' yang terdistribusi seragam. Yang penting, distribusi k' tidak bergantung pada m; ini berarti k' tidak mengandung informasi tentang m.

LEMMA 11.15 Misalkan $\mathbb{G}$ adalah grup berhingga, dan misalkan $m \in \mathbb{G}$ bersifat sembarang. Kemudian memilih seragam $k \in \mathbb{G}$ dan mengatur $k' := k \cdot m$ memberikan distribusi yang sama untuk k' dengan memilih seragam $k' \in \mathbb{G}$. Dengan kata lain, untuk setiap $\hat{g} \in \mathbb{G}$ yang kita punya

$$\Pr[k \cdot m = \hat{g}] = 1/|\mathbb{G}|,$$

dimana probabilitasnya diambil alih pilihan seragam $k \in \mathbb{G}$.

BUKTI Misalkan $\hat{g} \in \mathbb{G}$ bersifat sembarang. Kemudian

$$\Pr[k \cdot m = \hat{g}] = \Pr[k = \hat{g} \cdot m^{-1}].$$

Karena k seragam, peluang k sama dengan elemen tetap $\hat{g} \cdot m^{-1}$ adalah tepat 1/| $\mathbb{G}$ |.

Lemma di atas menyarankan cara untuk membangun skema enkripsi *kunci privat* yang sangat rahasia dengan ruang pesan $\mathbb{G}$. Pengirim dan penerima berbagi elemen seragam $k \in \mathbb{G}$ sebagai kunci rahasia mereka. Untuk mengenkripsi pesan $m \in \mathbb{G}$, pengirim menghitung ciphertext $k' := k \cdot m$. Penerima dapat memulihkan pesan dari ciphertext k' dengan menghitung $m := k'/k$. Kerahasiaan yang sempurna muncul dari lemma di atas. Faktanya, kita telah melihat skema ini dalam bentuk yang berbeda—skema enkripsi one-time pad adalah contoh dari pendekatan ini, dengan grup yang mendasarinya adalah himpunan string dengan panjang tetap di bawah operasi XOR bit-wise.

Kita dapat mengadaptasi ide-ide di atas ke dalam pengaturan kunci publik dengan menyediakan cara bagi para pihak untuk menghasilkan nilai bersama yang “tampak acak” dengan berinteraksi melalui saluran publik. Ini seharusnya terdengar familier karena itulah yang disediakan oleh protokol Diffie–Hellman. Kami melanjutkan dengan detailnya.

Seperti pada Bagian 8.3, misalkan algoritma waktu polinomial yang mengambil input 1^n dan (kecuali mungkin dengan probabilitas yang dapat diabaikan) menghasilkan deskripsi grup siklik $\mathbb{G}$, ordenya q (dengan $\|q\| = n$), dan a pembangkit g. Skema enkripsi El Gamal dijelaskan dalam Konstruksi 11.16.

CONSTRUCTION 11.16

Let $\mathcal{G}$ be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. Then choose a uniform $x \in \mathbb{Z}_q$ and compute $h := g^x$. The public key is $\langle \mathbb{G}, q, g, h \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$. The message space is $\mathbb{G}$.
- Enc: on input a public key $pk = \langle \mathbb{G}, q, g, h \rangle$ and a message $m \in \mathbb{G}$, choose a uniform $y \in \mathbb{Z}_q$ and output the ciphertext

$$\langle g^y,\ h^y \cdot m \rangle.$$

- Dec: on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $\langle c_1, c_2 \rangle$, output

$$\hat{m} := c_2 / c_1^x.$$

Skema enkripsi El Gamal.

Agar dekripsi berhasil, misalkan $\langle c_1, c_2 \rangle = \langle g^y, h^y \cdot m \rangle$ dengan $h = g^x$. Kemudian

$$\hat{m} = \frac{c_2}{c_1^x} = \frac{h^y \cdot m}{(g^y)^x} = \frac{(g^x)^y \cdot m}{g^{xy}} = \frac{g^{xy} \cdot m}{g^{xy}} = m.$$

Contoh 11.17

Misalkan $q = 83$ dan $p = 2q + 1 = 167$, dan misalkan $\mathbb{G}$ menyatakan kelompok residu kuadrat (yaitu persegi) modulo p. (Karena p dan q adalah bilangan prima, $\mathbb{G}$ adalah subgrup dari $\mathbb{Z}_p^*$ dengan orde q. Lihat Bagian 8.3.) Karena orde $\mathbb{G}$ adalah bilangan prima, setiap elemen $\mathbb{G}$ kecuali 1 adalah generator; ambil $g = 2^2 = 4 \bmod 167$. Katakanlah penerima memilih kunci rahasia $37 \in \mathbb{Z}_{83}$ sehingga kunci publiknya adalah

$$pk = \langle p, q, g, h \rangle = \langle 167, 83, 4, [4^{37} \bmod 167] \rangle = \langle 167, 83, 4, 76 \rangle,$$

di mana kita menggunakan p untuk mewakili $\mathbb{G}$ (diasumsikan bahwa penerima mengetahui bahwa grup tersebut adalah himpunan residu kuadrat modulo p).

Katakanlah pengirim mengenkripsi pesan $m = 65 \in \mathbb{G}$ (catatan $65 = 30^2 \bmod 167$ dan 65 adalah elemen dalam subgrup). Jika $y = 71$, maka cipherteksnya adalah

$$\langle [4^{71} \bmod 167],\ [76^{71} \cdot 65 \bmod 167] \rangle = \langle 132, 44 \rangle.$$

Untuk mendekripsi, penerima terlebih dahulu menghitung $124 = [132^{37} mod\ 167]$; kemudian, karena $66 = [124^{-1} \bmod 167]$, penerima memulihkan $m = 65 = [44 \cdot 66 \bmod 167]$.

Kami sekarang membuktikan keamanan skema tersebut. (Pembaca mungkin ingin membandingkan bukti berikut dengan bukti Teorema 3.18 dan 10.3.)

TEOREMA 11.18 Jika permasalahan DDH relatif sulit, maka skema enkripsi El Gamal aman terhadap CPA.

BUKTI Misalkan Π menunjukkan skema enkripsi El Gamal. Kami membuktikan bahwa Π memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap; berdasarkan Proposisi 11.3, ini berarti aman untuk CPA.

Biarkan $\mathcal{A}$ menjadi musuh waktu polinomial yang probabilistik. Kami ingin menunjukkan bahwa ada fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n).$$

Pertimbangkan "skema enkripsi" yang dimodifikasi $\widehat{\Pi}$ di mana Gen sama dengan Π, tetapi enkripsi pesan m sehubungan dengan kunci publik $\langle \mathbb{G}, q, g, h \rangle$ dilakukan dengan memilih seragam $y, z \in \mathbb{Z}_q$ dan menghasilkan keluaran teks tersandi tersebut

$$\langle g^y,\ g^z \cdot m \rangle.$$

Meskipun $\widehat{\Pi}$ sebenarnya bukan skema enkripsi (karena tidak ada cara bagi penerima untuk mendekripsi), eksperimen $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\widehat{\Pi}}(n)$ masih terdefinisi dengan baik karena eksperimen tersebut hanya bergantung pada pembuatan kunci dan algoritme enkripsi.

Lemma 11.15 dan pembahasan selanjutnya mengimplikasikan bahwa komponen kedua dari ciphertext dalam skema $\widehat{\Pi}$ adalah elemen grup yang terdistribusi secara seragam dan, khususnya, tidak bergantung pada pesan m yang dienkripsi. (Ingat bahwa g^z adalah elemen seragam dari $\mathbb{G}$ ketika z dipilih secara seragam dari $\mathbb{Z}_q$.) Komponen pertama dari ciphertext tidak bergantung pada m. Secara keseluruhan, ini berarti bahwa seluruh ciphertext tidak mengandung informasi tentang m. Oleh karena itu

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\widetilde{\Pi}}(n) = 1] = \frac{1}{2}.$$

Sekarang perhatikan algoritma PPT D berikut yang mencoba menyelesaikan masalah DDH relatif terhadap. Ingat bahwa $\mathcal{G}$ menerima $(\mathbb{G}, q, g, h_1, h_2, h_3)$ dimana $h_1 = g^x, h_2 = g^y$, dan h_3 adalah g^{xy} atau g^z (untuk seragam x, y, z); tujuan D adalah untuk menentukan kasusnya.

Algoritma D:

Algoritma diberikan $(\mathbb{G}, q, g, h_1, h_2, h_3)$ sebagai masukan.

- Tetapkan $pk = \langle \mathbb{G}, q, g, h_1 \rangle$ dan jalankan $\mathcal{A}(pk)$ untuk mendapatkan dua pesan $m_0, m_1 \in \mathbb{G}$.

- Pilih bit b yang seragam, lalu atur $c_1 := h_2$ dan $c_2 := h_3 \cdot m_b$
- Berikan ciphertext $\langle c_1, c_2 \rangle$ ke $\mathcal{A}$ dan dapatkan bit keluaran b'. Jika $b' = b$, keluaran 1; jika tidak, keluaran 0.

Mari kita analisa perilaku D. Ada dua kasus yang perlu dipertimbangkan:

Kasus 1: Katakanlah input ke D dihasilkan dengan menjalankan $\mathcal{G}(1^n)$ untuk mendapatkan $(\mathbb{G}, q, g)$, lalu memilih seragam $x, y, z \in \mathbb{Z}_q$, dan terakhir menyetel $h_1 := g^x, h_2 := g^y$, dan $h_3 := g^z$. Kemudian D menjalankan $\mathcal{A}$ pada kunci publik yang dikonstruksikan sebagai:

$$pk = \langle \mathbb{G}, q, g, g^x \rangle$$

dan ciphertext dikonstruksikan sebagai:

$$\langle c_1, c_2 \rangle = g^y, g^z \cdot m_b$$

Kita melihat bahwa dalam kasus ini tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)$. Karena D mengeluarkan 1 tepat ketika keluaran b' dari $\mathcal{A}$ sama dengan b, kita mendapatkan bahwa

$$\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] = \Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1\right] = \frac{1}{2}$$

Kasus 2: Katakanlah input ke D dihasilkan dengan menjalankan $\mathcal{G}(1^n)$ untuk mendapatkan $(\mathbb{G}, q, g)$, lalu memilih seragam $x, y, \in \mathbb{Z}_q$ dan terakhir menyetel $h_1 := g^x, h_2 := g^y$, dan $h_3 := g^{xy}$. Kemudian D menjalankan $\mathcal{A}$ pada kunci publik yang dikonstruksikan sebagai

$$pk = \langle \mathbb{G}, q, g, g^x \rangle$$

dan ciphertext dikonstruksikan sebagai

$$\langle c_1, c_2 \rangle = \langle g^y, g^z \cdot m_b \rangle = \langle g^y, (g^x)^y \cdot m_b \rangle$$

Kita melihat bahwa dalam kasus ini tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)$. Karena D menghasilkan 1 tepat ketika keluaran b' dari $\mathcal{A}$ sama dengan b, kita mendapatkan bahwa

$$\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1] = \Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1\right]$$

Dengan asumsi bahwa masalah DDH relatif sulit, terdapat fungsi yang dapat diabaikan sehingga

$$\begin{aligned}\mathrm{negl}(n) &\geq \left|\Pr[D(\mathbb{G}, q, g, g^x, g^y, g^z) = 1] - \Pr[D(\mathbb{G}, q, g, g^x, g^y, g^{xy}) = 1]\right| \\ &= \left|\frac{1}{2} - \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1]\right|.\end{aligned}$$

Ini menyiratkan

$$\Pr\left[\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$$

melengkapi buktinya.

Masalah Implementasi El Gamal

Kami secara singkat membahas beberapa masalah praktis terkait enkripsi El Gamal.

Berbagi parameter publik. Deskripsi kami tentang skema enkripsi El Gamal di Konstruksi 11.16 mengharuskan penerima dijalankan $\mathcal{G}$ untuk menghasilkan $\mathbb{G}, q, g$. Dalam praktiknya, parameter-parameter ini biasanya dibuat dan diperbaiki "sekali dan untuk selamanya", dan kemudian dibagikan kepada banyak penerima. (Tentu saja, setiap penerima harus memilih nilai rahasianya sendiri x dan mempublikasikan kunci publiknya sendiri $h = g^x$.) Misalnya, NIST telah menerbitkan serangkaian parameter yang direkomendasikan yang cocok untuk digunakan dalam skema enkripsi El Gamal. Berbagi parameter dengan cara ini tidak berdampak pada keamanan (dengan asumsi parameter dibuat dengan benar dan jujur). Ke depan, kami mencatat bahwa hal ini berbeda dengan kasus RSA, di mana parameter tidak dapat dibagikan dengan aman (lihat Bagian 11.5).

Pilihan kelompok. Sebagaimana dibahas dalam Bagian 8.3, orde grup q umumnya dipilih sebagai bilangan prima. Bagi kelompok tertentu, kurva elips adalah salah satu pilihan yang semakin populer; alternatifnya adalah dengan membiarkan $\mathbb{G}$ menjadi subgrup orde prima dari $\mathbb{Z}_p^*$, untuk p prima. Kami mengacu pada Bagian 9.3 untuk tabulasi panjang kunci yang direkomendasikan untuk mencapai tingkat keamanan yang berbeda.

Ruang pesan. Aspek yang tidak menyenangkan dari skema enkripsi El Gamal adalah bahwa ruang pesan adalah grup $\mathbb{G}$ dan bukan bit-string dengan panjang tertentu. Untuk beberapa pilihan grup, hal ini dapat diatasi dengan mendefinisikan pengkodean bit-string yang dapat dibalik sebagai elemen grup. Dalam kasus seperti ini, pengirim dapat mengkodekan pesannya terlebih dahulu $m \in \{0,1\}^\ell$ sebagai elemen grup $\hat{m} \in \mathbb{G}$ dan kemudian menerapkan enkripsi El Gamal ke $\hat{m}$. Penerima dapat mendekripsi seperti pada Konstruksi 11.16 untuk mendapatkan pesan yang dikodekan $\hat{m}$, dan kemudian membalik pengkodean untuk memulihkan pesan asli m.

Pendekatan yang lebih sederhana adalah dengan menggunakan (varian dari) enkripsi El Gamal sebagai bagian dari skema enkripsi hibrid. Misalnya, pengirim dapat memilih elemen grup seragam $m \in \mathbb{G}$, mengenkripsinya menggunakan skema enkripsi El Gamal, dan kemudian mengenkripsi pesan sebenarnya menggunakan skema enkripsi kunci pribadi dan kunci $H(m)$, di mana $H: \mathbb{G} \to \{0,1\}^n$ adalah fungsi penurunan kunci yang sesuai (lihat bagian berikut). Dalam hal ini, akan lebih efisien jika menggunakan KEM berbasis DDH yang akan kami jelaskan selanjutnya.

Enkapsulasi Kunci Berbasis DDH

Pada akhir bagian sebelumnya kami mencatat bahwa enkripsi El Gamal dapat digunakan sebagai bagian dari skema enkripsi hibrid hanya dengan mengenkripsi elemen grup seragam m dan menggunakan hash dari elemen tersebut sebagai kunci. Tapi ini sia-sia! Bukti keamanan enkripsi El Gamal menunjukkan bahwa c_1^x(dimana c_1 adalah komponen pertama dari ciphertext, dan x adalah kunci privat penerima) sudah tidak dapat dibedakan dari elemen grup yang seragam, sehingga pengirim/penerima sebaiknya juga menggunakan itu. Konstruksi 11.19 menggambarkan KEM yang mengikuti pendekatan ini. Perhatikan bahwa enkapsulasi yang dihasilkan hanya terdiri dari satu elemen grup. Sebaliknya, jika kita menggunakan enkripsi El Gamal pada elemen grup yang seragam, ciphertext akan berisi dua elemen grup.

CONSTRUCTION 11.19

Let $\mathcal{G}$ be as in the previous section. Define a KEM as follows:

- **Gen:** on input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. choose a uniform $x \in \mathbb{Z}_q$ and set $h := g^x$. Also specify a function $H : \mathbb{G} \to \{0,1\}^{\ell(n)}$ for some function ℓ (see text). The public key is $\langle \mathbb{G}, q, g, h, H \rangle$ and the private key is $\langle \mathbb{G}, q, g, x \rangle$.
- **Encaps:** on input a public key $pk = \langle \mathbb{G}, q, g, h, H \rangle$ choose a uniform $y \in \mathbb{Z}_q$ and output the ciphertext g^y and the key $H(h^y)$.
- **Decaps:** on input a private key $sk = \langle \mathbb{G}, q, g, x \rangle$ and a ciphertext $c \in \mathbb{G}$, output the key $H(c^x)$.

Seperti dijelaskan, konstruksinya membiarkan fungsi derivasi kunci H tidak ditentukan, dan ada beberapa opsi untuk itu. (Lihat Bagian 5.6 untuk informasi lebih lanjut mengenai derivasi kunci secara umum.) Salah satu kemungkinannya adalah dengan memilih fungsi $H: \mathbb{G} \to \{0,1\}^n$ yang (mendekati) regular, artinya untuk setiap kemungkinan kunci $k \in \{0,1\}^\ell$adalah bilangan elemen grup yang dipetakan ke k kira-kira sama. (Secara formal, kita memerlukan fungsi yang dapat diabaikan sehingga untuk setiap $k \in \{0,1\}^\ell$.

$$2^\ell \cdot \left|\Pr[H(g) = k] - 2^{-\ell}\right| \leq \text{negl}(n)$$

dimana probabilitas diambil alih pilihan seragam $g \in \mathbb{G}$. Hal ini memastikan bahwa distribusi pada kunci k secara statistik mendekati seragam.) Kompleksitas H, serta panjang kunci yang dapat dicapai ℓ, akan bergantung pada grup spesifik $\mathbb{G}$ sedang digunakan.

Kemungkinan kedua adalah membiarkan H menjadi fungsi berkunci, dimana kunci (seragam) untuk H dimasukkan sebagai bagian dari kunci publik penerima. Ini berfungsi jika H adalah ekstraktor yang kuat, seperti yang disebutkan secara singkat di Bagian 5.6. Pilihan ℓ yang tepat di sini (untuk memastikan bahwa kunci yang dihasilkan mendekati seragam secara statistik) akan bergantung pada ukuran $\mathbb{G}$.

Dalam salah satu kasus di atas, bukti keamanan CPA berdasarkan asumsi keputusan Diffie–Hellman (DDH) dapat diikuti dengan mudah dengan mengadaptasi bukti keamanan untuk protokol pertukaran kunci Diffie–Hellman (Teorema 10.3).

TEOREMA 11.20 Jika permasalahan DDH relatif sulit, dan H dipilih seperti yang dijelaskan, maka Konstruksi 11.19 adalah KEM aman-CPA.

Jika seseorang ingin memodelkan H sebagai ramalan acak, maka Konstruksi 11.19 dapat dibuktikan aman terhadap CPA berdasarkan asumsi komputasi Diffie–Hellman (CDH) (yang lebih lemah). Kami membahas ini di bagian berikut.

***KEM Berbasis CDH dalam Model Random-Oracle**

Pada bagian ini, kami menunjukkan bahwa jika seseorang ingin memodelkan H sebagai oracle acak, maka Konstruksi 11.19 dapat dibuktikan aman terhadap CPA berdasarkan asumsi CDH. (Pembaca mungkin ingin meninjau Bagian 5.5 untuk mengingatkan diri mereka tentang model oracle acak.) Secara intuitif, asumsi CDH menyiratkan bahwa penyerang yang mengamati $h = g^x$ (dari kunci publik) dan ciphertext $c = g^y$ tidak dapat menghitung $\mathrm{DH}_g(h, c) = h^y$. Secara khusus, penyerang tidak dapat menanyakan h^y ke oracle acak. Namun ini berarti kunci $H(h^y)$ yang dienkapsulasi benar-benar acak dari sudut pandang penyerang. Intuisi ini diubah menjadi bukti formal di bawah ini.

Seperti yang ditunjukkan oleh intuisi di atas, pembuktian secara inheren bergantung pada pemodelan H sebagai ramalan acak. Secara khusus, pembuktian bergantung pada fakta bahwa (1) satu-satunya cara untuk mempelajari $H(h^y)$ adalah dengan secara eksplisit menanyakan h^y ke H, yang mana berarti penyerang telah menyelesaikan instance CDH (ini disebut "ekstraksi" di Bagian 5.5), dan (2) jika penyerang tidak menanyakan h^y ke H, maka nilai $H(h^y)$ seragam dari milik penyerang sudut pandang. Properti ini hanya berlaku—bahkan masuk akal—jika H dimodelkan sebagai oracle acak.

TEOREMA 11.21 Jika permasalahan CDH relatif sulit terhadap $\mathcal{G}$, dan H dimodelkan sebagai ramalan acak, maka Konstruksi 11.19 aman untuk CPA.

BUKTI Misalkan Π menyatakan Konstruksi 11.19, dan misalkan $\mathcal{A}$ adalah musuh PPT. Kami ingin menunjukkan bahwa ada fungsi yang dapat diabaikan sehingga

$$\Pr\left[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathrm{negl}(n)$$

Probabilitas di atas juga diambil alih pilihan fungsi H yang seragam, dimana $\mathcal{A}$ diberi akses oracle.

Pertimbangkan pelaksanaan eksperimen $\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n)$ yang kunci publiknya adalah $\langle \mathbb{G}, q, g, h \rangle$ dan teks sandinya adalah $c = g^y$, dan misalkan Query adalah kejadian $\mathcal{A}$ menanyakan $\mathrm{DH}_g(h, c) = h^y$ to H. Kita punya

$$\begin{aligned}\Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1] &= \Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathrm{Query}}] \\ &\quad + \Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \mathrm{Query}] \\ &\leq \Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathrm{Query}}] + \Pr[\mathrm{Query}]. \quad (11.18)\end{aligned}$$

$$\text{If } \Pr[\overline{\mathrm{Query}}] = 0 \text{ then } \Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathrm{Query}}] = 0. \text{ Otherwise,}$$

$$\begin{aligned}\Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \wedge \overline{\mathrm{Query}}] &= \Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \mid \overline{\mathrm{Query}}] \cdot \Pr[\overline{\mathrm{Query}}] \\ &\leq \Pr[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 \mid \overline{\mathrm{Query}}].\end{aligned}$$

Dalam percobaan $\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n)$, musuh $\mathcal{A}$ diberikan kunci publik dan teks sandi, ditambah kunci yang dienkapsulasi $k \stackrel{\text{def}}{=} H(h^y)$ atau kunci seragam. Jika Query tidak terjadi, maka k terdistribusi secara seragam dari sudut pandang musuh, sehingga $\mathcal{A}$ tidak mungkin membedakan kedua kemungkinan ini. Artinya

$$\Pr\left[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1 | \overline{\mathrm{Query}}\right] = \frac{1}{2}$$

Kembali ke Persamaan (11.18), kita mendapatkan

$$\Pr\left[\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \Pr\left[\mathrm{Query}\right]$$

Kami selanjutnya menunjukkan bahwa $\Pr[\mathrm{Query}]$ dapat diabaikan, melengkapi pembuktiannya.

Misalkan $t = t(n)$ adalah batas atas (polinomial) dari jumlah kueri yang dibuat $\mathcal{A}$ terhadap oracle acak H. Tentukan algoritma PPT berikut $\mathcal{A}'$ untuk masalah CDH relatif terhadap $\mathbb{G}$:

Algoritma $\mathcal{A}'$:

Algoritma diberikan $\mathbb{G}, q, g, h, c$ sebagai input.

- Tetapkan $pk = \langle \mathbb{G}, q, g, h \rangle$ dan pilih seragam $k \in \{0,1\}^{\ell}$.
- Jalankan $\mathcal{A}(pk, c, k)$. Saat $\mathcal{A}$ membuat kueri ke H, jawablah dengan memilih string ℓ −bit yang baru dan seragam.
- Di akhir eksekusi $\mathcal{A}$, misalkan $y_1, \ldots, y_t$ adalah daftar pertanyaan yang dibuat $\mathcal{A}$ terhadap H. Pilih indeks seragam $i \in \{1, \ldots, t\}$ dan keluaran y_i.

Kita tertarik pada probabilitas dimana $\mathcal{A}'$ menyelesaikan permasalahan CDH, yaitu $\Pr\left[\mathcal{A}'(\mathbb{G}, q, g, h, c) = \mathrm{DH}_g(h, c)\right]$ dimana probabilitasnya diambil alih $\mathbb{G}, q, g$ keluaran oleh $\mathcal{G}(1^n)$, seragam $h, c \in \mathbb{G}$, dan keacakan $\mathcal{A}'$. Untuk menganalisis probabilitas ini, perhatikan terlebih dahulu bahwa peristiwa Query masih terdefinisi dengan baik dalam eksekusi $\mathcal{A}'$,

meskipun $\mathcal{A}'$ tidak dapat mendeteksi apakah peristiwa tersebut terjadi. Selain itu, probabilitas kejadian Query ketika $\mathcal{A}$ dijalankan sebagai sub-rutin oleh $\mathcal{A}'$ identik dengan probabilitas kejadian Query dalam eksperimen $\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n)$. Hal ini terjadi karena tampilan $\mathcal{A}$ identik dalam kedua kasus hingga peristiwa Query terjadi: dalam setiap kasus, $\mathbb{G}, q, g$ dihasilkan oleh $\mathcal{G}(1^n)$; dalam setiap kasus, h dan c adalah elemen seragam dari $\mathbb{G}$ dan k adalah string ℓ-bit yang seragam; dan dalam setiap kasus, pertanyaan ke H selain $\mathrm{DH}_g(h,c)$dijawab dengan string ℓ-bit yang seragam. (Dalam $\mathrm{KEM}^{\mathrm{cpa}}_{\mathcal{A},\Pi}(n)$, $\mathrm{DH}_g(h,c)$) dijawab dengan kunci enkapsulasi aktual, yaitu sama dengan k dengan probabilitas 1/2, sedangkan ketika $\mathcal{A}$ dijalankan sebagai subrutin oleh $\mathcal{A}'$ kueri $H(\mathrm{DH}_g(h,c))$ dijawab dengan string ℓ-bit seragam yang tidak bergantung pada k. Namun saat kueri ini dibuat, peristiwa Query terjadi.)

Terakhir, amati bahwa ketika Query muncul maka $\mathrm{DH}_g(h,c) \in y_1,\ldots,y_t$ menurut definisi, sehingga $\mathcal{A}'$ mengeluarkan hasil yang benar $\mathrm{DH}_g(h,c)$ dengan probabilitas setidaknya $1/t$. Oleh karena itu kami menyimpulkan bahwa:

$$\Pr\left[\mathcal{A}'(\mathbb{G},q,g,h,c) = \mathrm{DH}_g(h,c)\right] \geq \Pr[\mathrm{Query}]/\mathrm{t}$$

atau $\Pr[\mathrm{Query}] \leq t \cdot \Pr\left[\mathcal{A}'(\mathbb{G},q,g,h,c) = \mathrm{DH}_g(h,c)\right]$ sejak masalah CDH sulit untuk $\mathbb{G}$, kemungkinan terakhir ini dapat diabaikan; karena t adalah polinomial, ini menyiratkan bahwa $\Pr[\mathrm{Query}]$ juga dapat diabaikan. Ini melengkapi buktinya.

Pada bagian berikutnya kita akan melihat bahwa Konstruksi 11.19 dapat terbukti aman terhadap CCA berdasarkan varian asumsi CDH yang lebih kuat (jika kita terus memodelkan H sebagai ramalan acak).

Keamanan Ciphertext Terpilih dan DHIES/ECIES

Skema enkripsi El Gamal rentan terhadap serangan teks sandi terpilih. Ini mengikuti fakta bahwa itu mudah ditempa. Ingatlah bahwa skema enkripsi dapat ditempa, secara informal, jika diberi ciphertext c yang merupakan enkripsi dari beberapa pesan yang tidak diketahui m, dimungkinkan untuk menghasilkan ciphertext yang dimodifikasi c' yang merupakan enkripsi dari sebuah pesan m' yang mempunyai hubungan yang diketahui dengan m . Dalam kasus enkripsi El Gamal, pertimbangkan musuh $\mathcal{A}$ yang menyadap teks sandi $c = \langle c_1, c_2\rangle$ yang dienkripsi menggunakan kunci publik $pk = \langle \mathbb{G}, q, g, h\rangle$, dan kemudian membuat teks sandi yang dimodifikasi $c' = \langle c_1, c'_2\rangle$dimana $c'_s = c_2 \cdot \alpha$ untuk beberapa $\alpha \in \mathbb{G}$. Jika c adalah enkripsi pesan $m \in \mathbb{G}$ (yang mungkin tidak diketahui oleh $\mathcal{A}$), kita mempunyai $c_1 = g^y$ dan $c_2 = h^y \cdot m$ untuk beberapa $y \in \mathbb{Z}_q$. Tapi kemudian

$$c_1 = g^y \text{ dan } c'_2 = h^y \cdot (\alpha \cdot m)$$

jadi c' adalah enkripsi yang valid untuk pesan $\alpha \cdot m$. Dengan kata lain, dapat mengubah enkripsi pesan (tidak diketahui) m menjadi enkripsi pesan (tidak diketahui) $\alpha \cdot m$. Seperti yang dibahas dalam Skenario 3 di Bagian 11.2, serangan semacam ini dapat menimbulkan konsekuensi yang serius.

KEM yang dibahas pada bagian sebelumnya mungkin juga dapat dibentuk tergantung pada fungsi derivasi kunci spesifik H yang digunakan. Namun, jika H dimodelkan sebagai

oracle acak, maka serangan seperti itu tampaknya tidak lagi mungkin dilakukan. Faktanya, dalam kasus ini kita dapat membuktikan bahwa Konstruksi 11.19 aman terhadap CCA berdasarkan apa yang disebut asumsi gap-CDH. Ingatlah bahwa asumsi CDH mengatakan bahwa elemen grup tertentu g^x dan g^y (untuk beberapa generator g), tidak mungkin menghitung g^{xy}. Asumsi gap-CDH mengatakan bahwa hal ini tetap tidak mungkin dilakukan bahkan jika diberikan akses ke oracle $\mathcal{O}$ sedemikian rupa sehingga $\mathcal{O}(U, V)$ mengembalikan 1 tepat ketika $V = U^y$. Dengan kata lain, masalah CDH tetap sulit bahkan jika ada oracle yang memecahkan masalah DDH. (Kami tidak memberikan definisi formal karena kami tidak akan menggunakan asumsi ini di sisa buku ini.) Asumsi ini diyakini berlaku untuk kelas-kelas kelompok yang telah kita bahas dalam buku ini. Pembuktian berikut ini sangat mirip dengan pembuktian Teorema 11.38.

TEOREMA 11.22 Jika permasalahan gap-CDH relatif sulit terhadap $\mathcal{G}$, dan H dimodelkan sebagai ramalan acak, maka Konstruksi 11.19 adalah KEM yang aman CCA.

Menarik untuk diamati bahwa konstruksi yang sama (yaitu, Konstruksi 11.19) dapat dianalisis berdasarkan asumsi yang berbeda dan model yang berbeda, sehingga menghasilkan hasil yang berbeda. Dengan asumsi bahwa permasalahan DDH sulit (dan untuk H dipilih dengan tepat), skema ini aman untuk CPA. Jika kita memodelkan H sebagai oracle acak (yang menerapkan persyaratan lebih ketat pada H), maka dengan asumsi CDH yang lebih lemah kita memperoleh keamanan CPA, dan dengan asumsi gap-CDH yang lebih kuat kita memperoleh keamanan CCA.

CONSTRUCTION 11.23

Let $\mathcal{G}$ be as in the text. Let $\Pi_E = (\mathsf{Enc}', \mathsf{Dec}')$ be a private-key encryption scheme, and let $\Pi_M = (\mathsf{Mac}, \mathsf{Vrfy})$ be a message authentication code. Define a public-key encryption scheme as follows:

- Gen: On input 1^n run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. choose a uniform $x \in \mathbb{Z}_q$, set $h := g^x$, and specify a function $H : \mathbb{G} \to \{0,1\}^{2n}$. The public key is $\langle \mathbb{G}, q, g, h, H \rangle$ and the private key is $\langle \mathbb{G}, q, g, x, H \rangle$.
- Enc: On input a public key $pk = \langle \mathbb{G}, q, g, h, H \rangle$, choose a uniform $y \in \mathbb{Z}_q$ and set $k_E \| k_M := H(h^y)$. Compute $c' \leftarrow \mathsf{Enc}'_{k_E}(m)$, and output the ciphertext $\langle g^y, c', \mathsf{Mac}_{k_M}(c') \rangle$.
- Dec: On input a private key $sk = \langle \mathbb{G}, q, g, x, H \rangle$ and a ciphertext $\langle c, c', t \rangle$, output $\perp$ if $c \notin \mathbb{G}$. Else, compute $k_E \| k_M := H(c^x)$. If $\mathsf{Vrfy}_{k_M}(c', t) \neq 1$ then output $\perp$; otherwise, output $\mathsf{Dec}'_{k_E}(c')$.

Enkripsi aman CCA dengan Konstruksi 11.19. Menggabungkan KEM dalam Konstruksi 11.19 dengan skema enkripsi kunci privat aman CCA akan menghasilkan skema enkripsi kunci publik aman CCA. (Lihat Teorema 11.14.) Instansiasi pendekatan ini menggunakan Konstruksi 4.18 untuk komponen kunci privat sesuai dengan apa yang dilakukan di DHIES/ECIES, variannya termasuk dalam standar ISO/IEC 18033-2 untuk enkripsi kunci publik. (Lihat Konstruksi 11.23.) Enkripsi pesan m dalam skema ini mengambil bentuk

$$\langle g^y, \mathrm{Enc}'_{K_E}(x), \mathrm{Mac}_{K_E}(c') \rangle$$

dimana Enc' menunjukkan skema enkripsi kunci pribadi aman-CPA dan c' menunjukkan $\mathrm{Enc}'_{K_E}(m)$. DHIES, Skema Enkripsi Terintegrasi Diffie–Hellman, dapat digunakan secara umum untuk merujuk pada skema apa pun dalam bentuk ini, atau untuk merujuk secara khusus pada kasus ketika grup $\mathbb{G}$ adalah subgrup siklik dari bidang berhingga. ECIES, Skema Enkripsi Terintegrasi Kurva Elliptik, mengacu pada kasus ketika $\mathbb{G}$ adalah grup kurva elips. Kami mencatat bahwa, dalam Konstruksi 11.23, sangat penting untuk memeriksa selama dekripsi bahwa c, komponen pertama teks sandi, ada di $\mathbb{G}$. Jika tidak, penyerang mungkin meminta dekripsi teks sandi yang salah formatnya $\langle c, c', t \rangle$ di mana $c \notin \mathbb{G}$; mendekripsi ciphertext seperti itu (yaitu, tanpa mengembalikan $\perp$) mungkin membocorkan informasi tentang kunci pribadi.

Berdasarkan Teorema 4.19, mengenkripsi pesan dan kemudian menerapkan kode otentikasi pesan (kuat) menghasilkan skema enkripsi kunci privat yang aman dengan CCA. Menggabungkan ini dengan Teorema 11.14, kita menyimpulkan:

HASIL 11.24 Misalkan Π_E adalah skema enkripsi kunci pribadi yang aman dengan CPA, dan Π_M adalah kode autentikasi pesan yang sangat aman. Jika masalah gap-CDH relatif sulit, dan H dimodelkan sebagai oracle acak, maka Konstruksi 11.23 adalah skema enkripsi kunci publik yang aman dengan CCA.

11.5 ENKRIPSI RSA

Pada bagian ini kita mengalihkan perhatian kita pada skema enkripsi berdasarkan asumsi RSA yang didefinisikan dalam Bagian 8.2. Kami mencatat bahwa meskipun enkripsi berbasis RSA digunakan secara luas saat ini, saat ini terdapat pergeseran bertahap dari penggunaan RSA—dan menuju penggunaan kriptosistem berbasis CDH/DDH yang mengandalkan kelompok kurva elips—karena panjang kunci yang lebih panjang. diperlukan untuk skema berbasis RSA. Kami mengacu pada Bagian 9.3 untuk diskusi lebih lanjut.

RSA Biasa

Kita mulai dengan menjelaskan skema enkripsi sederhana berdasarkan masalah RSA. Meskipun skema ini tidak aman, skema ini memberikan titik awal yang berguna untuk skema aman berikutnya.

Misalkan GenRSA adalah algoritma PPT yang, pada input 1^n, menghasilkan modulus N yang merupakan hasil kali dua n-bit bilangan prima, bersama dengan bilangan bulat e, d yang memuaskan $ed = 1 \bmod \emptyset(N)$. (Seperti biasa, algoritma mungkin gagal dengan probabilitas yang dapat diabaikan tetapi kita mengabaikannya di sini.) Ingat dari Bagian 8.2.4 bahwa algoritma seperti itu dapat dengan mudah dibangun dari algoritma $\mathrm{GenModulus}$ apa pun yang menghasilkan modulus komposit N beserta faktorisasinya; lihat Algoritma 11.25.

ALGORITHM 11.25
RSA key generation GenRSA

Input: Security parameter 1^n
Output: N, e, d as described in the text

$(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$
$\phi(N) := (p-1)(q-1)$
choose $e > 1$ such that $\gcd(e, \phi(N)) = 1$
compute $d := [e^{-1} \bmod \phi(N)]$
return N, e, d

Misalkan N, e, d seperti di atas, dan misalkan $c = m^e \bmod N$. Enkripsi RSA bergantung pada fakta bahwa seseorang yang mengetahui d dapat memulihkan m dari c dengan menghitung $[c^d \bmod N]$; ini berhasil karena

$$[c^d = (m^e)^d = m^{ed} = m \bmod N];$$

seperti yang dibahas dalam Bagian 8.2. Di sisi lain, tanpa pengetahuan tentang d (bahkan jika N dan e diketahui) asumsi RSA (lih. Definisi 8.46) menyiratkan bahwa sulit untuk memulihkan m dari c, setidaknya jika m dipilih secara seragam dari $\mathbb{Z}_N^*$. Hal ini tentu saja menunjukkan skema enkripsi kunci publik yang ditunjukkan pada Konstruksi 11.26: Penerima menjalankan GenRSA untuk mendapatkan N, e, d; ia menerbitkan N dan e sebagai kunci publiknya, dan menyimpan d sebagai kunci privatnya. Untuk mengenkripsi pesan $m \in \mathbb{Z}_N^*$, pengirim menghitung ciphertext $c := [m^e \bmod N]$. Seperti yang baru saja kita catat, penerima—siapa yang mengetahui d—dapat mendekripsi c dan memulihkan m.

CONSTRUCTION 11.26

Let GenRSA be as in the text. Define a public-key encryption scheme as follows:

- Gen: on input 1^n run $\mathsf{GenRSA}(1^n)$ to obtain N, e, and d. The public key is $\langle N, e \rangle$ and the private key is $\langle N, d \rangle$.
- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the ciphertext
$$c := [m^e \bmod N].$$
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute the message
$$m := [c^d \bmod N].$$

Berikut ini adalah contoh kerja di atas (lihat juga Contoh 8.49).

Contoh 11.27

Katakanlah keluaran $\mathrm{GenRSA}\ (N, e, d) = (391, 3, 235)$. (Perhatikan bahwa $391 = 17 \cdot 23$ dan jadi $\emptyset(391) = 16 \cdot 22 = 352$. Selain itu, $3 \cdot 235 = 1 \bmod 352$.) Jadi kunci publiknya adalah $(391, 3)$ dan kunci privatnya adalah $(391, 235)$.

Untuk mengenkripsi pesan $m = 158 \in \mathbb{Z}_{391}^*$ menggunakan kunci publik $(391, 3)$, kita cukup menghitung $c := [158^3 \bmod 391] = 295$; ini adalah teks tersandi. Untuk mendekripsi, penerima menghitung $[295^{235} \bmod 391] = 158$.

Apakah skema enkripsi RSA biasa aman? Asumsi pemfaktoran menyiratkan bahwa secara komputasi tidak mungkin bagi penyerang yang diberi kunci publik untuk mendapatkan kunci privat yang sesuai; lihat Bagian 8.2. Hal ini perlu—tetapi tidak cukup—agar skema enkripsi kunci publik aman. Asumsi RSA menyiratkan bahwa jika pesan m dipilih secara seragam dari $\mathbb{Z}_N^*$ maka penyadap yang diberikan $N, e,$ dan c (yaitu, kunci publik dan teks tersandi) tidak dapat memulihkan m. Namun jaminan ini lemah dan jauh dari tingkat keamanan yang kita inginkan! Secara khusus, mereka membuka kemungkinan bahwa penyerang dapat memulihkan pesan ketika pesan tersebut tidak dipilih secara seragam dari $\mathbb{Z}_N^*$ (tentu saja, ketika m dipilih dari rentang yang kecil, mudah untuk melihat bahwa penyerang dapat menghitung m dari publik kunci dan teks sandi). Selain itu, tidak menutup kemungkinan bahwa penyerang dapat mempelajari sebagian informasi tentang pesan tersebut, meskipun informasi tersebut seragam (pada kenyataannya, hal ini diketahui mungkin terjadi). Selain itu, enkripsi RSA biasa bersifat deterministik sehingga pasti tidak aman, seperti yang telah kita bahas di Bagian 11.2.

Lebih Banyak Serangan di RSA Biasa

Kami telah mencatat bahwa enkripsi RSA biasa tidak aman terhadap CPA. Namun demikian, mungkin ada godaan untuk menggunakan RSA biasa untuk mengenkripsi "pesan acak" dan/atau dalam situasi di mana membocorkan sedikit informasi tentang pesan tersebut dapat diterima. Kami memperingatkan terhadap hal ini secara umum, dan di sini hanya memberikan beberapa contoh tentang apa yang bisa salah.

(Beberapa serangan berikutnya mengasumsikan $e = 3$. Dalam beberapa kasus, serangan dapat diperluas, setidaknya sebagian, ke e yang lebih besar; dalam hal apa pun, seperti disebutkan dalam Bagian 8.2, pengaturan $e = 3$ sering dilakukan dalam praktiknya. Serangan tersebut harus dianggap sebagai demonstrasi bahwa Konstruksi 11.26 tidak memadai, bukan sebagai indikasi bahwa pengaturan $e = 3$ merupakan pilihan yang buruk.)

Peningkatan kuadrat dalam memulihkan m. Karena enkripsi RSA biasa bersifat deterministik, kita tahu bahwa jika $m < B$ maka penyerang dapat menentukan m dari ciphertext $c = [m^e \bmod N]$ dalam waktu $\mathcal{O}(B)$ menggunakan serangan brute-force yang dibahas di Bagian 11.2. Namun, kita mungkin berharap bahwa enkripsi RSA biasa dapat digunakan jika B besar, yaitu jika pesan dipilih dari sekumpulan nilai yang cukup besar. Salah satu skenario yang mungkin terjadi adalah dalam konteks enkripsi hibrid (lihat Bagian 11.3), di mana "pesan" adalah kunci n-bit acak sehingga $B = 2^n$. Sayangnya, ada serangan cerdas yang memulihkan m, dengan probabilitas tinggi, dalam waktu sekitar $\mathcal{O}(\sqrt{B})$. Hal ini dapat membuat perbedaan yang signifikan dalam praktiknya: serangan 2^{80} kali (katakanlah) tidak mungkin dilakukan, namun serangan yang dijalankan dalam waktu 2^{40} relatif mudah dilakukan.

Deskripsi serangan diberikan sebagai Algoritma 11.28. Dalam uraian kita, kita asumsikan $B = 2^n$ dan misalkan $\alpha \in \left(\frac{1}{2}, 1\right)$ menunjukkan suatu konstanta tetap (lihat di bawah).

ALGORITHM 11.28
An attack on plain RSA encryption

Input: Public key $\langle N, e\rangle$; ciphertext c
Output: $m < 2^n$ such that $m^e = c \bmod N$

set $T := 2^{\alpha n}$
for $r = 1$ to T:
  $x_r := [c/r^e \bmod N]$
sort the pairs $\{(r, x_r)\}_{r=1}^{T}$ by their second component
for $s = 1$ to T:
  if $x_r \stackrel{?}{=} [s^e \bmod N]$ for some r
    return $[r \cdot s \bmod N]$

Kompleksitas waktu algoritma didominasi oleh waktu yang dibutuhkan untuk mengurutkan $2^{\alpha n}$ pasangan (r, x_r); ini dapat dilakukan dalam waktu $\mathcal{O}(n \cdot 2^{\alpha n})$. Pencarian biner digunakan pada baris kedua hingga terakhir untuk memeriksa apakah terdapat r dengan $x_r = [s^e \bmod N]$.

Kami sekarang membuat sketsa mengapa serangan tersebut memulihkan m dengan probabilitas tinggi. Misalkan $c = m^e \bmod N$. Untuk pemilihan $\alpha > 1/2$ yang tepat, dapat ditunjukkan bahwa jika m adalah bilangan bulat n-bit yang seragam, maka dengan probabilitas tinggi terdapat r, s dengan $1 < r \leq s \leq 2^{\alpha n}$ dengan $m = r \cdot s$. (Misalnya, jika n = 64 dan m adalah string acak 64-bit, maka dengan probabilitas 0,35 terdapat r, s dengan panjang paling banyak 34 bit sehingga $m = r \cdot s$. Lihat referensi di akhir bab untuk detailnya.) Dengan asumsi hal ini terjadi, algoritma di atas menemukan m sejak itu

$$c = m^e = (\mathrm{r} \cdot \mathrm{s})^e = r^e \cdot s^e \bmod N$$

dan $x_r = c/r^e = s^e \bmod N$ dengan $r, s < T$.

Mengenkripsi pesan singkat menggunakan *e*. Serangan sebelumnya menunjukkan cara memulihkan pesan m yang diketahui lebih kecil dari batas B dalam waktu kira-kira $\mathcal{O}(\sqrt{B})$. Di sini kami menunjukkan cara melakukan hal yang sama dalam waktu $\mathrm{poli}(\|N\|)$ jika $B \leq N^{1/e}$ (yang berarti akar e dari N sebagai bilangan real).

Serangan ini bergantung pada pengamatan bahwa ketika $m < N^{-1/e}$, menaikkan m ke modulo daya eth N tidak melibatkan reduksi modular; yaitu, $[m^e \bmod N]$ sama dengan bilangan bulat m^e. Ini berarti bahwa dengan ciphertext $c = [m^e \ mod \ N]$, penyerang dapat menentukan m dengan menghitung $m := c^{1/e}$ pada bilangan bulat (yaitu, bukan modulo N); ini dapat dilakukan dengan mudah dalam waktu $\mathrm{poly}(\|c\|) = \mathrm{poly}(\|N\|)$ karena menemukan akar eth mudah dilakukan pada bilangan bulat dan sulit hanya ketika menggunakan $\bmod N$.

Untuk e kecil, hal ini menunjukkan kelemahan serius enkripsi RSA biasa. Misalnya, jika kita mengambil $e = 3$ dan mengasumsikan $\|N\| \approx 1024$ bit, maka serangan akan berhasil bahkan ketika m adalah bilangan bulat 300-bit yang seragam; hal ini sekali lagi

mengesampingkan keamanan RSA biasa bahkan ketika digunakan sebagai bagian dari skema enkripsi hibrid.

Mengenkripsi pesan yang diketahui sebagian. Serangan ini dapat dipandang sebagai generalisasi dari serangan sebelumnya. Ini mengasumsikan pengirim yang mengenkripsi pesan, yang sebagiannya diketahui (sesuatu yang tidak boleh mengakibatkan serangan saat menggunakan skema enkripsi yang aman). Di sini kami mengandalkan hasil kuat dari Coppersmith yang kami nyatakan tanpa bukti:

TEOREMA 11.29 Misalkan $p(x)$ adalah polinomial berderajat e. Kemudian pada waktunya $\text{poli}(\|N\|, e)$ seseorang dapat menemukan semua m sedemikian rupa sehingga $p(m) = 0 \; mod \; N$ dan $|m| \leq N^{1/e}$.

Karena ketergantungan waktu berjalan pada e, serangan hanya praktis untuk e kecil. Berikut ini kita asumsikan $e = 3$ untuk kekonkretan.

Asumsikan pengirim mengenkripsi pesan $m = m_1 \| m_2$ ke penerima dengan kunci publik$\langle N, 3\rangle$, dimana bagian pertama m_1 dari pesan diketahui tetapi bagian kedua m_2 tidak. Agar lebih konkrit, misalkan m_2 panjangnya k bit, jadi $m = B \cdot m_1 + m_2$ dimana kita misalkan $B = 2^k$. Mengingat ciphertext yang dihasilkan $c = [(m_1 \| m_2)^3 \bmod N\,]$, seorang penyadap dapat mendefinisikan $p(x) \stackrel{\text{def}}{=} (2^k \cdot m_1 + x)^3 - c$, sebuah polinomial kubik. Polinomial ini memiliki m_2 sebagai akar (modulo N), dan $|m_2| < B$. Teorema 11.29 dengan demikian menyiratkan bahwa penyerang dapat menghitung m_2 secara efisien selama $B \leq N\,1/3$.

Serangan serupa bekerja ketika m_2 diketahui namun m_1 tidak. Mengenkripsi pesan terkait. Serangan ini mengasumsikan pengirim mengenkripsi dua pesan terkait ke penerima yang sama (sesuatu yang tidak boleh mengakibatkan serangan saat menggunakan skema enkripsi aman). Asumsikan pengirim mengenkripsi m dan $m + \delta$ ke penerima dengan kunci publik $\langle$ N,$e\rangle$, dimana offset δ diketahui tetapi m tidak. Mengingat dua ciphertext $c_1 = [m^e \bmod N\,]$ dan $c_2 = [(m+\delta)^e \bmod N]$, seorang penyadap dapat mendefinisikan dua polinomial $f_1(x) \stackrel{\text{def}}{=} x^e - c_2$ dan $f_2(x) \stackrel{\text{def}}{=} (x+\delta)^e - c_2$ $(modulo\; N\,)$, masing-masing berderajat e. Perhatikan bahwa $x = m$ adalah akar (modulo N) dari kedua polinomial, sehingga suku linier $(x - m)$ adalah faktor dari keduanya. Jadi, jika pembagi persekutuan terbesar dari $f_1(x)$ dan $f_2(x)$ (sebagai polinomial pada $\mathbb{Z}_N^*$) adalah linier, maka akan diperoleh m. Pembagi persekutuan terbesar dapat dihitung dalam waktu $poli(\|N\|\,, e)$ menggunakan algoritma yang serupa.

Mengirim pesan yang sama ke beberapa penerima. Serangan terakhir kita mengasumsikan pengirim yang mengenkripsi pesan yang sama ke beberapa penerima (sesuatu yang, sekali lagi, tidak boleh mengakibatkan serangan ketika menggunakan skema enkripsi yang aman). Misal $e = 3$, dan misalkan pesan yang sama m dienkripsi ke tiga pihak berbeda yang memegang kunci publik $pk_1 = \langle N_1, 3\rangle$, $pk_2 = \langle N_2, 3\rangle$, $pk_3 = \langle N_3, 3\rangle$. Asumsikan $\gcd(N_i, N_j\,) = 1$ untuk i, j; jika tidak, maka setidaknya salah satu modul dapat segera difaktorkan dan pesan m dapat dengan mudah dipulihkan. Seorang penyadap melihat

$$c_1 = [m^3 \bmod N_1], \quad c_2 = [m^3 \bmod N_2], \quad \text{and} \quad c_3 = [m^3 \bmod N_3].$$

Misalkan $N^* = N_1 N_2 N_3$. Versi lanjutan dari teorema sisa Cina mengatakan bahwa terdapat bilangan bulat non-negatif unik $\hat{c} < N^*$ sehingga

$$\hat{c} = c_1 \bmod N_1$$
$$\hat{c} = c_2 \bmod N_2$$
$$\hat{c} = c_3 \bmod N_3$$

Selain itu, dengan menggunakan teknik serupa dengan yang ditunjukkan pada Bagian 8.1, kita dapat menghitung m^3 secara efisien dengan menggunakan kunci publik dan teks sandi di atas. Perhatikan akhirnya bahwa memenuhi persamaan di atas, dan $m^3 < N^*$ karena $m < \min\{N_1, N_2, N_3\}$. Seperti pada serangan sebelumnya, ini berarti bahwa $\hat{c} < m^3$ pada bilangan bulat (yaitu, tanpa terjadi reduksi modular), sehingga pesan m dapat dipulihkan dengan menghitung akar pangkat tiga bilangan bulat dari $\hat{c}$.

RSA dan PKCS #1 v1.5

Meskipun RSA biasa tidak aman, RSA menyarankan satu pendekatan umum terhadap enkripsi kunci publik berdasarkan masalah RSA: untuk mengenkripsi pesan m menggunakan kunci publik $\langle N, e \rangle$, petakan m terlebih dahulu ke elemen $\widehat{m} \in \mathbb{Z}_N^*$; kemudian hitung ciphertext $c = [\widehat{m}^e \bmod N]$. Untuk mendekripsi ciphertext c, penerima menghitung m^$\widehat{m} = [c^d \bmod N]$ dan kemudian memulihkan pesan asli m. Agar penerima dapat memulihkan pesan, pemetaan dari pesan ke elemen $\mathbb{Z}_N^*$ harus (secara efisien) dapat dibalik. Agar skema yang mengikuti pendekatan ini memiliki harapan aman terhadap CPA, pemetaannya harus diacak sehingga enkripsi tidak bersifat deterministik. Tentu saja hal ini merupakan kondisi yang diperlukan namun belum cukup, dan keamanan skema enkripsi sangat bergantung pada pemetaan spesifik yang digunakan.

Salah satu implementasi sederhana dari ide di atas adalah dengan memasukkan pesan secara acak sebelum melakukan enkripsi. Artinya, untuk memetakan pesan m (dilihat sebagai bit-string) ke elemen $\mathbb{Z}_N^*$, pengirim memilih bit-string seragam $r \in \{0,1\}^\ell$ (untuk beberapa ℓ yang sesuai) dan menyetel $\widehat{m} := r \| m$; nilai yang dihasilkan secara alami dapat diinterpretasikan sebagai bilangan bulat di $\mathbb{Z}_N^*$, dan pemetaan ini jelas dapat dibalik. Lihat Konstruksi 11.30. (Batas pada $\ell(n)$ dan panjang m memastikan bahwa bilangan bulat $\widehat{m}$ lebih kecil dari N .)

Konstruksinya diparameterisasi dengan nilai ℓ yang menentukan panjang padding acak yang digunakan. Keamanan skema tergantung pada ℓ. Terdapat serangan brute force yang jelas pada skema yang berjalan pada waktu 2^ℓ, jadi jika ℓ terlalu pendek (khususnya, jika $\ell(n) = \mathcal{O}(\log n)$), skema tersebut tidak aman. Pada ekstrem yang lain, kami tunjukkan di bagian berikut (pada dasarnya) bahwa ketika padding sebesar mungkin, dan m hanya satu bit, maka dimungkinkan untuk membuktikan keamanan berdasarkan asumsi RSA. Dalam kasus-kasus menengah, situasinya kurang jelas: untuk rentang ℓ tertentu kita tidak dapat

membuktikan keamanan berdasarkan asumsi RSA tetapi tidak ada serangan waktu polinomial yang diketahui. Kami menunda diskusi lebih lanjut sampai setelah pembahasan PKCS #1 v1.5 berikutnya.

RSA PKCS #1 v1.5. Standar Kriptografi Kunci Publik Laboratorium RSA (PKCS) #1 versi 1.5, yang diterbitkan pada tahun 1993, menggunakan varian enkripsi RSA berlapis. Untuk kunci publik $pk = \langle N, e \rangle$ dalam bentuk biasa, misalkan k menyatakan panjang N dalam byte; yaitu, k adalah bilangan bulat yang memuaskan $2^{8(k-1)} \leq N < 2^{8k}$. Pesan m yang akan dienkripsi diasumsikan memiliki panjang kelipatan 8 bit dan dapat memiliki panjang antara satu hingga k — 11 byte. Enkripsi pesan m yang panjangnya D —byte dihitung sebagai

$$[(0\text{x}00 \| 0\text{x}02 \| r \| 0\text{x}00 \| m)^e \bmod N],$$

di mana r adalah string byte $(k - D - 3)$ yang dihasilkan secara acak dan tidak ada byte yang sama dengan 0x00. (Kondisi terakhir ini memungkinkan pesan dipulihkan secara jelas setelah dekripsi.) Perhatikan bahwa panjang maksimum yang diperbolehkan m memastikan bahwa panjang r setidaknya 8 byte.

CONSTRUCTION 11.30

Let GenRSA be as before, and let ℓ be a function with $\ell(n) \leq 2n - 4$ for all n. Define a public-key encryption scheme as follows:

- Gen: on input 1^n, run GenRSA(1^n) to obtain (N, e, d). Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0,1\}^{\|N\| - \ell(n) - 2}$, choose a uniform string $r \in \{0,1\}^{\ell(n)}$ and interpret $\hat{m} := r \| m$ as an element of $\mathbb{Z}_N^*$. Output the ciphertext
$$c := [\hat{m}^e \bmod N].$$
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute
$$\hat{m} := [c^d \bmod N],$$
and output the $\|N\| - \ell(n) - 2$ least-significant bits of $\hat{m}$.

Skema enkripsi RSA berlapis.

Sayangnya, PKCS #1 v1.5 seperti yang ditentukan tidak aman terhadap CPA karena memungkinkan penggunaan padding acak yang terlalu pendek. Hal ini paling baik diilustrasikan dengan menunjukkan bahwa penyerang dapat menentukan bagian awal dari sebuah pesan yang diketahui memiliki banyak angka 0 di belakangnya. Untuk mempermudah, katakanlah $m = b \| \underbrace{0 \cdots 0}_{L}$ dengan $b \in \{0,1\}$ tidak diketahui dan m sepanjang mungkin (jadi $L = 8 \cdot (k - 11) - 1$). Enkripsi m menghasilkan ciphertext c dengan

$$c = (0 \times 00 \| 0 \times 02 \| r \| 0 \times 00 \| b \| 0 \cdots 0)^e \bmod N$$

Seorang penyerang dapat menghitung $c' = c/(2^L)^e \mod N$; perhatikan itu

$$c' = \left(\frac{0 \times 00 \| 0 \times 02 \| r \| 0 \times 00 \| b \| 0 \cdots 0}{10 \cdots 0}\right)^e = c = (0 \times 00 \| 0 \times 02 \| r \| 0 \times 00 \| b)^e \bmod N$$

Bilangan bulat $0 \times 02 \| r \| 0 \times 00 \| b$ panjangnya 75 bit (perhatikan bahwa $0 \times 02 = 0000\ 0010$, dan semua bit 0 tingkat tinggi tidak dihitung), sehingga penyerang sekarang dapat menerapkan "serangan pesan singkat", atau serangan pesan singkat. serangan berdasarkan mengenkripsi pesan yang diketahui sebagian, dari bagian sebelumnya. Untuk menghindari serangan ini kita perlu mengambil panjang r paling sedikit $\|N\|/e$. Sekalipun e besar, namun "serangan peningkatan kuadrat" dari bagian sebelumnya menunjukkan bahwa r dapat diperoleh kembali, dengan probabilitas tinggi, dalam waktu sekitar $2^{\|r\|/2}$.

Jika kita memaksa r menjadi sekitar setengah panjang N, dan dengan demikian mengurangi panjang pesan maksimum, maka masuk akal untuk menduga bahwa skema enkripsi di PKCS #1 v1.5 aman terhadap CPA. (Kami menekankan, bagaimanapun, bahwa tidak ada bukti keamanan berdasarkan asumsi RSA yang diketahui.) Namun demikian, karena serangan ciphertext terpilih yang serius terhadap skema tersebut.

***Enkripsi Aman CPA tanpa Oracle Acak**

Pada bagian ini kami menunjukkan skema enkripsi yang dapat dibuktikan aman terhadap CPA berdasarkan asumsi RSA. Kita mulai dengan mendeskripsikan predikat hard-core tertentu untuk masalah RSA dan kemudian menunjukkan bagaimana menggunakan predikat hard-core tersebut untuk mengenkripsi satu bit. Kami kemudian memperluas skema ini dengan memberikan KEM.

Skema yang diuraikan dalam bagian ini sebagian besar hanya untuk kepentingan teoretis dan tidak digunakan dalam praktik. Hal ini karena konstruksi tersebut kurang efisien dibandingkan konstruksi alternatif berbasis RSA yang dapat dibuktikan aman dalam model random-oracle (lihat Bagian 5.5). Kita akan melihat contoh skema enkripsi tersebut di bagian selanjutnya.

Predikat inti untuk masalah RSA. Secara sederhana, asumsi RSA mengatakan bahwa jika diberikan N, e, dan $[x^e \ mod \ N]$ (untuk x yang dipilih secara seragam dari $\mathbb{Z}_N^*$), maka x tidak mungkin diperoleh kembali. Dengan sendirinya, hal ini tidak menjelaskan apa pun tentang kesulitan komputasi dalam menghitung beberapa informasi spesifik tentang x. Bisakah kita mengisolasi sedikit informasi tertentu tentang x yang sulit dihitung dari N, e, dan $[x^e \ mod \ N]$? Gagasan mengenai predikat hard-core menangkap dengan tepat persyaratan ini. (Predikat inti diperkenalkan di Bagian 7.1. Fakta bahwa asumsi RSA memberikan rangkaian permutasi satu arah dibahas di Bagian 8.4. Namun, perlakuan kami di sini mandiri.) Ternyata bit paling tidak signifikan dari x, dilambangkan dengan $\mathrm{lsb}(x)$, adalah predikat inti untuk masalah RSA.

Tentukan eksperimen berikut untuk algoritma tertentu GenRSA (dengan perilaku biasa) dan algoritma $\mathcal{A}$:
Eksperimen predikat hard-core $RSA - \text{lsb}_{\mathcal{A},\text{GenRSA}}(1^n)$:

1. Jalankan GenRSA(1^n) untuk mendapatkan (N, e, d).
2. pilih seragam $x \in \mathbb{Z}_N^*$ dan hitung $y := [x^e \ mod \ N]$?.
3. $\mathcal{A}$ diberikan N, e, y, dan keluarannya a bit b.
4. Hasil percobaan adalah 1 jika dan hanya jika $\text{lsb}(x) = b$.

Perhatikan bahwa $lsb(x)$ adalah bit seragam jika $x \in \mathbb{Z}_N^*$ seragam. $\mathcal{A}$ dapat menebak $\text{lsb}(x)$ dengan probabilitas 1/2 hanya dengan mengeluarkan bit seragam b. Teorema berikut menyatakan bahwa jika permasalahan RSA sulit, maka tidak ada algoritma efisien yang dapat melakukan lebih baik dari ini; yaitu, bit yang paling tidak signifikan adalah predikat hard-core dari permutasi RSA.

TEOREMA 11.31 Jika permasalahan RSA relatif sulit terhadap GenRSA maka untuk semua algoritma waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi negl yang dapat diabaikan sehingga $\Pr[\text{RSA} - \text{lsb}_{\mathcal{A},\text{GenRSA}}(n) = 1] \leq \frac{1}{2} + \text{negl}(n)$.

Bukti lengkap teorema ini berada di luar cakupan buku ini. Namun, kami memberikan beberapa intuisi untuk teorema tersebut dengan membuat sketsa bukti dari hasil yang lebih lemah: bahwa asumsi RSA menyiratkan $\Pr[\text{RSA} - \text{lsb}_{\mathcal{A},\text{GenRSA}}(n) = 1] < 1$ semua waktu polinomial probabilistik $\mathcal{A}$. Untuk membuktikannya, kami menunjukkan bahwa efisiensi algoritma yang selalu menghitung $\text{lsb}(r)$ dengan benar dari N, e dan $[x^e \bmod N]$ dapat digunakan untuk memulihkan x secara efisien (secara keseluruhan) dari N, e dan $[x^e \bmod N]$.

Perbaiki N dan e, dan biarkan $\mathcal{A}$ menjadi suatu algoritma sehingga $\mathcal{A}(r^e \bmod N = \text{lsb}(r)$. Diketahui N, e, dan $y = [x^e \bmod N]$, kita akan memulihkan bit x satu per satu, dari yang terkecil hingga yang paling signifikan. Untuk menentukan $lsb(x)$ kita cukup menjalankan $\mathcal{A}(y)$. Sekarang ada dua kasus:

Kasus 1: $\text{lsb}(x) = 0$. Perhatikan bahwa $y/2^e = x/2^e \bmod N$, dan karena x genap (yaitu, $lsb(x) = 0$), 2 membagi bilangan bulat x. Jadi $x/2$ hanyalah pergeseran bit ke kanan dari x, dan $\text{lsb}(x/2)$ sama dengan $2\text{sb}(x)$, bit x yang paling tidak signifikan ke-2. Jadi kita dapat memperoleh $2\text{sb}(x)$ dengan menghitung $y' := [y/2^e \bmod N]$ dan kemudian menjalankan $\mathcal{A}(y')$.

Kasus 2: $\text{lsb}(x) = 1$. Di sini $[x/2 \bmod N\ ] = (x + N)/2$. Jadi $\text{lsb}([x/2 \bmod N\ ])$ sama dengan $2\text{sb}(x + N)$; yang terakhir sama dengan $1 \oplus 2\text{sb}(N) \oplus 2\text{sb}(x)$ (kita mempunyai bit carry di posisi kedua karena x dan N keduanya ganjil). Jadi jika kita menghitung $y' := [y/2^e \bmod N\ ]$, maka $2\text{sb}(x) = A(y') \oplus 1 \oplus 2\text{sb}(N)$.

Melanjutkan cara ini, kita dapat memulihkan semua bit x.

Mengenkripsi satu bit. Kita dapat menggunakan predikat hard-core yang diidentifikasi di atas untuk mengenkripsi satu bit. Idenya sederhana: untuk mengenkripsi pesan $m \in \{0,1\}$, pengirim memilih seragam $r \in \mathbb{Z}_N^*$ dengan batasan $\text{lsb}(r) = m$; teks sandinya adalah $c := [r^e \bmod N]$. Lihat Konstruksi 11.32.

CONSTRUCTION 11.32

Let GenRSA be as usual, and define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\text{GenRSA}(1^n)$ to obtain (N, e, d). Output the public key $pk = \langle N, e \rangle$, and the private key $sk = \langle N, d \rangle$.
- Enc: on input a public key $pk = \langle N, e \rangle$ and a message $m \in \{0,1\}$, choose a uniform $r \in \mathbb{Z}_N^*$ subject to the constraint that $\text{lsb}(r) = m$. Output the ciphertext $c := [r^e \bmod N]$.
- Dec: on input a private key $sk = \langle N, d \rangle$ and a ciphertext c, compute $r := [c^d \bmod N]$ and output $\text{lsb}(r)$.

TEOREMA 11.33 Jika masalah RSA relatif sulit terhadap GenRSA maka Konstruksi 11.32 aman terhadap CPA.

BUKTI Misalkan Π menyatakan Konstruksi 11.32. Kami membuktikan bahwa Π memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap; berdasarkan Proposisi 11.3, ini berarti aman untuk CPA.

Misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik. Tanpa kehilangan sifat umum, kita dapat mengasumsikan $m_0 = 0$ dan $m_1 = 1$ dalam eksperimen $\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n)$. Jadi

$$\begin{aligned}\Pr[\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathcal{A}(N, e, c) = 0 \mid c \text{ is an encryption of } 0] \\ + \frac{1}{2} \cdot \Pr[\mathcal{A}(N, e, c) = 1 \mid c \text{ is an encryption of } 1].\end{aligned}$$

Enkripsi bit tunggal menggunakan predikat hard-core untuk RSA.

Pertimbangkan untuk menjalankan $\mathcal{A}$ dalam eksperimen $\text{RSA} - \text{lsb}$. Menurut definisi,

$$\Pr[\text{RSA-lsb}_{\mathcal{A},\text{GenRSA}}(n) = 1] = \Pr[\mathcal{A}(N, e, [r^e \bmod N]) = \text{lsb}(r)],$$

dimana r seragam dalam $\mathbb{Z}_N^*$. Karena $\Pr[\text{lsb}(r) = 1] = 1/2$, kita punya

$$\begin{aligned}\Pr[\text{RSA-lsb}_{\mathcal{A},\text{GenRSA}}(n) = 1] = \frac{1}{2} \cdot \Pr[\mathcal{A}(N, e, [r^e \bmod N]) = 0 \mid \text{lsb}(r) = 0] \\ + \frac{1}{2} \cdot \Pr[\mathcal{A}(N, e, [r^e \bmod N]) = 1 \mid \text{lsb}(r) = 1].\end{aligned}$$

Memperhatikan bahwa mengenkripsi $m \in \{0,1\}$ sama persis dengan memilih seragam r dengan batasan bahwa $\text{lsb}(r) = m$, kita melihat bahwa

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \Pr[\mathsf{RSA\text{-}lsb}_{\mathcal{A},\mathsf{GenRSA}}(n) = 1].$$

Teorema 11.31 dengan demikian menyiratkan bahwa terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathsf{negl}(n),$$

seperti yang diinginkan.

Membangun KEM. Sekarang kita tunjukkan bagaimana cara memperluas Konstruksi 11.32 sehingga diperoleh KEM dengan panjang kunci n. Cara naif untuk melakukan hal ini adalah dengan memilih kunci n-bit k yang seragam dan kemudian mengenkripsi bit-bit k satu per satu menggunakan n pemanggilan Konstruksi 11.32. Hal ini akan menghasilkan ciphertext yang agak panjang yang terdiri dari n elemen $\mathbb{Z}_N$.

Pendekatan yang lebih baik adalah pengirim menerapkan permutasi RSA (yaitu, menaikkan modulo pangkat eth N) berulang kali, dimulai dari nilai awal yang seragam c_1. Artinya, pengirim akan menghitung c_1^e secara berturut-turut, diikuti oleh $(c_1^e)^e = c_1^{e^2}$, dan seterusnya, hingga $c_1^{e^n}$ (semua modulo N). Nilai akhir $[c_1^{e^n} \bmod N\ ]$ akan menjadi ciphertext, dan urutan bit $\text{lsb}(c_1), \text{lsb}(c_1^e), \ldots, \text{lsb}(c_1^{e^{n-1}})$ adalah kuncinya. Untuk mendekripsi ciphertext c, penerima hanya membalikkan proses ini, berturut-turut menghitung $c^d, (c^d)^d = c^{d^2}$ hingga c^{d^n} (sekali lagi, semua modulo N) untuk memulihkan nilai awal $c_1 = c^{d^n}$ yang digunakan oleh pengirim. Setelah memulihkan c_1, penerima kemudian dapat menghitung ulang $c_1^e, \cdots, c_1^{e^n}$ dan dapatkan kuncinya.

Dekripsi dapat diimplementasikan dengan lebih efisien dengan menggunakan fakta bahwa penerima mengetahui urutan grup $\mathbb{Z}_N^*$. Pada waktu pembuatan kunci, penerima dapat melakukan pra-komputasi $d' := [d^n \; mod \; \phi(N)]$ dan menyimpan d' sebagai bagian dari kunci privatnya. Kemudian, daripada menghitung n eksponen modular berturut-turut c^d, c^{d^2}, . . . untuk memperoleh c_1, penerima dapat langsung menghitung $c_1 := c^{d'} \bmod N$. Tentu saja ini berhasil

$$c^{d^n} \bmod N = c^{[d^n \bmod \phi(N)]} = c^{d'} \bmod N.$$

Di atas secara resmi digambarkan sebagai Konstruksi 11.34.

CONSTRUCTION 11.34

Let GenRSA be as usual, and define a KEM as follows:

- Gen: on input 1^n, run GenRSA(1^n) to obtain (N, e, d). Then compute $d' := [d^n \bmod \phi(N)]$ (note that $\phi(N)$ can be computed from (N, e, d) or obtained during the course of running GenRSA). Output $pk = \langle N, e\rangle$ and $sk = \langle N, d'\rangle$.
- Encaps: on input $pk = \langle N, e\rangle$ and 1^n, choose a uniform $c_1 \in \mathbb{Z}_N^*$. Then for $i = 1, \ldots, n$ do:
  1. Compute $k_i := \mathsf{lsb}(c_i)$.
  2. Compute $c_{i+1} := [c_i^e \bmod N]$.

  Output the ciphertext c_{n+1} and the key $k = k_1 \cdots k_n$.
- Decaps: on input $sk = \langle N, d'\rangle$ and a ciphertext c, compute $c_1 := [c^{d'} \bmod N]$. Then for $i = 1, \ldots, n$ do:
  1. Compute $k_i := \mathsf{lsb}(c_i)$.
  2. Compute $c_{i+1} := [c_i^e \bmod N]$.

  Output the key $k = k_1 \cdots k_n$.

KEM yang menggunakan predikat hard-core untuk RSA.

Konstruksinya mengingatkan pada pendekatan yang digunakan untuk membuat generator acak semu dari permutasi satu arah menjelang akhir Bagian 7.4. Jika kita membiarkan f menunjukkan permutasi RSA relatif terhadap beberapa kunci publik $\langle N, e\rangle$ (yaitu, $f(x) := [x^e \bmod N]$), maka keamanan CPA Konstruksi 11.34 setara dengan keacakan semu dari $\mathrm{lsb}(f^{n-1}(c_1)), \ldots, \mathrm{lsb}(c_1)$ bahkan dikondisikan pada nilai $c = f^n(c_1)$. Hal ini, selanjutnya, dapat dibuktikan dengan menggunakan Teorema 11.31 dan teknik dari Bagian 7.4. (Satu-satunya perbedaan adalah bahwa dalam Bagian 7.4 nilai $f^n(c_1)$ itu sendiri merupakan string n-bit yang seragam, sedangkan di sini merupakan elemen seragam dari $\mathbb{Z}_N^*$. Keacakan semu dari predikat hard-core yang berurutan tidak bergantung pada domain f.) Meringkas:

TEOREMA 11.35 Jika masalah RSA relatif sulit terhadap GenRSA maka Konstruksi 11.34 adalah KEM yang aman terhadap CPA.

Efisiensi. Konstruksi 11.34 cukup efisien. Untuk lebih konkretnya, asumsikan bahwa $n = 128$, modulus RSA N panjangnya 2048 bit, dan eksponen publik e adalah 3 sehingga eksponensial pangkat e modulo N dapat dihitung menggunakan dua perkalian modular. Enkripsi memerlukan $2n = 256$ perkalian modular. Dekripsi dapat dilakukan dengan satu eksponensial modular penuh (dengan biaya sekitar $1{,}5 \cdot 2048 = 3072$ perkalian modular) ditambah 256 perkalian modular tambahan. Oleh karena itu, biaya dekripsi hanya sekitar 8% lebih efisien dibandingkan skema enkripsi RSA biasa. Sebaliknya, enkripsi jauh lebih mahal dibandingkan dengan RSA biasa, namun dalam banyak aplikasi, waktu dekripsi jauh lebih

penting (karena dapat diterapkan oleh server yang melakukan ribuan dekripsi secara bersamaan).

OAEP dan RSA PKCS #1 v2.0

Sejauh ini kami belum mempertimbangkan skema enkripsi berbasis RSA yang aman terhadap CCA. Kami pertama-tama menunjukkan bahwa semua skema enkripsi berbasis RSA yang kami lihat sejauh ini rentan terhadap serangan teks sandi terpilih.

Enkripsi RSA biasa. RSA biasa bahkan tidak aman untuk CPA. Namun hal ini memastikan bahwa jika $m \in \mathbb{Z}_N^*$ seragam maka penyerang yang menguping enkripsi $c = [m^e \bmod N]$ dari m sehubungan dengan kunci publik $\langle N, e \rangle$ tidak dapat memulihkan m. Bahkan jaminan yang lemah ini tidak lagi berlaku dalam situasi di mana serangan teks sandi yang dipilih mungkin terjadi. Seperti dalam kasus enkripsi El Gamal, hal ini merupakan konsekuensi dari fakta bahwa RSA biasa dapat ditempa: mengingat enkripsi $c = [m^e \bmod N]$ dari pesan yang tidak diketahui m, mudah untuk menghasilkan ciphertext c' yang merupakan enkripsi $[2m \ mod \ N]$ dengan pengaturan

$$\begin{aligned} c' &:= [2^e \cdot c \bmod N] \\ &= 2^e \cdot m^e = (2m)^e \bmod N. \end{aligned}$$

RSA PKCS #1 v1.5. Enkripsi RSA berlapis, yang diperkirakan aman terhadap CPA jika pengaturan parameternya tepat, pada dasarnya rentan terhadap serangan yang sama seperti enkripsi RSA biasa. Namun ada juga serangan teks sandi terpilih yang lebih menarik pada enkripsi PKCS #1 v1.5 yang, berbeda dengan serangan yang disajikan di atas, tidak memerlukan akses penuh ke oracle dekripsi; itu hanya memerlukan akses ke oracle dekripsi "parsial" yang menunjukkan apakah dekripsi beberapa ciphertext menghasilkan kesalahan atau tidak. Hal ini membuat serangan menjadi jauh lebih praktis, karena serangan dapat dilakukan setiap kali penyerang dapat membedakan perilaku penerima setelah dekripsi berhasil dari perilakunya setelah kegagalan dekripsi, seperti dalam kasus serangan padding-Oracle yang ditunjukkan pada Bagian 3.7.

Ingatlah bahwa skema enkripsi kunci publik yang didefinisikan dalam standar PKCS #1 v1.5 menggunakan varian enkripsi RSA berlapis yang mana bantalannya dilakukan dengan cara tertentu. Khususnya, dua byte tingkat tinggi dari pesan berlapis selalu $0\text{x}00 \| 0\text{x}02$. Saat mendekripsi, penerima harus memeriksa apakah dua byte tingkat tinggi cocok dengan nilai-nilai ini, dan mengembalikan kesalahan jika tidak demikian. Pada tahun 1998, Bleichenbacher mengembangkan serangan teks sandi terpilih yang mengeksploitasi fakta bahwa pemeriksaan ini telah dilakukan. Secara kasar, dengan ciphertext c yang sesuai dengan enkripsi jujur dari beberapa pesan yang tidak diketahui m sehubungan dengan kunci publik $\langle N, e \rangle$, serangan berulang kali memilih seragam $s \in \mathbb{Z}_N^*$ dan mengirimkan ciphertext $c' := [s^e \cdot c \bmod N]$ ke penerima. Katakanlah $c = [\widehat{m}^e \bmod N]$ di mana

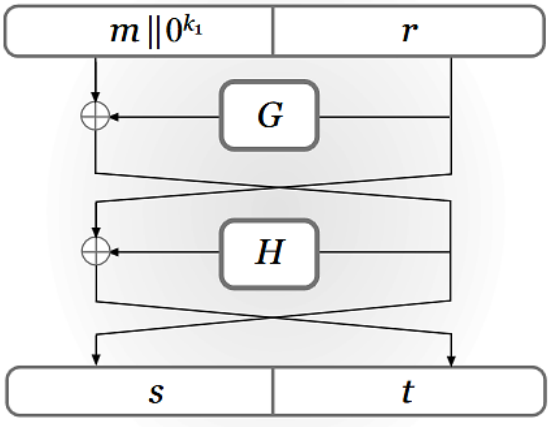
$$\widehat{m} = 0\text{x}00 \| 0\text{x}02 \| r \| 0\text{x}00 \| m$$

Sebagaimana ditentukan oleh PKCS #1 v1.5. Kemudian dekripsi c' akan memberikan hasil antara $\hat{m}' = [s \cdot \hat{m} \bmod N]$, dan penerima akan mengembalikan kesalahan kecuali dua byte teratas $\hat{m}'$ tepat $0\text{x}00\|0\text{x}02$. (Pemeriksaan lainnya juga dilakukan, namun kami mengabaikannya demi kesederhanaan.) Setiap kali dekripsi berhasil, penyerang mengetahui bahwa dua byte teratas dari $s \cdot \hat{m} \bmod N$ adalah $0\text{x}00\|0\text{x}02$, dimana s diketahui. Sejumlah persamaan jenis ini cukup bagi penyerang untuk mempelajari $\hat{m}$ dan memulihkan semua pesan asli m.

KEM yang aman dari CPA. Di Bagian 11.5 kami menunjukkan konstruksi KEM yang terbukti aman CPA berdasarkan asumsi RSA. Konstruksi tersebut juga tidak aman terhadap serangan teks sandi yang dipilih; kami meninggalkan detailnya sebagai latihan.

RSA-OAEP

Pada bagian ini kita mengeksplorasi konstruksi enkripsi aman CCA berbasis RSA menggunakan padding enkripsi asimetris optimal (OAEP). Skema RSA-OAEP yang dihasilkan mengikuti gagasan (digunakan juga di Bagian 11.5) untuk mengambil pesan m, mengubahnya menjadi elemen $\hat{m} \in \mathbb{Z}_N^*$, dan kemudian membiarkan $c = [\hat{m}^e \bmod N]$ menjadi teks tersandi. Namun transformasi di sini lebih kompleks dibandingkan sebelumnya. Versi RSA-OAEP telah distandarisasi sebagai bagian dari RSA PKCS #1 sejak versi 2.0.



GAMBAR 11.4: Mekanisme OAEP.

Misalkan $\ell(n)$, $k_0(n), k_1(n)$ adalah fungsi bernilai bilangan bulat dengan $k_0(n), k_1(n) = \Theta(n)$ dan sedemikian hingga $\ell(n) + k_0(n) + k_1(n)$ kurang dari panjang bit minimum keluaran moduli oleh $\text{GenRSA}(1^n)$. Perbaiki n, dan misalkan $\ell = \ell(n) = k_0 = k_0(n), dan\ k_1 = k_1(n)$. Misalkan $G : \{0,1\}^{k_0} \to \{0,1\}^{\ell + k_1}\ dan\ H : \{0,1\}^{\ell + k_1} \to \{0,1\}^{k_0}$ menjadi dua fungsi hash yang akan dimodelkan sebagai oracle acak independen. (Meskipun menggunakan lebih dari satu oracle acak tidak dibahas dalam Bagian 5.5.1, hal ini ditafsirkan secara alami.) Transformasi yang didefinisikan oleh OAEP adalah jaringan Feistel dua putaran dengan G dan H sebagai fungsi putaran; lihat Gambar 11.4. Secara detail, padding pesan $m \in \{0,1\}^{\ell}$ dilakukan sebagai berikut: set pertama $m' := m\|0^{k_1}$ dan pilih seragam $r \in \{0,1\}^{k_0}$. Kemudian hitung

$$s := m' \oplus G(r) \in \{0,1\}^{\ell+k_1}, \quad t := r \oplus H(s) \in \{0,1\}^{k_0},$$

dan atur $\hat{m} := s \| t$.

Untuk mengenkripsi pesan m sehubungan dengan kunci publik $\langle \mathrm{N}, \mathrm{e} \rangle$, pengirim menghasilkan $\hat{\mathrm{m}}$ seperti di atas dan mengeluarkan teks sandi $\mathrm{c} := \hat{\mathrm{m}}^e \bmod \mathrm{N}$. (Perhatikan bahwa $\hat{\mathrm{m}}$, ditafsirkan sebagai bilangan bulat, lebih kecil dari N karena batasan pada ℓ, k_0, k_1.) Untuk mendekripsi, penerima menghitung $\hat{\mathrm{m}} := [\mathrm{c}^d \bmod \mathrm{N}]$ dan membiarkan $\mathrm{s}|\mathrm{t} := \hat{\mathrm{m}}$ dengan s dan t dengan panjang yang sesuai. Kemudian membalikkan jaringan Feistel dengan menghitung $\mathrm{r} := \mathrm{H(s)} \oplus \mathrm{t}$ dan $\mathrm{m}' := \mathrm{G(r)} \oplus \mathrm{s}$. Yang penting, penerima kemudian memverifikasi bahwa k_1 bit tambahan dari m' semuanya 0; jika tidak, ciphertext ditolak dan pesan kesalahan dikembalikan. Jika tidak, k_1 yang paling tidak signifikan $0s\ dari\ m'$ akan dibuang, dan ℓ bit m' yang tersisa dikeluarkan sebagai pesan.

Bukti keamanan CCA untuk RSA-OAEP agak rumit, dan kami tidak memberikannya di sini. Sebaliknya, kami hanya memberikan sedikit intuisi. Pertama pertimbangkan keamanan CPA. Selama enkripsi, pengirim menghitung

$$m' := m \| 0^{k_1}, \quad s := m' \oplus G(r), \quad t := r \oplus H(s)$$

atau seragam r; teks sandinya adalah $[\ (s\|t)^e \bmod \mathrm{N}\ ]$. Jika penyerang tidak pernah menanyakan r ke G maka, karena kita memodelkan G sebagai fungsi acak, nilai G(r) seragam dari sudut pandang penyerang sehingga m ditutupi dengan string seragam seperti pada pad satu kali skema enkripsi. Jadi, jika penyerang tidak pernah menanyakan r ke G maka tidak ada informasi tentang pesan yang bocor.

Bisakah penyerang menanyakan r ke G? Perhatikan bahwa nilai r itu sendiri ditutupi oleh H(s). Jadi penyerang tidak mempunyai informasi tentang r kecuali ia terlebih dahulu menanyakan s ke H. Jika penyerang tidak menanyakan s ke H maka penyerang mungkin beruntung dan tetap menebak r, tetapi jika kita mengatur panjang r (i.e.., k_0) secukupnya lama maka kemungkinan hal ini dapat diabaikan.

Jadi, satu-satunya cara bagi penyerang untuk mempelajari sesuatu tentang m adalah dengan menanyakan s ke H terlebih dahulu. Hal ini akan mengharuskan penyerang untuk menghitung s dari ciphertext (seragam) $[\ (s\|t)^e \bmod \mathrm{N}\ ]$. Perhatikan bahwa komputasi s dari $[\ (s\|t)^e \bmod \mathrm{N}\ ]$ bukanlah masalah RSA, melainkan melibatkan komputasi s dan t. Namun demikian, untuk pengaturan parameter yang tepat kita dapat menggunakan Teorema 11.29 untuk menunjukkan bahwa pemulihan s memungkinkan pemulihan t dalam waktu polinomial, sehingga pemulihan s secara komputasi tidak mungkin dilakukan jika permasalahan RSA sulit.

Memperdebatkan keamanan CCA melibatkan komplikasi tambahan, namun ide dasarnya adalah untuk menunjukkan bahwa setiap permintaan oracle dekripsi c yang dibuat oleh penyerang termasuk dalam salah satu dari dua kategori: penyerang memperoleh c dengan mengenkripsi beberapa pesan m secara legal (dalam hal ini penyerang tidak belajar apa pun dari permintaan dekripsi), atau dekripsi c akan menghasilkan kesalahan. Hal ini merupakan konsekuensi dari fakta bahwa penerima memeriksa bahwa k_1 bit orde rendah $\hat{m}$

adalah 0 selama dekripsi; jika penyerang tidak membuat ciphertext c dengan mengenkripsi pesan secara legal, kemungkinan kondisi ini dapat diabaikan. Pembuktian formal diperumit oleh fakta bahwa pertanyaan dekripsi-Oracle penyerang harus dijawab dengan benar tanpa mengetahui kunci privat, yang berarti harus ada cara yang efisien untuk menentukan apakah akan mengembalikan kesalahan atau tidak dan, jika tidak, pesan apa yang harus dikembalikan. Hal ini dicapai dengan melihat pertanyaan musuh ke ramalan acak G, H.

CONSTRUCTION 11.36

Let GenRSA be as in the previous sections, and ℓ, k_0, k_1 be as described in the text. Let $G : \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$ and $H : \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$ be functions. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n, run GenRSA(1^n) to obtain (N, e, d). The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
- Enc: on input a public key $\langle N, e\rangle$ and a message $m \in \{0,1\}^\ell$, set $m' := m \| 0^{k_1}$ and choose a uniform $r \in \{0,1\}^{k_0}$. Then compute

$$s := m' \oplus G(r), \quad t := r \oplus H(s)$$

  and set $\hat{m} := s\|t$. Output the ciphertext $c := [\hat{m}^e \bmod N]$.
- Dec: on input a private key $\langle N, d\rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $\hat{m} := [c^d \bmod N]$. If $\|\hat{m}\| > \ell + k_0 + k_1$, output $\perp$. Otherwise, parse $\hat{m}$ as $s\|t$ with $s \in \{0,1\}^{\ell+k_1}$ and $t \in \{0,1\}^{k_0}$. Compute $r := H(s) \oplus t$ and $m' := G(r) \oplus s$. If the least-significant k_1 bits of m' are not all 0, output $\perp$. Otherwise, output the ℓ most-significant bits of $\hat{m}$.

Skema enkripsi RSA-OAEP.

Serangan teks sandi pilihan Manger pada PKCS #1 v2.0. Pada tahun 2001, James Manger menunjukkan serangan teks sandi terpilih terhadap implementasi tertentu dari skema enkripsi RSA yang ditentukan dalam PKCS #1 v2.0—meskipun yang ditentukan adalah varian RSA-OAEP! Karena Konstruksi 11.36 aman untuk CCA (dengan asumsi masalah RSA sulit), bagaimana mungkin?

Dengan memeriksa algoritme dekripsi dalam Konstruksi 11.36, perhatikan bahwa ada dua kemungkinan terjadinya kesalahan: $\hat{\mathrm{m}} \in Z_N^*$ terlalu besar, atau $\mathrm{m}' \in \{0,1\}^{\mathrm{A}+k_1}$ tidak memiliki angka 0s yang cukup. Dalam Konstruksi 11.36, penerima seharusnya mengembalikan kesalahan yang sama (dilambangkan $\perp$) dalam kedua kasus. Namun dalam beberapa implementasi, penerima akan mengeluarkan kesalahan yang berbeda tergantung pada langkah mana yang gagal. Sedikit informasi tambahan ini memungkinkan penyerang melakukan serangan teks sandi terpilih yang memulihkan pesan m secara keseluruhan dari enkripsi pesan tersebut, hanya menggunakan sekitar $\|\mathrm{N}\|$ kueri ke oracle yang membocorkan pesan kesalahan saat dekripsi. Hal ini menunjukkan pentingnya penerapan skema kriptografi persis seperti yang ditentukan, karena bukti dan analisis yang dihasilkan mungkin tidak berlaku lagi jika aspek skema diubah. Bahkan jika kesalahan yang sama dikembalikan pada

kedua kasus, penyerang dapat menentukan di mana kesalahan tersebut terjadi jika waktu untuk mengembalikan kesalahan tersebut berbeda. (Ini adalah contoh yang bagus tentang bagaimana seorang penyerang tidak hanya sekedar memeriksa masukan/keluaran suatu algoritma, namun dapat menggunakan informasi saluran samping untuk menyerang suatu skema.) Implementasinya harus hati-hati untuk memastikan bahwa waktu untuk mengembalikan kesalahan sudah tepat. identik terlepas dari di mana kesalahan terjadi.

KEM Aman CCA dalam Model Random-Oracle

Di sini kami tunjukkan konstruksi KEM berbasis RSA yang aman CCA dalam model random-oracle. (Ingat Teorema 11.14 bahwa konstruksi seperti itu dapat digunakan bersama dengan skema enkripsi kunci privat aman CCA untuk menghasilkan skema enkripsi kunci publik aman CCA.) Dibandingkan dengan skema RSA-OAEP dari skema sebelumnya keuntungan utamanya adalah kesederhanaan konstruksi dan bukti keamanannya. Kerugian utamanya adalah menghasilkan cipherteks yang lebih panjang ketika mengenkripsi pesan pendek karena memerlukan paradigma KEM/DEM sedangkan RSA-OAEP tidak. Namun, untuk mengenkripsi pesan yang panjang, RSA-OAEP juga akan digunakan sebagai bagian dari skema enkripsi hibrid, dan akan menghasilkan skema enkripsi yang memiliki efisiensi serupa dengan apa yang dapat diperoleh dengan menggunakan KEM yang ditunjukkan di sini.

KEM yang kami jelaskan disertakan sebagai bagian dari standar ISO/IEC 18033-2 untuk enkripsi kunci publik. Dalam skema tersebut, kunci publik mencakup ⟨N, e⟩ seperti biasa, dan fungsi $H : Z_N^* \rightarrow \{0,1\}^n$ ditentukan yang akan dimodelkan sebagai oracle acak dalam analisis. (Fungsi ini dapat didasarkan pada beberapa fungsi hash kriptografi yang mendasarinya, seperti yang dibahas dalam Bagian 5.5. Kami menghilangkan detailnya.) Untuk merangkum kunci, pengirim memilih seragam $r \in Z_N^*$ dan kemudian menghitung teks sandi $c := [r^e \ mod \ N]$ dan kuncinya $k := H(r)$. Untuk mendekripsi ciphertext c, penerima cukup memulihkan r dengan cara biasa dan kemudian mendapatkan kembali kunci *yang sama* $k := H(r)$. Lihat Konstruksi 11.37.

CONSTRUCTION 11.37

Let GenRSA be as usual, and construct a KEM as follows:

- Gen: on input 1^n, run GenRSA(1^n) to compute (N, e, d). The public key is $\langle N, e\rangle$, and the private key is $\langle N, d\rangle$.
  As part of key generation, a function $H : \mathbb{Z}_N^* \rightarrow \{0,1\}^n$ is specified, but we leave this implicit.
- Encaps: on input public key $\langle N, e\rangle$ and 1^n, choose a uniform $r \in \mathbb{Z}_N^*$. Output the ciphertext $c := [r^e \bmod N]$ and the key $k := H(r)$.
- Decaps: on input private key $\langle N, d\rangle$ and a ciphertext $c \in \mathbb{Z}_N^*$, compute $r := [c^d \bmod N]$ and output the key $k := H(r)$.

KEM yang aman dengan CCA (dalam model oracle acak).

Keamanan CPA pada skema ini bersifat langsung. Memang benar, ciphertext c sama dengan $[r^e \ mod \ N]$ untuk seragam $r \in Z_N^*$, sehingga asumsi RSA menyiratkan bahwa penyadap yang mengamati c tidak akan mampu menghitung r . Hal ini berarti, pada gilirannya,

bahwa penyadap tidak akan menanyakan r ke H, dan dengan demikian nilai kunci $k \stackrel{\text{def}}{=} H(r)$ tetap seragam dari sudut pandang penyerang.

Faktanya, hal di atas juga menunjukkan keamanan CCA. Hal ini karena menjawab kueri dekapsulasi-Oracle untuk teks tersandi apa pun $\tilde{c} \neq c$ hanya melibatkan evaluasi H pada beberapa masukan $[\tilde{c^d} \ mod \ N] = \tilde{c} \neq r$. Dengan demikian, kueri oracle dekapsulasi penyerang tidak mengungkapkan informasi tambahan apa pun tentang kunci H(r) yang dienkapsulasi oleh teks sandi tantangan. (Pembuktian formal sedikit lebih rumit karena kita harus menunjukkan bagaimana mungkin untuk mensimulasikan jawaban terhadap pertanyaan dekapsulasi-Oracle tanpa mengetahui kunci privat. Namun demikian, hal ini ternyata tidak terlalu sulit.)

TEOREMA 11.38 Jika masalah RSA relatif sulit terhadap GenRSA dan H dimodelkan sebagai oracle acak, maka Konstruksi 11.37 aman untuk CCA.

BUKTI Misalkan Π menyatakan Konstruksi 11.37, dan misalkan A adalah suatu probabilistik musuh waktu polinomial. Demi kenyamanan, dan karena ini adalah bukti pertama di mana kami menggunakan kekuatan penuh model oracle acak, kami menjelaskan secara eksplisit langkah-langkah eksperimen $\mathrm{KEM}^{cca}_{\mathcal{A}\Pi}(\mathrm{n})$:

1. GenRSA(1^n) dijalankan untuk mendapatkan $(\mathrm{N}, \mathrm{e}, \mathrm{d})$. Selain itu, fungsi acak $\mathrm{H} : \mathrm{Z} * \mathrm{N} \rightarrow \{0,1\}\mathrm{n}$ dipilih.
2. Seragam $\mathrm{r} \in \mathrm{Z} * \mathrm{N}$ dipilih, dan ciphertext $\mathrm{c} := [\mathrm{re} \bmod \mathrm{N}]$ dan kunci $\mathrm{k} := \mathrm{H}(\mathrm{r})$ dihitung.
3. Bit seragam $\mathrm{b} \in \{0,1\}$ dipilih. Jika $\mathrm{b} = 0$ himpunan $\hat{k} := \mathrm{k}$. Jika $\mathrm{b} = 1$ maka pilihlah seragam $\hat{k} \in \{0,1\}\mathrm{n}$.
4. $\mathcal{A}$ diberikan $\mathrm{pk} = \langle \mathrm{N}, \mathrm{e} \rangle$, c, dan $\hat{k}$, dan dapat menanyakan $\mathrm{H}(\cdot)$ (pada input apa pun) dan oracle dekapsulasi $\mathrm{Decaps}_{\langle \mathrm{N},\mathrm{d} \rangle}(\cdot)$ pada ciphertext mana pun $\hat{c}$ 6 $=$ c .
5. $\mathcal{A}$ mengeluarkan sedikit b'. Keluaran percobaan didefinisikan sebagai 1 jika $\mathrm{b}' = \mathrm{b}$, dan 0 jika tidak.

Dalam pelaksanaan eksperimen $\mathrm{KEM}^{cca}_{\mathcal{A}\Pi}(\mathrm{n})$, misalkan Query adalah peristiwa yang, pada titik mana pun selama pelaksanaannya, $\mathcal{A}$ menanyakan r ke oracle acak H. Kita biarkan Sukses menyatakan peristiwa yang b' = b (yaitu, hasil percobaan 1). Kemudian

$$\begin{aligned} \Pr[\text{Success}] &= \Pr\left[\text{Success} \wedge \overline{\text{Query}}\right] + \Pr[\text{Success} \wedge \text{Query}] \\ &\leq \Pr\left[\text{Success} \wedge \overline{\text{Query}}\right] + \Pr[\text{Query}], \end{aligned}$$

dimana semua probabilitas diambil alih keacakan yang digunakan dalam percobaan $\mathrm{KEM}^{cca}_{\mathcal{A}\Pi}(\mathrm{n})$. Kami menunjukkan bahwa $\Pr[\text{Sukses} \wedge \overline{\text{Kueri}}] \leq \frac{1}{2}$ dan $Pr[\overline{Kueri}]$ dapat diabaikan. Teorema berikut. Kami pertama-tama berpendapat bahwa $\Pr[\text{Success} \wedge \overline{\text{Query}}] \leq \frac{1}{2}$. Jika $\Pr[\overline{\text{Query}}] = 0$, ini akan langsung terjadi. Jika tidak, $\Pr[\text{Sukses} \wedge$

$\overline{\text{Kueri}}] \leq \Pr[\text{Sukses}|\overline{\text{Kueri}}]$. Sekarang, dikondisikan pada Query, nilai kunci yang benar $k = H(r)$ adalah seragam karena H adalah fungsi acak. Perhatikan informasi $\mathcal{A}$ tentang k dalam percobaan $\text{KEM}_{\mathcal{A}\Pi}^{cca}(n)$. Kunci publik pk dan ciphertext c, dengan sendirinya, tidak mengandung informasi apapun tentang k. (Mereka menentukan r secara unik, tetapi karena H dipilih secara independen, hal ini tidak memberikan informasi tentang $H(r)$.)

Kueri yang dibuat $\mathcal{A}$ ke H juga tidak mengungkapkan informasi apa pun tentang r, kecuali $\mathcal{A}$ menanyakan $r\ ke\ H$ (dalam hal ini Kueri terjadi); ini, sekali lagi, bergantung pada fakta bahwa H adalah fungsi acak. Terakhir, pertanyaan yang dibuat $\mathcal{A}$ terhadap oracle dekapsulasinya hanya mengungkapkan $H(\tilde{r})$ untuk $\tilde{r} = r$. Hal ini mengikuti fakta bahwa $\text{Decaps}_{\langle N,d\rangle}(\tilde{c}) = H(\tilde{r})$ di mana $\tilde{r} = [\tilde{c}^{\sim d} \bmod N]$, tetapi $\tilde{c} = c$ menyiratkan $\tilde{r} = r$. Sekali lagi, ini dan fakta bahwa H adalah fungsi acak berarti bahwa tidak ada informasi tentang $H(r)$ yang terungkap kecuali Query terjadi.

Gambar di atas menunjukkan bahwa, selama Query tidak terjadi, nilai kunci k yang benar adalah seragam bahkan dengan pandangan A terhadap kunci publik, teks tersandi, dan dia menjawab semua pertanyaan oraclenya. Maka dalam hal ini, A tidak mungkin dapat membedakan (yang lebih baik daripada menebak secara acak) apakah k adalah kunci yang benar atau merupakan kunci yang seragam dan independen. Oleh karena itu, $\Pr[\text{Sukses}|\overline{\text{Kueri}}] = \frac{1}{2}$.

Kami menyoroti bahwa dalam argumen di atas kami tidak mengandalkan fakta bahwa A dibatasi secara komputasi, dan pada kenyataannya $\Pr[\text{Sukses} \wedge \overline{\text{Kueri}}] \leq \frac{1}{2}$ bahkan jika tidak ada batasan komputasi yang diterapkan pada A. Hal ini menunjukkan sebagian dari kekuatan dari model oracle acak. Untuk melengkapi pembuktian teorema, kami tunjukkan

KLAIM 11.39 Jika permasalahan RSA relatif sulit terhadap GenRSA dan H dimodelkan sebagai oracle acak, maka Pr[Query] dapat diabaikan.

Untuk membuktikannya, kami membuat algoritma $\mathcal{A}'$yang menggunakan $\mathcal{A}$ sebagai subrutin. $\mathcal{A}'$ diberikan sebuah contoh N, e, c dari masalah RSA, dan tujuannya adalah untuk menghitung r yang $re = c\ mod\ N$. Untuk melakukannya, ia akan menjalankan $\mathcal{A}$, menjawab pertanyaannya ke H dan Decaps. Menangani kueri ke H itu sederhana, karena $\mathcal{A}'$ nya dapat mengembalikan nilai acak. Namun, kueri ke Decaps lebih rumit karena $\mathcal{A}'$ tidak mengetahui kunci privat yang terkait dengan kunci publik efektif $\langle N, e\rangle$.

Namun, jika dipikir lebih jauh, pertanyaan dekapsulasi juga mudah dijawab karena $\mathcal{A}'$ juga dapat mengembalikan nilai acak di sini. Artinya, meskipun kueri $\text{Decaps}(\tilde{c})$ seharusnya dihitung dengan terlebih dahulu menghitung $\tilde{r}$sehingga $\tilde{r}^e = \tilde{c} \bmod N$ dan kemudian mengevaluasi $H(\tilde{r})$, hasilnya hanyalah nilai seragam. Jadi, $\mathcal{A}'$ dapat dengan mudah mengembalikan nilai acak tanpa melakukan perhitungan perantara. Satu-satunya "tangkapan" adalah bahwa $\mathcal{A}'$ harus memastikan konsistensi antara jawabannya terhadap pertanyaan-H dan pertanyaan-pertanyaan Decaps; yaitu, ia harus memastikan bahwa untuk setiap $\tilde{r}$, $\tilde{c}$ dengan $\tilde{r}^e = \tilde{c} \bmod N$ ia menyatakan bahwa $H(\tilde{r})$ $\text{Decaps}(\tilde{c})$ Hal ini ditangani

dengan menggunakan pembukuan sederhana dan mencantumkan LH dan LDecaps yang melacak jawaban yang diberikan $\mathcal{A}'$ sebagai tanggapan terhadap pertanyaan oracle masing-masing. Kami sekarang memberikan rinciannya.

Algoritma $\mathcal{A}'$:
Algoritma diberikan $(\mathrm{N}, \mathrm{e}, \mathrm{c})$ sebagai masukan.

1. Inisialisasi daftar kosong L_H, $\mathrm{L}_{\mathrm{Decaps}}$. pilih seragam $\mathrm{k} \in \{0,1\}^n$ dan simpan (c, k) di $\mathrm{L}_{\mathrm{Decaps}}$.
2. Pilih bit yang seragam $\mathrm{b} \in \{0,1\}$. Jika $\mathrm{b} = 0$ himpunan $\hat{\mathrm{k}} := \mathrm{k}$. Jika $\mathrm{b} = 1$ maka pilihlah seragam $\hat{\mathrm{k}} \in \{0,1\}^n$. Jalankan $\mathcal{A}$ pada $\langle \mathrm{N}, \mathrm{e} \rangle$, c, dan $\hat{\mathrm{k}}$.

Ketika $\mathcal{A}$ membuat pertanyaan $H(\tilde{r})$, jawablah sebagai berikut:

- Jika ada entri di LH dalam bentuk $(\tilde{\mathrm{r}}, \mathrm{k})$ untuk beberapa k, kembalikan k.
- Jika tidak, biarkan $\tilde{\mathrm{c}} := [\tilde{\mathrm{r}}^e \bmod \mathrm{N}]$. Jika ada entri dalam $\mathrm{L}_{\mathrm{Decaps}}$ dalam bentuk $(\tilde{\mathrm{c}}, \mathrm{k})$ untuk beberapa k, kembalikan k dan simpan $(\tilde{\mathrm{r}}, \mathrm{k})$ di L_H .
- Jika tidak, pilih seragam $\mathrm{k} \in \{0,1\}^n$, kembalikan k, dan simpan $(\tilde{}\mathrm{r}, \mathrm{k})$ di L_H .

Saat $\mathcal{A}$ membuat kueri $\mathrm{Decaps}(\tilde{\mathrm{c}})$, jawablah sebagai berikut:

- Jika ada entri dalam $\mathrm{L}_{\mathrm{Decaps}}$ dalam bentuk $(\tilde{\mathrm{c}}, \mathrm{k})$ untuk beberapa k, kembalikan k.
- Jika tidak, untuk setiap entri $(\tilde{r}, \mathrm{k}) \in \mathrm{L}_H$, periksa apakah $\tilde{\mathrm{r}}^e = \tilde{\mathrm{c}} \bmod \mathrm{N}$ dan, jika demikian, keluaran k.
- Jika tidak, pilih seragam $\mathrm{k} \in \{0,1\}^n$, kembalikan k, dan simpan $(\tilde{\mathrm{c}}, \mathrm{k})$ di LDecaps.

Di akhir eksekusi $\mathcal{A}$, jika ada entri (r, k) di LH yang $\mathrm{r}^e = \mathrm{c} \bmod \mathrm{N}$ maka kembalikan r.

Jelasnya $\mathcal{A}'$ berjalan dalam waktu polinomial, dan tampilan A ketika dijalankan sebagai subrutin oleh $\mathcal{A}'$ dalam eksperimen $RSA - inv_{A',GenRSA}(n)$ identik dengan tampilan $\mathcal{A}$ dalam eksperimen $\mathrm{KEM}_{\mathcal{A}\Pi}^{cca}(\mathrm{n})$: masukan yang diberikan kepada A jelas memiliki distribusi yang benar, jawaban terhadap pertanyaan oracle $\mathcal{A}$ konsisten, dan tanggapan terhadap semua pertanyaan$-H$ seragam dan independen. Akhirnya, $\mathcal{A}'$ mengeluarkan solusi yang benar tepat ketika Query muncul. Kekerasan masalah RSA relatif terhadap $GenRSA$ menyiratkan bahwa $Pr[Query]$ dapat diabaikan, sesuai kebutuhan.

Perlu diperhatikan berbagai properti model oracle acak (lihat Bagian 5.5) yang digunakan dalam pembuktian di atas. Pertama, kita mengandalkan fakta bahwa nilai H(r) seragam kecuali r ditanyakan ke H bahkan jika H ditanyakan pada beberapa nilai lain ˜r 6 = r. Kami juga, secara implisit, menggunakan kemampuan ekstrak untuk menyatakan bahwa penyerang tidak dapat menanyakan r ke H; jika tidak, kita dapat menggunakan penyerang ini untuk memecahkan masalah RSA. Terakhir, pembuktiannya bergantung pada kemampuan program untuk mensimulasikan kueri dekapsulasi-oracle musuh.

Masalah dan Kendala Implementasi RSA

Kami menutup bagian ini dengan diskusi singkat tentang beberapa permasalahan terkait implementasi skema berbasis RSA, dan beberapa kendala yang harus diperhatikan. Menggunakan sisa bahasa Cina. Dalam implementasi enkripsi berbasis RSA, penerima dapat menggunakan teorema sisa Cina (Bagian 8.1) untuk mempercepat komputasi akar eth modulo N selama dekripsi. Secara khusus, misalkan N = pq dan katakanlah penerima ingin menghitung

akar eth dari beberapa nilai y menggunakan d = [e−1 mod ϕ(N)]. Penerima dapat menggunakan korespondensi [yd mod N] ↔ ([yd mod p], [yd mod q]) untuk menghitung hasil parsial

$$x_p := [y^d \bmod p] = \left[y^{[d \bmod (p-1)]} \bmod p\right] \qquad (11.19)$$

Dan

$$x_q := [y^d \bmod q] = \left[y^{[d \bmod (q-1)]} \bmod q\right], \qquad (11.20)$$

lalu gabungkan keduanya untuk mendapatkan $x \leftrightarrow (x_p, x_q)$, seperti yang dibahas di Bagian 8.1.5. Perhatikan bahwa $[d\ mod\ (p-1)]\ dan\ [d\ mod\ (q-1)]$ dapat dihitung terlebih dahulu karena tidak bergantung pada y.

Mengapa ini lebih baik? Asumsikan modul eksponensial sebuah bilangan bulat `-bit membutuhkan $\gamma \cdot \ell^3$ operasi untuk beberapa konstanta γ. Jika p, q panjangnya masing-masing n bit, maka komputasi naif $[y^d\ mod\ N]$ membutuhkan $\gamma \cdot (2n)^3 = 8\gamma \cdot n^3$ langkah ($karena\ \|N\| = 2n$). Menggunakan sisa bahasa Mandarin akan menguranginya menjadi sekitar $2 \cdot (\gamma \cdot n^3)$ langkah (karena ‖p‖ = ‖q‖ = n), atau kira-kira 1/4 waktunya.

Contoh 11.40

Kita meninjau kembali Contoh 8.49. Ingatlah bahwa $N = 143 = 11 \cdot 13$ dan $d = 103$, dan $y = 64$. Untuk menghitung $[64^{103}\ mod\ 143]$ kita menghitung

$$\begin{aligned}([64 \bmod 11],\ [64 \bmod 13])^{103} &= ([(-2)^{103} \bmod 11],\ [(-1)^{103} \bmod 13]) \\ &= \left([(-2)^{[103 \bmod 10]} \bmod 11],\ -1\right) \\ &= ([-8 \bmod 11],\ -1) = (3,\ -1).\end{aligned}$$

Kita dapat menghitung $1p = 78 \leftrightarrow (1,0)\ dan\ 1q = 66 \leftrightarrow (0,1)$, seperti yang dibahas di Bagian 8.1.5. (Perhatikan bahwa nilai-nilai ini dapat dihitung terlebih dahulu, karena tidak bergantung pada y.) Maka $(3,-1) \leftrightarrow 3 \cdot 1_p - 1_q = 3 \cdot 78 - 66 = 168 = 25\ mod\ 143$, sesuai dengan jawaban sebelumnya diperoleh.

Serangan kesalahan saat menggunakan sisa bahasa Mandarin. Saat menggunakan sisa bahasa Mandarin seperti yang baru saja dijelaskan, seseorang harus waspada terhadap potensi serangan yang dapat dilakukan jika terjadi kesalahan (atau dapat disebabkan oleh penyerang, misalnya dengan gangguan perangkat keras) selama proses komputasi.

Pertimbangkan apa yang terjadi jika $[y^d\ mod\ N]$ dihitung dua kali: pertama kali tanpa kesalahan (memberikan hasil x yang benar), tetapi kedua kalinya dengan kesalahan saat menghitung Persamaan (11.20) tetapi tidak pada Persamaan (11.19) (sama serangan berlaku dalam kasus sebaliknya). Perhitungan kedua menghasilkan hasil yang salah x' dimana $x' =$

$x \bmod p$ *tetapi* $x' = x \bmod q$. Artinya $p \mid (x' - x)$ *tetapi* $q \mid (x' - x)$. Tapi *kemudian* $gcd(x' - x, N) = p$, menghasilkan faktorisasi N .

Salah satu tindakan pencegahan yang mungkin dilakukan adalah memverifikasi kebenaran hasil sebelum menggunakannya, dengan memeriksa bahwa $x^e = y \bmod N$. (Sejak $\|e\| \leqslant \|d\|$, menggunakan sisa bahasa Mandarin masih memberikan efisiensi yang lebih baik.) Hal ini direkomendasikan dalam implementasi perangkat keras.

Kunci publik yang bergantung I. Ketika beberapa penerima ingin menggunakan skema enkripsi yang sama, mereka harus menggunakan kunci publik yang independen. Serangan ini dan serangan berikutnya menunjukkan apa yang bisa menjadi salah jika hal ini tidak dilakukan.

Bayangkan sebuah perusahaan ingin menggunakan modulus N yang sama untuk setiap karyawannya. Karena pesan yang dienkripsi ke satu karyawan tidak diinginkan untuk dibaca oleh karyawan lain, perusahaan mengeluarkan pasangan (e_i, d_i) yang berbeda untuk setiap karyawan. Artinya, kunci publik karyawan ke i adalah $pk_i = \langle N, e_i \rangle$ dan kunci privatnya adalah $sk = \langle N, d_i \rangle$, dengan $e_i \cdot d_i = 1 \bmod \emptyset(N)$ untuk semua i.

Pendekatan ini tidak aman dan memungkinkan setiap karyawan membaca pesan terenkripsi ke semua karyawan lainnya. Alasannya adalah, seperti disebutkan dalam Bagian 8.2.4, dengan mengetahui N dan ei, di dengan ei · di = 1 mod ϕ(N), faktorisasi N dapat dihitung secara efisien. Mengingat faktorisasi N , tentu saja, dimungkinkan untuk menghitung $dj := e - 1j \bmod \varphi(N)$ untuk j apa pun.

Kunci publik dependen II. Serangan yang baru saja ditampilkan memungkinkan setiap karyawan untuk mendekripsi pesan yang dikirim ke karyawan lain. Hal ini masih menyisakan kemungkinan bahwa berbagi modulus N diperbolehkan selama semua karyawan saling percaya (atau, alternatifnya, selama kerahasiaan hanya perlu dijaga terhadap pihak luar tetapi tidak terhadap anggota perusahaan lainnya). Di sini kami menunjukkan skenario yang menunjukkan bahwa berbagi modulus masih merupakan ide buruk, setidaknya ketika enkripsi RSA biasa digunakan.

Katakanlah pesan yang sama m dienkripsi dan dikirim ke dua karyawan berbeda (dikenal) dengan kunci publik (N, e_1) dan (N, e_2) di mana $e_1 \neq e_2$. Asumsikan lebih lanjut bahwa $gcd(e_1, e_2) = 1$. Kemudian seorang penyadap melihat dua teks tersandi

$$c_1 = m^{e_1} \bmod N \quad \text{and} \quad c_2 = m^{e_2} \bmod N.$$

Karena $gcd(e_1, e_2) = 1$, maka terdapat bilangan bulat X, Y sehingga $Xe_1 + Ye_1 = 1$ dengan Proposisi 8.2. Selain itu, mengingat eksponen publik $e1$ dan $e2$, maka dimungkinkan untuk menghitung X dan Y secara efisien menggunakan algoritma Euclidean yang diperluas (lihat Lampiran B.1.2). Kami mengklaim bahwa $m = [c_1^X \cdot c_2^Y \bmod N]$, yang dapat dihitung dengan mudah. Ini benar karena

$$c_1^X \cdot c_2^Y = m^{Xe_1} m^{Ye_2} = m^{Xe_1 + Ye_2} = m^1 = m \bmod N.$$

Serangan serupa berlaku saat menggunakan RSA empuk atau RSA-OAEP jika pengirim menggunakan pesan transformasi yang sama mˆ saat mengenkripsi ke dua pengguna.

Kualitas keacakan dalam pembuatan kunci RSA. Sepanjang buku ini, kami selalu berasumsi bahwa pihak yang jujur mempunyai akses terhadap keacakan yang cukup dan berkualitas tinggi. Jika asumsi ini dilanggar maka keamanan mungkin gagal dipertahankan.

Khususnya, jika string `-bit dipilih dari beberapa himpunan $S \subset \{0,1\}^{\ell`}$ dan bukannya dipilih secara seragam dari $\{0,1\}^{\ell}$, maka penyerang dapat melakukan penelusuran brute-force (dalam waktu $\mathcal{O}(|S|)$) untuk menyerang sistem. Dalam beberapa kasus, situasinya mungkin lebih buruk lagi. Pertimbangkan khususnya kasus pembangkitan kunci RSA, di mana satu sampel bit acak r_P digunakan untuk memilih P prima pertama, dan sampel kedua r_q digunakan untuk menghasilkan q prima kedua. Asumsikan lebih lanjut bahwa banyak kunci publik/pribadi dihasilkan menggunakan sumber keacakan berkualitas buruk yang sama, di mana r_P, r_q dipilih secara seragam dari beberapa himpunan S berukuran 2^s. Setelah menghasilkan kunci publik kira-kira $2^{s/2}$(lihat Lampiran A.4), kita berharap memperoleh dua moduli N, N ’ berbeda yang dihasilkan menggunakan keacakan identik $r_P = r'_P$. Kedua modulus ini mempunyai faktor prima yang sama yang dapat dengan mudah dicari dengan menghitung gcd(N, N ’). Dengan demikian, seorang penyerang dapat mencari sejumlah besar kunci publik RSA di Internet, menghitung Gcd berpasangannya, dan berharap untuk memfaktorkan beberapa subset dari kunci tersebut. Meskipun menghitung gcd berpasangan sebesar $2^{s/2}$ modulus secara naif memerlukan waktu $\mathcal{O}(2^s)$, ternyata hal ini dapat ditingkatkan secara signifikan dengan menggunakan pendekatan “divide-and-conquer” yang berada di luar cakupan buku ini. Hasilnya adalah penyerang dapat melakukan pencarian brute force dalam waktu kurang dari 2 detik. Terlebih lagi, serangan tersebut berhasil meskipun set S tidak diketahui oleh penyerang!

Skenario di atas telah diverifikasi secara eksperimental oleh dua tim peneliti yang bekerja secara independen, yang melakukan serangan seperti di atas terhadap kunci publik yang diambil melalui Internet dan berhasil memfaktorkan sebagian besar kunci yang mereka temukan.

BAB 12
SKEMA TANDA TANGAN DIGITAL

12.1 TANDA TANGAN DIGITAL

Pada bab sebelumnya kita mengeksplorasi bagaimana enkripsi kunci publik dapat digunakan untuk mencapai kerahasiaan dalam pengaturan kunci publik. Integritas (atau keaslian) dalam pengaturan kunci publik disediakan dengan menggunakan skema tanda tangan digital. Ini dapat dilihat sebagai analogi kunci publik dari kode otentikasi pesan, meskipun, seperti yang akan kita lihat, ada beberapa perbedaan penting antara primitif ini. Skema tanda tangan memungkinkan penandatangan S yang telah membuat kunci publik pk untuk "menandatangani" pesan menggunakan kunci pribadi sk yang terkait sedemikian rupa sehingga siapa pun yang mengetahui pk (dan mengetahui bahwa kunci publik ini dibuat oleh S) dapat memverifikasi bahwa pesan berasal dari S dan tidak diubah saat transit. (Perhatikan bahwa, berbeda dengan enkripsi kunci publik, dalam konteks tanda tangan digital, pemilik kunci publik bertindak sebagai pengirim.) Sebagai aplikasi prototipikal, pertimbangkan perusahaan perangkat lunak yang ingin menyebarkan pembaruan perangkat lunak dengan cara yang diautentikasi; yaitu, ketika perusahaan merilis pembaruan, kliennya harus dapat memverifikasi bahwa pembaruan tersebut asli, dan pihak ketiga yang jahat tidak boleh dapat menipu klien agar menerima pembaruan yang sebenarnya tidak dirilis oleh perusahaan. perusahaan.

Untuk melakukan hal ini, perusahaan dapat menghasilkan pk kunci publik bersama dengan sk kunci pribadi, dan kemudian mendistribusikan pk dengan cara yang dapat diandalkan kepada kliennya sambil menjaga rahasia sk. (Seperti dalam kasus enkripsi kunci publik, kami berasumsi bahwa distribusi awal kunci publik ini dilakukan dengan benar sehingga semua klien memiliki salinan pk yang benar. Dalam contoh saat ini, pk dapat digabungkan dengan perangkat lunak asli yang dibeli oleh klien.) Saat merilis pembaruan perangkat lunak m, perusahaan menghitung tanda tangan digital σ pada m menggunakan kunci pribadinya sk, dan mengirimkan (m, σ) ke setiap klien. Setiap klien dapat memverifikasi keaslian m dengan memeriksa bahwa σ adalah tanda tangan yang benar pada m sehubungan dengan kunci publik pk.

Pihak jahat mungkin mencoba mengeluarkan pembaruan palsu dengan mengirimkan (m', σ') ke klien, di mana m' mewakili pembaruan yang tidak pernah dirilis oleh perusahaan. Ini mungkin merupakan versi modifikasi dari beberapa pembaruan sebelumnya, atau mungkin benar-benar baru dan tidak terkait dengan pembaruan sebelumnya. Namun, jika skema tanda tangan "aman" (dalam artian kami akan segera mendefinisikannya dengan lebih hati-hati), maka ketika klien mencoba memverifikasi σ' ia akan menemukan bahwa ini adalah tanda tangan yang tidak valid pada m' sehubungan dengan pk, dan akan oleh karena itu tolak tanda tangannya. Klien akan menolak meskipun m' dimodifikasi hanya sedikit dari pembaruan asli m. Hal di atas bukan hanya sekedar penerapan teori tanda tangan digital, namun yang digunakan secara luas saat ini untuk mendistribusikan pembaruan perangkat lunak.

Perbandingan dengan Kode Otentikasi Pesan

Kode otentikasi pesan dan skema tanda tangan digital digunakan untuk memastikan integritas pesan yang dikirimkan. Meskipun pembahasan pada Bab 10 yang membandingkan pengaturan kunci publik dan kunci privat berfokus terutama pada enkripsi, pembahasan tersebut juga berlaku pada integritas pesan. Menggunakan tanda tangan digital dibandingkan kode autentikasi pesan akan menyederhanakan distribusi dan pengelolaan kunci, terutama ketika pengirim perlu berkomunikasi dengan banyak penerima seperti pada contoh pembaruan perangkat lunak di atas. Dengan menggunakan skema tanda tangan digital, pengirim tidak perlu membuat kunci rahasia yang berbeda dengan masing-masing calon penerima, dan menghindari keharusan menghitung tag MAC terpisah untuk setiap kunci tersebut. Sebaliknya, pengirim hanya perlu menghitung satu tanda tangan yang dapat diverifikasi oleh semua penerima.

Keuntungan kualitatif yang dimiliki tanda tangan digital dibandingkan dengan kode otentikasi pesan adalah bahwa tanda tangan dapat diverifikasi secara publik. Artinya jika penerima memverifikasi bahwa tanda tangan pada pesan tertentu adalah sah, maka semua pihak lain yang menerima pesan yang ditandatangani ini juga akan memverifikasi keasliannya. Fitur ini tidak dicapai dengan kode otentikasi pesan jika penandatangan berbagi kunci terpisah dengan masing-masing penerima: dalam pengaturan seperti itu, pengirim yang jahat mungkin menghitung tag MAC yang benar sehubungan dengan kunci yang dibagikannya dengan penerima A tetapi tag MAC yang salah sehubungan dengan itu. ke kunci yang dibagikannya dengan pengguna B yang berbeda. Dalam hal ini, A mengetahui bahwa ia menerima pesan asli dari pengirim tetapi tidak memiliki jaminan bahwa B akan menyetujuinya.Verifikasi publik menyiratkan bahwa tanda tangan dapat dipindahtangankan: tanda tangan σ pada pesan m oleh penandatangan S dapat ditunjukkan kepada pihak ketiga, yang kemudian dapat memverifikasi dirinya sendiri bahwa σ adalah tanda tangan sah pada m sehubungan dengan kunci publik S (di sini, kita asumsikan pihak ketiga ini juga mengetahui kunci publik S). Dengan membuat salinan tanda tangan, pihak ketiga ini kemudian dapat menunjukkan tanda tangan tersebut kepada pihak lain dan meyakinkan mereka bahwa S mengautentikasi m, dan seterusnya. Verifikasi publik dan kemampuan transfer sangat penting dalam penerapan tanda tangan digital pada sertifikat dan infrastruktur kunci publik, seperti yang akan kita bahas di Bagian 12.7. Skema tanda tangan digital juga memberikan sifat non-penyangkalan yang sangat penting. Ini berarti bahwa ketika S menandatangani sebuah pesan, dia tidak dapat menyangkal telah melakukannya (dengan asumsi kunci publik dari S dipublikasikan dan didistribusikan secara luas). Aspek tanda tangan digital ini sangat penting untuk penerapan hukum di mana penerima mungkin perlu membuktikan kepada pihak ketiga (misalnya hakim) bahwa penandatangan memang "mensertifikasi" pesan tertentu (misalnya kontrak): dengan asumsi publik S kuncinya diketahui oleh hakim, atau tersedia untuk umum, tanda tangan yang sah pada suatu pesan menjadi bukti yang meyakinkan bahwa S memang menandatangani pesan tersebut. Kode otentikasi pesan tidak bisa memberikan non-penyangkalan. Untuk melihatnya, katakanlah pengguna S dan R berbagi kunci kSR, dan S mengirimkan pesan m ke R bersama dengan tag MAC (valid) t yang dihitung menggunakan kunci ini. Karena juri tidak

mengetahui kSR (memang kunci ini dirahasiakan oleh S dan R), maka tidak ada cara bagi juri untuk menentukan valid atau tidaknya t. Jika R mengungkapkan kunci kSR kepada juri, tidak ada cara bagi juri untuk mengetahui apakah ini adalah kunci "sebenarnya" yang dibagikan oleh S dan R, atau apakah itu adalah kunci "palsu" yang dibuat oleh R. Akhirnya , bahkan jika kita berasumsi bahwa juri dapat memperoleh kunci kSR sebenarnya yang dibagikan oleh para pihak, tidak ada cara bagi juri untuk membedakan apakah S menghasilkan t atau apakah R yang menghasilkan ini mengikuti fakta bahwa kode otentikasi pesan adalah simetris- kunci primitif; apa pun yang bisa dilakukan S, R juga bisa melakukannya.

Seperti dalam kasus enkripsi kunci pribadi vs. kunci publik, kode autentikasi pesan memiliki keuntungan karena lebih pendek dan kira-kira 2–3 lipat lebih efisien untuk dihasilkan/diverifikasi dibandingkan tanda tangan digital. Jadi, dalam situasi dimana verifikasi publik, transferabilitas, dan/atau non-penyangkalan tidak diperlukan, dan pengirim berkomunikasi terutama dengan satu penerima (dengan siapa pengirim dapat berbagi kunci rahasia), kode otentikasi pesan harus digunakan. .

Kaitannya dengan Enkripsi Kunci Publik

Tanda tangan digital sering disalahartikan sebagai "kebalikan" dari enkripsi kunci publik, dengan peran pengirim dan penerima yang dipertukarkan. Secara historis,1 faktanya, telah disarankan bahwa tanda tangan digital dapat diperoleh dengan "membalikkan" enkripsi kunci publik, yaitu, menandatangani pesan m dengan mendekripsinya (menggunakan kunci privat) untuk mendapatkan σ, dan memverifikasi tanda tangan σ dengan mengenkripsinya (menggunakan kunci publik yang sesuai) dan memeriksa apakah hasilnya m. Saran untuk membangun skema tanda tangan dengan cara ini sama sekali tidak berdasar: dalam banyak kasus, skema ini tidak dapat diterapkan, dan jika diterapkan, hal ini akan menghasilkan skema tanda tangan yang tidak aman.

12.2 DEFINISI

Tanda tangan digital adalah kunci publik dari kode otentikasi pesan, dan sintaksis serta jaminan keamanannya serupa. Algoritme yang diterapkan pengirim pada pesan di sini dilambangkan dengan Tanda (bukan Mac), dan keluaran dari algoritma ini sekarang disebut tanda tangan (bukan tag). Algoritme yang diterapkan penerima pada pesan dan tanda tangan untuk memverifikasi validitas masih dilambangkan dengan Vrfy.

DEFINISI 12.1 Skema tanda tangan (digital) terdiri dari tiga algoritma waktu polinomial probabilistik (Gen, Sign, Vrfy) sehingga:

1. Algoritme pembangkitan kunci Gen mengambil parameter keamanan 1^n sebagai masukan dan mengeluarkan sepasang kunci (pk, sk). Ini masing-masing disebut kunci publik dan kunci privat. Kita asumsikan bahwa pk dan sk masing-masing memiliki panjang paling sedikit n, dan n dapat ditentukan dari pk atau sk.
2. Algoritma penandatanganan Sign mengambil input kunci pribadi sk dan pesan m dari beberapa ruang pesan (yang mungkin bergantung pada pk). Ini menghasilkan tanda tangan σ, dan kami menulisnya sebagai $\sigma \rightarrow Sign_{sk}(m)$.

3. Algoritma verifikasi deterministik Vrfy mengambil input kunci publik pk, pesan m, dan tanda tangan σ. Outputnya sedikit b, dengan b = 1 berarti valid dan b = 0 berarti tidak valid. Kami menulis ini sebagai $b := Vrfy_{pk}(m, \sigma)$.

Diperlukan bahwa kecuali dengan probabilitas yang dapat diabaikan atas keluaran (pk, sk) oleh $Gen(1^n), Vrfy_{pk}(m, Sign_{sk}(m)) = 1$ untuk setiap pesan (legal) m. Jika ada fungsi l sehingga untuk setiap (pk, sk) keluaran $Gen(1^n)$, ruang pesannya adalah $\{0,1\}^{\ell(n)}$, maka kita katakan bahwa (Gen, Sign, Vrfy) adalah skema tanda tangan untuk pesan dengan panjang $\ell(n)$. Kita menyebut σ sebagai tanda tangan yang valid pada pesan m (sehubungan dengan beberapa kunci publik pk yang dipahami dari konteksnya) jika $Vrfy_{pk}(m, \sigma) = 1$.

Skema tanda tangan digunakan dengan cara berikut. Salah satu pihak S, yang bertindak sebagai pengirim, menjalankan $Gen(1^n)$, untuk mendapatkan kunci (pk, sk). Kunci publik pk kemudian dipublikasikan sebagai milik S; misalnya, S dapat meletakkan kunci publik di halaman webnya atau menempatkannya di beberapa direktori publik. Seperti halnya enkripsi kunci publik, kami berasumsi bahwa pihak lain mana pun dapat memperoleh salinan kunci publik S yang sah (lihat pembahasan di bawah). Ketika S ingin mengautentikasi pesan m, ia menghitung tanda tangan σ $Sign_{sk}(m)$ dan mengirimkan (m, σ). Setelah menerima (m, σ), penerima yang mengetahui pk dapat memverifikasi keaslian m dengan memeriksa apakah $Vrfy_{pk}(m, \sigma) = 1$. Hal ini membuktikan bahwa S mengirim m, dan juga bahwa m tidak diubah dalam perjalanan. Namun, seperti halnya kode autentikasi pesan, kode ini tidak menyebutkan kapan m dikirim, dan serangan replay masih mungkin terjadi (lihat Bagian 4.2).

Asumsi bahwa pihak-pihak dapat memperoleh salinan kunci publik S yang sah menyiratkan bahwa S mampu mengirimkan setidaknya satu pesan (yaitu, pk itu sendiri) dengan cara yang dapat diandalkan dan diautentikasi. Namun, jika kita mampu mengirimkan pesan dengan andal, lalu mengapa kita memerlukan skema tanda tangan? Jawabannya adalah bahwa distribusi pk yang dapat diandalkan adalah tugas yang sulit dan mahal, namun menggunakan skema tanda tangan berarti bahwa hal ini hanya perlu dilakukan sekali, setelah itu pesan dalam jumlah tak terbatas selanjutnya dapat dikirim dengan andal. Lebih jauh lagi, seperti yang akan kita bahas di Bagian 12.7, skema tanda tangan itu sendiri digunakan untuk memastikan distribusi kunci publik lainnya dapat diandalkan. Dengan demikian, mereka berfungsi sebagai alat utama untuk menyiapkan "infrastruktur kunci publik" untuk mengatasi masalah distribusi kunci.

Keamanan skema tanda tangan. Untuk kunci publik tetap pk yang dihasilkan oleh penanda tangan S, pemalsuan adalah pesan m bersama dengan tanda tangan yang valid σ, dimana m sebelumnya tidak ditandatangani oleh S. Keamanan skema tanda tangan berarti bahwa musuh tidak dapat menghasilkan pemalsuan bahkan jika ia memperoleh tanda tangan pada banyak pesan lain pilihannya. Ini adalah analogi langsung dari definisi keamanan untuk kode otentikasi pesan, dan kami merujuk pembaca ke Bagian 4.2 untuk motivasi dan diskusi lebih lanjut.

Kami sekarang menyajikan definisi formal keamanan, yang pada dasarnya sama dengan Definisi 4.2. Misalkan $\Pi = (Gen, Sign, Vrfy)$ adalah skema tanda tangan, dan pertimbangkan eksperimen berikut untuk musuh $\mathcal{A}$ dan parameter n:

Eksperimen tanda tangan $Sig - forge_{\mathrm{A},\Pi}(n)$:

1. $Gen(1^n)$, dijalankan untuk mendapatkan kunci (pk, sk).
2. Musuh $\mathcal{A}$ diberikan pk dan akses ke oracle $Sign_{sk}(\cdot)$. Musuh kemudian mengeluarkan (m, σ). Biarkan Q menunjukkan himpunan semua pertanyaan yang ditanyakan $\mathcal{A}$ kepada oraclenya.
3. $\mathcal{A}$ berhasil jika dan hanya jika (1) $Vrfy_{pk}(m, \sigma) = 1$. dan (2) $m \notin Q$.

Dalam hal ini keluaran percobaan didefinisikan sebagai 1.

DEFINISI 12.2 $\mathcal{A}$ Skema tanda tangan $\Pi = (Gen, Sign, Vrfy)$ secara eksistensial tidak dapat dipalsukan di bawah serangan pesan pilihan adaptif, atau hanya aman, jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi yang dapat diabaikan sehingga:

$$\Pr[\text{Sig-forge}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

12.3 PARADIGMA HASH DAN TANDA

Seperti dalam kasus enkripsi kunci publik vs. kunci privat, skema tanda tangan "asli" kurang efisien dibandingkan kode autentikasi pesan. Untungnya, seperti halnya enkripsi hibrid (lihat Bagian 11.3), fungsionalitas tanda tangan digital dapat diperoleh dengan biaya asimtotik dari operasi kunci pribadi, setidaknya untuk pesan yang cukup panjang. Hal ini dapat dilakukan dengan menggunakan pendekatan hash-and-sign, yang akan dibahas selanjutnya.

CONSTRUCTION 12.3

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme for messages of length $\ell(n)$, and let $\Pi_H = (\mathsf{Gen}_H, H)$ be a hash function with output length $\ell(n)$. Construct signature scheme $\Pi' = (\mathsf{Gen}', \mathsf{Sign}', \mathsf{Vrfy}')$ as follows:

- Gen': on input 1^n, run $\mathsf{Gen}(1^n)$ to obtain (pk, sk) and run $\mathsf{Gen}_H(1^n)$ to obtain s; the public key is $\langle pk, s\rangle$ and the private key is $\langle sk, s\rangle$.
- Sign': on input a private key $\langle sk, s\rangle$ and a message $m \in \{0,1\}^*$, output $\sigma \leftarrow \mathsf{Sign}_{sk}(H^s(m))$.
- Vrfy': on input a public key $\langle pk, s\rangle$, a message $m \in \{0,1\}^*$, and a signature σ, output 1 if and only if $\mathsf{Vrfy}_{pk}(H^s(m), \sigma) \stackrel{?}{=} 1$.

Paradigma hash-dan-tanda tangan.

Intuisi di balik pendekatan hash-and-sign sangatlah jelas. Katakanlah kita mempunyai skema tanda tangan untuk pesan dengan panjang l, dan ingin menandatangani pesan (yang lebih panjang) m 0, 1 ∗. Daripada menandatangani m itu sendiri, kita bisa menggunakan fungsi hash H untuk meng-hash pesan ke dalam intisari dengan panjang tetap dengan panjang l, dan

kemudian menandatangani intisari yang dihasilkan. Pendekatan ini analog dengan pendekatan hash dan MAC yang dibahas di Bagian 5.3.

TEOREMA 12.4 Jika Π adalah skema tanda tangan aman untuk pesan dengan panjang l dan ΠH tahan benturan, maka Konstruksi 12.3 adalah skema tanda tangan aman (untuk pesan dengan panjang sembarang).

Pembuktian teorema ini hampir sama dengan pembuktian Teorema 5.6.

12.4 TANDA TANGAN RSA

Kami memulai pertimbangan kami mengenai skema tanda tangan konkrit dengan pembahasan skema berdasarkan asumsi RSA.

RSA Biasa

Kami pertama kali menjelaskan skema tanda tangan sederhana berbasis RSA. Meskipun skema ini tidak aman, skema ini dapat menjadi titik awal yang berguna.

CONSTRUCTION 12.5

Let GenRSA be as in the text. Define a signature scheme as follows:

- Gen: on input 1^n run GenRSA(1^n) to obtain (N, e, d). The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
- Sign: on input a private key $sk = \langle N, d\rangle$ and a message $m \in \mathbb{Z}_N^*$, compute the signature

$$\sigma := [m^d \bmod N].$$

- Vrfy: on input a public key $pk = \langle N, e\rangle$, a message $m \in \mathbb{Z}_N^*$, and a signature $\sigma \in \mathbb{Z}_N^*$, output 1 if and only if

$$m \stackrel{?}{=} [\sigma^e \bmod N].$$

Skema tanda tangan RSA biasa.

Seperti biasa, biarkan GenRSA menjadi algoritma ppt yang, pada input 1n, menghasilkan modulus N yang merupakan produk dari dua n-bit bilangan prima (kecuali dengan probabilitas yang dapat diabaikan), bersama dengan bilangan bulat e, d yang memuaskan $ed = 1\, mod\, \emptyset(N)$. Pembuatan kunci dalam RSA biasa hanya dengan menjalankan GenRSA, dan mengeluarkan $\langle N, e\rangle$ sebagai kunci publik dan $\langle N, d\rangle$ sebagai kunci privat. Untuk menandatangani pesan $m \in Z_N^*$, penandatangan menghitung $\sigma := [m^d\ mod\ N]$. Verifikasi tanda tangan σ pada pesan m sehubungan dengan kunci publik $\langle N, e\rangle$ dilakukan dengan memeriksa apakah $m\ ?\ =\ \sigma e\ mod\ N$. Lihat Konstruksi 12.5.

Sangat mudah untuk melihat bahwa verifikasi tanda tangan yang dibuat secara sah selalu berhasil sejak saat itu

$$\sigma^e = (m^d)^e = m^{[ed \bmod \phi(N)]} = m^1 = m \bmod N.$$

Seseorang mungkin berharap skema ini aman karena, bagi musuh yang hanya mengetahui kunci publik N, e, menghitung tanda tangan yang valid pada pesan m tampaknya memerlukan penyelesaian masalah RSA (karena tanda tangan tersebut persis merupakan akar eth dari m). Sayangnya, alasan ini tidak benar. Untuk satu hal, asumsi RSA hanya mengimplikasikan kesulitan menghitung tanda tangan (yaitu, menghitung akar eth) dari pesan seragam m; ia tidak menjelaskan apa pun tentang kesulitan dalam menghitung tanda tangan pada m yang tidak seragam atau pada pesan m yang dipilih penyerang. Selain itu, asumsi RSA tidak menjelaskan apa pun tentang apa yang mungkin dapat dilakukan penyerang setelah mengetahui tanda tangan pada pesan lain. Contoh berikut menunjukkan bahwa kedua pengamatan ini mengarah pada serangan terhadap skema tanda tangan RSA biasa.

Seseorang mungkin berharap skema ini aman karena, bagi musuh yang hanya mengetahui kunci publik N, e, menghitung tanda tangan yang valid pada pesan m tampaknya memerlukan penyelesaian masalah RSA (karena tanda tangan tersebut persis merupakan akar eth dari m). Sayangnya, alasan ini tidak benar. Untuk satu hal, asumsi RSA hanya mengimplikasikan kesulitan menghitung tanda tangan (yaitu, menghitung akar eth) dari pesan seragam m; ia tidak menjelaskan apa pun tentang kesulitan dalam menghitung tanda tangan pada m yang tidak seragam atau pada pesan m yang dipilih penyerang. Selain itu, asumsi RSA tidak menjelaskan apa pun tentang apa yang mungkin dapat dilakukan penyerang setelah mengetahui tanda tangan pada pesan lain. Contoh berikut menunjukkan bahwa kedua pengamatan ini mengarah pada serangan terhadap skema tanda tangan RSA biasa.

Serangan tanpa pesan. Serangan pertama yang kami jelaskan menghasilkan pemalsuan hanya dengan menggunakan kunci publik, tanpa mendapatkan tanda tangan apa pun dari penanda tangan yang sah. Serangannya bekerja sebagai berikut: diberi kunci publik $pk = \langle N, e \rangle$, pilih seragam $\sigma \in Z_N^*$ dan hitung $m := [\sigma^e \ mod \ N \ ]$. Kemudian keluar*kan pemalsuan* (m, σ). Jelas sekali bahwa σ adalah tanda tangan yang sah pada m, dan ini merupakan pemalsuan karena tidak ada tanda tangan sama sekali yang dikeluarkan oleh pemilik kunci publik. Kami menyimpulkan bahwa skema tanda tangan RSA biasa tidak memenuhi Definisi 12.2. Ada yang mungkin berargumentasi bahwa hal ini bukan merupakan serangan yang "realistis" karena pihak lawan "tidak punya kendali" atas pesan yang dibuatnya dengan tanda tangan yang sah. Hal ini tidak relevan sejauh menyangkut Definisi 12.2, dan kita telah membahas (di Bab 4) mengapa berbahaya untuk mengasumsikan semantik apa pun untuk pesan yang akan diautentikasi menggunakan skema kriptografi apa pun. Selain itu, musuh mempunyai kendali atas m: misalnya, dengan memilih beberapa nilai σ yang seragam, ia dapat (dengan probabilitas tinggi) memperoleh m dengan beberapa bit yang diatur dengan cara yang diinginkan. Dengan memilih σ dengan cara tertentu, dimungkinkan juga untuk mempengaruhi pesan yang dihasilkan yang menghasilkan keluaran palsu.

Memalsukan tanda tangan pada pesan sewenang-wenang. Serangan yang lebih merusak terhadap skema tanda tangan RSA biasa mengharuskan musuh untuk mendapatkan dua tanda tangan dari penandatangan, namun memungkinkan musuh untuk mengeluarkan tanda tangan palsu pada pesan apa pun yang dipilihnya. Katakanlah musuh ingin memalsukan tanda tangan pada pesan $m \in Z_N^*$ sehubungan dengan kunci publik $pk = \langle N, e \rangle$. Musuh memilih $m_1, m_2 \in Z_N^*$ secara sembarang berbeda dari m sehingga $m = m_1 \cdot m_2 \; mod \; N$.

Ia kemudian memperoleh tanda tangan σ_1, σ_2 pada m_1, m_2, masing-masing. Akhirnya, ia menghasilkan $\sigma := [\sigma_1 \cdot \sigma_2 \; mod \; N]$ sebagai tanda tangan yang valid pada m. Ini berhasil karena

$$\sigma^e = (\sigma_1 \cdot \sigma_2)^e = (m_1^d \cdot m_2^d)^e = m_1^{ed} \cdot m_2^{ed} = m_1 \cdot m_2 = m \bmod N,$$

menggunakan fakta bahwa $\sigma 1, \sigma 2$ adalah tanda tangan sah pada m_1, m_2.

Mampu memalsukan tanda tangan pada pesan sewenang-wenang sangatlah menyedihkan. Namun demikian, ada yang berpendapat bahwa serangan ini tidak realistis karena musuh tidak akan mampu meyakinkan penandatangan untuk menandatangani pesan m_1 dan m_2. Sekali lagi, hal ini tidak relevan jika menyangkut Definisi 12.2. Selain itu, berbahaya jika membuat asumsi tentang pesan apa yang ingin atau tidak ingin ditandatangani oleh penandatangan. Misalnya, klien dapat menggunakan skema tanda tangan untuk mengautentikasi ke server dengan menandatangani tantangan acak yang dikirim oleh server. Di sini, server jahat akan bisa mendapatkan tanda tangan pada pesan apa pun pilihannya. Secara umum, pihak lawan mungkin saja memilih m_1 dan m_2 sebagai pesan "sah" yang akan disetujui oleh pihak yang menandatanganinya. Terakhir, perhatikan bahwa serangan dapat digeneralisasikan: jika musuh memperoleh tanda tangan yang valid pada q pesan acak M = $m_1, \ldots, m_q$, maka musuh dapat mengeluarkan tanda tangan yang valid pada salah satu dari $2^q - q$ pesan lain yang diperoleh dengan mengambil produk dari himpunan bagian M (yang ukurannya berbeda dari 1).

RSA-FDH dan PKCS #1 v2.1

Seseorang dapat mencoba untuk mencegah serangan dari bagian sebelumnya dengan menerapkan beberapa transformasi pada pesan sebelum menandatanganinya. Artinya, penandatangan sekarang akan menentukan sebagai bagian dari kunci publiknya sebuah fungsi (deterministik) H dengan sifat kriptografi tertentu (dijelaskan di bawah) yang memetakan pesan ke Z_N^* ; tanda tangan pada pesan m adalah $\sigma := [H(m)^d \; mod \; N]$, dan verifikasi tanda tangan σ pada pesan m akan dilakukan dengan memeriksa apakah $\sigma^{\mathrm{e}} =? \; \mathrm{H(m)} \bmod \mathrm{N}$. Lihat Konstruksi 12.6.

CONSTRUCTION 12.6

Let GenRSA be as in the previous sections, and construct a signature scheme as follows:

- Gen: on input 1^n, run GenRSA(1^n) to compute (N, e, d). The public key is $\langle N, e\rangle$ and the private key is $\langle N, d\rangle$.
  As part of key generation, a function $H : \{0,1\}^* \to \mathbb{Z}_N^*$ is specified, but we leave this implicit.
- Sign: on input a private key $\langle N, d\rangle$ and a message $m \in \{0,1\}^*$, compute
  $$\sigma := [H(m)^d \bmod N].$$
- Vrfy: on input a public key $\langle N, e\rangle$, a message m, and a signature σ, output 1 if and only if $\sigma^e \stackrel{?}{=} H(m) \bmod N$.

Skema tanda tangan RSA-FDH.

Properti apa yang dibutuhkan H agar konstruksi ini aman? Minimal, untuk mencegah serangan tanpa pesan, penyerang tidak dapat memulai dengan σ, hitung $\hat{m} := [\sigma^e \, mod \, N]$, dan kemudian temukan pesan m sedemikian rupa sehingga $H(m) = \hat{m}$. Hal ini, khususnya, berarti bahwa H seharusnya sulit untuk dibalik. Untuk mencegah serangan kedua, kita memerlukan H yang tidak menerima "hubungan perkalian", yaitu sulit untuk menemukan tiga pesan m, m_1, m_2 dengan $H(m) = H(m_1) \cdot H(m_2) \, mod \, N$. Terakhir, pasti sulit menemukan tumbukan di H: jika $H(m_1) = H(m_2)$, maka m_1 dan m_2 mempunyai tanda tangan yang sama dan pemalsuan menjadi hal yang sepele.

Tidak ada cara yang diketahui untuk memilih H sehingga Konstruksi 12.6 dapat dibuktikan aman. Namun, keamanan dapat dibuktikan jika H dimodelkan sebagai oracle acak yang memetakan masukannya secara seragam ke Z_N^* ; skema dalam hal ini disebut skema tanda tangan RSA full-domain hash (RSA-FDH). Kita dapat memeriksa bahwa fungsi acak semacam ini memenuhi persyaratan yang dibahas di paragraf sebelumnya: fungsi acak (dengan rentang yang besar) sulit untuk dibalik, tidak memiliki hubungan perkalian yang mudah ditemukan, dan tahan benturan. Tentu saja, alasan informal ini tidak mengesampingkan semua kemungkinan serangan, namun bukti keamanan di bawah ini mengecualikannya.

Sebelum beralih ke bukti formal, kami memberikan beberapa intuisi. Tujuan kami adalah untuk membuktikan bahwa jika masalah RSA relatif sulit dibandingkan GenRSA, maka RSA-FDH aman ketika H dimodelkan sebagai oracle acak. Kami mempertimbangkan keamanan pertama terhadap serangan tanpa pesan, yaitu ketika musuh A tidak dapat meminta tanda tangan apa pun. Di sini musuh dibatasi untuk membuat pertanyaan ke oracle acak, dan kita asumsikan tanpa kehilangan keumuman bahwa A selalu membuat pertanyaan q (berbeda) dengan tepat ke H dan jika musuh menghasilkan keluaran palsu (m, σ) maka ia sebelumnya telah menanyakan m ke H.

Katakanlah ada musuh efisien $\mathcal{A}$yang melakukan serangan tanpa pesan dan membuat q query ke H. Kita membangun algoritma efisien $\mathcal{A}'$ yang memecahkan masalah RSA relatif terhadap GenRSA. Diberi masukan (N, e, y), algoritma $\mathcal{A}'$ menjalankan $\mathcal{A}$ pada kunci publik

$pk = \langle N, e \rangle$. Misalkan $m_1, \ldots, m_q$menunjukkan kueri q (berbeda) yang dibuat $\mathcal{A}$ ke H. Algoritme kita $\mathcal{A}'$ menjawab kueri oracle acak dari $\mathcal{A}$ dengan elemen seragam Z_N^* kecuali untuk satu kueri misalnya, kueri ke-i, dipilih secara seragam dari kueri oracle dari $\mathcal{A}$—yang dijawab dengan y itu sendiri. Perhatikan bahwa, dari sudut pandang $\mathcal{A}$ semua pertanyaan oracle acaknya dijawab dengan elemen seragam Z_N^* (ingat bahwa y juga seragam, meskipun tidak dipilih oleh $\mathcal{A}'$), sehingga A tidak memiliki informasi tentang saya. Terlebih lagi, tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}'$ didistribusikan secara identik ke tampilan $\mathcal{A}$ ketika menyerang skema tanda tangan asli.

Jika $\mathcal{A}$ menghasilkan keluaran palsu (m, σ) maka, karena $m \in \{m_1, \ldots, m_q\}$, dengan probabilitas 1/q kita akan mendapatkan m = mi. Dalam hal itu,

$$\sigma^e = H(m) = H(m_i) = y \bmod N$$

dan $\mathcal{A}'$dapat menampilkan σ sebagai solusi untuk instance RSA yang diberikan (N, e, y). Kami menyimpulkan bahwa jika $\mathcal{A}$ menghasilkan pemalsuan dengan probabilitas ε, maka $\mathcal{A}'$ menyelesaikan masalah RSA dengan probabilitas ε/q. Karena q adalah polinomial, kami menyimpulkan bahwa ε harus dapat diabaikan jika masalah RSA relatif sulit dibandingkan dengan GenRSA.

Menangani kasus ketika musuh diizinkan untuk meminta tanda tangan pada pesan pilihannya adalah hal yang lebih sulit. Kerumitan muncul karena algoritme kami $\mathcal{A}'$ di atas tidak mengetahui eksponen dekripsi d, namun sekarang harus menghitung tanda tangan yang valid pada pesan yang ditanyakan oleh $\mathcal{A}$ ke oracle penandatanganannya. Hal ini tampaknya mustahil (dan mungkin bahkan bertentangan!) sampai kita menyadari bahwa $\mathcal{A}'$dapat menghitung tanda tangan pada pesan m dengan benar selama ia menetapkan $H(m)$ sama dengan [σe mod N] untuk nilai σ yang diketahui. (Di sini kita menggunakan fakta bahwa oracle acak "dapat diprogram.") Jika σ seragam maka [σe mod N] juga seragam, sehingga oracle acak masih ditiru "dengan baik" oleh $\mathcal{A}'$.

Intuisi di atas diformalkan dalam pembuktian berikut ini:

TEOREMA 12.7 Jika masalah RSA relatif sulit terhadap GenRSA dan H dimodelkan sebagai oracle acak, maka Konstruksi 12.6 aman.

BUKTI Misalkan Π = $(\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ menunjukkan Konstruksi 12.6, dan misalkan A adalah musuh waktu polinomial probabilistik. Kita berasumsi tanpa kehilangan keumumannya bahwa jika $\mathcal{A}$ meminta tanda tangan pada sebuah pesan m, atau mengeluarkan sebuah pemalsuan (m, σ), maka sebelumnya ia menanyakan m ke H. Misalkan q(n) adalah batas atas polinomial pada jumlah kueri $\mathcal{A}$ menjadi H pada parameter keamanan n; kami berasumsi tanpa kehilangan keumuman bahwa $\mathcal{A}$ membuat q(n) kueri yang berbeda dengan H.

Untuk memudahkan, kami mencantumkan langkah-langkah percobaan Sig-forgeA,Π(n):

1. GenRSA (1^n) dijalankan untuk mendapatkan (N, e, d). Fungsi acak $H : \{0,1\}^* \rightarrow Z_N^*$ dipilih.
2. Musuh $\mathcal{A}$ diberikan pk = 〈N, e〉 , dan dapat menanyakan H serta penandatanganan Oracle Sign 〈N,d〉 (·) yang, saat memasukkan pesan m, mengembalikan σ := [H(m)d mod N].
3. Keluaran $\mathcal{A}$ (m, σ), yang sebelumnya tidak meminta tanda tangan pada m. Keluaran percobaan adalah 1 jika dan hanya jika $\sigma^e = H(m) \; mod \; N$.

Kami mendefinisikan eksperimen yang dimodifikasi Sig-forge' A,Π(n) di mana tebakan dibuat sejak awal mengenai pesan mana (dari antara q pesan yang ditanyakan A ke H) yang akan sesuai dengan pemalsuan akhirnya (jika ada) keluaran oleh A:

1. Pilih seragam $j \in \{1, \ldots, q\}$.
2. GenRSA(1n) dijalankan untuk mendapatkan (N, e, d). Fungsi acak $H : \{0,1\}^* \rightarrow Z_N^*$ dipilih.
3. Musuh A diberikan $pk = \langle N, e \rangle$, dan dapat menanyakan H serta tanda oracle penandatanganan $Sign_{\langle N,d \rangle}(\cdot)$ yang, pada input pesan m, mengembalikan $\sigma := [H(m)^d \; mod \; N]$.
4. Keluaran $A(m, \sigma)$, yang sebelumnya tidak meminta tanda tangan pada m. Misalkan i sedemikian rupa sehingga $m = m_i$[2]. Keluaran percobaan adalah 1 jika dan hanya jika $\sigma^e = H(m) \; mod \; N \; dan \; j = i$.

Karena j seragam dan tidak bergantung pada segala sesuatu yang lain, peluang bahwa $j = i$ (walaupun dikondisikan pada kejadian A menghasilkan pemalsuan) adalah tepat $1/q$. Oleh karena itu $Pr\left[Sig - forge'_{A,\Pi}(n) = 1\right] = \frac{1}{q(n)} \cdot Pr[Sig - forge'_{A,\Pi}(n) = 1]$.

Sekarang pertimbangkan eksperimen yang dimodifikasi $Sig - forge'_{A,\Pi}(n)$ di mana eksperimen dibatalkan jika A meminta tanda tangan pada pesan mj (di mana mj menunjukkan pesan ke-j yang ditanyakan ke H, dan j adalah nilai seragam yang dipilih pada awalnya). Hal ini tidak mengubah probabilitas bahwa keluaran percobaan adalah 1, karena jika A meminta tanda tangan pada mj maka ia tidak mungkin menghasilkan pemalsuan pada mj . Dalam kata kata,

$$\begin{aligned} \Pr[\text{Sig-forge}''_{\mathcal{A},\Pi}(n) = 1] &= \Pr[\text{Sig-forge}'_{\mathcal{A},\Pi}(n) = 1] \\ &= \frac{\Pr[\text{Sig-forge}_{\mathcal{A},\Pi}(n) = 1]}{q(n)}. \end{aligned} \qquad (12.1)$$

Terakhir, pertimbangkan algoritma A' berikut untuk memecahkan masalah RSA:
Algoritma $\mathcal{A}$:
Algoritma diberikan (N, e, y) sebagai masukan.

1. Pilih seragam $j \in \{1, \ldots, Q\}$.
2. Jalankan $\mathcal{A}$ pada input kunci publik $pk = \langle N, e \rangle$. Simpan tripel (·, ·, ·) dalam sebuah tabel, awalnya kosong. Entri (m_i, σ_i, y_i) menunjukkan bahwa $\mathcal{A}'$ tel*ah mengatur* $H(mi) = yi, dan\ \sigma e\ i = yi\ mod\ N$.
3. Saat $\mathcal{A}$ membuat kueri oracle acak ke-i H(mi), jawablah sebagai berikut:
   - Jika $i = j$, kembalikan y sebagai jawaban pertanyaan.
   - Jika tidak, pilih seragam σi ∈ Z_N^*, hitung yi := [σe i mod N], kembalikan yi sebagai jawaban pertanyaan, dan simpan (m_i, σ_i, y_i) dalam tabel.

   Ketika $\mathcal{A}$ meminta tanda tangan pada pesan m, misalkan i sedemikian rupa sehingga m = mi dan menjawab pertanyaan sebagai berikut:
   - Jika $i = j$ maka $\mathcal{A}'$ batal.
   - Jika $i \neq j$ maka terdapat entri (m_i, σ_i, y_i) pada tabel.

   Kembalikan σi sebagai jawaban atas pertanyaan.
4. Di akhir eksekusi $\mathcal{A}$, dihasilkan (m, σ). Jika $m = m_j$dan $\sigma^e = y$ mod N , maka keluaran σ.

Jelasnya, $\mathcal{A}'$ berjalan dalam waktu polinomial probabilistik. Katakanlah masukan (N, e, y) ke $\mathcal{A}$ dihasilkan dengan menjalankan GenRSA(1n) untuk memperoleh (N, e, d), lalu memilih seragam y ∈ Z_N^*. Pengamatan penting adalah bahwa tampilan A ketika dijalankan sebagai subrutin oleh $\mathcal{A}'$ identik dengan tampilan A dalam eksperimen $Sig - forge'_{A,\Pi}(n)$

Secara khusus, semua kueri Sign-Oracle dijawab dengan benar, dan masing-masing kueri Oracle acak dari $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}'$ dijawab dengan elemen seragam Z_N^*:

- Pertanyaan $H(m_j)$ dijawab dengan y, elemen seragam dari Z_N^*.
- Pertanyaan $H(m_i)$ dengan $i \neq j$ dijawab dengan $yi = [\sigma_i^e\ mod\ N]$, dimana σi seragam dalam Z_N^*. Karena eksponensial pangkat eth merupakan fungsi satu-satu, yi juga terdistribusi secara seragam.

Terakhir, amati bahwa setiap kali eksperimen $Sig - forge'_{A,\Pi}(n)$menghasilkan 1, maka $\mathcal{A}'$ menghasilkan solusi yang benar untuk instance RSA yang diberikan. Hal ini terjadi karena $Sig - forge'_{A,\Pi}(n) = 1$ menyiratkan bahwa $j = i$ dan $\sigma^e = H(m_i)\ mod\ N$. Sekarang, ketika j = i, algoritma $\mathcal{A}'$ tidak dibatalkan dan sebagai tambahan $H(m_i) = y$. Jadi, $\sigma^e = H(m_i) = y\ mod\ N$, sehingga σ adalah invers yang diinginkan. Menggunakan Persamaan (12.1), artinya

$$\begin{aligned} \Pr[\text{RSA-inv}_{\mathcal{A}',\text{GenRSA}}(n) = 1] &= \Pr[\text{Sig-forge}''_{\mathcal{A},\Pi}(n) = 1] \\ &= \frac{\Pr[\text{Sig-forge}_{\mathcal{A},\Pi}(n) = 1]}{q(n)}. \end{aligned} \qquad (12.2)$$

Jika permasalahan RSA relatif sulit dibandingkan dengan GenRSA, terdapat fungsi negl yang dapat diabaikan sehingga $\Pr[\text{RSA} - \text{inv}_{\text{A}',\text{GenRSA}}(\text{n}) = 1] \leq \text{negl}(\text{n})$. Karena $q(n)$

adalah polinomial, kita menyimpulkan dari Persamaan (12.2) bahwa $\Pr[Sig-forge'_{A,\Pi}(n)=1]$juga dapat diabaikan. Ini melengkapi buktinya.

RSA PKCS #1 v2.1. Standar RSA PKCS #1 v2.1 mencakup skema tanda tangan yang dapat dipandang sebagai varian RSA-FDH. Lebih tepatnya, standar mendefinisikan skema di mana tanda tangan pada pesan bergantung pada garam (yaitu, nilai acak) yang dipilih oleh penanda tangan pada saat pembuatan tanda tangan. Jika garam ini ditetapkan ke NULL oleh penandatangan—sesuatu yang diperbolehkan oleh standar—skema yang dihasilkan sangat mirip dengan RSA-FDH.

Rentang H harus (mendekati) seluruh Z_N^* ; secara khusus tidak cukup hanya membiarkan H menjadi fungsi hash kriptografi yang "siap pakai" seperti SHA-1. (Panjang keluaran SHA-1 jauh lebih kecil daripada panjang modul RSA yang digunakan dalam praktik.) Memang benar, serangan praktis pada Konstruksi 12.6 diketahui jika panjang keluaran H terlalu kecil (misalnya, jika panjang keluaran adalah 160 bit seperti halnya jika SHA-1 digunakan langsung sebagai H). Dalam skema tanda tangan PKCS #1 v2.1, H dibangun melalui penerapan berulang dari fungsi hash kriptografi yang mendasarinya.

12.5 TANDA TANGAN DARI SOAL LOGARITMA DISKRIT

Skema penandatanganan juga dapat didasarkan pada asumsi logaritma diskrit, meskipun asumsi tersebut tidak mudah untuk ditandatangani seperti halnya asumsi RSA.

Skema Tanda Tangan Schnorr

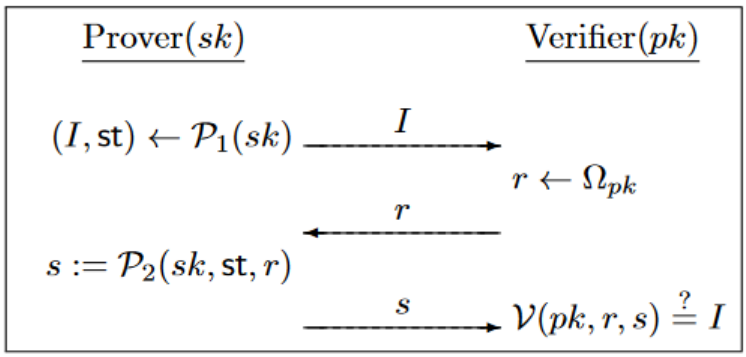
Intuisi yang mendasari skema tanda tangan Schnorr paling baik dijelaskan dengan mengambil sedikit jalan memutar untuk mendiskusikan skema identifikasi (kunci publik). Kami kemudian menjelaskan transformasi Fiat-Shamir yang dapat digunakan untuk mengubah skema identifikasi menjadi skema tanda tangan dalam model random-oracle. Terakhir, kami menyajikan skema identifikasi Schnorr dan skema tanda tangan yang sesuai berdasarkan masalah logaritma diskrit.

Skema Identifikasi

Skema identifikasi adalah protokol interaktif yang memungkinkan satu pihak membuktikan identitasnya (yaitu mengautentikasi dirinya sendiri) kepada pihak lain. Ini adalah gagasan yang sangat alami, dan saat ini adalah hal yang umum untuk mengautentikasi diri sendiri saat masuk ke situs web. Kami menyebut pihak yang mengidentifikasi dirinya (misalnya pengguna) sebagai "pembukti", dan pihak yang memverifikasi identitas (misalnya server web) sebagai "verifikator". Di sini, kami tertarik pada pengaturan kunci publik di mana pembuktian dan pemverifikasi tidak membagikan informasi rahasia apa pun (seperti kata sandi) sebelumnya; sebaliknya, pemverifikasi hanya mengetahui kunci publik dari pembuktian. Keberhasilan pelaksanaan protokol identifikasi meyakinkan pemverifikasi bahwa ia berkomunikasi dengan pembuktian yang dituju dan bukan penipu.

Kami hanya akan mempertimbangkan protokol identifikasi tiga putaran dengan bentuk tertentu, di mana pembukti ditentukan oleh dua algoritma P_1, P_2 dan sisi protokol pemverifikasi ditentukan oleh algoritma $\mathcal{V}$. Pembukti menjalankan $P_1(sk)$ menggunakan

kunci pribadinya sk untuk mendapatkan pesan awal I bersama dengan beberapa status st, dan memulai protokol dengan mengirimkan I ke verifikator. Sebagai tanggapan, pemverifikasi mengirimkan tantangan r yang dipilih secara seragam dari beberapa himpunan Ω_{pk} yang ditentukan oleh kunci publik pk pembuktian. Selanjutnya, peribahasa menjalankan $P_2(sk, st, r)$ untuk menghitung respons yang dikirim kembali ke pemverifikasi. Terakhir, verifikator menghitung $\mathcal{V}(pk, r, s)$ dan menerima jika dan hanya jika hal ini menghasilkan pesan awal I; lihat Gambar 12.1. Tentu saja, untuk kebenarannya kami memerlukan bahwa jika pembukti yang sah menjalankan protokol dengan benar maka pemverifikasi harus selalu menerima.



GAMBAR 12.1: Skema identifikasi tiga putaran.

Untuk alasan teknis, kami mengasumsikan skema identifikasi bersifat "non-degenerate," yang secara intuitif berarti bahwa ada banyak kemungkinan pesan awal I, dan tidak ada satu pun yang mempunyai kemungkinan besar untuk terkirim. Secara formal, suatu skema dikatakan non-degenerasi jika untuk setiap sk kunci privat dan setiap pesan awal tetap I, probabilitas bahwa $P_1(sk)$ keluaran I dapat diabaikan. (Skema identifikasi apa pun dapat dengan mudah dimodifikasi menjadi non-degenerasi dengan mengirimkan string n-bit yang seragam bersama dengan pesan awal.)

Persyaratan keamanan dasar dari skema identifikasi adalah bahwa musuh yang tidak mengetahui kunci rahasia pengirim tidak akan mampu mengelabui verifikator agar menerima. Hal ini akan tetap berlaku bahkan jika penyerang dapat secara pasif menguping beberapa eksekusi protokol (yang jujur) antara pembukti dan pemverifikasi. Kami meresmikan penyadapan tersebut melalui Oracle Transsk yang, ketika dipanggil tanpa masukan apa pun, menjalankan eksekusi protokol secara jujur dan mengembalikan seluruh transkrip (I, r, s) interaksi kepada musuh.

Misalkan $\Pi = (\mathrm{Gen}, P_1, P_2, \mathcal{V})$ adalah skema identifikasi, dan pertimbangkan eksperimen berikut untuk musuh $\mathcal{A}$dan parameter n:

Eksperimen identifikasi $\mathrm{Ident}_{\mathrm{A},\Pi}(n)$:

1. $Gen(1^n)$ dijalankan untuk mendapatkan kunci (pk, sk).
2. Musuh $\mathcal{A}$ diberikan pk dan akses ke Oracle $Trans_{sk}$ yang dapat ditanyakan sesering yang diinginkannya.
3. Kapan saja selama percobaan, $\mathcal{A}$ mengeluarkan pesan I.

4. Tantangan seragam r $\in \Omega_{pk}$ dipilih dan diberikan kepada $\mathcal{A}$, yang merespons dengan beberapa s. ($\mathcal{A}$ dapat terus menanyakan Transsk bahkan setelah menerima r.)
   Hasil percobaan 1 jika dan hanya jika $\mathcal{V}(pk, r, s) \stackrel{?}{=} I$

DEFINISI 12.8 Skema identifikasi $\Pi = (\mathrm{Gen}, \mathrm{P1}, \mathrm{P2}, \mathrm{V})$ aman terhadap serangan pasif, atau cukup aman, jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi yang dapat diabaikan sehingga:

$$\Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}(n).$$

Kita juga dapat mempertimbangkan gagasan keamanan yang lebih kuat, misalnya, ketika musuh juga dapat melakukan serangan aktif terhadap protokol dengan menyamar sebagai pemverifikasi dan mungkin mengirimkan nilai r yang dipilih secara jahat. Kami tidak memerlukan ini untuk aplikasi kami pada skema tanda tangan.

Dari Skema Identifikasi hingga Tanda Tangan

Transformasi Fiat–Shamir (Konstruksi 12.9) menyediakan cara untuk mengubah skema identifikasi (interaktif) menjadi skema tanda tangan (non-interaktif). Ide dasarnya adalah penandatangan bertindak sebagai pembuktian, menjalankan protokol identifikasi sendiri. Artinya, untuk menandatangani pesan m, penandatangan terlebih dahulu menghitung I, dan selanjutnya menghasilkan tantangan r dengan menerapkan beberapa fungsi H pada I dan m. Ini kemudian memperoleh respons yang benar s. Tanda tangan pada m adalah (r, s), yang dapat dibuktikan dengan (1) menghitung ulang $I := \mathcal{V}(pk, r, s)$ dan kemudian (2) m*emeriksa bahwa* $H(I, m) \stackrel{?}{=} r$.

CONSTRUCTION 12.9

Let $(\mathsf{Gen}_{\mathsf{id}}, \mathcal{P}_1, \mathcal{P}_2, \mathcal{V})$ be an identification scheme, and construct a signature scheme as follows:

- Gen: on input 1^n, simply run $\mathsf{Gen}_{\mathsf{id}}(1^n)$ to obtain keys pk, sk. The public key pk specifies a set of challenges Ω_{pk}. As part of key generation, a function $H : \{0,1\}^* \to \Omega_{pk}$ is specified, but we leave this implicit.
- Sign: on input a private key sk and a message $m \in \{0,1\}^*$, do:
  1. Compute $(I, \mathsf{st}) \leftarrow \mathcal{P}_1(sk)$.
  2. Compute $r := H(I, m)$.
  3. Compute $s := \mathcal{P}_2(sk, \mathsf{st}, r)$.

  Output the signature (r, s).
- Vrfy: on input a public key pk, a message m, and a signature (r, s), compute $I := \mathcal{V}(pk, r, s)$ and output 1 if and only if $H(I, m) \stackrel{?}{=} r$.

Transformasi Fiat – Shamir.

Tanda tangan (r, s) "terikat" pada pesan tertentu m karena r merupakan fungsi dari I dan m; mengubah m dengan demikian menghasilkan r yang sama sekali berbeda. Jika H

dimodelkan sebagai input pemetaan oracle acak secara seragam ke Ωpk, maka tantangannya r seragam; secara intuitif, akan sulit bagi musuh (yang tidak mengetahui sk) untuk menemukan tanda tangan yang valid (r, s) pada pesan m seperti halnya meniru pembukti dalam pelaksanaan protokol yang jujur. Intuisi ini diformalkan dalam pembuktian teorema berikut.

TEOREMA 12.10 Misalkan Π adalah skema identifikasi, dan misalkan Π′ adalah skema tanda tangan yang dihasilkan dengan menerapkan transformasi Fiat – Shamir padanya. Jika Π aman dan H dimodelkan sebagai oracle acak, maka Π′ aman.

BUKTI Misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik yang menyerang skema tanda tangan Π′, dengan q = q(n) batas atas jumlah kueri yang $\mathcal{A}'$ buat ke H. Kita membuat sejumlah asumsi penyederhanaan tanpa kehilangan keumuman. Pertama, kita asumsikan bahwa $\mathcal{A}'$ membuat query tertentu ke H hanya sekali. Kita juga berasumsi bahwa setelah diberi tanda tangan (r, s) pada pesan m dengan V(pk, r, s) = I, musuh $\mathcal{A}'$ tidak pernah menanyakan H(I, m) (karena ia mengetahui jawabannya adalah R). Terakhir, kita berasumsi bahwa jika $\mathcal{A}'$ mengeluarkan tanda tangan palsu (r, s) pada pesan m dengan V(pk, r, s) = I, maka $\mathcal{A}'$ sebelumnya telah menanyakan H(I, m).

Kami membangun musuh efisien $\mathcal{A}$ yang menggunakan $\mathcal{A}'$ sebagai subrutin dan menyerang skema identifikasi Π:

Algoritma $\mathcal{A}$:
Algoritma diberikan pk dan akses ke Oracle Transsk.

1. Pilih seragam $j \in \{1, \ldots, Q\}$.
2. Jalankan $\mathcal{A}'$ (pk). Jawab pertanyaannya sebagai berikut:

Saat A′ membuat kueri oracle acak ke-i $H(I_i, m_i)$, jawablah sebagai berikut:

- Jika $i = j$, keluarkan Ij dan terima tantangan r sebagai balasannya. Kembalikan r ke $\mathcal{A}'$ sebagai jawaban atas pertanyaannya.
- Jika $i \neq j$, pilih $\mathrm{r} \in \Omega\mathrm{pk}$ yang seragam dan kembalikan r sebagai jawaban pertanyaan.

  Apabila $\mathcal{A}'$ meminta tanda tangan pada m, jawablah sebagai berikut:
  (a) Permintaan $Trans_{sk}$ untuk mendapatkan transkrip (I, r, s) dari pelaksanaan protokol yang jujur.
  (b) Mengembalikan tanda tangan (r, s).

3. Jika $\mathcal{A}$ mengeluarkan tanda tangan palsu (r, s) pada pesan m, hitung $I := V(pk, r, s)$ dan periksa apakah $(I, m) \stackrel{?}{=} (I_j, m_j)$. Jika demikian, maka keluarkan s. Jika tidak, batalkan.

Tampilan $\mathcal{A}'$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}$ pada eksperimen $\mathrm{Ident}_{\mathcal{A},\Pi}(n)$ hampir identik dengan tampilan $\mathcal{A}'$ pada eksperimen $\mathrm{Sig-forge}_{\mathcal{A}',\Pi'}(n)$. Memang benar, semua kueri H yang dibuat $\mathcal{A}'$ dijawab dengan nilai seragam dari Ω_{pk}, dan semua kueri penandatanganan yang dibuat $\mathcal{A}'$ dijawab dengan tanda tangan valid yang memiliki distribusi

benar. Satu-satunya perbedaan antara pandangan adalah ketika $\mathcal{A}'$ dijalankan sebagai subrutin oleh $\mathcal{A}$, ada kemungkinan terdapat ketidakkonsistenan dalam jawaban yang diterima $\mathcal{A}'$ dari kuerinya ke H: khususnya, ini terjadi jika $\mathcal{A}$ pernah menjawab kueri penandatanganan untuk pesan m menggunakan transkrip (I, r, s) yang $H(I, m)$ sudah didefinisikan (yaitu, $\mathcal{A}'$ sebelumnya telah menanyakan (I, m) ke H) dan $H(I, m) \neq r$. Namun, jika Π tidak merosot maka hal ini hanya terjadi dengan probabilitas yang dapat diabaikan. Jadi, probabilitas bahwa $\mathcal{A}'$ menghasilkan pemalsuan ketika dijalankan sebagai subrutin oleh $\mathcal{A}$ adalah $\mathrm{Sig} - \mathrm{forge}_{\mathcal{A}',\Pi'}(n) - \mathrm{negl}(n)$ untuk beberapa fungsi negl yang dapat diabaikan.

Pertimbangkan pelaksanaan eksperimen $\mathrm{Ident}_{\mathcal{A},\Pi}(n)$ di mana $\mathcal{A}'$ mengeluarkan tanda tangan palsu (r, s) pada pesan m, dan misalkan $\mathrm{I} := \mathcal{V}(pk, r, s)$. Karena j seragam dan tidak bergantung pada semua hal lainnya, peluang bahwa $(I, m) = (I_j m_j)$ (walaupun dikondisikan pada kejadian $\mathcal{A}'$ menghasilkan pemalsuan) adalah tepat $1/q$. (Ingat kita berasumsi bahwa jika $\mathcal{A}'$ mengeluarkan tanda tangan palsu (r, s) pada pesan m dengan $\mathcal{V}(pk, r, s) = I$, maka $\mathcal{A}'$ sebelumnya telah menanyakan $H(I, m).$) Ketika kedua kejadian terjadi, $\mathcal{A}$ berhasil menyamar sebagai pembuktian. Memang benar, $\mathcal{A}$ mengirimkan I_j sebagai pesan awalnya, menerima tantangan r sebagai respons, dan merespons dengan s.

Tetapi $H(I_j, m_j) = r$ dan (karena tanda tangan palsu itu sah) $\mathcal{V}(pk, r, s) = I$. Jika digabungkan, kita melihat bahwa

$$\Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] \geq \frac{1}{q(n)} \cdot \left(\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1] - \mathsf{negl}(n)\right)$$

Atau

$$\Pr[\mathsf{Sig\text{-}forge}_{\mathcal{A}',\Pi'}(n) = 1] \leq q(n) \cdot \Pr[\mathsf{Ident}_{\mathcal{A},\Pi}(n) = 1] + \mathsf{negl}(n).$$

Jika Π aman maka $\Pr[\mathrm{Ident}_{\mathcal{A},\Pi}(n) = 1]$ dapat diabaikan; karena $q(n)$ adalah polinomial, hal ini menyiratkan bahwa $\Pr[\mathrm{Sig} - \mathrm{forge}_{\mathcal{A}',\Pi'}(n) = 1]$ juga dapat diabaikan. Karena $\mathcal{A}'$ bersifat arbitrer, berarti Π' aman.

Skema Identifikasi Schnorr

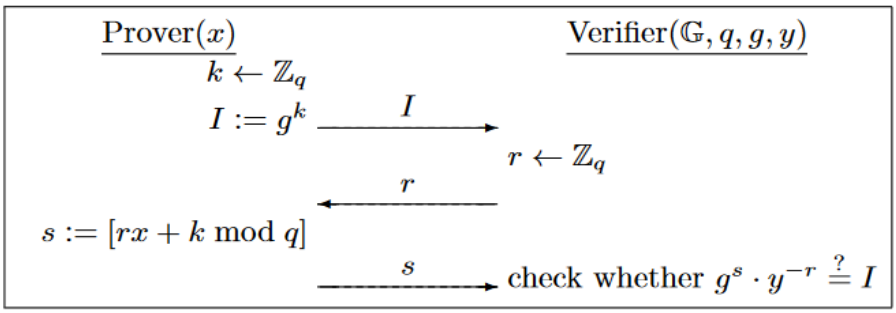
Skema identifikasi Schnorr didasarkan pada kekerasan masalah logaritma diskrit. Misalkan $\mathcal{G}$ adalah algoritma waktu polinomial yang mengambil input 1^n dan (kecuali mungkin dengan probabilitas yang dapat diabaikan) menghasilkan deskripsi grup siklik $\mathbb{G}$, ordenya q (dengan $\| q \| = n$), dan generator g. Untuk menghasilkan kuncinya, peribahasa menjalankan $\mathcal{G}(1^n)$ untuk mendapatkan $(\mathbb{G}, q, g)$, memilih seragam $x \in \mathbb{Z}_q$, dan menetapkan $y := g^x$; kunci publiknya adalah $\langle \mathbb{G}, q, g, y \rangle$ dan kunci privatnya adalah x. Untuk menjalankan protokol (lihat Gambar 12.2), pembuktian dimulai dengan memilih seragam $k \in \mathbb{Z} *_q$ dan pengaturan $I := g^k$; itu mengirimkan I sebagai pesan awal. Verifikator memilih dan mengirimkan

tantangan seragam $r \in \mathbb{Z}_q$; sebagai tanggapan, pepatah menghitung $s := [rx + k \bmod q]$. Verifikator menerima jika dan hanya jika $g^s \cdot y^{-r} \stackrel{?}{=} I$.

Kebenaran berlaku karena

$$g^s \cdot y^{-r} = g^{rx+k} \cdot (g^x)^{-r} = g^k = I.$$

Perhatikan bahwa I seragam di $\mathbb{G}$, sehingga skemanya tidak merosot.
Sebelum memberikan bukti, kami memberikan beberapa intuisi tingkat tinggi. Pengamatan penting pertama adalah bahwa penyadapan pasif tidak membantu penyerang. Alasannya adalah penyerang dapat mensimulasikan sendiri transkrip eksekusi yang jujur, hanya berdasarkan kunci publik dan tanpa sepengetahuan kunci privat. Untuk melakukan hal ini, penyerang hanya membalikkan urutan langkah-langkahnya: pertama-tama ia memilih $r, s \in \mathbb{Z}_q$ yang seragam dan independen, lalu menetapkan $I := g^s \cdot y^{-r}$. Dalam transkrip jujur (I, r, s), pesan awal I adalah elemen seragam dari $\mathbb{G}$, tantangannya adalah elemen $\mathbb{Z}_q$ yang independen dan seragam, dan s kemudian ditentukan secara unik sebagai $s = \log_g(I \cdot y^r)$. Transkrip simulasi yang dibuat oleh penyerang



GAMBAR 12.2: Eksekusi skema identifikasi Schnorr.

memiliki distribusi yang sama: $r \in \mathbb{Z}_q$ seragam dan, karena s seragam di $\mathbb{Z}_q$ dan tidak bergantung pada r, kita melihat bahwa I seragam di $\mathbb{G}$ dan tidak bergantung pada r. Akhirnya, s secara unik ditentukan memenuhi batasan yang sama seperti sebelumnya. Oleh karena itu, kita dapat berasumsi bahwa ketika menyerang skema identifikasi, penyerang tidak menguping eksekusi yang jujur sama sekali.

Jadi, kita telah mereduksinya menjadi seorang penyerang yang mendapatkan kunci publik y, mengirimkan pesan awal I, diberikan tantangan seragam r sebagai respons, dan kemudian harus mengirimkan respons s yang mana $g^s \cdot y^{-r} = I$. Secara informal, jika Jika seorang penyerang mampu melakukan hal ini dengan probabilitas yang tinggi, maka ia harus, khususnya, mampu menghitung respons yang benar s_1, s_2 terhadap setidaknya dua tantangan berbeda $r_1, r_2 \in \mathbb{Z}_q$. Catatan

$$g^{s_1} \cdot y^{-r_1} = I = g^{s_2} \cdot y^{-r_2},$$

jadi $g^{s_1-s_2} = y^{r_1-r_2}$. Namun hal ini menyiratkan bahwa penyerang (yang, ingat, mampu menghasilkan s_1 sebagai respons terhadap r_1, dan s_2 sebagai respons terhadap r_2) secara implisit dapat menghitung logaritma diskrit.

$$\log_g y = [(s_1 - s_2) \cdot (r_1 - r_2)^{-1} \bmod q],$$

bertentangan dengan asumsi kekerasan dari masalah logaritma diskrit.

TEOREMA 12.11 Jika permasalahan logaritma diskrit relatif sulit terhadap G, maka skema identifikasi Schnorr aman.

BUKTI Misalkan Π menyatakan skema identifikasi Schnorr, dan misalkan $\mathcal{A}$ adalah musuh ppt yang menyerang skema tersebut. Kami membuat algoritma ppt berikut $\mathcal{A}'$ untuk memecahkan masalah logaritma diskrit relatif terhadap $\mathcal{G}$:

Algoritma $\mathcal{A}'$:
Algoritma diberikan $\mathbb{G}, q, g, y$ sebagai masukan.

1. Jalankan $\mathcal{A}(pk)$, jawab semua pertanyaannya ke Trans_{sk} seperti yang dijelaskan dalam intuisi yang diberikan sebelumnya.
2. Ketika $\mathcal{A}$ mengeluarkan I, pilih $r_1 \in \mathbb{Z}_q$ yang seragam sebagai tantangannya. Berikan r_1 kepada $\mathcal{A}$, yang menjawab dengan s_1.
3. Jalankan $\mathcal{A}(pk)$ untuk kedua kalinya (dari awal), dengan menggunakan keacakan yang sama seperti sebelumnya kecuali untuk $r_2 \in \mathbb{Z}_q$ yang seragam dan independen. Akhirnya, $\mathcal{A}$ merespons dengan s_2.
4. Jika $g^{s_1} \cdot h^{-r_1} = I$ dan $g^{s_2} \cdot h^{-r_2} = I$ dan $r_1 \neq r_2$ maka keluaran $[(s_1 - s_2) \cdot (r_1 - r_2)^{-1} \bmod q]$. Jika tidak, tidak menghasilkan apa pun.

Mengingat satu kali eksekusi $\mathcal{A}$ sebagai subrutin dari $\mathcal{A}'$, misalkan ω menunjukkan keacakan yang digunakan dalam eksekusi tersebut kecuali untuk tantangan itu sendiri. Jadi, ω terdiri dari segala keacakan yang digunakan oleh $\mathcal{G}$, pilihan kunci privat (yang tidak diketahui) x, segala keacakan yang digunakan oleh $\mathcal{A}$ itu sendiri, dan keacakan yang digunakan oleh $\mathcal{A}'$ saat menjawab pertanyaan ke Trans_{sk}. Definisikan $V(\omega, r)$ sama dengan 1 jika dan hanya jika $\mathcal{A}$ merespons tantangan r dengan benar ketika keacakan ω digunakan dalam sisa eksekusi. Untuk ω tetap apa pun, tentukan $\delta_\omega \stackrel{\text{def}}{=} \Pr_r[V(\omega, r) = 1]$; setelah menetapkan ω, ini adalah probabilitas atas pilihan tantangan r yang ditanggapi $\mathcal{A}$ dengan benar. Definisikan $\delta(n) \stackrel{\text{def}}{=} \Pr[\mathrm{Ident}_{\mathcal{A},\Pi}(n) = 1]$. Karena simulasi oracle Trans_{sk} sempurna, kita punya

$$\delta(n) = \Pr_{\omega,r}[V(\omega, r) = 1] = \sum_\omega \Pr[\omega] \cdot \delta_\omega.$$

Selain itu, intuisi sebelum pembuktian menunjukkan bahwa $\mathcal{A}'$ dengan benar menghitung logaritma diskrit y setiap kali $\mathcal{A}$ berhasil dua kali dan $r_1 \neq r_2$. Dengan demikian:

$$\begin{aligned}\Pr[\mathrm{DLog}_{\mathcal{A}',\mathcal{G}}(n)=1] &= \Pr_{\omega,r_1,r_2}[V(\omega,r_1) \wedge V(\omega,r_2) \wedge r_1 \neq r_2] \\ &\geq \Pr_{\omega,r_1,r_2}[V(\omega,r_1) \wedge V(\omega,r_2)] - \Pr_{\omega,r_1,r_2}[r_1=r_2] \\ &= \textstyle\sum_\omega \Pr[\omega]\cdot(\delta_\omega)^2 - 1/q \\ &\geq \left(\textstyle\sum_\omega \Pr[\omega]\cdot\delta_\omega\right)^2 - 1/q \\ &= \delta(n)^2 - 1/q,\end{aligned}$$

menggunakan pertidaksamaan Jensen[4] pada langkah kedua hingga terakhir. Jika soal logaritma diskrit relatif sulit terhadap $\mathcal{G}$ maka $\Pr[\mathrm{DLog}_{\mathcal{A}',\mathcal{G}}(n)=1]$ dapat diabaikan. Karena $1/q$ dapat diabaikan (karena $\|q\|=n$), hal ini berarti $\delta(n)$ juga dapat diabaikan, sehingga Π merupakan skema identifikasi yang aman.

Skema tanda tangan Schnorr diperoleh dengan menerapkan transformasi Fiat – Shamir ke skema identifikasi Schnorr. Lihat Konstruksi 12.12

CONSTRUCTION 12.12

Let $\mathcal{G}$ be as described in the text.

- Gen: run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G},q,g)$. Choose a uniform $x\in\mathbb{Z}_q$ and set $y:=g^x$. The private key is x and the public key is $(\mathbb{G},q,g,y)$. As part of key generation, a function $H:\{0,1\}^*\to\mathbb{Z}_q$ is specified, but we leave this implicit.
- Sign: on input a private key x and a message $m\in\{0,1\}^*$, choose uniform $k\in\mathbb{Z}_q$ and set $I:=g^k$. Then compute $r:=H(I,m)$, followed by $s:=[rx+k \bmod q]$. Output the signature (r,s).
- Vrfy: on input a public key $(\mathbb{G},q,g,y)$, a message m, and a signature (r,s), compute $I:=g^s\cdot y^{-r}$ and output 1 if $H(I,m)\stackrel{?}{=}r$.

Skema tanda tangan Schnorr.

Pertidaksamaan Jensen[4] mengatakan bahwa $\sum_i a_i\cdot b_i^2 \geq \left(\sum_i a_i\right)^{-1}\cdot\left(\sum_i a_i\cdot b_i\right)^2$ untuk bentuk positif $\{a_1\}$.

DSA dan ECDSA

Algoritma Tanda Tangan Digital (DSA) dan Algoritma Tanda Tangan Digital Kurva Elliptic (ECDSA) didasarkan pada masalah logaritma diskrit di kelas grup yang berbeda. Mereka telah ada dalam beberapa bentuk sejak tahun 1991, dan keduanya termasuk dalam Standar Tanda Tangan Digital (DSS) yang dikeluarkan oleh NIST. Kedua skema tersebut mengikuti pola umum dan dapat dilihat sebagai konstruksi dari skema identifikasi yang mendasarinya (lihat bagian sebelumnya). Misalkan $\mathbb{G}$ adalah grup siklik orde prima q dengan generator g. Perhatikan skema identifikasi berikut dimana kunci privat pembuktiannya adalah x dan kunci publiknya adalah $(\mathbb{G},q,g,y)$ dengan $y=g^x$:

1. Pepatah memilih seragam $k\in\mathbb{Z}*_q$ dan mengirimkan $I:=g^k$.
2. Verifikator memilih dan mengirimkan seragam $\propto, r\in\mathbb{Z}_q$ sebagai tantangan.
3. Pepatah mengirimkan $s:=[k^{-1}\cdot(\propto+xr) \bmod q]$ sebagai respon.

4. Verifikator menerima jika $s \neq 0$ dan $g^{\propto s^{-1}} \cdot y^{rs^{-1}} \stackrel{?}{=} I$.

Catatan $s \neq 0$ kecuali $\propto = -xr \bmod q$, yang terjadi dengan probabilitas yang dapat diabaikan. Dengan asumsi $s \neq 0$, invers $s^{-1} \bmod q$ ada dan

$$g^{\alpha s^{-1}} \cdot y^{rs^{-1}} = g^{\alpha s^{-1}} \cdot g^{xrs^{-1}} = g^{(\alpha + xr)\cdot s^{-1}} = g^{(\alpha+xr)\cdot k\cdot(\alpha+xr)^{-1}} = I.$$

Dengan demikian, kita melihat bahwa kebenaran tetap ada, namun kemungkinannya dapat diabaikan.

Kita dapat menunjukkan bahwa skema identifikasi ini aman jika permasalahan logaritma diskrit relatif sulit terhadap $\mathcal{G}$. Kita hanya membuat sketsa argumennya, dengan asumsi sudah familiar dengan hasil bagian sebelumnya. Pertama-tama, transkrip eksekusi yang jujur dapat disimulasikan: untuk melakukannya, cukup pilih seragam $\propto, r \in \mathbb{Z}_q$ dan $s \in \mathbb{Z}*_q$, lalu atur $I := g^{\propto s^{-1}} \cdot y^{rs^{-1}}$. (Ini tidak lagi memberikan simulasi yang sempurna, namun cukup mendekati.) Selain itu, jika penyerang mengeluarkan pesan awal I yang dapat memberikan respon yang benar $s_1, s_2 \in \mathbb{Z}*_q$ terhadap tantangan yang berbeda $(\propto, r_1), (\propto, r_2)$ lalu

$$g^{\alpha s_1^{-1}} \cdot y^{r_1 s_1^{-1}} = I = g^{\alpha s_2^{-1}} \cdot y^{r_2 s_2^{-1}},$$

sehingga $g^{\propto(s_1^{-1} - s_2^{-1})} = h^{r_1 s_1^{-1} - r_2 s_2^{-1}}$ dan $\log_g h$ dapat dihitung seperti pada bagian sebelumnya. Hal yang sama juga berlaku jika penyerang memberikan respon yang benar terhadap tantangan yang berbeda $(\propto_1, r), (\propto_2, r)$.

Skema tanda tangan DSA/ECDSA dibangun dengan "meruntuhkan" skema identifikasi di atas menjadi algoritma non-interaktif yang dijalankan oleh penandatangan. Berbeda dengan transformasi Fiat-Shamir, transformasi di sini bisa dilihat sebagai berikut (lihat Konstruksi 12.13):

- Atur $\propto := H(m)$, dimana m adalah pesan yang ditandatangani dan H adalah fungsi hash kriptografi.
- Atur $r := F(I)$ untuk fungsi (yang ditentukan) $F : \mathbb{G} \to \mathbb{Z}_q$. Di sini, F adalah fungsi "sederhana" yang tidak dimaksudkan untuk bertindak seperti ramalan acak.

Fungsi F bergantung pada grup $\mathbb{G}m$, yang selanjutnya bergantung pada skema. Dalam DSA, $\mathbb{G}$ dianggap subgrup order-q dari $\mathbb{Z}*_p$, untuk p prima, dan $F(I) \stackrel{\text{def}}{=} [I \bmod q]$. Dalam ECDSA, $\mathbb{G}$ adalah subgrup orde- q dari grup kurva elips $E(\mathbb{Z}_p)$, untuk p prima.[5] Ingat dari Bagian 8.3 bahwa setiap elemen dari grup tersebut dapat direpresentasikan sebagai pasangan $(x, y) \in \mathbb{Z}_p \times \mathbb{Z}_p$. Fungsi F dalam hal ini didefinisikan sebagai $F((x, y)) \stackrel{\text{def}}{=} [x \bmod q]$.

CONSTRUCTION 12.13

Let $\mathcal{G}$ be as in the text.

- Gen: on input 1^n, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$. Choose uniform $x \in \mathbb{Z}_q$ and set $y := g^x$. The public key is $\langle \mathbb{G}, q, g, y \rangle$ and the private key is x.

  As part of key generation, two functions $H : \{0,1\}^* \to \mathbb{Z}_q$ and $F : \mathbb{G} \to \mathbb{Z}_q$ are specified, but we leave this implicit.

- Sign: on input the private key x and a message $m \in \{0,1\}^*$, choose uniform $k \in \mathbb{Z}_q^*$ and set $r := F(g^k)$. Then compute $s := [k^{-1} \cdot (H(m) + xr) \bmod q]$. (If $r = 0$ or $s = 0$ then start again with a fresh choice of k.) Output the signature (r, s).

- Vrfy: on input a public key $\langle \mathbb{G}, q, g, y \rangle$, a message $m \in \{0,1\}^*$, and a signature (r, s) with $r, s \neq 0 \bmod q$, output 1 if and only if

$$r \stackrel{?}{=} F\left(g^{H(m)\cdot s^{-1}} y^{r \cdot s^{-1}}\right).$$

DSA dan ECDSA—secara abstrak.

Dengan asumsi kekerasan masalah logaritma diskrit, DSA dan ECDSA dapat dibuktikan aman jika H dan F dimodelkan sebagai ramalan acak. Namun, seperti yang telah kita bahas di atas, meskipun model oracle acak mungkin masuk akal untuk H, namun model tersebut bukanlah model yang tepat untuk F. Tidak ada bukti keamanan yang diketahui untuk pilihan spesifik F dalam standar. Namun demikian, DSA dan ECDSA telah digunakan dan dipelajari selama beberapa dekade tanpa ditemukan adanya serangan.

Generasi k yang tepat. Skema DSA/ECDSA menetapkan bahwa penandatangan harus memilih $k \in \mathbb{Z} *_q$ yang seragam saat menghitung tanda tangan. Kegagalan dalam memilih k dengan benar (misalnya, karena pembuatan bilangan acak yang buruk) dapat menyebabkan hasil yang sangat buruk. Sebagai permulaan, jika penyerang dapat memprediksi nilai k yang digunakan untuk menghitung tanda tangan (r, s) pada pesan m, maka mereka dapat menghitung kunci privat penanda tangan. Hal ini benar karena $s = k^{-1} \cdot (H(m) + xr) \bmod q$, dan jika k diketahui maka satu-satunya yang tidak diketahui adalah kunci privat x.

Bahkan jika k tidak dapat diprediksi, penyerang dapat menghitung kunci pribadi penanda tangan jika k yang sama digunakan untuk menghasilkan dua tanda tangan yang berbeda. Penyerang dapat dengan mudah mengetahui kapan hal ini terjadi karena r juga akan mengulanginya. Katakanlah (r, s_1) dan (r, s_2) masing-masing adalah tanda tangan pada pesan m_1 dan m_2. Kemudian

$$s_1 = k^{-1} \cdot (H(m_1) + xr) \bmod q$$
$$s_2 = k^{-1} \cdot (H(m_2) + xr) \bmod q.$$

Pengurangan menghasilkan $s_1 - s_2 = k^{-1}\big(H(m_1) - H(m_2)\big) \bmod q$, yang darinya k dapat dihitung; mengingat k, penyerang dapat menentukan kunci privat x seperti pada paragraf sebelumnya. Serangan ini digunakan oleh peretas untuk mengekstrak kunci pribadi utama dari Sony PlayStation (PS3) pada tahun 2010.

12.6 TANDA TANGAN DARI FUNGSI HASH

Menariknya—dan mungkin agak mengejutkan—skema tanda tangan dapat dibangun berdasarkan fungsi hash kriptografi, tanpa bergantung pada asumsi teori bilangan. (Hal ini berbeda dengan enkripsi kunci publik, yang tampaknya memerlukan masalah sulit dengan beberapa struktur aljabar.) Pada bagian ini kita mengeksplorasi konstruksi tersebut. Skema yang akan kita lihat juga menarik karena tidak bergantung pada ramalan acak, berbeda dengan semua konstruksi yang telah kami jelaskan sebelumnya dalam bab ini.

Skema Tanda Tangan Lamport

Kami memulai studi kami tentang skema tanda tangan berdasarkan fungsi hash dengan mempertimbangkan gagasan yang relatif lemah tentang skema tanda tangan satu kali yang aman. Secara informal, skema tersebut "aman" selama kunci privat tertentu digunakan untuk menandatangani satu pesan saja. Skema yang memenuhi gagasan keamanan ini mungkin sesuai untuk beberapa aplikasi, dan juga berfungsi sebagai "blok bangunan" yang berguna untuk mencapai gagasan keamanan yang lebih kuat, seperti yang akan kita lihat di bagian berikut.

Misalkan $\Pi = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ adalah skema tanda tangan, dan pertimbangkan eksperimen berikut untuk musuh $\mathcal{A}$ dan parameter n:

eksperimen tanda tangan satu kali $\mathrm{Sig}-\mathrm{forge}^{1-\mathrm{time}}_{A,\Pi}(n)$:

1. $\mathrm{Gen}(1^n)$ dijalankan untuk mendapatkan kunci (pk, sk).
2. Musuh $\mathcal{A}$ diberikan pk dan menanyakan satu pertanyaan m' padanya ramalan $\mathrm{Sign}_{sk}(\cdot)$. $\mathcal{A}$ kemudian mengeluarkan (m, σ) dengan $m \neq m'$.
3. Keluaran percobaan didefinisikan 1 jika dan hanya jika $\mathrm{Vrfy}_{pk}(m, \sigma) = 1$.

DEFINISI 12.14 Skema tanda tangan $\Pi = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ secara eksistensial tidak dapat dipalsukan dalam serangan pesan tunggal, atau skema tanda tangan satu kali aman, jika untuk semua musuh waktu polinomial probabilistik $\mathcal{A}$, terdapat fungsi negl yang dapat diabaikan seperti yang:

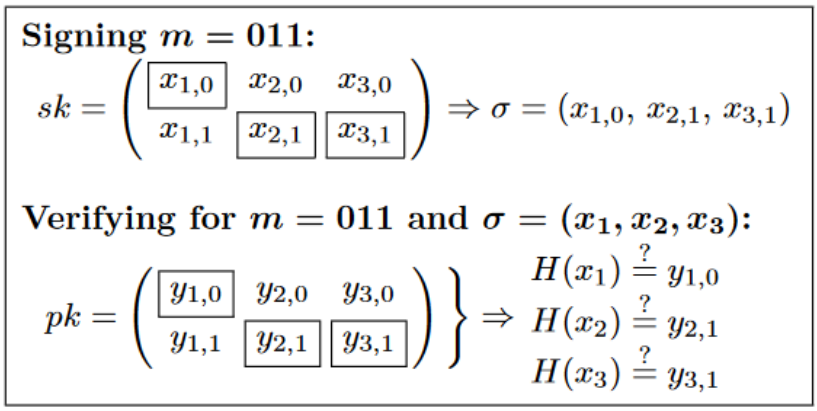
$$\Pr\left[\text{Sig-forge}^{1\text{-time}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \text{negl}(n).$$

Leslie Lamport menunjukkan konstruksi skema tanda tangan satu kali aman pada tahun 1979. Kami mengilustrasikan ide untuk kasus penandatanganan pesan 3-bit. Biarkan H menjadi fungsi hash kriptografi. Kunci pribadi terdiri dari enam nilai seragam $x_{1,0}, x_{1,1}, x_{2,0}, x_{2,1}, x_{3,0}, x_{3,1} \in \{0,1\}^n$, dan kunci publik terkait berisi hasil yang diperoleh dengan menerapkan H untuk masing-masing elemen ini. Kunci-kunci ini dapat divisualisasikan sebagai array dua dimensi:

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & y_{3,0} \\ y_{1,1} & y_{2,1} & y_{3,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & x_{3,0} \\ x_{1,1} & x_{2,1} & x_{3,1} \end{pmatrix}.$$

Untuk menandatangani pesan $m = m_1 m_2 m_3$ (di mana $m_i \in \{0,1\}$), penandatangan melepaskan gambar awal yang sesuai x_{i,m_i} untuk setiap bit pesan; tanda tangan σ terdiri dari tiga nilai $(x_{1,m_1}, x_{2,m_2}, x_{3,m_3})$. Verifikasi dilakukan secara alamiah: ditunjukkan dengan tanda tangan calon (x_1, x_2, x_3) pada pesan $m = m_1 m_2 m_3$, terima jika dan hanya jika $H(x_i) \stackrel{?}{=} y_{i,m_i}$ untuk $1 \leq i \leq 3$.

Hal ini ditunjukkan secara grafis pada Gambar 12.3, dan kasus umum—untuk pesan dengan panjang berapa pun ℓ—dijelaskan secara formal dalam Konstruksi 12.15.



GAMBAR 12.3: Skema Lamport yang digunakan untuk menandatangani pesan $m = 011$.

Setelah mengamati tanda tangan pada sebuah pesan, penyerang yang ingin memalsukan tanda tangan pada pesan lain harus menemukan gambar awal salah satu dari tiga elemen "yang tidak digunakan" dalam kunci publik. Jika H satu arah (lihat Definisi 8.72), maka sulit untuk menemukan bayangan awal seperti itu secara komputasi.

TEOREMA 12.16 Misalkan ℓ adalah polinomial apa pun. Jika H adalah fungsi satu arah, maka Konstruksi 12.15 adalah skema tanda tangan satu kali aman.

CONSTRUCTION 12.15

Let $H : \{0,1\}^* \to \{0,1\}^*$ be a function. Construct a signature scheme for messages of length $\ell = \ell(n)$ as follows:

- Gen: on input 1^n, proceed as follows for $i \in \{1, \ldots, \ell\}$:
  1. Choose uniform $x_{i,0}, x_{i,1} \in \{0,1\}^n$.
  2. Compute $y_{i,0} := H(x_{i,0})$ and $y_{i,1} := H(x_{i,1})$.

  The public key pk and the private key sk are

$$pk = \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix} \quad sk = \begin{pmatrix} x_{1,0} & x_{2,0} & \cdots & x_{\ell,0} \\ x_{1,1} & x_{2,1} & \cdots & x_{\ell,1} \end{pmatrix}.$$

- Sign: on input a private key sk as above and a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, output the signature $(x_{1,m_1}, \ldots, x_{\ell,m_\ell})$.
- Vrfy: on input a public key pk as above, a message $m \in \{0,1\}^\ell$ with $m = m_1 \cdots m_\ell$, and a signature $\sigma = (x_1, \ldots, x_\ell)$, output 1 if and only if $H(x_i) = y_{i,m_i}$ for all $1 \leq i \leq \ell$.

Skema tanda tangan Lamport.

BUKTI Misalkan $\ell = \ell(n)$ seluruhnya. Seperti disebutkan beberapa saat yang lalu, observasi kuncinya adalah ini: katakanlah penyerang $\mathcal{A}$ meminta tanda tangan pada pesan m', dan pertimbangkan pesan lainnya $m \neq m'$. Setidaknya harus ada satu posisi $i^* \in \{1, \ldots, \ell\}$ yang m dan m' berbeda. Katakanlah $m_{i^*} = b \neq m'_{i^*}$. Kemudian memalsukan tanda tangan pada m memerlukan, setidaknya, menemukan preimage (di bawah H) elemen y_{i^*,b^*} dari kunci publik. Karena H satu arah, hal ini tidak mungkin dilakukan. Kami sekarang meresmikan intuisi ini.

Misalkan Π menunjukkan skema Lamport, dan misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik. Dalam eksekusi tertentu $\text{Sig} - \text{forge}^{1-\text{time}}_{\mathcal{A},\Pi}(n)$, misalkan m' menunjukkan pesan yang tanda tangannya diminta oleh $\mathcal{A}$ (kita berasumsi tanpa kehilangan keumuman bahwa $\mathcal{A}$ selalu meminta tanda tangan pada sebuah pesan), dan misalkan (m, σ) adalah keluaran akhir dari $\mathcal{A}$. Kita katakan bahwa $\mathcal{A}$ menghasilkan pemalsuan pada (i, b) jika $\text{Vrfy}_{pk}(m, \sigma) = 1$ dan selanjutnya $m_i \neq m'_i$ (yaitu, pesan m dan m' berbeda pada posisi ke-i) dan $m_i = b \neq m'_i$. Perhatikan bahwa setiap kali $\mathcal{A}$ menghasilkan pemalsuan, ia menghasilkan pemalsuan di beberapa (i, b). Pertimbangkan algoritma ppt berikut $\mathcal{I}$ yang saya coba untuk membalikkan H:

Algoritma $\mathcal{I}$:
Algoritma diberikan 1^n dan y sebagai masukan.
1. Pilih seragam $i^* \in \{1, \ldots, \ell\}$ dan $b^* \in \{0,1\}$. Atur $y_{i^*,b^*} := y$.
2. Untuk semua $i \in \{1, \ldots, \ell\}$ dan $b \in \{0,1\}$ dengan $(i, b) \neq (i^*, b^*)$:
   - Pilih seragam $x_{i,b} \in \{0,1\}^n$ dan atur $y_{i,b} := H(x_{i,b})$.
3. Jalankan $\mathcal{A}$ pada masukan

$$pk := \begin{pmatrix} y_{1,0} & y_{2,0} & \cdots & y_{\ell,0} \\ y_{1,1} & y_{2,1} & \cdots & y_{\ell,1} \end{pmatrix}.$$

4. Saat $\mathcal{A}$ meminta tanda tangan pada pesan m':
   - Jika $m'_{i^*} = b^*$, maka $\mathcal{I}$ batalkan eksekusinya.
   - Jika tidak, $\mathcal{I}$ mengembalikan tanda tangan $\sigma = (x_{1,m'_1}, \ldots, x_{\ell,m'_\ell})$.
5. Ketika $\mathcal{A}$ mengeluarkan (m, σ) dengan $\sigma = (x_1, \ldots, x_\ell)$:
   - Jika $\mathcal{A}$ menghasilkan keluaran palsu pada (i^*, b^*), maka keluaran x_{i^*}.

Setiap kali $\mathcal{A}$ menghasilkan keluaran palsu pada (i^*, b^*), algoritma $\mathcal{I}$ berhasil membalikkan masukan yang diberikan y. Kami tertarik pada probabilitas bahwa hal ini terjadi ketika input ke $\mathcal{I}$ dihasilkan dengan memilih seragam $x \in \{0,1\}^n$ dan mengatur $y := H(x)$ (lih. Definisi 8.72). Bayangkan sebuah "eksperimen mental" yang mana $\mathcal{I}$ diberikan x di awal, himpunan x_i, bx, dan kemudian selalu mengembalikan tanda tangan ke $\mathcal{A}$ pada langkah 4 (yakni, meskipun $m'_{i^*} = b^*$) . Tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{I}$ dalam eksperimen mental ini terdistribusi secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen $\text{Sig} - \text{forge}^{1-\text{time}}_{\mathcal{A},\Pi}(n)$. Karena $(i^* b^*)$ dipilih secara seragam pada awal percobaan, dan pandangan $\mathcal{A}$ tidak bergantung pada pilihan ini, probabilitas bahwa $\mathcal{A}$ menghasilkan pemalsuan pada $(i^* b^*)$, dikondisikan pada fakta bahwa $\mathcal{A}$ menghasilkan pemalsuan sama

sekali, setidaknya $1/2\ell$. (Hal ini karena pemalsuan tanda tangan berarti pemalsuan paling sedikit pada satu titik (i, b). Karena terdapat $2\ell(n)$ titik, maka peluang pemalsuan berada pada $(i^* b^*)$ paling sedikit $1/2\ell$.) Kita menyimpulkan bahwa, dalam eksperimen mental ini, probabilitas $\mathcal{A}$ menghasilkan pemalsuan pada $(i^* b^*)$ setidaknya $\frac{1}{2\ell} \cdot \Pr[\text{Sig} - \text{forge}^{1-\text{time}}_{\mathcal{A},\Pi}(n) = 1]$.

Kembali ke eksperimen nyata yang melibatkan $\mathcal{I}$ seperti yang dijelaskan di awal, observasi utamanya adalah probabilitas bahwa $\mathcal{A}$ menghasilkan pemalsuan pada $(i^* b^*)$ tidak berubah. Hal ini karena eksperimen mental dan eksperimen sebenarnya terjadi bersamaan jika $\mathcal{A}$ menghasilkan keluaran palsu pada $(i^* b^*)$. Artinya, eksperimen hanya berbeda jika $m'_{i^*} = b^*$, tetapi jika hal ini terjadi maka tidak mungkin (menurut definisi) bagi $\mathcal{A}$ untuk kemudian menghasilkan pemalsuan pada $(i^* b^*)$. Jadi peluang $\mathcal{A}$ menghasilkan pemalsuan pada $(i^* b^*)$ setidaknya masih $\frac{1}{2\ell} \cdot \Pr[\text{Sig} - \text{forge}^{1-\text{time}}_{\mathcal{A},\Pi}(n) = 1]$. Dengan kata lain,

$$\Pr[\text{Invert}_{\mathcal{I},H}(n) = 1] \geq \frac{1}{2\ell} \cdot \Pr[\text{Sig-forge}^{1-\text{time}}_{\mathcal{A},\Pi}(n) = 1].$$

Karena H merupakan fungsi satu arah, maka fungsi negl tersebut dapat diabaikan

$$\text{negl}(n) \geq \Pr[\text{Invert}_{\mathcal{I},H}(n) = 1].$$

Karena ℓ adalah polinomial, hal ini menyiratkan bahwa $\Pr[\text{Sig} - \text{forge}^{1-\text{time}}_{\mathcal{A},\Pi}(n) = 1]$ dapat diabaikan, sehingga melengkapi pembuktiannya.

KOROLLARIS 12.17 Jika ada fungsi satu arah, maka untuk polinomial apa pun l terdapat skema tanda tangan satu kali aman untuk pesan dengan panjang ℓ.

Tanda Tangan Berbasis Rantai

Mampu menandatangani hanya satu pesan dengan kunci pribadi tertentu jelas merupakan kelemahan yang signifikan. Di sini kami menunjukkan sebuah pendekatan berdasarkan fungsi hash anti-benturan yang memungkinkan penandatangan menandatangani banyak pesan secara sewenang-wenang, dengan mengorbankan status yang harus diperbarui setelah setiap tanda tangan dihasilkan. Di Bagian 12.6 kita membahas varian yang lebih efisien dari pendekatan ini (yang masih memerlukan status), dan kemudian menjelaskan bagaimana konstruksi yang dimodifikasi ini dapat dibuat tanpa kewarganegaraan. Hasilnya menunjukkan bahwa skema tanda tangan yang memenuhi Definisi 12.2 dapat dibangun berdasarkan fungsi hash yang tahan benturan.

Kami pertama-tama mendefinisikan skema tanda tangan yang memungkinkan penanda tangan mempertahankan status yang diperbarui setelah setiap tanda tangan dibuat.

DEFINISI 12.18 Skema tanda tangan stateful adalah rangkaian algoritma waktu polinomial probabilistik (Gen, Sign, Vrfy) yang memenuhi hal berikut:

1. Algoritme pembangkitan kunci Gen mengambil input parameter keamanan 1^n dan output (pk, sk, s_0). Ini masing-masing disebut kunci publik, kunci privat, dan keadaan awal. Kita asumsikan pk dan sk masing-masing memiliki panjang paling sedikit n, dan n dapat ditentukan dari pk, sk.
2. Algoritma penandatanganan Sign mengambil input kunci pribadi sk, nilai s_{i-1}, dan pesan $m \in \{0,1\}^*$. Ini menghasilkan tanda tangan σ dan nilai s_i.
   Algoritma verifikasi deterministik Vrfy mengambil input kunci publik pk, pesan m, dan tanda tangan σ. Outputnya sedikit b.
3. Kita memerlukannya untuk setiap n, setiap (pk, sk, s_0) keluaran Gen(1^n), dan pesan apa pun $m1, \ldots, m_t \in \{0,1\}^*$, jika kita menghitung secara iteratif $(\sigma_i, s_i) \leftarrow \text{Sign}_{sk,s_{i-1}}(m_i)$ untuk $i = 1, \ldots, t$, maka untuk setiap $i \in \{1, \ldots, t\}$, dinyatakan bahwa $\text{Vrfy}_{pk}(m_i, \sigma_i) = 1$.

Kami menekankan bahwa verifikator tidak perlu mengetahui status penandatangan untuk memverifikasi tanda tangan; bahkan, dalam beberapa skema, negara harus dirahasiakan oleh penandatangan agar keamanan tetap terjaga. Skema tanda tangan yang tidak mempertahankan status (seperti dalam Definisi 12.1) disebut tanpa kewarganegaraan untuk membedakannya dari skema berstatus negara. Jelasnya, skema tanpa kewarganegaraan (stateless) lebih disukai (walaupun skema yang bernegara masih berpotensi bermanfaat). Kami memperkenalkan tanda tangan negara sebagai batu loncatan menuju konstruksi tanpa negara.

Keamanan untuk skema tanda tangan stateful sama persis dengan Definisi 12.2, dengan satu-satunya kehalusan adalah bahwa oracle penandatanganan hanya mengembalikan tanda tangan (dan bukan negara), dan bahwa oracle penandatanganan memperbarui keadaan setiap kali dipanggil.

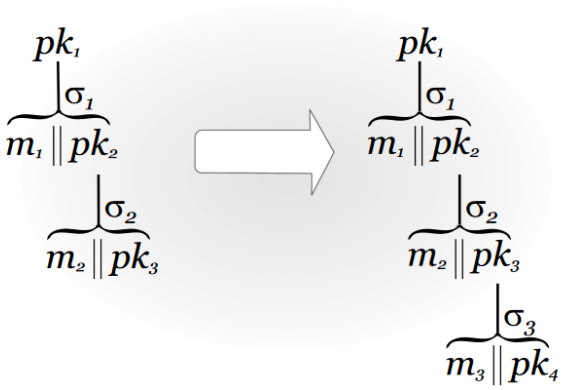
Untuk polinomial apa pun $t = t(n)$, kita dapat dengan mudah membuat skema tanda tangan stateful "t-time-secure". (Definisi keamanan di sini adalah generalisasi dari Definisi 12.14.) Kita dapat melakukan hal ini dengan membiarkan kunci publik (resp., kunci privat) terdiri dari t kunci publik yang dihasilkan secara independen (resp., kunci privat)) untuk skema tanda tangan satu kali yang aman; yaitu, setel $pk := \langle pk_1, \ldots, pk_t \rangle$ dan $sk := \langle sk_1, \ldots, sk_t \rangle$ yang masing-masing (pk_i, sk_i) merupakan pasangan kunci yang dibuat secara independen untuk skema tanda tangan satu kali yang aman. Keadaannya adalah penghitung i yang awalnya disetel ke 1. Untuk menandatangani pesan m menggunakan kunci privat sk dan keadaan saat ini $i \leq t$, hitung $\sigma \leftarrow \text{Sign}_{sk_i}(m)$ (yaitu, buat tanda tangan pada m menggunakan kunci privat sk_i) dan keluaran (σ, i); keadaan diperbarui menjadi $i := i + 1$. Karena keadaan dimulai dari 1, ini berarti pesan ke-i ditandatangani menggunakan sk_i. Verifikasi tanda tangan (σ, i) pada pesan m dilakukan dengan memeriksa apakah σ merupakan tanda tangan yang sah pada m terhadap pk_i. Skema ini aman jika digunakan untuk

menandatangani t pesan karena setiap kunci privat dari skema one-time-secure yang mendasari hanya digunakan untuk menandatangani satu pesan.

Seperti dijelaskan, tanda tangan memiliki panjang yang konstan (yaitu, tidak bergantung pada t), tetapi kunci publik memiliki panjang linier dalam t. Dimungkinkan untuk menukar panjang kunci publik dan tanda tangan dengan meminta penandatangan menghitung pohon Merkle $h := \mathcal{MT}_t(pk_1, \ldots, pk_t)$ (lihat Bagian 5.6) pada t kunci publik yang mendasarinya skema satu kali aman. Artinya, kunci publik sekarang akan menjadi $\langle t, h \rangle$, dan tanda tangan pada pesan ke-i akan mencakup (σ, i), seperti sebelumnya, bersama dengan nilai ke- i pk_1 dan bukti π_1 bahwa ini adalah nilai yang benar sesuai dengan h. (Verifikasi dilakukan dengan cara alami.) Kunci publik sekarang memiliki ukuran konstan, dan panjang tanda tangan hanya bertambah secara logaritmik dengan t.

Karena t dapat berupa polinomial sembarang, mengapa skema sebelumnya tidak memberikan solusi yang kita cari? Kelemahan utamanya adalah mereka memerlukan batas atas t pada jumlah pesan yang dapat ditandatangani untuk ditetapkan terlebih dahulu, pada saat pembuatan kunci. Ini berpotensi menjadi batasan yang parah karena setelah batas atas tercapai, kunci publik baru harus dibuat dan didistribusikan. Kami lebih memilih untuk memiliki satu kunci publik tetap yang dapat digunakan untuk menandatangani pesan dalam jumlah tak terbatas. Misalkan $\Pi = (\mathrm{Gen}, \mathrm{Sign}, \mathrm{Vrfy})$ adalah skema tanda tangan satu kali yang aman. Dalam skema yang baru saja kita jelaskan (mengabaikan optimasi Merkle-tree), penandatangan menjalankan t pemanggilan Gen untuk mendapatkan kunci publik $pk_1, \ldots, pk_t$, dan memasukkan masing-masingnya ke dalam kunci publik sebenarnya pk. Penandatangan kemudian dibatasi untuk menandatangani paling banyak t pesan. Kita dapat melakukan yang lebih baik dengan menggunakan skema "berbasis rantai" di mana penandatangan menghasilkan kunci publik tambahan saat diperlukan, sesuai kebutuhan.

Dalam skema berbasis rantai, kunci publik hanya terdiri dari satu kunci publik pk_1 yang dihasilkan menggunakan Gen, dan kunci privat hanyalah kunci privat terkait sk_1. Untuk menandatangani pesan pertama m_1, penandatangan terlebih dahulu membuat pasangan kunci baru (pk_2, sk_2) menggunakan Gen, lalu menandatangani m_1 dan pk_2 menggunakan sk_1 untuk mendapatkan $\sigma_1 \leftarrow Sign_{sk_1}(m_1 \parallel pk_2)$. Tanda tangan yang dihasilkan mencakup pk2 dan σ1, dan penandatangan menambahkan $(m_1, pk_2, sk_2, \sigma_1)$ ke kondisi saat ini. Secara umum, ketika tiba waktunya untuk menandatangani pesan ke-i, penandatangan akan menyimpan $\{(m_j, pk_{j+1}, sk_{j+1}, \sigma_j)\}_{j=1}^{i-1}$ sebagai bagian dari statusnya. Untuk menandatangani pesan ke-i m_i, penandatangan pertama-tama membuat pasangan kunci baru (pk_{i+1}, sk_{i+1}) menggunakan Gen, lalu menandatangani m_i dan pk_{i+1} menggunakan sk_i untuk mendapatkan tanda tangan $\sigma_i \rightarrow \mathrm{Sign}_{sk_i}(m_i \parallel pk_{i+1})$. Tanda tangan aktual yang dihasilkan mencakup pk_{i+1}, σ_i, dan juga nilai $\{(m_j, pk_{j+1}, \sigma_j)\}_{j=1}^{i-1}$. Penandatangan kemudian menambahkan $(m_i, pk_{i+1}, sk_{i+1}, \sigma_i)$ ke statusnya. Lihat Gambar 12.4 untuk gambaran grafis dari proses ini.



GAMBAR 12.4: Tanda tangan berbasis rantai: situasi sebelum dan sesudah penandatanganan pesan ketiga m_3.

Untuk memverifikasi tanda tangan $(pk_{i+1}, \sigma_i, \{m_j, pk_{j+1}, \sigma_j\}_{j=1}^{i-1})$ pada pesan $m = m_i$ sehubungan dengan kunci publik pk_1, penerima memverifikasi setiap tautan antara kunci publik pk_j dan kunci publik berikutnya kunci publik pk_{j+1} dalam rantai, serta hubungan antara kunci publik terakhir pk_i dan m. Artinya, verifikasi menghasilkan 1 jika dan hanya jika $\text{Vrfy}_{pk_j}(m_j \parallel pk_{j+1}, \sigma_j) \stackrel{?}{=} 1$ untuk semua $j \in \{1, \ldots, i\}$. (Lihat Gambar 12.4.)

Tidak sulit untuk diyakinkan—setidaknya pada tingkat intuitif—bahwa skema tanda tangan ini secara eksistensial tidak dapat dipalsukan di bawah serangan pesan pilihan yang adaptif (terlepas dari berapa banyak pesan yang ditandatangani). Secara informal, hal ini sekali lagi disebabkan oleh kenyataan bahwa setiap pasangan kunci (pk_i, sk_i) digunakan untuk menandatangani hanya satu "pesan", dimana dalam kasus ini "pesan" sebenarnya adalah pasangan pesan/kunci publik $m_i \parallel pk_{i+1}$. Karena kita akan membuktikan keamanan skema yang lebih efisien di bagian berikutnya, kita tidak membuktikan keamanan skema berbasis rantai di sini.

Dalam skema berbasis rantai, setiap pk_i kunci publik digunakan untuk menandatangani pesan dan kunci publik lainnya. Oleh karena itu, skema tanda tangan satu kali aman yang mendasarinya Π harus mampu menandatangani pesan lebih lama dibandingkan kunci publik. Skema Lamport yang disajikan pada Bagian 12.6 tidak memiliki sifat ini. Namun, jika kita menerapkan paradigma hash-dan-tanda dari Bagian 12.3 ke skema Lamport, kita memperoleh skema tanda tangan satu kali aman yang dapat menandatangani pesan dengan panjang yang berubah-ubah. (Meskipun Teorema 12.4 dinyatakan hanya berkenaan dengan skema tanda tangan yang memenuhi Definisi 12.2, tidak sulit untuk melihat bahwa pembuktian yang sama berlaku untuk skema tanda tangan satu kali aman.) Karena hasil ini sangat penting untuk bagian selanjutnya, kami menyatakannya secara formal.

LEMMA 12.19 Jika ada fungsi hash yang tahan benturan, maka terdapat skema tanda tangan satu kali aman (untuk pesan dengan panjang sembarang).

Skema tanda tangan berbasis rantai adalah skema tanda tangan stateful yang secara eksistensial tidak dapat diubah dalam serangan pesan pilihan adaptif. Namun, ia memiliki sejumlah kelemahan. Pertama, tidak ada cara langsung untuk menghapuskan negara (ingatlah

bahwa tujuan akhir kita adalah skema tanpa kewarganegaraan yang memenuhi Definisi 12.2). Hal ini juga tidak terlalu efisien, karena panjang tanda tangan, ukuran negara, dan waktu verifikasi semuanya linier dalam jumlah pesan yang telah ditandatangani. Terakhir, setiap tanda tangan mengungkapkan semua pesan yang telah ditandatangani sebelumnya, dan hal ini mungkin tidak diinginkan dalam beberapa konteks.

Tanda Tangan Berbasis Pohon

Penandatangan dalam skema berbasis rantai pada bagian sebelumnya dapat dilihat sebagai mempertahankan pohon derajat 1, berakar pada kunci publik pk_1, dan dengan kedalaman yang sama dengan jumlah pesan yang ditandatangani sejauh ini (lihat Gambar 12.4). Cara alami untuk meningkatkan efisiensi adalah dengan menggunakan pohon biner di mana setiap node memiliki derajat 2. Seperti sebelumnya, tanda tangan akan berhubungan dengan jalur "yang ditandatangani" di pohon dari daun ke akar; selama pohon tersebut memiliki kedalaman polinomial (walaupun ukurannya eksponensial!), verifikasi dapat dilakukan dalam waktu polinomial.

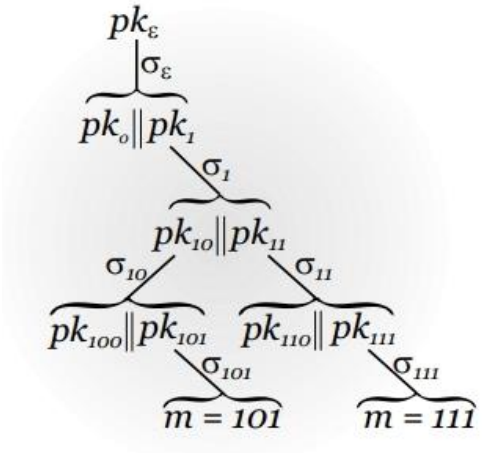
Konkritnya, untuk menandatangani pesan dengan panjang n kita akan bekerja dengan pohon biner dengan kedalaman n yang memiliki 2^n daun. Seperti sebelumnya, penandatangan akan menambahkan node ke pohon "on-the-fly," sesuai kebutuhan. Berbeda dengan skema berbasis rantai, hanya daun (dan bukan node internal) yang akan digunakan untuk menandatangani pesan. Setiap daun pohon akan berhubungan dengan salah satu kemungkinan pesan dengan panjang n.

Secara lebih rinci, kita bayangkan sebuah pohon biner dengan kedalaman n dimana akar diberi label dengan ε (yaitu, string kosong), dan sebuah node yang diberi label dengan string biner w (yang panjangnya kurang dari n) memiliki label anak kiri $w0$ dan anak kanan berlabel $w1$. Pohon ini tidak pernah dibuat secara keseluruhan (perhatikan bahwa ia memiliki ukuran eksponensial), namun dibuat oleh penandatangan sesuai kebutuhan.

Untuk setiap node w, kami mengaitkan sepasang kunci pk_w, sk_w untuk skema tanda tangan satu kali aman Π. Kunci publik dari akar, pk_ε, adalah kunci publik sebenarnya dari penandatangan. Untuk menandatangani pesan $m \in \{0,1\}^n$, penandatangan melakukan hal berikut:

1. Pertama-tama buatlah kunci (sesuai kebutuhan) untuk semua node pada jalur dari akar ke daun berlabel m. (Beberapa kunci publik ini mungkin telah dihasilkan dalam proses penandatanganan pesan sebelumnya, dan dalam hal ini tidak akan dihasilkan lagi.)
2. Selanjutnya, ia "mensertifikasi" jalur dari akar ke daun berlabel m dengan menghitung tanda tangan pada $pk_{w0} \parallel pk_{w1}$, menggunakan kunci pribadi sk_w, untuk setiap string w yang merupakan awalan m yang tepat.
3. Terakhir, ia "mensertifikasi" m sendiri dengan menghitung tanda tangan pada m menggunakan kunci privat sk_m.

Tanda tangan akhir pada m terdiri dari tanda tangan pada m sehubungan dengan pk_m, serta semua informasi yang diperlukan untuk memverifikasi jalur dari daun berlabel



GAMBAR 12.5: Tanda tangan berbasis pohon (secara konseptual).

m ke akar; lihat Gambar 12.5. Selain itu, penandatangan memperbarui statusnya dengan menyimpan semua kunci yang dihasilkan sebagai bagian dari proses penandatanganan di atas. Penjelasan formal skema ini diberikan sebagai Konstruksi 12.20.

Perhatikan bahwa masing-masing kunci yang mendasari skema ini digunakan untuk menandatangani hanya satu "pesan." Setiap kunci yang terkait dengan node internal menandatangani sepasang kunci publik, dan kunci pada daun digunakan untuk menandatangani hanya satu pesan. Karena setiap kunci digunakan untuk menandatangani sepasang kunci lainnya, kita memerlukan skema tanda tangan one-time-secure Π agar mampu menandatangani pesan lebih lama dibandingkan kunci publik. Lemma 12.19 menunjukkan bahwa skema tersebut dapat dibangun berdasarkan fungsi hash yang tahan benturan.

Sebelum membuktikan keamanan pendekatan berbasis pohon ini, perlu diperhatikan bahwa pendekatan ini meningkatkan skema berbasis rantai dalam beberapa hal. Itu masih memungkinkan untuk menandatangani pesan dalam jumlah tidak terbatas. (Meskipun hanya ada 2^n daun, ruang pesan hanya berisi 2^n pesan. Bagaimanapun, 2^n pada akhirnya lebih besar dari fungsi polinomial n.) Dalam hal efisiensi, panjang tanda tangan dan waktu verifikasi kini sebanding dengan panjang pesan n tetapi tidak bergantung pada jumlah pesan yang ditandatangani. Skema ini masih bersifat stateful, namun kita akan melihat bagaimana hal ini dapat dihindari setelah kita membuktikan hasil berikut.

TEOREMA 12.21 Misalkan Π adalah skema tanda tangan one-time-secure. Maka Konstruksi 12.20 adalah skema tanda tangan yang aman.

BUKTI Misalkan Π^* menyatakan Konstruksi 12.20. Misalkan $\mathcal{A}^*$ adalah musuh waktu polinomial probabilistik, misalkan $\ell^* = \ell^*(n)$ menjadi batas atas (polinomial) pada jumlah kueri penandatanganan yang dibuat oleh $\mathcal{A}^*$, dan himpunan $\ell = \ell(n) \stackrel{\text{def}}{=} 2n\ell^*(n) + 1$.

CONSTRUCTION 12.20

Let $\Pi = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vrfy})$ be a signature scheme. For a binary string m, let $m|_i \stackrel{\text{def}}{=} m_1 \cdots m_i$ denote the i-bit prefix of m (with $m|_0 \stackrel{\text{def}}{=} \varepsilon$, the empty string). Construct the scheme $\Pi^* = (\mathsf{Gen}^*, \mathsf{Sign}^*, \mathsf{Vrfy}^*)$ as follows:

- Gen^*: on input 1^n, compute $(pk_\varepsilon, sk_\varepsilon) \leftarrow \mathsf{Gen}(1^n)$ and output the public key pk_ε. The private key and initial state are sk_ε.
- Sign^*: on input a message $m \in \{0,1\}^n$, carry out the following.
  1. For $i = 0$ to $n-1$:
     - If $pk_{m|_i 0}, pk_{m|_i 1}$, and $\sigma_{m|_i}$ are not in the state, compute $(pk_{m|_i 0}, sk_{m|_i 0}) \leftarrow \mathsf{Gen}(1^n)$, $(pk_{m|_i 1}, sk_{m|_i 1}) \leftarrow \mathsf{Gen}(1^n)$, and $\sigma_{m|_i} \leftarrow \mathsf{Sign}_{sk_{m|_i}}(pk_{m|_i 0} \,\|\, pk_{m|_i 1})$. In addition, add all of these values to the state.
  2. If σ_m is not yet included in the state, compute $\sigma_m \leftarrow \mathsf{Sign}_{sk_m}(m)$ and store it as part of the state.
  3. Output the signature $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1}, \sigma_m\right)$.
- Vrfy^*: on input a public key pk_ε, message m, and signature $\left(\{\sigma_{m|_i}, pk_{m|_i 0}, pk_{m|_i 1}\}_{i=0}^{n-1}, \sigma_m\right)$, output 1 if and only if:
  1. $\mathsf{Vrfy}_{pk_{m|_i}}(pk_{m|_i 0} \,\|\, pk_{m|_i 1}, \sigma_{m|_i}) \stackrel{?}{=} 1$ for all $i \in \{0, \ldots, n-1\}$.
  2. $\mathsf{Vrfy}_{pk_m}(m, \sigma_m) \stackrel{?}{=} 1$.

Skema tanda tangan “berbasis pohon”

Perhatikan bahwa ℓ membatasi jumlah kunci publik dari Π yang diperlukan untuk menghasilkan ℓ^* tanda tangan menggunakan Π^*. Hal ini karena setiap tanda tangan di Π^* memerlukan paling banyak $2n$ kunci baru dari Π (dalam kasus terburuk), dan satu kunci tambahan dari Π digunakan sebagai kunci publik sebenarnya pk_ε.

Pertimbangkan musuh PPT berikut $\mathcal{A}$ yang menyerang skema tanda tangan satu kali aman Π:

Musuh $\mathcal{A}$:

$\mathcal{A}$ diberikan sebagai masukan kunci publik pk (parameter keamanan n bersifat implisit).

- Pilih indeks seragam $i^* \in \{1, \ldots, \ell\}$. Buatlah daftar $pk^1, \ldots, pk^\ell$ dari kunci sebagai berikut:
  – Tetapkan $pk^{i^*} := pk$.
  – Untuk $i \neq i^*$, hitung $(pk^i, sk^i) \leftarrow \mathrm{Gen}(1^n)$.
- Jalankan $\mathcal{A}^*$ pada input kunci publik $pk_\varepsilon = pk^1$. Ketika $\mathcal{A}^*$ meminta tanda tangan pada pesan m lakukan:
  1. Untuk $i = 0$ sampai $n-1$:
     – Jika nilai $pk_{m|_i 0}, pk_{m|_i 1}$, dan $\sigma_{m|_i}$ belum ditentukan didenda, lalu setel $pk_{m|_i 0}$ dan $pk_{m|_i 1}$ sama dengan dua kunci publik berikutnya yang tidak terpakai pk^j dan pk^{j+1}, dan hitung tanda tangan $\sigma_{m|_i}$ pada $pk_{m|_i 0} \| pk_{m|_i 1}$ terhadap $pk_{m|_i}$.[6]

2. Jika σ_m belum terdefinisi, hitung tanda tangan σ_m pada m terhadap pk_m (lihat catatan kaki 6).
3. Berikan $(\{\sigma_{m|_i}, pk_{m|_i0}, pk_{m|_i1}\}_{i=0}^{n-1}, \sigma_m)$ ke $\mathcal{A}^*$.

- Misalkan $\mathcal{A}^*$ menghasilkan pesan m (yang sebelumnya tidak diminta tanda tangannya) dan tanda tangan $(\{\sigma'_{m|_i}, pk'_{m|_i0}, pk'_{m|_i1}\}_{i=0}^{n-1}, \sigma'_m)$. Jika ini adalah tanda tangan yang sah pada m, maka:
  Kasus 1: Misalkan terdapat $j \in \{0, \ldots, n-1\}$ yang mana $pk'_{m|_j0} \neq pk_{m|_j0}$ atau $pk'_{m|_j1} \neq pk_{m|_j1}$; ini termasuk kasus ketika $pk_{m|_j0}$ atau $pk_{m|_j1}$ tidak pernah didefinisikan oleh $\mathcal{A}$. Ambil j yang minimal, dan misalkan i sedemikian rupa sehingga $pk^i = pk_{m|_j} = pk'_{m|_j}$ (i seperti itu ada oleh minimalitas j). Jika $i = i^*$, keluaran $(pk'_{m|_j0} \parallel pk'_{m|_j1}, \sigma'_{m|_j})$.
  Kasus 2: Jika kasus 1 tidak berlaku, maka $pk'_m = pk_m$. Misalkan i sedemikian sehingga $pk^i = pk_m$. Jika $i = i^*$, keluaran (m, σ'_m).

Dalam eksperimen $\text{Sig} - \text{forge}_{\mathcal{A},\Pi}^{1-\text{time}}(n)$, tampilan $\mathcal{A}^*$ yang dijalankan sebagai subrutin oleh $\mathcal{A}$ didistribusikan secara identik dengan tampilan $\mathcal{A}^*$ dalam eksperimen $\text{Sig} - \text{forge}_{\mathcal{A}^*,\Pi^*}(n)$.[7]

Jadi, probabilitas bahwa $\mathcal{A}^*$ menghasilkan pemalsuan adalah tepat $\delta(n)$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}$ dalam percobaan ini. Mengingat bahwa $\mathcal{A}^*$ menghasilkan keluaran palsu, pertimbangkan masing-masing dari dua kemungkinan kasus yang dijelaskan di atas:

Kasus 1: Karena i^* seragam dan tidak bergantung pada pandangan $\mathcal{A}^*$, peluang $i = i^*$ tepat$1/\ell$. Jika $i = i^*$, maka $\mathcal{A}$ meminta tanda tangan pada pesan $pk_{m|_j0} \parallel pk_{m|_j1}$ sehubungan dengan kunci publik $pk = pk^{i^*} = pk_{m|_j}$ yang telah diberikan (dan tidak meminta tanda tangan lainnya). Lebih-lebih lagi,

$$pk'_{m|_j0} \| pk'_{m|_j1} \neq pk_{m|_j0} \| pk_{m|_j1}$$

Namun $\sigma'_{m|_j}$ adalah tanda tangan yang sah pada $pk'_{m|_j0} \parallel pk'_{m|_j1}$ terhadap pk. Jadi, $\mathcal{A}$ menghasilkan pemalsuan dalam kasus ini.

Kasus 2: Sekali lagi, karena i^* dipilih secara acak dan seragam dan tidak bergantung pada pandangan $\mathcal{A}^*$, probabilitas $i = i^*$ tepat $1/\ell$. Jika $i = i^*$, maka $\mathcal{A}$ tidak meminta tanda tangan apa pun sehubungan dengan kunci publik $pk = pk^i = pk_m$ namun σ'_m adalah tanda tangan yang sah pada m terhadap pk.

Kita melihat bahwa, dengan syarat $\mathcal{A}^*$ menghasilkan pemalsuan, $\mathcal{A}$ menghasilkan pemalsuan dengan probabilitas tepat $1/\ell$. Artinya

$$\Pr[\text{Sig-forge}^{1\text{-time}}_{\mathcal{A},\Pi}(n) = 1] = \Pr[\text{Sig-forge}_{\mathcal{A}^*,\Pi^*}(n) = 1]/\ell(n).$$

Karena Π adalah skema tanda tangan satu kali aman, maka ada fungsi yang dapat diabaikan

$$\Pr[\text{Sig-forge}^{1\text{-time}}_{\mathcal{A},\Pi}(n) = 1] \leq \text{negl}(n).$$

Karena ℓ adalah polinomial, ini berarti $\Pr[\text{Sig} - \text{forge}_{\mathcal{A}^*,\Pi^*}(n) = 1]$ dapat diabaikan.

Solusi Tanpa Kewarganegaraan

Seperti yang dijelaskan, penandatangan membuat status dengan cepat sesuai kebutuhan. Namun, kita dapat membayangkan penandatangan menghasilkan informasi yang diperlukan untuk semua node di seluruh pohon terlebih dahulu, pada saat pembuatan kunci. (Artinya, pada saat pembuatan kunci, penandatangan dapat membuat kunci $\{(pk_w, sk_w)\}$ dan tanda tangan $\{\sigma_w\}$ untuk semua string biner w dengan panjang paling banyak n.) Jika pembuatan kunci dilakukan dengan cara ini, penandatangan tidak perlu memperbarui statusnya sama sekali; semua nilai ini dapat disimpan sebagai bagian dari kunci pribadi (yang sangat besar), dan kita akan memperoleh skema tanpa kewarganegaraan. Masalah dengan pendekatan ini, tentu saja, menghasilkan semua nilai ini memerlukan waktu eksponensial, dan menyimpan semuanya memerlukan memori eksponensial.

Alternatifnya adalah dengan menyimpan beberapa keacakan yang dapat digunakan untuk menghasilkan nilai $\{(pk_w, sk_w)\}$ dan $\{\sigma_w\}$, sesuai kebutuhan, daripada menyimpan nilai itu sendiri. Artinya, penandatangan dapat menyimpan string acak r_w untuk setiap w, dan setiap kali nilai pk_w, sk_w diperlukan, penandatangan dapat menghitung $(pk_w, sk_w) := \text{Gen}(1^n; r_w)$, yang mana hal ini menandakan pembuatan suatu panjang-n kunci menggunakan koin acak r_w. Demikian pula, jika prosedur penandatanganan bersifat probabilistik, penandatangan dapat menyimpan r'_w dan kemudian menetapkan $\sigma_w := \text{Sign}_{sk_w}(pk_{w0} \parallel pk_{w1}; r'_w)$ (dengan asumsi di sini bahwa $|w| < n$). Namun, menghasilkan dan menyimpan string acak dalam jumlah yang cukup masih memerlukan waktu dan memori yang eksponensial.

Modifikasi sederhana dari alternatif ini menghasilkan solusi waktu polinomial.
Daripada menyimpan r_w dan r'_w acak seperti yang disarankan di atas, penandatangan dapat menyimpan dua kunci k, k' untuk fungsi pseudorandom F. Bila diperlukan, nilai pk_w, sk_w kini dapat dihasilkan dengan proses dua langkah berikut:
1. Hitung $r_w := F_k(w)$.[8]
2. Hitung $(pk_w, sk_w) := \text{Gen}(1^n; r_w)$ (seperti sebelumnya).

Selain itu, kunci k' digunakan untuk menghasilkan nilai r'_w yang digunakan untuk menghitung tanda tangan σ_w. Hal ini memberikan skema tanpa kewarganegaraan di mana pembuatan kunci (serta penandatanganan dan verifikasi) dapat dilakukan dalam waktu polinomial. Secara intuitif, hal ini aman karena menyimpan fungsi acak setara dengan menyimpan semua nilai r_w dan r'_w yang diperlukan, dan menyimpan fungsi pseudorandom "sama baiknya." Kami membiarkannya sebagai latihan untuk memberikan bukti formal bahwa skema yang dimodifikasi ini tetap aman.

Karena keberadaan fungsi hash yang tahan benturan menyiratkan adanya fungsi satu arah (lih. Latihan 7.4), dan fungsi satu arah menyiratkan adanya fungsi pseudorandom (lihat Bab 7), kita mempunyai:

TEOREMA 12.22 Jika ada fungsi hash yang tahan benturan, maka terdapat skema tanda tangan aman (tanpa kewarganegaraan).

Kami mencatat bahwa adalah mungkin untuk membangun skema tanda tangan yang memenuhi Definisi 12.2 dari asumsi (minimal) bahwa ada fungsi satu arah; bukti dari hasil ini berada di luar cakupan buku ini.

12.7 SERTIFIKAT DAN INFRASTRUKTUR KUNCI PUBLIK

Pada bagian ini kita membahas secara singkat salah satu aplikasi utama tanda tangan digital: distribusi kunci publik yang aman. Hal ini membawa kita pada pembahasan penuh mengenai kriptografi kunci publik. Dalam bab ini dan bab sebelumnya kita telah melihat bagaimana menggunakan kriptografi kunci publik setelah kunci publik didistribusikan dengan aman. Sekarang kami menunjukkan bagaimana kriptografi kunci publik itu sendiri dapat digunakan untuk mendistribusikan kunci publik dengan aman. Ini mungkin terdengar melingkar, tapi sebenarnya tidak. Apa yang akan kami tunjukkan adalah ketika sebuah kunci publik, milik pihak terpercaya, didistribusikan dengan cara yang aman, kunci tersebut dapat digunakan untuk "bootstrap" distribusi aman dari banyak kunci publik lainnya. Jadi, setidaknya pada prinsipnya, masalah distribusi kunci yang aman hanya perlu diselesaikan satu kali saja.

Gagasan utamanya di sini adalah sertifikat digital, yang merupakan tanda tangan yang mengikat suatu entitas ke beberapa kunci publik. Untuk lebih konkritnya, katakanlah pihak Charlie telah menghasilkan pasangan kunci (pk_C, sk_C) untuk skema tanda tangan digital yang aman (di bagian ini, kita hanya akan membahas skema tanda tangan yang memenuhi Definisi 12.2). Asumsikan lebih lanjut bahwa pihak lain Bob juga telah menghasilkan pasangan kunci (pk_B, sk_B) (dalam diskusi ini, ini mungkin kunci untuk skema tanda tangan atau skema enkripsi kunci publik), dan bahwa Charlie mengetahui bahwa pk_B adalah milik Bob. kunci publik. Kemudian Charlie dapat menghitung tanda tangannya

$$\text{cert}_{C \to B} \stackrel{\text{def}}{=} \text{Sign}_{sk_C}(\text{`Bob's key is } pk_B\text{'})$$

dan berikan tanda tangan ini kepada Bob. Kami menyebut $\mathrm{cert}_{C\to B}$ sebagai sertifikat untuk kunci Bob yang dikeluarkan oleh Charlie. Dalam praktiknya, sertifikat harus secara jelas mengidentifikasi pihak yang memegang kunci publik tertentu sehingga istilah deskriptif yang lebih unik daripada "Bob" akan digunakan, misalnya, nama lengkap dan alamat email Bob, atau URL situs web Bob.

Sekarang katakanlah Bob ingin berkomunikasi dengan pihak lain Alice yang sudah mengetahui pk_C. Bob dapat mengirim $(pk_B, \mathrm{cert}_{C\to B})$ ke Alice, yang kemudian dapat memverifikasi bahwa $\mathrm{cert}_{C\to B}$ memang merupakan tanda tangan yang valid pada pesan 'Bob's key is $pk_B{}'$ sehubungan dengan pk_C. Dengan asumsi verifikasi berhasil, Alice sekarang mengetahui bahwa Charlie telah menandatangani pesan yang ditunjukkan. Jika Alice mempercayai Charlie, dia dapat menerima pk_B sebagai kunci publik Bob yang sah.

Semua komunikasi antara Bob dan Alice dapat terjadi melalui saluran yang tidak aman dan tidak diautentikasi. Jika musuh aktif mengganggu transmisi $(pk_B, \mathrm{cert}_{C\to B})$ dari Bob ke Alice, musuh tersebut tidak akan dapat menghasilkan sertifikat valid yang menghubungkan Bob ke pk'_B kunci publik lainnya kecuali Charlie sebelumnya telah menandatangani beberapa sertifikat lain menghubungkan Bob dengan pk'_B (dalam hal ini hal ini bukanlah sebuah serangan). Ini semua mengasumsikan bahwa Charlie tidak jujur dan kunci pribadinya belum dibobol. Kami telah menghilangkan banyak detail dalam uraian di atas. Yang paling menonjol, kita belum membahas bagaimana Alice mempelajari pk_C; bagaimana Charlie bisa yakin bahwa pk_B adalah kunci publik Bob; dan bagaimana Alice memutuskan apakah akan memercayai Charlie. Menentukan secara lengkap rincian tersebut (dan lainnya) mendefinisikan infrastruktur kunci publik (PKI) yang memungkinkan distribusi kunci publik secara luas. Berbagai macam model PKI telah diusulkan, dan kami akan menyebutkan beberapa model yang lebih populer saat ini. Perlakuan kami di sini akan dijaga pada tingkat yang relatif tinggi, dan pembaca yang tertarik pada rincian lebih lanjut disarankan untuk membaca referensi di akhir bab ini.

Otoritas sertifikat tunggal. PKI yang paling sederhana mengasumsikan satu otoritas sertifikat (CA) yang sepenuhnya dipercaya oleh semua orang dan yang menerbitkan sertifikat untuk kunci publik semua orang. Otoritas sertifikat biasanya tidak berupa perorangan, namun lebih mungkin berupa perusahaan yang bisnisnya melakukan sertifikasi kunci publik, lembaga pemerintah, atau mungkin departemen dalam suatu organisasi (walaupun dalam kasus terakhir ini CA kemungkinan hanya akan melakukan sertifikasi kunci publik). digunakan oleh orang-orang dalam organisasi). Siapapun yang ingin mengandalkan layanan CA harus mendapatkan salinan sah dari kunci publik CA pk_{CA}. Jelasnya, langkah ini harus dilakukan dengan cara yang aman karena jika beberapa pihak memperoleh versi pk_{CA} yang salah maka pihak tersebut mungkin tidak dapat memperoleh salinan asli kunci publik orang lain. Ini berarti pk_{CA} harus didistribusikan melalui saluran yang diautentikasi. Cara termudah untuk melakukan hal ini adalah melalui sarana fisik: misalnya, jika CA berada dalam suatu organisasi maka setiap karyawan dapat memperoleh salinan asli pk_{CA} langsung dari CA pada hari pertama mereka bekerja. Jika CA adalah sebuah perusahaan, maka pengguna lain harus pergi

ke perusahaan ini suatu saat dan, misalnya, mengambil stik USB yang berisi kunci publik CA. Langkah merepotkan ini hanya perlu dilakukan sekali.

Cara umum bagi CA untuk mendistribusikan kunci publiknya dalam praktiknya adalah dengan "menggabungkan" kunci publik ini dengan beberapa perangkat lunak lain. Misalnya, hal ini terjadi saat ini di banyak browser web populer: kunci publik CA disediakan bersama dengan browser, dan browser diprogram untuk secara otomatis memverifikasi sertifikat saat sertifikat tersebut tiba. (Sebenarnya, browser web modern memiliki kunci publik dari beberapa CA yang tertanam dalam kodenya, sehingga lebih akurat masuk ke dalam model "multiple CA" yang dibahas di bawah.)

Mekanisme dimana CA menerbitkan sertifikat kepada pihak tertentu Bob juga harus dikontrol dengan sangat hati-hati, meskipun rinciannya mungkin berbeda dari CA ke CA. Sebagai salah satu contoh, Bob mungkin harus datang sendiri dengan membawa salinan kunci publik pkB miliknya bersama dengan identifikasi yang membuktikan bahwa namanya (atau alamat emailnya) adalah yang dia klaim. Hanya dengan begitu CA akan menerbitkan sertifikat tersebut. Dalam model dimana terdapat satu CA, para pihak sepenuhnya mempercayai CA ini untuk menerbitkan sertifikat hanya jika diperlukan; Inilah sebabnya mengapa sangat penting untuk melakukan proses verifikasi terperinci sebelum sertifikat diterbitkan. Sebagai konsekuensinya, jika Alice menerima sertifikat $\mathrm{cert}_{CA\to B}$ yang menyatakan bahwa pk_B adalah kunci publik Bob, Alice akan menerima pernyataan ini sebagai valid, dan menggunakan pk_B sebagai kunci publik Bob.

Beberapa otoritas sertifikat. Meskipun model yang hanya memiliki satu CA sederhana dan menarik, namun tidak terlalu praktis. Di satu sisi, di luar satu organisasi, kecil kemungkinan semua orang memercayai CA yang sama. Hal ini tidak berarti bahwa ada orang yang menganggap CA korup; hal ini bisa saja terjadi ketika seseorang menganggap proses verifikasi $\mathrm{CA's}$ tidak mencukupi (misalnya, CA hanya meminta satu bentuk identifikasi saat membuat sertifikat, namun Alice lebih memilih dua bentuk identifikasi yang digunakan). Selain itu, CA adalah satu titik kegagalan seluruh sistem. Jika CA korup, atau dapat disuap, atau bahkan jika CA lemah dalam melindungi kunci privatnya, legitimasi sertifikat yang diterbitkan mungkin dipertanyakan. Juga merepotkan semua pihak yang menginginkan sertifikat harus menghubungi CA ini. Salah satu pendekatan untuk mengatasi permasalahan ini adalah dengan mengandalkan beberapa CAs. Pihak Bob yang ingin mendapatkan sertifikat pada kunci publiknya dapat memilih $\mathrm{CA(s)}$ mana yang ingin diterbitkan sertifikatnya, dan pihak Alice yang diberikan sebuah sertifikat, atau bahkan beberapa sertifikat yang diterbitkan oleh CAs berbeda, dapat memilih yang mana Sertifikat $\mathrm{CA's}$ yang dia percayai. Tidak ada salahnya jika Bob mendapatkan sertifikat dari lebih dari satu CA (selain menimbulkan ketidaknyamanan dan biaya bagi Bob), namun Alice harus lebih berhati-hati karena keamanan komunikasinya pada akhirnya hanya akan sebaik CA yang paling tidak aman yang dia percaya. Artinya, Alice mempercayai dua CAs CA_1 dan CA_2, dan CA_2 dirusak oleh musuh. Kemudian, meskipun musuh ini tidak dapat memalsukan sertifikat yang dikeluarkan oleh CA_1, ia akan dapat menerbitkan sertifikat palsu atas nama CA_2 untuk identitas/kunci publik apa pun yang dipilihnya. Ini adalah masalah nyata dalam sistem saat ini. Seperti disebutkan sebelumnya,

sistem operasi/browser web biasanya sudah dikonfigurasi sebelumnya dengan banyak kunci publik CAs', dan pengaturan defaultnya adalah agar semua CAs ini diperlakukan sama-sama dapat dipercaya. Pada dasarnya, perusahaan mana pun yang bersedia membayar dapat dimasukkan sebagai CA. Jadi, daftar CAs yang telah dikonfigurasi sebelumnya mencakup beberapa perusahaan terkemuka dan mapan serta perusahaan baru lainnya yang kepercayaannya tidak dapat dibangun dengan mudah. Pengguna harus mengonfigurasi pengaturannya secara manual agar hanya menerima sertifikat dari CAs yang dipercaya pengguna.

Rantai delegasi dan sertifikat. Pendekatan lain yang meringankan sebagian beban pada satu CA (tetapi tidak mengatasi masalah keamanan karena hanya ada satu titik kegagalan) adalah dengan menggunakan rantai sertifikat. Kami menyajikan ide untuk rantai sertifikat dengan panjang 2, meskipun mudah untuk melihat bahwa semua yang kami katakan digeneralisasikan ke rantai dengan panjang sewenang-wenang.

Katakanlah Charlie, yang bertindak sebagai CA, menerbitkan sertifikat untuk Bob seperti dalam diskusi awal kita. Asumsikan lebih lanjut bahwa kunci Bob pk_B adalah kunci publik untuk skema tanda tangan. Bob, pada gilirannya, bisa menerbitkan sertifikatnya sendiri untuk pihak lain. Misalnya, Bob dapat menerbitkan sertifikat formulir untuk Alice

$$\mathrm{cert}_{B\to A} \stackrel{\mathrm{def}}{=} \mathrm{Sign}_{sk_B}(\text{`Alice's key is } pk_A\text{'}).$$

Sekarang, jika Alice ingin berkomunikasi dengan pihak keempat Dave yang mengetahui kunci publik Charlie (tetapi bukan kunci publik Bob), maka Alice dapat mengirimkan

$$pk_A,\ \mathrm{cert}_{B\to A},\ pk_B,\ \mathrm{cert}_{C\to B},$$

kepada Dave. Apa yang bisa Dave simpulkan dari hal ini? Yah, pertama-tama dia dapat memverifikasi bahwa Charlie, yang dia percayai dan kunci publiknya sudah ada di tangannya, telah menandatangani sertifikat $\mathrm{cert}_{C\to B}$ yang menunjukkan bahwa pk_B memang milik seseorang bernama Bob. Dave juga dapat memverifikasi bahwa orang bernama Bob ini telah menandatangani sertifikat $\mathrm{cert}_{B\to A}$ yang menunjukkan bahwa pk_A memang milik Alice. Jika Dave memercayai Charlie untuk mengeluarkan sertifikat hanya kepada orang yang dapat dipercaya, maka Dave dapat menerima pk_A sebagai kunci asli Alice.

Kami menyoroti bahwa dalam contoh ini semantik yang lebih kuat dikaitkan dengan sertifikat $\mathrm{cert}_{C\to B}$. Dalam diskusi kita sebelumnya, sertifikat formulir ini hanya merupakan pernyataan bahwa Bob memegang kunci publik pk_B. Sekarang, sebuah sertifikat menegaskan bahwa Bob memegang kunci publik pk_B dan Bob dipercaya untuk menerbitkan sertifikat lainnya. Ketika Charlie menandatangani sertifikat untuk Bob yang memiliki semantik yang lebih kuat, Charlie sebenarnya mendelegasikan kemampuannya untuk menerbitkan sertifikat kepada Bob. Bob sekarang dapat bertindak sebagai wakil Charlie, menerbitkan sertifikat atas nama Charlie.

Kembali ke PKI berbasis CA, kita dapat membayangkan satu CA "root" dan n "tingkat kedua" CAs $\mathrm{CA}_1, \ldots, \mathrm{CA}_n$. CA root dapat menerbitkan sertifikat untuk masing-masing CAs tingkat kedua, yang kemudian dapat menerbitkan sertifikat untuk prinsip-prinsip lain yang memegang kunci publik. Hal ini meringankan beban CA root, dan juga memudahkan para pihak untuk mendapatkan sertifikat (karena mereka sekarang dapat menghubungi CA tingkat kedua yang paling dekat dengan mereka, misalnya). Di sisi lain, mengelola CAs tingkat kedua ini mungkin sulit, dan kehadiran mereka berarti kini terdapat lebih banyak titik serangan dalam sistem.

Model "jaringan kepercayaan". Contoh terakhir dari PKI yang akan kita bahas adalah model yang terdistribusi sepenuhnya, tanpa titik pusat kepercayaan, yang disebut "jaringan kepercayaan". Varian model ini digunakan oleh perangkat lunak enkripsi email PGP ("Pretty Good Privacy") untuk distribusi kunci publik.

Dalam model "web kepercayaan", siapa pun dapat menerbitkan sertifikat kepada orang lain dan setiap pengguna harus membuat keputusan sendiri tentang seberapa besar kepercayaan yang akan diberikan pada sertifikat yang diterbitkan oleh pengguna lain. Sebagai contoh cara kerjanya, katakanlah pengguna Alice sudah memiliki kunci publik pk_1, pk_2, pk_3 untuk beberapa pengguna C_1, C_2, C_3. (Di bawah ini kita akan membahas bagaimana kunci publik ini pada awalnya dapat diperoleh oleh Alice.) Pengguna lain, Bob, yang ingin berkomunikasi dengan Alice mungkin memiliki sertifikat $\mathrm{cert}_{C_1 \to B}, \mathrm{cert}_{C_3 \to B}$, dan $\mathrm{cert}_{C_4 \to B}$, dan akan mengirimkan sertifikat ini (bersama dengan miliknya kunci publik pk_B) ke Alice. Alice tidak dapat memverifikasi $\mathrm{cert}_{C_4 \to B}$ (karena dia tidak memiliki kunci publik C_4), namun dia dapat memverifikasi dua sertifikat lainnya. Sekarang dia harus memutuskan seberapa besar kepercayaannya pada C_1 dan C_3. Ia dapat memutuskan untuk menerima pk_B jika ia benar-benar memercayai C_1, atau jika ia memercayai C_1 dan C_3 pada tingkat yang lebih rendah. (Contohnya, ia mungkin menganggap C_1 atau C_3 kemungkinan besar korup, namun menganggap kecil kemungkinan keduanya korup.)

Dalam model ini, seperti yang dijelaskan, pengguna diharapkan mengumpulkan kunci publik pihak lain, serta sertifikat pada kunci publik mereka sendiri. Dalam konteks PGP, hal ini biasanya dilakukan di "pesta penandatanganan kunci" di mana para pengguna PGP berkumpul (misalnya, di sebuah konferensi), saling memberikan salinan asli kunci publik mereka, dan menerbitkan sertifikat untuk satu sama lain. Secara umum para pengguna di pihak penandatanganan kunci mungkin tidak mengenal satu sama lain, namun mereka dapat memeriksa SIM, misalnya, sebelum menerima atau menerbitkan sertifikat untuk kunci publik seseorang.

Kunci publik dan sertifikat juga dapat disimpan dalam database pusat, dan ini dilakukan untuk PGP. Ketika Alice ingin mengirim pesan terenkripsi ke Bob, dia dapat mencari kunci publik Bob di database ini; bersama dengan kunci publik Bob, database akan mengembalikan daftar semua sertifikat yang dimilikinya yang telah diterbitkan untuk kunci publik Bob. Mungkin juga beberapa kunci publik untuk Bob akan ditemukan dalam database, dan masing-masing kunci publik ini dapat disertifikasi oleh sertifikat yang diterbitkan oleh pihak-pihak yang

berbeda. Sekali lagi, Alice kemudian perlu memutuskan seberapa besar kepercayaan yang akan diberikan pada salah satu kunci publik ini sebelum menggunakannya.

Model jaringan kepercayaan menarik karena tidak memerlukan kepercayaan pada otoritas pusat mana pun. Di sisi lain, meskipun mungkin berfungsi dengan baik bagi rata-rata pengguna yang mengenkripsi email mereka, hal ini tampaknya tidak sesuai untuk pengaturan yang mengutamakan keamanan, atau untuk distribusi kunci publik organisasi (misalnya, untuk e-niaga di web) . Jika seorang pengguna ingin berkomunikasi dengan banknya, misalnya, kecil kemungkinannya dia akan memercayai orang-orang yang dia temui di sebuah konferensi untuk mengesahkan kunci publik banknya, dan juga kecil kemungkinannya bahwa perwakilan bank akan mendatangi pihak yang menandatangani kunci untuk mendapatkan kunci publik tersebut. kunci bank bersertifikat.

Sertifikat Tidak Valid

Salah satu persoalan penting yang belum kami bahas sama sekali adalah kenyataan bahwa sertifikat pada umumnya tidak berlaku selamanya. Seorang karyawan dapat meninggalkan perusahaan, dalam hal ini dia tidak lagi diperbolehkan menerima komunikasi terenkripsi dari orang lain di dalam perusahaan; kunci pribadi pengguna mungkin juga dicuri, sehingga pengguna (dengan asumsi mereka mengetahui pencurian tersebut) ingin membuat pasangan kunci baru dan menghapus kunci publik lama dari peredaran. Dalam salah satu skenario ini, kita memerlukan cara untuk membuat sertifikat yang diterbitkan sebelumnya menjadi tidak valid.

Pendekatan untuk menangani isu-isu ini beragam dan kompleks, dan kami hanya akan menyebutkan dua gagasan yang relatif sederhana, yang dalam beberapa hal mewakili ekstrem yang berlawanan. (Meningkatkan metode ini merupakan area aktif dalam penelitian keamanan jaringan di dunia nyata.)

Kedaluwarsa. Salah satu metode untuk mencegah penggunaan sertifikat tanpa batas waktu adalah dengan mencantumkan tanggal kedaluwarsa sebagai bagian dari sertifikat. Sertifikat yang dikeluarkan oleh CA Charlie untuk kunci publik Bob sekarang mungkin memiliki formulir tersebut

$$\mathrm{cert}_{C\to B} \stackrel{\mathrm{def}}{=} \mathrm{Sign}_{sk_C}(\text{`Bob's key is } pk_B\text{', date}),$$

dimana tanggal adalah suatu tanggal di masa depan yang mana sertifikat tersebut menjadi tidak berlaku. (Misalnya, satu tahun sejak sertifikat diterbitkan.) Ketika pengguna lain memverifikasi sertifikat ini, mereka perlu mengetahui tidak hanya pk_B tetapi juga tanggal kedaluwarsanya, dan mereka sekarang perlu memeriksa tidak hanya apakah tanda tangannya valid, tetapi juga validitasnya. juga tanggal kadaluwarsanya belum lewat. Pengguna yang memegang sertifikat harus menghubungi CA untuk mendapatkan sertifikat baru yang diterbitkan setiap kali sertifikat mereka saat ini habis masa berlakunya; pada titik ini, CA memverifikasi identitas/kredensial pengguna lagi sebelum menerbitkan sertifikat lain.

Penggunaan tanggal kedaluwarsa memberikan solusi yang sangat kasar terhadap masalah yang disebutkan sebelumnya. Jika seorang karyawan meninggalkan perusahaan sehari setelah mendapatkan sertifikat, dan sertifikat tersebut habis masa berlakunya satu

tahun setelah tanggal penerbitannya, maka karyawan tersebut dapat menggunakan kunci publiknya secara tidak sah selama satu tahun penuh hingga tanggal habis masa berlakunya terlewati. Oleh karena itu, pendekatan ini biasanya digunakan bersama dengan metode lain seperti yang akan kami jelaskan selanjutnya.

Pencabutan. Ketika seorang karyawan meninggalkan organisasi, atau kunci pribadi pengguna dicuri, kami ingin sertifikat yang telah dikeluarkan untuk kunci publik mereka segera menjadi tidak valid, atau setidaknya sesegera mungkin. Hal ini dapat dicapai dengan meminta CA secara eksplisit mencabut sertifikatnya. Untuk mempermudah, kami mengasumsikan satu CA, namun semua yang kami katakan berlaku lebih umum jika pengguna memiliki sertifikat yang diterbitkan oleh beberapa CAs.

Ada banyak cara berbeda untuk menangani pencabutan. Salah satu kemungkinan (satu-satunya yang akan kita bahas) adalah CA menyertakan nomor seri di setiap sertifikat yang diterbitkannya; artinya, sertifikat sekarang akan memiliki formulir

$$\mathrm{cert}_{C\to B} \stackrel{\mathrm{def}}{=} \mathrm{Sign}_{sk_C}(\text{`Bob's key is } pk_B\text{', ###}),$$

di mana "###" mewakili nomor seri sertifikat ini. Setiap sertifikat harus memiliki nomor seri unik, dan CA akan menyimpan informasi $(\mathrm{Bob}, pk_B, \#\#\#)$ untuk setiap sertifikat yang dihasilkannya.

Jika kunci privat milik pengguna Bob yang berhubungan dengan kunci publik pk_B dicuri, Bob dapat memperingatkan CA mengenai fakta ini. (CA harus memverifikasi identitas Bob di sini, untuk mencegah pengguna lain melakukan pencabutan palsu atas sertifikat yang diterbitkan untuk Bob.) CA kemudian akan mencari database-nya untuk menemukan nomor seri yang terkait dengan sertifikat yang diterbitkan untuk Bob dan pk_B. Pada akhir setiap hari, katakanlah, CA kemudian akan membuat daftar pencabutan sertifikat (CRL) dengan nomor seri semua sertifikat yang dicabut, dan menandatangani CRL dan tanggal saat ini. CRL yang ditandatangani kemudian didistribusikan secara luas atau disediakan kepada calon verifikator. Verifikasi sertifikat sekarang memerlukan pemeriksaan apakah tanda tangan dalam sertifikat itu valid, pemeriksaan bahwa nomor seri tidak muncul pada daftar pencabutan terkini, dan verifikasi tanda tangan CA's pada daftar pencabutan itu sendiri.

Dalam pendekatan yang telah kami jelaskan ini, terdapat jeda paling lama satu hari sebelum sertifikat menjadi tidak valid. Pendekatan ini menawarkan lebih banyak fleksibilitas dibandingkan pendekatan yang hanya berdasarkan tanggal kadaluarsa.

12.8 MENYATUKAN SEMUANYA – SSL/TLS

Sebagai puncak dari apa yang telah kami bahas dalam buku ini sejauh ini, kami membahas protokol Transport Layer Security (TLS) yang digunakan secara luas untuk mengamankan komunikasi melalui web; TLS adalah protokol yang digunakan oleh browser Anda setiap kali Anda terhubung ke situs web menggunakan https, bukan http. Kita harus jelas bahwa tujuan kita di sini adalah untuk fokus pada kriptografi dasar yang digunakan dalam

protokol inti TLS, dan bukan berbagai aspek lain yang, meskipun menarik dari sudut pandang keamanan jaringan, tidak relevan dengan keprihatinan kita. Seperti biasa, kami telah sedikit menyederhanakan dan mengabstraksi bagian-bagian protokol untuk menyampaikan poin utama, dan uraian kami tidak boleh diandalkan untuk implementasi. Terakhir, kami tidak secara formal mendefinisikan atau mengklaim keamanan protokol; memang, analisis formal TLS adalah subjek penelitian aktif.

TLS adalah protokol standar berdasarkan pendahulunya yang disebut SSL (atau Secure Socket Layer) yang dikembangkan oleh Netscape pada pertengahan 1990an; versi terakhir yang tersedia adalah SSL 3.0. TLS versi 1.0 dirilis pada tahun 1999, diperbarui ke versi 1.1 pada tahun 2006, dan diperbarui lagi ke versi 1.2 (versi saat ini) pada tahun 2008. Pada saat artikel ini ditulis, sekitar 50% situs web masih menggunakan TLS 1.0 dibandingkan versi yang lebih baru. versi terbaru; semua browser web utama mendukung TLS 1.2, meskipun dalam beberapa kasus versi TLS yang lebih lama digunakan secara default. Untuk sebagian besar, uraian kami di bawah ini berada pada tingkat yang cukup tinggi sehingga perbedaan antar versi tidak penting untuk tujuan kami. Namun kami memperingatkan bahwa ada beberapa serangan yang diketahui terjadi pada versi sebelumnya.

TLS memungkinkan klien (misalnya browser web) dan server (misalnya situs web) untuk menyetujui serangkaian kunci bersama dan kemudian menggunakan kunci tersebut untuk mengenkripsi dan mengautentikasi komunikasi selanjutnya. Ini terdiri dari dua bagian: protokol jabat tangan yang melakukan pertukaran kunci yang diautentikasi untuk menetapkan kunci bersama, dan protokol lapisan rekaman yang menggunakan kunci bersama tersebut untuk mengenkripsi/mengautentikasi komunikasi para pihak. Meskipun TLS memungkinkan klien untuk mengautentikasi ke server, TLS terutama digunakan hanya untuk mengautentikasi server ke klien karena biasanya hanya server yang memiliki sertifikat. (Setelah sesi TLS dibuat, otentikasi pengguna-ke-server—jika diinginkan—dapat dilakukan pada lapisan aplikasi tumpukan jaringan dengan, misalnya, mengirimkan kata sandi.)

Protokol jabat tangan. Kami sekarang menjelaskan aliran dasar protokol jabat tangan. Pada awal protokol, klien C memegang satu set kunci publik CA's $\{pk_1, \ldots, pk_n\}$, dan server S memiliki pasangan kunci (pk_S, sk_S) untuk KEM[9] bersama dengan sertifikat $\mathrm{cert}_{i \to S}$ yang dikeluarkan oleh salah satu CAs yang kunci publiknya diketahui oleh C. Untuk terhubung ke S, para pihak menjalankan langkah-langkah berikut.

1. C dimulai dengan mengirimkan pesan ke S yang mencakup informasi tentang versi protokol yang didukung oleh klien, ciphersuite yang didukung oleh klien (misalnya, fungsi hash atau cipher blok mana yang diizinkan klien), dan nilai seragam (sebuah "tidak ada") N_C.
2. S merespons dengan memilih versi terbaru dari protokol yang didukungnya serta ciphersuite yang sesuai. Selain itu, ia mengirimkan kunci publiknya pk_S, sertifikatnya $\mathrm{cert}_{i \to S}$, dan nilai seragamnya sendiri N_S.
3. C memeriksa apakah salah satu kunci publik CA's yang dimilikinya, misalnya pk_i, cocok dengan CA yang menerbitkan sertifikat S's. Jika demikian, C memverifikasi sertifikat (dan juga memeriksa apakah sertifikat tersebut belum kedaluwarsa atau

dicabut) dan, jika berhasil, mengetahui bahwa pk_S adalah kunci publik S's. Kemudian berjalan $(c, \text{pmk}) \leftarrow \text{Encaps}_{pk_S}(1^n)$ (lihat Bagian 11.3) untuk mendapatkan ciphertext c dan apa yang disebut pmk kunci pra-master. Ini mengirimkan c ke server. pmk digunakan untuk memperoleh kunci master mk menggunakan fungsi derivasi kunci (lih. Bagian 5.6.4) yang diterapkan pada pmk, N_C, dan N_S. Klien kemudian menerapkan generator pseudorandom ke mk untuk mendapatkan empat kunci k_C, k'_C, k_S, k'_S.

Terakhir, C menghitung $\tau_C \leftarrow \text{Mac}_{\text{mk}}(\text{transkrip})$, di mana transkrip menunjukkan semua pesan yang dipertukarkan antara C dan S sejauh ini. Klien kemudian mengirimkan τ_C ke S. (Faktanya, τ_C sendiri dienkripsi dan diautentikasi seperti yang dilakukan untuk komunikasi di lapisan rekaman, dijelaskan di bawah.)

4. S menghitung $\text{pmk} := \text{Decaps}_{sk_S}(c)$, yang darinya ia dapat memperoleh mk dan k_C, k'_C, k_S, k'_S seperti yang dilakukan klien. Jika $\text{Vrfy}_{\text{mk}}(\text{transkrip}, \tau_C) \neq 1$, maka S dibatalkan. Jika tidak, ia menetapkan $\tau_C \leftarrow \text{Mac}_{\text{mk}}(\text{transkrip}')$, di mana $\text{transkrip}'$ menunjukkan semua pesan yang dipertukarkan antara C dan S sejauh ini (yaitu, termasuk pesan terakhir yang diterima dari C). S kemudian mengirimkan τ_C ke C. (Sekali lagi, τ_C sebenarnya dienkripsi dan diautentikasi sebagaimana lalu lintas lapisan rekaman.)
5. Jika $\text{Vrfy}_{\text{mk}}(\text{transkrip}', \tau_C) \neq 1$, klien dibatalkan.

Pada akhir keberhasilan eksekusi protokol jabat tangan, C dan S berbagi satu set empat kunci simetris k_C, k'_C, k_S, k'_S.

TLS 1.2 mendukung dua KEM: KEM berbasis CDH/DDH seperti pada Konstruksi 11.19, atau skema enkripsi berbasis RSA PKCS #1 v1.5. Untuk mencegah serangan gaya Bleichenbacher pada skema dalam kasus terakhir, server tidak boleh melaporkan kesalahan dekripsi jika Decaps gagal pada langkah 4. Sebaliknya, jika dekripsi gagal, server harus memilih pmk yang seragam dan kemudian terus menjalankan protokol menggunakan nilai tersebut. (Dalam hal ini, tentu saja, nilai τ_C tidak akan terverifikasi dan S akan dibatalkan; intinya adalah penyerang tidak dapat membedakan apakah hal ini disebabkan oleh kegagalan dekripsi atau bukan.)

Intuisi untuk keamanan protokol jabat tangan adalah karena C memverifikasi sertifikat, ia mengetahui bahwa hanya server sah S yang dapat mempelajari pmk dan mk. Jadi, jika protokol berhasil dihentikan, C mengetahui bahwa ia berbagi kunci k_C, k'_C, k_S, k'_S dengan S yang sah, dan tidak ada musuh yang mengetahui apa pun tentang kunci tersebut. Hal ini seharusnya berlaku bahkan di hadapan penyerang aktif, meskipun bukti formal mengenai hal ini cukup rumit. Kami mencatat, bagaimanapun, bahwa kode otentikasi pesan pada transkrip digunakan untuk mencegah penyerang man-in-the-middle mengubah versi protokol dan ciphersuite yang dikirim pada awal protokol jabat tangan. Hal ini mencegah penyerang menyebabkan S atau C menggunakan protokol versi lama yang lebih lemah, atau cipher blok dan kunci pendek yang lemah.

Protokol lapisan rekaman. Setelah kunci disetujui oleh C dan S, para pihak menggunakan kunci tersebut untuk mengenkripsi dan mengotentikasi semua komunikasi selanjutnya. C menggunakan k_C (resp., k'_C) untuk mengenkripsi (resp., mengautentikasi) semua pesan yang

dikirim ke S; sama halnya, S menggunakan k_S dan k'_S untuk mengenkripsi dan mengautentikasi semua pesan yang dikirimkannya. Nomor urut digunakan untuk mencegah serangan ulangan, seperti yang dibahas di Bagian 4.5. TLS 1.2 menggunakan pendekatan autentikasi lalu enkripsi, yang, seperti telah kita lihat di Bagian 4.5, dapat menimbulkan masalah.

12.9 SKRIPSI TANDA TANGAN

Untuk menutup bab ini, kami secara singkat dan informal membahas isu kerahasiaan dan integritas bersama dalam pengaturan kunci publik. Walaupun hal ini sejajar dengan pembahasan kita pada Bagian 4.5, fakta bahwa kita sekarang berada dalam setting kunci publik menimbulkan beberapa komplikasi tambahan.

Untuk mempermudah, kami mempertimbangkan jaringan di mana semua pihak terkait memiliki pasangan kunci publik/pribadi untuk enkripsi dan penandatanganan. Kita biarkan (ek, dk) menunjukkan kunci enkripsi (publik) dan kunci dekripsi (pribadi), dan menggunakan (vk, sk) untuk kunci verifikasi (publik) dan kunci penandatanganan (pribadi). Kami berasumsi semua pihak mengetahui kunci publik orang lain.

Secara informal, tujuan kami adalah merancang mekanisme yang memungkinkan pengirim S mengirim pesan m ke penerima R sambil memastikan bahwa (1) tidak ada pihak lain dalam jaringan yang dapat mengetahui informasi apa pun tentang m (yaitu kerahasiaan) dan (2) R yakin bahwa pesan tersebut datang dari S (yaitu, integritas). Kami ingin mempertimbangkan kedua properti keamanan ini bahkan terhadap serangan aktif (misalnya, teks tersandi yang dipilih) oleh pihak lain dalam jaringan.

Mengikuti diskusi kita di Bagian 4.5, ide alaminya adalah menggunakan pendekatan "enkripsi-lalu autentikasi" di mana S mengirimkan $\langle S, c, \mathrm{Sign}_{sk_S}(c)\rangle$ ke R, di mana c adalah enkripsi m menggunakan kunci enkripsi R ek_R. (Kami secara eksplisit menyertakan identitas pengirim di sini untuk kenyamanan.) Namun, ada serangan ciphertext yang dipilih dengan cerdas di sini terlepas dari skema enkripsi yang digunakan. Setelah mengamati transmisi seperti di atas, pihak A yang lain (yang berlawanan) dapat menghapus tanda tangan S dan menggantinya dengan miliknya sendiri, mengirimkan $\langle A, c, \mathrm{Sign}_{sk_A}A(c)\rangle$ ke R. Dalam hal ini, R tidak akan mendeteksi sesuatu yang salah, dan salah mengira bahwa A telah mengiriminya pesan m. Jika R membalas A, atau berperilaku terhadap A dengan cara yang bergantung pada isi pesan, maka A berpotensi mempelajari pesan yang tidak diketahui m.

(Masalah lain dalam skema ini, meskipun agak independen dari pembahasan kita di sini, adalah bahwa skema ini tidak lagi memberikan non-penyangkalan. Artinya, R tidak dapat dengan mudah membuktikan kepada pihak ketiga bahwa S telah menandatangani pesan m, setidaknya tanpa mengungkapkan pesannya. kunci dekripsi sendiri dk_R.)

Sebagai gantinya, seseorang dapat mencoba pendekatan "otentikasi lalu enkripsi". Di sini, S pertama-tama akan menghitung tanda tangan $\sigma \rightarrow \mathrm{Sign}_{sk_S}(m)$ dan kemudian mengirimkannya

$$\langle S, \mathrm{Enc}_{ek_R}(m \| \sigma) \rangle.$$

(Perhatikan bahwa ini memecahkan masalah non-penyangkalan yang disebutkan di atas.) Jika skema enkripsi hanya aman terhadap CPA, maka masalah seperti yang disebutkan di Bagian 4.5 juga berlaku, jadi mari kita asumsikan skema enkripsi aman CCA yang digunakan. Meski begitu, ada serangan yang dapat dilakukan oleh R yang berbahaya. Setelah menerima $\langle S, \mathrm{Enc}_{ek_R}(m \parallel \sigma) \rangle$ dari S, R yang jahat dapat mendekripsi untuk mendapatkan $m \parallel \sigma$, lalu mengenkripsi ulang dan mengirim $\langle S, \mathrm{Enc}_{ek_{R'}}(m \parallel \sigma) \rangle$ ke penerima lain R'. Penerima (jujur) R' ini kemudian akan berpikir bahwa S mengiriminya pesan m. Hal ini dapat menimbulkan konsekuensi yang serius, misalnya jika m adalah pesan "Saya berhutang $100 padamu."
Serangan-serangan ini dapat dicegah jika pihak-pihak lebih berhati-hati dalam menangani pengidentifikasi. Saat mengenkripsi, pengirim harus menyertakan identitasnya sendiri bersama dengan pesannya; pada saat penandatanganan, salah satu pihak harus menandatangani identitas penerima yang dituju beserta apa yang ditandatangani. Misalnya, pendekatan kedua akan dimodifikasi sehingga S terlebih dahulu menghitung $\sigma \rightarrow \mathrm{Sign}_{sk_S}(m \parallel R)$, dan kemudian mengirimkan $\langle S, \mathrm{Enc}_{ek_R}(S \parallel m \parallel \sigma) \rangle$ ke R. Saat mendekripsi, penerima harus memeriksa bahwa nilai dekripsi yang dihasilkan mencakup (konon) identitas pengirim; ketika memverifikasi, penerima harus memeriksa bahwa apa yang ditandatangani mengandung identitasnya sendiri. Saat menyertakan identitas dengan cara ini, autentikasi-lalu-enkripsi dan enkripsi-lalu-otentikasi aman jika skema enkripsi aman CCA dan skema tanda tangan yang sangat aman (yang definisinya sejajar dengan Definisi 4.3 untuk MAC) digunakan.

BAB 13
TOPIK LANJUTAN DALAM ENKRIPSI KUNCI PUBLIK

Pada Bab 11 kita melihat beberapa contoh skema enkripsi kunci publik yang digunakan dalam praktik. Di sini, kami mengeksplorasi beberapa skema yang saat ini lebih bersifat teoritis—walaupun dalam beberapa kasus ada kemungkinan bahwa skema ini (atau variannya) akan digunakan secara lebih luas di masa depan.

Kita mulai dengan pembahasan permutasi pintu jebakan, generalisasi permutasi satu arah, dan menunjukkan bagaimana menggunakannya untuk membangun skema enkripsi kunci publik. Permutasi pintu jebakan dengan rapi merangkum karakteristik utama dari permutasi RSA yang membuatnya sangat berguna. Oleh karena itu, mereka sering kali memberikan abstraksi yang berguna untuk merancang sistem kriptografi baru. Selanjutnya kami sajikan tiga skema berdasarkan permasalahan yang berkaitan dengan anjak piutang:

- Skema enkripsi Paillier merupakan salah satu contoh skema enkripsi yang bersifat homomorfik. Properti ini ternyata berguna untuk membangun protokol kriptografi yang lebih kompleks, sesuatu yang akan kita bahas secara singkat di Bagian 13.3.
- Skema enkripsi Goldwasser–Micali memiliki sejarah yang menarik sebagai skema pertama yang terbukti aman terhadap CPA. Ia juga homomorfik, dan menggunakan beberapa teori bilangan menarik yang dapat diterapkan dalam konteks lain.
- Terakhir, kita membahas permutasi pintu jebakan Rabin, yang dapat digunakan untuk membangun skema enkripsi kunci publik. Meskipun secara dangkal mirip dengan permutasi pintu jebakan RSA, permutasi pintu jebakan Rabin dibedakan berdasarkan fakta bahwa keamanannya didasarkan langsung pada kekerasan pemfaktoran. (Ingat dari Bagian 8.2 bahwa beratnya masalah RSA tampaknya merupakan asumsi yang lebih kuat.)

13.1 ENKRIPSI DARI PERMUTASI TRAPDOOR

Pada Bagian 11.5 kita melihat bagaimana membangun skema enkripsi kunci publik yang aman dengan CPA berdasarkan asumsi RSA. Dengan menyaring properti RSA yang digunakan dalam konstruksi, dan mendefinisikan gagasan abstrak yang merangkum properti tersebut, kami memperoleh templat umum untuk membangun skema enkripsi aman berdasarkan primitif apa pun yang memenuhi kumpulan properti yang sama. Permutasi pintu jebakan ternyata merupakan abstraksi yang “benar” di sini. Pada bagian berikut ini kita mendefinisikan (keluarga) permutasi pintu jebakan dan mengamati bahwa keluarga permutasi satu arah RSA (Konstruksi 8.77) memenuhi persyaratan tambahan yang diperlukan untuk menjadi keluarga permutasi pintu jebakan. Pada Bagian 13.1 kami menggeneralisasi konstruksi dari Bagian 11.5 dan menunjukkan bahwa enkripsi kunci publik dapat dibangun dari permutasi pintu jebakan apa pun. Hasil ini akan digunakan lagi di Bagian 13.5, dimana kami

menunjukkan contoh kedua dari permutasi pintu jebakan, kali ini berdasarkan langsung pada asumsi pemfaktoran.

Permutasi Pintu Trap

Ingat kembali definisi rumpun fungsi dan rumpun permutasi satu arah dari Bagian 8.4. Pada bagian tersebut, kami menunjukkan bahwa asumsi RSA secara alami menimbulkan permutasi satu arah. Pembaca yang cerdik mungkin telah memperhatikan bahwa konstruksi yang kami berikan (Konstruksi 8.77) memiliki sifat khusus yang tidak disebutkan di sana: yaitu, algoritma pembuatan parameter Gen mengeluarkan beberapa informasi tambahan bersama dengan I yang memungkinkan inversi efisien dari f_I. Kami menyebut informasi tambahan tersebut sebagai pintu jebakan, dan menyebut keluarga permutasi satu arah dengan properti tambahan ini sebagai keluarga permutasi pintu jebakan. Definisi formal berikut ini.

DEFINISI 13.1 Tupel algoritma waktu polinomial $(\mathrm{Gen}, \mathrm{Samp}, \mathrm{f}, \mathrm{Inv})$ adalah keluarga permutasi pintu jebakan (atau permutasi pintu jebakan) jika:

- Algoritma pembangkitan parameter probabilistik Gen, pada input 1^n, output (I, td) dengan $|I| \geq n$. Setiap nilai I mendefinisikan himpunan $\mathcal{D}_I$ yang membentuk domain dan rentang permutasi (yaitu, bijeksi) $f_I : \mathcal{D}_I \to \mathcal{D}_I$.
- Anggaplah Gen_1 menunjukkan algoritma yang dihasilkan dengan menjalankan Gen dan hanya mengeluarkan keluaran I. Maka $(\mathrm{Gen}_1, \mathrm{Samp}, \mathrm{f})$ adalah suatu kelompok permutasi satu arah.
- Misal (I, td) merupakan keluaran dari $\mathrm{Gen}(1^n)$. Algoritma pembalik deterministik Inv, pada masukan td dan $y \in \mathcal{D}_I$, menghasilkan keluaran $x \in \mathcal{D}_I$. Kami menyatakannya dengan $x := \mathrm{Inv}_{\mathrm{td}}(y)$. Diperlukan bahwa dengan probabilitas yang dapat diabaikan atas keluaran (I, td) oleh $\mathrm{Gen}(1^n)$ dan pilihan $x \in \mathcal{D}_I$ yang seragam, kita mempunyai

$$\mathrm{Inv}_{\mathrm{td}}(f_I(x)) = x.$$

Sebagai singkatannya, kami menghilangkan penyebutan Samp secara eksplisit dan hanya merujuk pada permutasi pintu jebakan $(\mathrm{Gen}, \mathrm{f}, \mathrm{Inv})$. Untuk keluaran (I, td) oleh Gen kita menulis $x \to \mathcal{D}_I$ untuk menunjukkan pemilihan seragam $x \in \mathcal{D}_I$ (dengan pemahaman bahwa ini dilakukan oleh algoritma Samp).

Kondisi kedua di atas menyiratkan bahwa f_I tidak dapat dibalik secara efisien tanpa td, namun kondisi terakhir berarti bahwa f_I dapat dibalik secara efisien dengan td. Konstruksi 8.77 dapat segera dimodifikasi untuk memberikan kumpulan permutasi pintu jebakan jika masalah RSA relatif sulit dibandingkan dengan GenRSA, sehingga kami menyebut konstruksi tersebut sebagai permutasi pintu jebakan RSA.

Enkripsi Kunci Publik dari Permutasi Trapdoor

Kami sekarang membuat sketsa bagaimana skema enkripsi kunci publik dapat dibangun dari rangkaian permutasi pintu jebakan yang sewenang-wenang. Konstruksi ini hanyalah sebuah generalisasi dari apa yang telah dilakukan untuk permutasi pintu jebakan RSA spesifik di Bagian 11.5.

Kita mulai dengan (kembali) memperkenalkan gagasan tentang predikat hard-core. Ini merupakan adaptasi alami dari Definisi 7.4 terhadap konteks kita, dan juga menggeneralisasi pembahasan kita sebelumnya mengenai satu predikat inti khusus untuk permutasi pintu jebakan RSA di Bagian 11.5.

DEFINISI 13.2 Misalkan $\Pi = (\mathrm{Gen}, \mathsf{f}, \mathrm{Inv})$ merupakan suatu keluarga permutasi pintu jebakan, dan misalkan hc adalah algoritma waktu polinomial deterministik yang, pada masukan I dan $x \in \mathcal{D}_I$, menghasilkan bit tunggal $\mathrm{hc}_I(x)$. Kita katakan bahwa hc adalah predikat inti dari Π jika untuk setiap algoritma waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan negl sehingga

$$\Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)] \leq \frac{1}{2} + \mathsf{negl}(n),$$

dimana probabilitas diambil alih dari eksperimen dimana $\mathrm{Gen}(1^n)$ dijalankan untuk menghasilkan (I, td) dan kemudian x dipilih secara seragam dari $\mathcal{D}_I$.

Asimetri yang diberikan oleh permutasi pintu jebakan menyiratkan bahwa siapa pun yang mengetahui td pintu jebakan yang terkait dengan I dapat memulihkan x dari $\mathsf{f}_I(x)$ dan dengan demikian menghitung $\mathrm{hc}_I(x)$ dari $\mathsf{f}_I(x)$. Namun jika hanya diberi I saja, maka tidak mungkin menghitung $\mathrm{hc}_I(x)$ dari $\mathsf{f}_I(x)$ untuk x yang seragam.
Hal berikut dapat dibuktikan dengan modifikasi Teorema 7.5 yang sesuai:

TEOREMA 13.3 Diberikan sebuah keluarga permutasi pintu jebakan Π, terdapat sebuah keluarga permutasi pintu jebakan $\widehat{\Pi}$ dengan predikat hard-core hc untuk $\widehat{\Pi}$.

Mengingat keluarga permutasi pintu jebakan $\widehat{\Pi} = (\widehat{\mathrm{Gen}}, \mathsf{f}, \mathrm{Inv})$ dengan predikat hard-core hc, kita dapat membangun skema enkripsi bit tunggal melalui pendekatan berikut (lihat Konstruksi 13.4 di bawah, dan bandingkan dengan Konstruksi 11.32): Untuk menghasilkan kunci, jalankan $\widehat{\mathrm{Gen}}(1^n)$ untuk mendapatkan (I, td); kunci publiknya adalah I dan kunci privatnya adalah td. Dengan adanya kunci publik I, enkripsi pesan $m \in \{0,1\}$ bekerja dengan memilih $r \in \mathcal{D}_I$ yang seragam dengan batasan $\mathrm{hc}_I(r) = m$, dan kemudian menetapkan ciphertext sama dengan $\mathsf{f}_I(r)$. Untuk mendekripsi, penerima menggunakan td untuk memulihkan r dari $\mathsf{f}_I(r)$ dan kemudian mengeluarkan pesan $m := \mathrm{hc}_I(r)$.

CONSTRUCTION 13.4

Let $\widehat{\Pi} = (\widehat{\mathsf{Gen}}, f, \mathsf{Inv})$ be a family of trapdoor permutations with hard-core predicate hc. Define a public-key encryption scheme as follows:

- Gen: on input 1^n, run $\widehat{\mathsf{Gen}}(1^n)$ to obtain (I, td). Output the public key I and the private key td.
- Enc: on input a public key I and a message $m \in \{0,1\}$, choose a uniform $r \in \mathcal{D}_I$ subject to the constraint that $\mathsf{hc}_I(r) = m$. Output the ciphertext $c := f_I(r)$.
- Dec: on input a private key td and a ciphertext c, compute the value $r := \mathsf{Inv}_I(c)$ and output the message $\mathsf{hc}_I(r)$.

Enkripsi kunci publik dari keluarga permutasi pintu jebakan mana pun.

Bukti keamanan mengikuti garis pembuktian Teorema 11.33.

TEOREMA 13.5 Jika $\widehat{\Pi}$ adalah keluarga permutasi pintu jebakan dengan hard-core

BUKTI Misalkan Π menyatakan Konstruksi 13.4. Kami membuktikan bahwa Π memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap; berdasarkan Proposisi 11.3, ini berarti aman untuk CPA.

Pertama-tama kita mengamati bahwa hc harus tidak memihak dalam pengertian berikut. Membiarkan

$$\delta_0(n) \stackrel{\text{def}}{=} \Pr_{(I,\mathsf{td})\leftarrow\widehat{\mathsf{Gen}}(1^n);x\leftarrow\mathcal{D}_I}[\mathsf{hc}_I(x)=0]$$

dan

$$\delta_1(n) \stackrel{\text{def}}{=} \Pr_{(I,\mathsf{td})\leftarrow\widehat{\mathsf{Gen}}(1^n);x\leftarrow\mathcal{D}_I}[\mathsf{hc}_I(x)=1].$$

Lalu ada fungsi yang dapat diabaikan negl sehingga

$$\delta_0(n), \delta_1(n) \geq \frac{1}{2} - \mathsf{negl}(n);$$

jika tidak, maka penyerang yang hanya mengeluarkan bit yang lebih sering muncul akan melanggar Definisi 13.2. Sekarang misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik. Tanpa kehilangan sifat umum, kita dapat mengasumsikan $m_0 = 0$ dan $m_1 = 1$ dalam eksperimen $\mathrm{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$.
Kami kemudian punya

$$\begin{aligned}\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)=1] = \frac{1}{2}\cdot\Pr[\mathcal{A}(pk,c)=0 \mid c \text{ is an encryption of } 0] \\ + \frac{1}{2}\cdot\Pr[\mathcal{A}(pk,c)=1 \mid c \text{ is an encryption of } 1].\end{aligned}$$

Tapi kemudian

$$
\begin{aligned}
&\Pr[\mathcal{A}(I, f_I(x)) = \mathsf{hc}_I(x)] \\
&= \delta_0(n) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 \mid \mathsf{hc}_I(x) = 0] \\
&\quad + \delta_1(n) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 \mid \mathsf{hc}_I(x) = 1] \\
&\geq \left(\frac{1}{2} - \mathsf{negl}(n)\right) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 \mid \mathsf{hc}_I(x) = 0] \\
&\quad + \left(\frac{1}{2} - \mathsf{negl}(n)\right) \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 \mid \mathsf{hc}_I(1) = 1] \\
&\geq \frac{1}{2} \cdot \Pr[\mathcal{A}(I, f_I(x)) = 0 \mid \mathsf{hc}_I(x) = 0] \\
&\quad + \frac{1}{2} \cdot \Pr[\mathcal{A}(I, f_I(x)) = 1 \mid \mathsf{hc}_I(1) = 1] - 2 \cdot \mathsf{negl}(n) \\
&= \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] - 2 \cdot \mathsf{negl}(n).
\end{aligned}
$$

Karena hc adalah predikat inti untuk $\widehat{\Pi}$, terdapat fungsi negl′ yang dapat diabaikan sehingga $\mathrm{negl}'(n) \geq \Pr[\mathcal{A}(I, f_I(x)) = \mathrm{hc}_I(x)]$; ini berarti itu

$$\Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \mathsf{negl}'(n) + 2 \cdot \mathsf{negl}(n),$$

melengkapi buktinya.

Mengenkripsi pesan yang lebih panjang. Dengan menggunakan Klaim 11.7, kita tahu bahwa kita dapat memperluas Konstruksi 13.4 untuk mengenkripsi pesan ℓ-bit menggunakan ciphertext ℓ kali lebih lama. Efisiensi yang lebih baik dapat diperoleh dengan membangun KEM, sesuai dengan Konstruksi 11.34. Kami meninggalkan detailnya sebagai latihan.

13.2 SKEMA ENKRIPSI PAILIER

Pada bagian ini kami menjelaskan skema enkripsi Paillier, skema enkripsi kunci publik yang keamanannya didasarkan pada asumsi yang terkait (tetapi tidak diketahui setara) dengan kekerasan pemfaktoran. Skema enkripsi ini sangat menarik karena memiliki beberapa sifat homomorfik yang bagus, seperti yang akan kita bahas lebih lanjut di Bagian 13.2

Skema enkripsi Paillier menggunakan grup $\mathbb{Z}*_{N^2}$, grup perkalian elemen dalam rentang $\{1, \ldots, N^2\}$ yang relatif prima terhadap N, untuk N hasil kali dua bilangan prima berbeda. Untuk memahami skema ini, ada gunanya memahami terlebih dahulu struktur $\mathbb{Z}*_{N^2}$. Karakterisasi yang berguna dari kelompok ini diberikan oleh proposisi berikut, yang antara lain mengatakan bahwa $\mathbb{Z}*_{N^2}$ isomorfik terhadap $\mathbb{Z}_N \times \mathbb{Z}*_N$ (lih. Definisi 8.23) untuk N dalam bentuk yang akan kita minati. Kami membuktikan proposisi di bagian selanjutnya. (Pembaca yang bersedia menerima proposisi tentang iman dapat melompat ke Bagian 13.2.2.)

PROPOSISI 13.6 Misalkan $N = pq$, dimana p, q adalah bilangan prima ganjil berlainan yang panjangnya sama. Kemudian:

1. $\gcd(N, \phi(N)) = 1$.
2. Untuk bilangan bulat $a \geq 0$, kita mempunyai $(1 + N)^a = (1 + aN) \bmod N^2$.

Akibatnya, orde $(1+N)$ pada $\mathbb{Z}*_{N^2}$ adalah N. Artinya, $(1+N)^N = 1 \bmod N^2$ dan $(1+N)^a \neq 1 \bmod N^2$ untuk sembarang $1 \leq a < N$.

3. $\mathbb{Z}_N \times \mathbb{Z}*_N$ isomorfik terhadap $\mathbb{Z}*_{N^2}$, dengan isomorfisme $f: \mathbb{Z}_N \times \mathbb{Z}*_N \rightarrow \mathbb{Z}*_{N^2}$ diberikan oleh

$$f(a,b) = [(1+N)^a \cdot b^N \bmod N^2].$$

Mengingat bagian terakhir dari proposisi di atas, kami memperkenalkan beberapa notasi yang mudah digunakan. Dengan memahami N, dan $x \in \mathbb{Z}*_{N^2}$, $a \in \mathbb{Z}_N$, $b \in \mathbb{Z}*_N$, kita tuliskan $x \leftrightarrow (a,b)$ jika $f(a,b) = x$ dimana f adalah isomorfisme dari proposisi di atas. Salah satu cara untuk memikirkan notasi ini adalah bahwa ini berarti "x dalam $\mathbb{Z}*_{N^2}$ sama dengan (a,b) dalam $\mathbb{Z}_N \times \mathbb{Z}*_N$." Kami telah menggunakan notasi yang sama dalam buku ini sehubungan dengan isomorfisme $\mathbb{Z}*_N \simeq \mathbb{Z}*_p \times \mathbb{Z}*_p$ yang diberikan oleh teorema sisa Cina; kami mempertahankan notasi tersebut karena dalam kedua kasus ini mengacu pada isomorfisme grup. Meskipun demikian, tidak boleh terjadi kebingungan karena golongan $\mathbb{Z}*_{N^2}$ dan proposisi di atas hanya digunakan pada bagian ini. Kami mencatat bahwa di sini isomorfisme—tetapi bukan kebalikannya—dapat dihitung secara efisien bahkan tanpa faktorisasi N.

Struktur $\mathbb{Z}*_{N^2}$

Bagian ini dikhususkan untuk pembuktian Proposisi 13.6. Secara keseluruhan, kita biarkan N, p, q seperti pada proposisi.

KLAIM 13.7 $\gcd(N, \phi(N)) = 1$.

BUKTI Ingatlah bahwa $\phi(N) = (p-1)(q-1)$. Asumsikan $p > q$ tanpa kehilangan keumuman. Karena p adalah bilangan prima dan $p > p-1 > q-1$, jelas $\gcd(p, \phi(N)) = 1$. Demikian pula, $\gcd(q, q-1) = 1$. Sekarang, jika $\gcd(q, p-1) \neq 1$ maka $\gcd(q, p-1) = q$ karena q adalah bilangan prima. Namun kemudian $(p-1)/q \geq 2$, bertentangan dengan asumsi bahwa p dan q memiliki panjang yang sama.

KLAIM 13.8 Untuk $a \geq 0$ bilangan bulat, kita mempunyai $(1+N)^a = 1 + aN \bmod N^2$. Jadi, orde $(1+N)$ pada $\mathbb{Z}*_{N^2}$ adalah N.

BUKTI Menggunakan teorema ekspansi binomial (Teorema A.1):

$$(1+N)^a = \sum_{i=0}^{a} \binom{a}{i} N^i.$$

Mengurangi modulo ruas kanan N^2, semua suku dengan $i \geq 2$ menjadi 0 sehingga $(1+N)^a = 1 + aN \bmod N^2$. Nilai bukan nol terkecil a sehingga $(1+N)^a = 1 \bmod N^2$ Oleh karena itu $a = N$.

KLAIM 13.9 Grup $\mathbb{Z}_N \times \mathbb{Z}*_N$ isomorfik terhadap grup $\mathbb{Z}*_{N^2}$, dengan isomorfisme $f: \mathbb{Z}_N \times \mathbb{Z}*_N \rightarrow \mathbb{Z}*_{N^2}$ diberikan oleh $f(a,b) = [(1+N)^a \cdot b^N \bmod N^2]$.

BUKTI Perhatikan bahwa $(1+N)^a \cdot b^N$ tidak mempunyai faktor persekutuan dengan N^2 karena $\gcd((1+N), N^2) = 1$ dan $\gcd(b, N^2) = 1$ (karena $b \in \mathbb{Z}*_N$). Jadi $[(1+N)^a \cdot b^N \bmod N^2]$ terletak pada $\mathbb{Z}*_{N^2}$. Sekarang kita buktikan bahwa f adalah isomorfisme.
Pertama-tama kita tunjukkan bahwa f adalah sebuah bijection. Sejak

$$|\mathbb{Z}^*_{N^2}| = \phi(N^2) = p \cdot (p-1) \cdot q \cdot (q-1) = pq \cdot (p-1)(q-1)$$
$$= |\mathbb{Z}_N| \cdot |\mathbb{Z}^*_N| = |\mathbb{Z}_N \times \mathbb{Z}^*_N|$$

(lihat Teorema 8.19 untuk persamaan kedua), cukuplah untuk menunjukkan bahwa ƒ adalah satu-satu. Katakanlah $a_1, a_2 \in \mathbb{Z}_N$ dan $b_1, b_2 \in \mathbb{Z}*_N$ sedemikian sehingga $f(a_1, b_1) = f(a_2, b_2)$.
Kemudian:

$$(1+N)^{a_1 - a_2} \cdot (b_1/b_2)^N = 1 \bmod N^2. \qquad (13.1)$$

(Perhatikan bahwa $b_2 \in \mathbb{Z}*_N$ dan dengan demikian $b_2 \in \mathbb{Z}*_{N^2}$, sehingga b_2 mempunyai modulo invers perkalian N^2.) Menaikkan kedua ruas ke pangkat $\phi(N)$ dan menggunakan fakta bahwa orde $\mathbb{Z}*_{N^2}$ adalah $\phi(N^2) = N \cdot \phi(N)$ kita peroleh

$$(1+N)^{(a_1 - a_2) \cdot \phi(N)} \cdot (b_1/b_2)^{N \cdot \phi(N)} = 1 \bmod N^2$$
$$\Rightarrow (1+N)^{(a_1 - a_2) \cdot \phi(N)} = 1 \bmod N^2 .$$

Berdasarkan Klaim 13.8, $(1+N)$ mempunyai orde N modulo N^2. Menerapkan Proposisi 8.53, kita melihat bahwa $(a_1 — a_2) \cdot \phi(N) = 0 \bmod N$ sehingga N membagi $(a_1 — a_2) \cdot \phi(N)$. Karena $\gcd(N, \phi(N)) = 1$ berdasarkan Klaim 13.7, maka $N | (a_1 — a_2)$. Karena $a_1, a_2 \in \mathbb{Z}_N$, hal ini hanya dapat terjadi jika $a_1 = a_2$.

Kembali ke Persamaan (13.1) dan menetapkan $a_1 = a_2$, kita mendapatkan $b_1^N = b_2^N \bmod N^2$. Ini berarti $b_1^N = b_2^N \bmod N$. Karena N relatif prima terhadap $\phi(N)$, orde $\mathbb{Z}*_N$, eksponensial pangkat N adalah bijeksi dalam $\mathbb{Z}*_N$ (lih. Akibat Akibat 8.17). Artinya $b_1 = b_2 \bmod N$; karena $b_1, b_2 \in \mathbb{Z}*_N$, kita mempunyai $b_1 = b_2$. Kami menyimpulkan bahwa f adalah satu-satu, dan karenanya merupakan bijection.

Untuk menunjukkan bahwa f merupakan isomorfisme, kita tunjukkan bahwa $f(a_1, b_1) \cdot f(a_2, b_2) = f(a_1 + a_2, b_1 \cdot b_2)$. (Perhatikan bahwa perkalian di ruas kiri persamaan terjadi modulo N^2, sedangkan penjumlahan/perkalian di ruas kanan terjadi modulo N.) Kita mempunyai:

$$f(a_1, b_1) \cdot f(a_2, b_2) = \left((1+N)^{a_1} \cdot b_1^N\right) \cdot \left((1+N)^{a_2} \cdot b_2^N\right) \bmod N^2$$
$$= (1+N)^{a_1+a_2} \cdot (b_1 b_2)^N \bmod N^2.$$

Karena $(1+N)$ mempunyai orde N modulo N^2 (menurut Klaim 13.8), kita dapat menerapkan Proposisi 8.52 dan memperoleh

$$f(a_1, b_1) \cdot f(a_2, b_2) = (1+N)^{a_1+a_2} \cdot (b_1 b_2)^N \bmod N^2$$
$$= (1+N)^{[a_1+a_2 \bmod N]} \cdot (b_1 b_2)^N \bmod N^2. \quad (13.2)$$

Kita belum selesai, karena $b_1 b_2$ pada Persamaan (13.2) merepresentasikan modulo perkalian N^2 sedangkan kita ingin modulo N. Misal $b_1 b_2 = r + \gamma N$, dengan γ, r adalah bilangan bulat dengan $1 \leq r < N$ (r tidak boleh bernilai 0 karena $b_1, b_2 \in \mathbb{Z}*_N$ sehingga hasil kali keduanya tidak habis dibagi N). Perhatikan bahwa $r = b_1 b_2 \bmod N$. Kami juga punya

$$(b_1 b_2)^N = (r + \gamma N)^N \bmod N^2$$
$$= \sum_{k=0}^{N} \binom{N}{k} r^{N-k} (\gamma N)^k \bmod N^2$$
$$= r^N + N \cdot r^{N-1} \cdot (\gamma N) = r^N = ([b_1 b_2 \bmod N])^N \bmod N^2,$$

menggunakan teorema ekspansi binomial seperti pada Klaim 13.8. Memasukkannya ke Persamaan (13.2) kita mendapatkan hasil yang diinginkan:

$$f(a_1, b_1) \cdot f(a_2, b_2) = (1+N)^{[a_1+a_2 \bmod N]} \cdot (b_1 b_2 \bmod N)^N \bmod N^2$$
$$= f(a_1 + a_2, b_1 b_2),$$

membuktikan bahwa f adalah isomorfisme dari $\mathbb{Z}_N \times \mathbb{Z}*_N$ ke $\mathbb{Z}*_{N^2}$.

Skema Enkripsi Paillier

Misalkan $N = pq$ adalah hasil kali dua bilangan prima berbeda yang panjangnya sama. Proposisi 13.6 mengatakan bahwa $\mathbb{Z}_N \times \mathbb{Z}*_N$ isomorfik terhadap $\mathbb{Z}*_{N^2}$, dengan isomorfisme yang diberikan oleh $f(a, b) = [(1+N)^a \cdot b^N \bmod N^2]$. Konsekuensinya adalah suatu unsur seragam $y \in \mathbb{Z}*_{N^2}$ bersesuaian dengan unsur seragam $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}*_N$ atau dengan kata lain unsur (a, b) yang seragam $a \in \mathbb{Z}_N$ dan seragam $b \in \mathbb{Z}*_N$. Sebut $y \in \mathbb{Z}*_{N^2}$ an N modul residu ke-N^2 jika y adalah pangkat ke-N, yakni jika terdapat $x \in \mathbb{Z}*_{N^2}$ dengan $y = x^N \bmod N^2$. Kita nyatakan himpunan ke- N residu modulo N^2 oleh Res(N^2). Mari kita karakterisasi residu ke-N di $\mathbb{Z}*_{N^2}$.

Mengambil $x \in \mathbb{Z}*_{N^2}$ dengan $x \leftrightarrow (a, b)$ dan menaikkannya ke pangkat N menghasilkan:

$$[x^N \bmod N^2] \leftrightarrow (a,b)^N = (N \cdot a \bmod N,\ b^N \bmod N) = (0, b^N \bmod N).$$

(Ingat bahwa operasi grup di $\mathbb{Z}_N \times \mathbb{Z}*_N$ adalah penjumlahan modulo N pada komponen pertama dan modulo perkalian N pada komponen kedua.) Selain itu, kita menyatakan bahwa setiap elemen y dengan $y \leftrightarrow (0,b)$ adalah elemen ke-N residu. Untuk melihatnya, ingatlah bahwa $\gcd(N, \phi(N)) = 1$ dan $d \stackrel{\text{def}}{=} [N^{-1} \bmod \phi(N)]$ ada. Jadi

$$(a,\ [b^d \bmod N])^N = (Na \bmod N, [b^{dN} \bmod N]) = (0,b) \leftrightarrow y$$

untuk setiap $a \in \mathbb{Z}_N$. Dengan demikian kita telah menunjukkan bahwa $\text{Res}(N^2)$ berkorespondensi dengan himpunan

$$\{(0,b) \mid b \in \mathbb{Z}_N^*\}.$$

Persamaan di atas juga menunjukkan bahwa banyaknya N akar-akar dari setiap $y \in \text{Res}(N^2)$ adalah tepat N, sehingga menghitung N pangkat adalah suatu fungsi N-ke-1. Dengan demikian, jika $r \in \mathbb{Z}*_{N^2}$ seragam maka $[r^N \bmod N^2]$ adalah elemen seragam dari $\text{Res}(N^2)$. Secara kasar, masalah residuositas gabungan yang dapat diputuskan adalah membedakan elemen seragam $\mathbb{Z}*_{N^2}$ dari elemen seragam $\text{Res}(N^2)$. Secara formal, biarkan GenModulus menjadi algoritma waktu polinomial yang, pada input 1^n, menghasilkan output (N,p,q) dengan $N = pq$, dan p dan q adalah n-bit bilangan prima (kecuali dengan probabilitas yang dapat diabaikan dalam n). Kemudian:

DEFINISI 13.10 Masalah residuositas gabungan keputusan relatif sulit terhadap GenModulus jika untuk semua algoritma waktu polinomial probabilistik D terdapat fungsi yang dapat diabaikan negl sehingga

$$\left|\Pr[D(N,[r^N \bmod N^2]) = 1] - \Pr[D(N,r) = 1]\right| \le \text{negl}(n),$$

di mana dalam setiap kasus probabilitas diambil alih eksperimen di mana $\text{GenModulus}(1^n)$ menghasilkan (N,p,q), dan kemudian $r \in \mathbb{Z}*_{N^2}$ yang seragam dipilih. (Ingat bahwa $[r^N \bmod N^2]$ adalah elemen seragam dari $\text{Res}(N^2)$.)

Asumsi residuositas komposit keputusan (DCR) adalah asumsi bahwa terdapat GenModulus yang relatif sulit untuk mengatasi masalah residuositas komposit keputusan. Seperti yang telah kita bahas, unsur $\mathbb{Z}*_{N^2}$ berbentuk (r',r) dengan r' dan r sembarang (dalam kelompok yang sesuai), sedangkan residu ke-N berbentuk $(0,r)$ dengan $r \in \mathbb{Z}*_N$ sewenang-wenang. Asumsi DCR adalah sulit membedakan elemen seragam tipe pertama dengan elemen seragam tipe kedua. Hal ini menyarankan cara abstrak berikut untuk mengenkripsi pesan $m \in$

$\mathbb{Z}_N$ sehubungan dengan kunci publik N: pilih residu ke N yang seragam $(0, r)$ dan atur cipherteksnya sama dengan

$$c \leftrightarrow (m, 1) \cdot (0, r) = (m + 0,\ 1 \cdot r) = (m, r).$$

Tanpa khawatir sekarang bagaimana hal ini dapat dilakukan secara efisien oleh pengirim, atau bagaimana penerima dapat mendekripsi, mari kita yakinkan diri kita sendiri (secara intuitif) bahwa ini aman. Karena residu ke-N yang seragam $(0, r)$ tidak dapat dibedakan dari elemen yang seragam (r', r), teks tersandi seperti yang dibangun di atas tidak dapat dibedakan (dari sudut pandang penyadap yang tidak mengetahui faktorisasi N) dari teks sandi

$$c' \leftrightarrow (m, 1) \cdot (r', r) = ([m + r' \bmod N], r)$$

untuk seragam $r' \in \mathbb{Z}_N$ dan $r \in \mathbb{Z}*_N$. Lemma 11.15 menunjukkan bahwa $[m + r' \, mod \, N]$ terdistribusi secara merata di $\mathbb{Z}_N$ sehingga, khususnya, ciphertext c' ini tidak bergantung pada pesan m. Keamanan CPA mengikuti. Bukti formal yang berjalan persis seperti ini diberikan lebih jauh di bawah.

Sebelum beralih ke deskripsi formal dan bukti keamanan, kami menunjukkan bagaimana enkripsi dan dekripsi dapat dilakukan secara efisien.

Enkripsi. Kami telah menjelaskan enkripsi di atas seolah-olah terjadi di $\mathbb{Z}_N \times \mathbb{Z}*_N$. Faktanya, ini terjadi pada grup isomorfik $\mathbb{Z}*_{N^2}$. Artinya, pengirim menghasilkan ciphertext $c \in \mathbb{Z}*_{N^2}$ dengan memilih seragam[1] $r \in \mathbb{Z}*_N$ dan kemudian menghitung

$$c := [(1 + N)^m \cdot r^N \bmod N^2].$$

Perhatikan itu

$$c = \left((1 + N)^m \cdot 1^N\right) \cdot \left((1 + N)^0 \cdot r^N\right) \bmod N^2 \leftrightarrow (m, 1) \cdot (0, r),$$

dan $c \leftrightarrow (m, r)$ sesuai keinginan.

Deskripsi. Kami sekarang menjelaskan bagaimana dekripsi dapat dilakukan secara efisien dengan faktorisasi N. Untuk c yang dibuat seperti di atas, kami mengklaim bahwa m dipulihkan dengan langkah-langkah berikut:

- Tetapkan $\hat{c} := [c^{\phi(N)} \bmod N^2]$.
- Tetapkan $\hat{m} := (\hat{c} — 1)/N$. (Perhatikan bahwa ini dilakukan pada bilangan bulat.)
- Tetapkan $m := [\hat{m} \cdot \phi(N)^{-1} \bmod N]$.

Untuk mengetahui mengapa ini berhasil, misalkan $c \leftrightarrow (m, r)$ untuk $r \in \mathbb{Z} *_N$ sembarang. Kemudian

$$\begin{aligned}\hat{c} &\stackrel{\text{def}}{=} [c^{\phi(N)} \bmod N^2] \\ &\leftrightarrow (m,\, r)^{\phi(N)} \\ &= \Big([m \cdot \phi(N) \bmod N],\ [r^{\phi(N)} \bmod N]\Big) \\ &= \big([m \cdot \phi(N) \bmod N],\ 1\big)\,.\end{aligned}$$

Dengan Proposisi 13.6(3), ini berarti $\hat{c} = (1+N)^{[m\cdot\phi(N) \bmod N]} \bmod N^2$. Dengan menggunakan Proposisi 13.6(2), kita mengetahui bahwa

$$\hat{c} = (1+N)^{[m\cdot\phi(N) \bmod N]} = (1 + [m \cdot \phi(N) \bmod N] \cdot N) \bmod N^2.$$

Karena $1 + [m \cdot \phi(N)\ mod\ N] \cdot N$ selalu lebih kecil dari N^2, kita dapat membuang mod N^2 di bagian akhir dan memandang persamaan di atas sebagai bilangan bulat. Jadi, $\hat{m} \stackrel{\text{def}}{=} (\hat{c} - 1)/N = [m \cdot \phi(N) \bmod N]$ dan, akhirnya,

$$m = [\hat{m} \cdot \phi(N)^{-1} \bmod N],$$

seperti yang dipersyaratkan. (Perhatikan bahwa $\phi(N)$ adalah modulo N yang dapat dibalik karena $\gcd(N, \phi(N)) = 1$.)

Kami memberikan gambaran lengkap tentang skema enkripsi Paillier, diikuti dengan contoh perhitungan di atas.

CONSTRUCTION 13.11

Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes (except with probability negligible in n). Define the following encryption scheme:

- Gen: on input 1^n run GenModulus(1^n) to obtain (N, p, q). The public key is N, and the private key is $\langle N, \phi(N)\rangle$.
- Enc: on input a public key N and a message $m \in \mathbb{Z}_N$, choose a uniform $r \leftarrow \mathbb{Z}_N^*$ and output the ciphertext

$$c := [(1+N)^m \cdot r^N \bmod N^2].$$

- Dec: on input a private key $\langle N, \phi(N)\rangle$ and a ciphertext c, compute

$$m := \left[\frac{[c^{\phi(N)} \bmod N^2] - 1}{N} \cdot \phi(N)^{-1} \bmod N\right].$$

Skema enkripsi Pailier.

Contoh 13.12
Misalkan $N = 11 \cdot 17 = 187$ (dan seterusnya $N^2 = 34969$), dan pertimbangkan untuk mengenkripsi pesan $m = 175$ dan kemudian mendekripsi teks sandi yang sesuai. Dengan memilih $r = 83 \in \mathbb{Z}*_{187}$, kita menghitung ciphertextnya

$$c := [(1 + 187)^{175} \cdot 83^{187} \bmod 34969] = 23911$$

sesuai dengan $(175, 83)$. Untuk mendekripsinya, perhatikan bahwa $\phi(N) = 160$. Jadi pertama-tama kita hitung $\hat{c} := [23911^{160} \bmod 34969] = 25620$. Mengurangi 1 dan membagi dengan 187 menghasilkan $\hat{m} := (25620— 1)/187 = 137$; karena $90 = [160— 1 \bmod 187]$, pesan dipulihkan sebagai $m := [137 \cdot 90 \bmod 187] = 175$.

TEOREMA 13.13 Jika masalah residuositas gabungan keputusan relatif sulit terhadap GenModulus, maka skema enkripsi Paillier aman terhadap CPA.

BUKTI Misalkan Π menunjukkan skema enkripsi Paillier. Kami membuktikan bahwa Π memiliki enkripsi yang tidak dapat dibedakan jika ada penyadap; berdasarkan Teorema 11.6 ini menyiratkan bahwa ini aman terhadap CPA.

Biarkan $\mathcal{A}$ menjadi musuh waktu polinomial probabilistik yang sewenang-wenang. Pertimbangkan algoritma PPT D berikut yang mencoba memecahkan masalah residuositas gabungan keputusan relatif terhadap GenModulus:

Algoritma D:
Algoritma diberikan N, y sebagai masukan.

- Atur $pk = N$ dan jalankan $\mathcal{A}(pk)$ untuk mendapatkan dua pesan m_0, m_1.
- Pilih bit b yang seragam dan atur $c := [(1 + N)^{m_b} \cdot y \bmod N^2]$.
- Berikan ciphertext c ke $\mathcal{A}$ dan dapatkan bit keluaran b'. Jika $b' = b$, keluaran 1; jika tidak, keluaran 0.

Mari kita analisa perilaku D. Ada dua kasus yang perlu dipertimbangkan:

Kasus 1: Katakanlah input ke D dihasilkan dengan menjalankan GenModulus(1^n) untuk mendapatkan (N, p, q), memilih seragam $r \in \mathbb{Z}*_{N^2}$, dan menyetel $y := [r^N \bmod N^2]$. (Artinya, y adalah elemen seragam dari Res(N^2).) Dalam kasus ini,

$$c = [(1 + N)^{m_b} \cdot r^N \bmod N^2]$$

untuk seragam $r \in \mathbb{Z}*_{N^2}$. Mengingat bahwa distribusi pada $[r^N \bmod N^2]$ adalah sama apakah r dipilih secara seragam dari $\mathbb{Z}*_N$ atau dari $\mathbb{Z}*_{N^2}$, kita melihat bahwa dalam kasus ini tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh D didistribusikan secara identik ke $\mathcal{A}$ lihat

di eksperimen $\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n)$. Karena D mengeluarkan 1 tepat ketika keluaran b' dari $\mathcal{A}$ sama dengan b, kita punya

$$\Pr\left[D(N,[r^N \bmod N^2]) = 1\right] = \Pr[\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n) = 1],$$

dimana probabilitas pertama diambil alih percobaan seperti pada Definisi 13.10.

Kasus 2: Katakanlah input ke D dihasilkan dengan menjalankan GenModulus(1^n) untuk mendapatkan (N,p,q) dan memilih seragam $y \in \mathbb{Z}*_{N^2}$. Kami mengklaim bahwa tampilan $\mathcal{A}$ dalam hal ini tidak bergantung pada bit b. Hal ini terjadi karena y adalah elemen seragam dari grup $\mathbb{Z}*_{N^2}$, sehingga teks tersandi c terdistribusi secara merata di $\mathbb{Z}*_{N^2}$ (lihat Lemma 11.15), tidak bergantung pada m. Jadi, peluang $b' = b$ dalam kasus ini adalah tepat $\frac{1}{2}$. Itu adalah,

$$\Pr[D(N,r) = 1] = \frac{1}{2},$$

dimana probabilitas diambil alih percobaan seperti pada Definisi 13.10.
Menggabungkan hal di atas, kita melihatnya

$$\begin{aligned}\left|\Pr\left[D(N,[r^N \bmod N^2]) = 1\right] - \Pr[D(N,r) = 1]\right| \\ = \left|\Pr[\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n) = 1] - \tfrac{1}{2}\right|.\end{aligned}$$

Dengan asumsi bahwa masalah residuositas gabungan keputusan relatif sulit dibandingkan GenModulus, terdapat pengabaian fungsi yang dapat diabaikan sehingga

$$\left|\Pr[\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n) = 1] - \tfrac{1}{2}\right| \leq \text{negl}(n).$$

Dengan demikian $\Pr\left[\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \text{negl}(n)$, melengkapi buktinya.

Enkripsi Homomorfik

Skema enkripsi Paillier berguna dalam sejumlah pengaturan karena bersifat homomorfik. Secara kasar, skema enkripsi homomorfik memungkinkan perhitungan (tertentu) dilakukan pada data terenkripsi, menghasilkan teks tersandi yang berisi hasil terenkripsi. Dalam kasus enkripsi Paillier, perhitungan yang dapat dilakukan adalah penjumlahan (modular). Secara khusus, perbaiki kunci publik $pk = N$. Kemudian skema Paillier mempunyai sifat yang mengalikan enkripsi m_1 dan enkripsi m_2 (dengan perkalian yang dilakukan modulo N^2) menghasilkan enkripsi $[m_1 + m_2 \bmod N]$; hal ini dikarenakan

$$\begin{aligned}&\left((1+N)^{m_1}\cdot r_1^N\right)\cdot\left((1+N)^{m_2}\cdot r_2^N\right)\\&=(1+N)^{[m_1+m_2 \bmod N]}\cdot(r_1r_2)^N \bmod N^2.\end{aligned}$$

Meskipun kemampuan untuk menambahkan nilai terenkripsi mungkin tidak terlalu berguna, hal ini cukup untuk beberapa aplikasi menarik termasuk voting, yang dibahas di bawah. Kami menyajikan definisi umum, di mana enkripsi Pailler merupakan kasus khusus.

DEFINISI 13.14 Skema enkripsi kunci publik (Gen, Enc, Des) bersifat homomorfik jika untuk semua n dan semua (pk, sk) keluaran Gen(1^n), dimungkinkan untuk mendefinisikan grup $\mathbb{M}, \mathbb{C}$ (bergantung pada pk saja) seperti yang:

Ruang pesannya adalah $\mathbb{M}$, dan semua teks tersandi yang dihasilkan oleh Enc_{pk} adalah elemen $\mathbb{C}$. Untuk kenyamanan notasi, kami menulis $\mathbb{M}$ sebagai grup aditif dan $\mathbb{C}$ sebagai grup perkalian.

Untuk setiap $m_1, m_2 \in \mathbb{M}$, setiap keluaran c_1 oleh $\mathrm{Enc}_{pk}(m_1)$, dan setiap keluaran c_2 oleh $\mathrm{Enc}_{pk}(m_2)$, dinyatakan bahwa

$$\mathrm{Dec}_{sk}(c_1 \cdot c_2) = m_1 + m_2.$$

Selain itu, distribusi pada ciphertext yang diperoleh dengan mengenkripsi m_1, mengenkripsi m_2, dan kemudian mengalikannya, hasilnya sama dengan distribusi pada ciphertext yang diperoleh dengan mengenkripsi $m_1 + m_2$.

Bagian terakhir dari definisi memastikan bahwa jika ciphertext $c_1 \leftarrow \mathrm{Enc}_{pk}(m_1)$ dan $c_2 \leftarrow \mathrm{Enc}_{pk}(m_2)$ dihasilkan dan hasilnya $c_3 := c_1 \cdot c_2$ dihitung, maka ciphertext c_3 yang dihasilkan tidak berisi informasi lebih lanjut tentang m_1 atau m_2 daripada jumlah m_3.

Skema enkripsi Paillier dengan $pk = N$ bersifat homomorfik dengan $\mathbb{M} = \mathbb{Z}_N$ dan $C = \mathbb{Z}_N^*$. Ini bukan contoh pertama skema enkripsi homomorfik yang kita lihat; Enkripsi El Gamal juga homomorfik. Khususnya, untuk kunci publik $pk = \langle \mathbb{G}, q, g, h \rangle$ kita dapat mengambil $\mathbb{M} = \mathbb{G}$ dan $\mathbb{C} = \mathbb{G} \times \mathbb{G}$; Kemudian

$$\langle g^{y_1},\ h^{y_1}\cdot m_1\rangle \cdot \langle g^{y_2},\ h^{y_2}\cdot m_2\rangle = \langle g^{y_1+y_2},\ h^{y_1+y_2}\cdot m_1m_2\rangle,$$

dimana perkalian ciphertext adalah berdasarkan komponen. Skema enkripsi Goldwasser–Micali yang akan kita lihat nanti juga bersifat homomorfik.

Fitur bagus dari enkripsi Paillier adalah homomorfik pada grup aditif besar (yaitu, $\mathbb{Z}_N$). Untuk melihat penerapan hal ini, pertimbangkan skema pemungutan suara terdistribusi

berikut ini, di mana ℓ pemilih dapat memilih "tidak" atau "ya" dan tujuannya adalah untuk membuat tabulasi jumlah suara "ya":

1. Otoritas pemungutan suara menghasilkan kunci publik N untuk skema enkripsi Paillier dan mempublikasikan N.
2. Misalkan 0 berarti "tidak", dan 1 berarti "ya". Setiap pemilih memberikan suaranya dengan mengenkripsinya. Artinya, pemilih i memberikan suaranya v_i dengan menghitung $c_i := [(1+N)^{v_i}(r_i)^N \bmod N^2]$ untuk seragam $r_i \in \mathbb{Z}_N^*$.
3. Setiap pemilih menyiarkan suaranya c_i. Suara-suara ini kemudian dikumpulkan secara publik melalui komputasi
$$c_{\text{total}} := \left[\prod_{i=1}^{\ell} c_i \bmod N^2\right].$$
4. Kewenangan yang diberikan secara c_{total}. (Kami berasumsi pihak berwenang belum dapat mengamati apa yang terjadi hingga saat ini.) Dengan mendekripsinya, pihak berwenang memperoleh total suara
$$v_{\text{total}} \stackrel{\text{def}}{=} \sum_{i=1}^{\ell} v_i \bmod N.$$
Jika ℓ kecil (sehingga $v_{total} \ll N$), tidak ada modulo N yang membungkus dan $v_{total} = \sum_{i=1}^{\ell} v_i$.

Ciri utama dari hal di atas adalah tidak ada pemilih yang mengetahui suara orang lain, dan penghitungan total suara dapat diverifikasi secara publik jika pihak berwenang dipercaya untuk menghitung dengan benar vtotal dari ctotal. Selain itu, pihak berwenang memperoleh jumlah yang benar tanpa mempelajari satu pun suara. (Di sini, kami berasumsi bahwa pihak berwenang tidak dapat melihat ciphertext pemilih. Dalam Bagian 13.3 kami menunjukkan sebuah protokol di mana suara tetap disembunyikan dari pihak berwenang meskipun mereka melihat seluruh komunikasinya.) Kami berasumsi semua pemilih bertindak jujur (dan hanya mencoba untuk mempelajari suara orang lain berdasarkan informasi yang mereka amati); seluruh bidang penelitian kriptografi didedikasikan untuk mengatasi potensi ancaman dari partisipan yang mungkin jahat dan tidak mengikuti protokol.

13.3 BERBAGI RAHASIA DAN ENKRIPSI AMBANG BATAS

Termotivasi oleh diskusi tentang pemungutan suara terdistribusi di bagian sebelumnya, kami secara singkat membahas protokol yang aman (interaktif). Protokol-protokol seperti itu bisa jauh lebih rumit daripada protokol-protokol kriptografi primitif (misalnya, skema enkripsi dan tanda tangan) yang telah kita fokuskan sampai saat ini, karena protokol-protokol tersebut dapat melibatkan banyak pihak untuk bertukar beberapa putaran pesan, dan juga karena protokol-protokol tersebut dimaksudkan untuk mewujudkan persyaratan keamanan yang lebih kompleks.

Tujuan utama dari bagian ini adalah untuk memberi pembaca gambaran tentang area yang menarik ini, dan tidak ada upaya yang dilakukan untuk menjelaskannya secara komprehensif atau lengkap. Meskipun protokol yang disajikan di sini dapat dibuktikan aman (sehubungan dengan definisi yang tepat), kami menghilangkan definisi formal, rincian, dan bukti dan lebih mengandalkan diskusi informal.

Berbagi Rahasia

Perhatikan permasalahan berikut ini. Seorang dealer memegang sebuah rahasia—misalnya, kode peluncuran nuklir—yang ingin dibagikan kepada sejumlah N pengguna $P_1, \ldots, P_N$ dengan memberikan bagian kepada setiap pengguna. Setiap t pengguna harus dapat mengumpulkan saham mereka dan merekonstruksi rahasianya, namun tidak ada koalisi yang berjumlah kurang dari t pengguna yang boleh mendapatkan informasi apa pun tentang s dari saham kolektif mereka (di luar informasi apa pun yang sudah mereka miliki). Kami menyebut mekanisme pembagian tersebut sebagai skema pembagian rahasia (t, N)-ambang batas. Skema seperti ini memastikan bahwa data s tidak terungkap tanpa otorisasi yang memadai, sekaligus menjamin ketersediaan data s ketika dibutuhkan (karena t pengguna mana pun dapat merekonstruksinya). Di luar penerapan langsungnya, skema pembagian rahasia juga merupakan landasan bagi banyak protokol kriptografi.

Pertimbangkan solusi sederhana untuk kasus ini $t = N$, dengan asumsi $s \in \{0,1\}^{\ell}$. Dealer memilih seragam $s_1, \ldots, s_{N-1} \in \{0,1\}^{\ell}$ dan himpunan $s_N := s \oplus (\bigoplus_{i=1}^{N-1} s_i)$; bagian pengguna P_i adalah s_i. Karena $\bigoplus_{i=1}^{N-1} s_i = s$ berdasarkan konstruksi, jelas semua pengguna bersama-sama dapat memulihkan s. Namun, bagian dari koalisi pengguna $N-1$ (bersama) seragam dan independen terhadap s, sehingga tidak mengungkapkan informasi tentang s. Hal ini terlihat jelas ketika koalisi berada pada $P_1, \ldots, P_{N-1}$. Dalam kasus umum, ketika koalisi mencakup semua orang kecuali $P_j (j \neq N)$, hal ini benar karena $s_i, \ldots, s_{j-1}, s_{j+1}, \ldots, s_{N-1}$ seragam dan tidak bergantung pada s berdasarkan konstruksi, dan

$$s_N = s \oplus \left(\bigoplus_{i<N, i\neq j} s_i\right) \oplus s_j;$$

dengan demikian, meskipun dikondisikan pada beberapa nilai tetap untuk s dan bagian anggota koalisi lainnya, bagian s_N dari pengguna P_N adalah seragam karena s_j seragam dan tidak bergantung pada s.

Kita dapat memperluasnya untuk mendapatkan solusi untuk $t < N$. Ide dasarnya adalah mereplikasi skema di atas untuk setiap subset $T \subset N$ dengan ukuran t. Artinya, untuk setiap subset $T = \{P_{i_1}, \ldots, P_{i_t}\}$, kita pilih bagian yang seragam $s_{T,i_1}, \ldots, s_{T,i_t}$ itu tunduk pada batasan bahwa $\bigoplus_{j=1}^{t} s_{T,i_j} = s$, dan memberikan s_{T,i_j} kepada pengguna P_{i_j}. Tidak sulit untuk melihat bahwa hal ini memenuhi persyaratan.

Sayangnya, perluasan skema awal ini tidak efisien. Setiap pengguna sekarang menyimpan $s_{T,i}$ untuk setiap subset T dimana dia menjadi anggotanya. Untuk setiap pengguna terdapat $\binom{N-1}{t-1}$ himpunan bagian tersebut, yang eksponensial dalam N jika $t \approx N/2$.

Skema Shamir. Untungnya, kita bisa melakukan hal yang jauh lebih baik dengan menggunakan skema berbagi rahasia yang diperkenalkan oleh Adi Shamir (dari RSA yang terkenal). Skema ini didasarkan pada polinomial^2 pada bidang berhingga $\mathbb{F}$, dimana $\mathbb{F}$ dipilih sehingga $s \in \mathbb{F}$ dan $|\mathbb{F}| > N$. (Lihat Lampiran A.5 untuk pembahasan singkat mengenai medan berhingga.) Sebelum menjelaskan skemanya, kami meninjau secara singkat beberapa latar belakang terkait polinomial pada bidang $\mathbb{F}$.

Nilai $x \in \mathbb{F}$ adalah akar dari suatu polinomial p jika $p(x) = 0$. Kita menggunakan fakta yang diketahui bahwa setiap polinomial bukan nol, derajat-t pada suatu bidang mempunyai paling banyak t akar.
Ini menyiratkan:

KORELARI 13.15 Dua polinomial derajat-t berbeda p dan q sepakat pada paling banyak t poin.

BUKTI Jika tidak, maka polinomial bukan nol berderajat t $p - q$ akan memiliki lebih dari t akar.

Skema Shamir bergantung pada fakta bahwa untuk setiap t pasang elemen $(x_1, y_1), \ldots, (x_t, y_t)$ dari $\mathbb{F}$ (dengan perbedaan $\{x_i\}$), terdapat polinomial unik p berderajat $(t-1)$ sehingga $p(x_i) = y_i$ untuk $1 \leq i \leq t$. Kita bisa membuktikannya dengan cukup mudah. Fakta bahwa terdapat p seperti itu menggunakan interpolasi polinomial standar.
Detailnya: untuk $i = 1, \ldots, t$, tentukan polinomial derajat-(t— 1).

$$\delta_i(X) \stackrel{\text{def}}{=} \frac{\prod_{j=1, j\neq i}^{t}(X - x_j)}{\prod_{j=1, j\neq i}^{t}(x_i - x_j)}.$$

Perhatikan bahwa $\delta_i(x_j) = 0$ untuk sembarang $j \neq i$, dan $\delta_i(x_i) = 1$. Jadi $p(X) \stackrel{\text{def}}{=} \sum_{i=1}^{t} \delta_i(X) \cdot y_i$ adalah polinomial berderajat $(t-1)$ dengan $p(x_i) = y_i$ untuk $1 \leq i \leq t$. (Kami mencatat bahwa hal ini, pada kenyataannya, menunjukkan bahwa polinomial p yang diinginkan dapat ditemukan secara efisien.) Keunikan mengikuti akibat wajar 13.15.

Kami sekarang menjelaskan skema pembagian rahasia Shamir (t, N)-threshold. Misalkan $\mathbb{F}$ adalah field berhingga yang memuat domain rahasia yang mungkin, dan dengan $|\mathbb{F}| > N$.

Misal $x_1, \ldots, x_N \in \mathbb{F}$ merupakan elemen berbeda dan bukan nol yang tetap dan diketahui publik. (Elemen seperti itu ada sejak $|\mathbb{F}| > N$.) Skemanya bekerja sebagai berikut:

Berbagi: Diberikan rahasia $s \in \mathbb{F}$, dealer memilih seragam $a_1, \ldots, a_{t-1} \in \mathbb{F}$ dan mendefinisikan polinomial $p(X) \stackrel{\text{def}}{=} s + \sum_{i=1}^{t-1} a_i X_i$. Ini adalah polinomial derajat seragam-$(t-1)$ dengan suku konstan s. Bagian pengguna P_i adalah $s_i := p(x_i) \in \mathbb{F}$.

Rekonstruksi: Katakanlah kepada pengguna $P_{i_1}, \ldots, P_{i_t}$ pool bagiannya $s_{i_1}, \ldots, s_{i_t}$. Dengan menggunakan interpolasi polinomial, mereka menghitung polinomial derajat-$(t-1)$ unik p' yang mana $p'(x_{i_j}) = s_{i_j}$ untuk $1 \leq j \leq t$. Rahasianya adalah $p'(0)$.

Jelas bahwa rekonstruksi berhasil karena $p' = p$ dan $p(0) = s$.
Masih terlihat bahwa setiap $t-1$ pengguna tidak mengetahui apa pun tentang rahasia s dari share mereka. Secara simetri, cukup dengan mempertimbangkan bagian pengguna $P_1, \ldots, P_{t-1}$. Kami mengklaim bahwa untuk rahasia apa pun s, bagiannya $s_1, \ldots, s_{t-1}$ (bersama) seragam. Karena dealer memilih $a_1, \ldots, a_{t-1}$ secara seragam, hal ini terjadi jika kita menunjukkan bahwa terdapat korespondensi satu-satu antara polinomial p yang dipilih oleh

dealer dan bagian $s_1, \ldots, s_{t-1}$. Namun hal ini merupakan konsekuensi langsung dari Akibat Kolelari 13.15.

Pembagian Rahasia yang Dapat Diverifikasi

Sejauh ini kami telah mempertimbangkan serangan pasif di mana $t-1$ pengguna mungkin mencoba menggunakan bagian mereka untuk mempelajari informasi tentang rahasia tersebut. Namun kita mungkin juga khawatir dengan perilaku aktif dan jahat. Di sini ada dua permasalahan yang berbeda: Pertama, dealer yang korup dapat memberikan bagian yang tidak konsisten kepada pengguna, yaitu, rahasia yang berbeda dapat dipulihkan tergantung pada t pengguna mana yang mengumpulkan bagian mereka. Kedua, dalam tahap rekonstruksi, pengguna jahat dapat memberikan bagian yang berbeda dari yang diberikan oleh dealer, dan dengan demikian mempengaruhi rahasia yang dipulihkan. (Meskipun hal ini dapat diatasi dengan meminta dealer menandatangani sahamnya, hal ini tidak akan berhasil jika dealer itu sendiri mungkin tidak jujur.) Skema pembagian rahasia yang dapat diverifikasi (VSS) mencegah kedua serangan ini.

Secara lebih formal, kami mengizinkan $t-1$ pengguna untuk melakukan korupsi dan berkolusi satu sama lain dan, mungkin, dengan dealer. Kami mengharuskan (1a) di akhir fase berbagi, sebuah rahasia s didefinisikan sedemikian rupa sehingga setiap koleksi yang mencakup t pengguna yang tidak rusak (baik koleksi ini juga mencakup beberapa pengguna yang rusak atau tidak) akan berhasil memulihkan s dalam fase rekonstruksi; apalagi (1b) jika dealer jujur, maka s sesuai dengan rahasia dealer. Selain itu, (2) ketika dealer jujur, maka, seperti sebelumnya, $t-1$ pengguna yang korup tidak mengetahui apa pun tentang rahasia saham mereka dan informasi publik apa pun yang dipublikasikan dealer. Karena kami ingin ada t pengguna yang tidak rusak meskipun $t-1$ pengguna rusak, kami memerlukan $N \geq t + (t-1) > 2(t-1)$; dengan kata lain, kami berasumsi sebagian besar pengguna tidak melakukan korupsi.

Kami menggambarkan skema VSS karena Feldman yang mengandalkan algoritma $\mathcal{G}$ yang relatif sulit mengatasi masalah logaritma diskrit. Untuk mempermudah, kami mendeskripsikan dalam model oracle acak dan biarkan H menunjukkan fungsi yang akan dimodelkan sebagai oracle acak. Kami juga berasumsi bahwa beberapa parameter terpercaya $(\mathbb{G}, q, g)$, yang dihasilkan menggunakan $\mathcal{G}(1^n)$, dipublikasikan terlebih dahulu, dengan q adalah bilangan prima dan $\mathbb{Z}_q$ adalah sebuah bidang. Terakhir, kami berasumsi bahwa semua pengguna memiliki akses ke kesaluran siaran, sehingga pesan yang disiarkan oleh pengguna mana pun dapat didengar oleh semua orang. Fase berbagi sekarang melibatkan N pengguna yang menjalankan protokol interaktif dengan dealer yang berlangsung sebagai berikut:

1. Untuk membagikan rahasia s, dealer memilih seragam $a_0 \in \mathbb{Z}_q$ dan kemudian membagikan a_0 seperti pada skema Shamir. Artinya, dealer memilih seragam $a_1, \ldots, a_{t-1} \in \mathbb{Z}_q$ dan mendefinisikan polinomial $p(X) \stackrel{\text{def}}{=} \sum_{i=1}^{t-1} a_i X_i$. Dealer mengirimkan bagian $s_i := p(i) = \sum_{i=1}^{t-1} a_i \cdot i^j$ ke pengguna P_i .[3]
   Selain itu, dealer menyiarkan nilai $A_0 := g^{a_0}, \ldots, A_{t-1} := g^{a_{t-1}}$, dan "rahasia terselubung" $c := H(a_0) \oplus s$.
2. Setiap pengguna P_i memverifikasi bahwa bagiannya memenuhi

$$g^{s_i} \stackrel{?}{=} \prod_{j=0}^{t-1} (A_j)^{i^j}. \qquad (13.3)$$

Jika tidak, P_i akan menyiarkan keluhannya secara terbuka.
Perhatikan bahwa jika dealer jujur, kita punya

$$\prod_{j=0}^{t-1} (A_j)^{i^j} = \prod_{j=0}^{t-1} \left(g^{a_j}\right)^{i^j} = g^{\sum_{j=0}^{t-1} a_j \cdot i^j} = g^{p(i)} = g^{s_i},$$

jadi tidak ada pengguna jujur yang akan mengeluh. Karena paling banyak ada $t-1$ pengguna yang rusak, paling banyak ada $t-1$ keluhan jika penyalurnya jujur.

3. Jika lebih dari $t-1$ pengguna mengeluh, dealer didiskualifikasi dan protokol dibatalkan. Jika tidak, dealer akan menanggapi keluhan dari P_i dengan menyiarkan s_i. Jika bagian ini tidak memenuhi Persamaan (13.3) (atau jika dealer menolak untuk menanggapi keluhan sama sekali), dealer didiskualifikasi dan protokol dibatalkan. Jika tidak, P_i menggunakan nilai siaran (bukan nilai yang diterima pada putaran pertama) sebagai bagiannya.

Dalam tahap rekonstruksi, katakanlah sekelompok pengguna (yang mencakup setidaknya t pengguna yang tidak dikorupsi) mengumpulkan bagian mereka. Bagian s_i yang disediakan oleh pengguna P_i dibuang jika tidak memenuhi Persamaan (13.3). Di antara sisa saham, sebanyak t di antaranya digunakan untuk memulihkan a_0 persis seperti skema Shamir. Rahasia aslinya kemudian dihitung sebagai $s := c \oplus H(a_0)$.

Kami sekarang berpendapat bahwa protokol ini memenuhi persyaratan keamanan yang diinginkan. Kami pertama-tama menunjukkan bahwa, dengan asumsi dealer tidak didiskualifikasi, nilai yang diperoleh kembali dalam tahap rekonstruksi ditentukan secara unik oleh informasi publik; khususnya, nilai yang dipulihkan adalah $c \oplus H(\log_g A_0)$. (Dikombinasikan dengan fakta bahwa dealer yang jujur tidak pernah didiskualifikasi, ini membuktikan bahwa kondisi (1a) dan (1b) berlaku.) Definisikan $a_i := \log_g A_i$ untuk $0 \le i \le t-1$; $\{a_i\}$ tidak dapat dihitung secara efisien jika permasalahan logaritma diskritnya sulit, namun permasalahan tersebut masih terdefinisi dengan baik. Definisikan polinomial $p(X) \stackrel{\text{def}}{=} \sum_{i=1}^{t-1} a_i X_i$. Setiap bagian s_i, yang disumbangkan oleh pihak P_i, yang tidak dibuang selama tahap rekonstruksi harus memenuhi Persamaan (13.3), dan karenanya memenuhi $s_i = p(i)$. Oleh karena itu, terlepas dari bagian mana yang digunakan, para pihak akan merekonstruksi polinomial p, menghitung $a_0 = p(0)$, dan kemudian memulihkan $s = c \oplus H(a_0)$.

Hal ini juga memungkinkan untuk menunjukkan bahwa kondisi (2) berlaku untuk musuh yang dibatasi secara komputasi jika masalah logaritma diskrit sulit untuk $\mathcal{G}$. (Berbeda dengan skema pembagian rahasia Shamir, kerahasiaan di sini tidak lagi tanpa syarat. Skema VSS yang aman tanpa syarat dimungkinkan. , tetapi berada di luar cakupan perlakuan kami.) Secara intuitif, hal ini karena rahasia s ditutupi oleh nilai acak $H(a_0)$, dan informasi yang diberikan kepada setiap pengguna $t-1$ dalam fase berbagi—yaitu, pembagian dan nilai publik $\{A_i\}$—hanya mengungkapkan g^{a_0}, sehingga sulit untuk menghitung a_0. Intuisi ini bisa dibuat ketat, tapi kita tidak melakukannya di sini.

Enkripsi Ambang Batas dan Pemungutan Suara Elektronik

Pada Bagian ini memperkenalkan gagasan skema enkripsi homomorfik dan memberikan skema enkripsi Paillier sebagai contoh. Di sini kami menunjukkan skema enkripsi homomorfik berbeda yang merupakan varian dari enkripsi El Gamal. Secara khusus, dengan kunci publik $pk = \langle \mathbb{G}, q, g, h \rangle$ seperti pada enkripsi El Gamal biasa, kita sekarang mengenkripsi pesan $m \in \mathbb{Z}_q$ dengan mengatur $M := g^m$, memilih seragam $y \in \mathbb{Z}_q$, dan mengirimkan ciphertext $c := \langle g^y h^y \cdot M \rangle$. Untuk mendekripsi, penerima memulihkan M seperti pada dekripsi standar El Gamal dan kemudian menghitung $m := \log^g M$. Meskipun hal ini tidak efisien jika m berasal dari domain yang besar, jika m berasal dari domain kecil—seperti yang akan diterapkan dalam aplikasi kita—maka penerima dapat menghitung $\log_g M$ secara efisien menggunakan pencarian menyeluruh. Keuntungan skema varian ini adalah homomorfik terhadap penjumlahan $\mathbb{Z}_q$. Itu adalah,

$$\langle g^{y_1},\ h^{y_1} \cdot g^{m_1} \rangle \cdot \langle g^{y_2},\ h^{y_2} \cdot g^{m_2} \rangle = \langle g^{y_1+y_2},\ h^{y_1+y_2} \cdot g^{m_1+m_2} \rangle.$$

Ingatlah bahwa pendekatan dasar pemungutan suara elektronik menggunakan enkripsi homomorfik membuat setiap pemilih i mengenkripsi suaranya $v_i \in \{0,1\}$ untuk mendapatkan ciphertext c_i. Setelah semua orang memilih, teks sandi dikalikan untuk mendapatkan enkripsi dengan jumlah $v_{total} \stackrel{\text{def}}{=} \sum_i v_i \mod q = \sum_i v_i$. (Nilai q, dalam prakteknya, cukup besar sehingga tidak terjadi modulo q yang membungkus.) Karena $0 \leq v_{total} \leq \ell$, dimana ℓ adalah jumlah total pemilih, otoritas dengan kunci privat dapat secara efisien mendeskripsi teks sandi akhir dan memulihkan v_{total}.

Kelemahan dari pendekatan ini adalah bahwa pihak yang berwenang dipercaya, baik untuk (dengan benar) mendekripsi ciphertext akhir maupun tidak untuk mendekripsi ciphertext milik masing-masing pemilih. (Dalam Bagian 13.2 kita berasumsi bahwa pihak berwenang tidak dapat melihat cipherteks pemilih individu.) Kita mungkin lebih memilih untuk mendistribusikan kepercayaan di antara sekumpulan N otoritas, sehingga kumpulan t otoritas mana pun dapat secara bersama-sama mendeskripsi kesepakatan yang telah disepakati ciphertext (hal ini memastikan ketersediaan meskipun beberapa otoritas tidak aktif atau tidak mau membantu mendeskripsi), namun tidak ada kumpulan $t-1$ otoritas yang mampu mendekripsi ciphertext apa pun sendiri (hal ini menjamin privasi selama kurang dari t otoritas yang rusak).

Pada pandangan pertama, tampaknya berbagi rahasia dapat menyelesaikan masalah. Jika kita membagi kunci privat di antara N otoritas, maka tidak ada kumpulan otoritas $t-1$ yang mempelajari kunci privat tersebut sehingga mereka tidak dapat mendekripsi. Di sisi lain, otoritas mana pun dapat mengumpulkan bagian mereka, memulihkan kunci privat, dan kemudian mendekripsi teks sandi apa pun yang diinginkan.

Sedikit pemikiran menunjukkan bahwa ini tidak berhasil. Jika pihak berwenang merekonstruksi kunci privat untuk mendekripsi beberapa ciphertext, maka sebagai bagian dari

proses ini semua pihak berwenang mempelajari kunci privat tersebut! Jadi, setelahnya, otoritas mana pun dapat mendekripsi teks tersandi apa pun yang mereka pilih sendiri.

Yang kita perlukan adalah pendekatan yang dimodifikasi dimana "rahasia" (yaitu kunci privat) tidak pernah direkonstruksi secara jelas, namun secara implisit direkonstruksi hanya cukup untuk memungkinkan dekripsi satu teks sandi yang telah disepakati. Kita dapat mencapai hal ini untuk kasus spesifik enkripsi El Gamal dengan cara berikut. Perbaiki kunci publik $pk = \langle \mathbb{G}, q, g, h \rangle$, dan misalkan $x \in \mathbb{Z}_q$ menjadi kunci privat, yaitu $g^x = h$. Setiap otoritas diberi bagian $x_i \in \mathbb{Z}_q$ persis seperti skema pembagian rahasia Shamir. Artinya, polinomial p berderajat seragam-$(t — 1)$ dengan $p(0) = x$ dipilih, dan otoritas ke-i diberikan $x_i := p(i)$. (Kami mengasumsikan dealer tepercaya yang mengetahui x dan menghapusnya dengan aman setelah dibagikan. Dealer dapat dihilangkan seluruhnya, tetapi hal ini berada di luar cakupan kami saat ini.)

Sekarang, katakanlah t beberapa otoritas $i_1, \ldots, i_t$ ingin bersama-sama mendeskripsi ciphertext $\langle c_1, c_2 \rangle$. Untuk melakukannya, otoritas i_j terlebih dahulu menerbitkan nilai $w_j := c_1^{x_{i_j}}$. Ingat dari bagian sebelumnya bahwa terdapat polinomial yang dapat dihitung secara publik $\{\delta_j(X)\}$ (yang bergantung pada identitas t otoritas ini) sehingga $p(X) \stackrel{\text{def}}{=} \sum_{j=1}^{t} \delta_j(X) \cdot x_{i_j}$. Dengan menetapkan $\delta_j \stackrel{\text{def}}{=} \delta_j(0)$, kita melihat bahwa terdapat nilai yang dapat dihitung secara publik $\delta_1, \ldots, \delta_t \in \mathbb{Z}_q$ yang $x = p(0) = \sum_{j=1}^{t} \delta_j \cdot x_{i_j}$. Otoritas mana pun kemudian dapat menghitung

$$M' := \frac{c_2}{\prod_{j=1}^{t} w_j^{\delta_j}}.$$

(Mereka kemudian dapat menghitung $\log_g M$ masing-masing, jika diinginkan.) Untuk memastikan bahwa tindakan ini memulihkan pesan dengan benar, ucapkan $c_1 = g^y$ dan $c_2 = h^y \cdot M$. Kemudian

$$\prod_{j=1}^{t} w_j^{\delta_j} = \prod_{j=1}^{t} c_1^{x_{i_j}\delta_j} = c_1^{\sum_{j=1}^{t} x_{i_j}\delta_j} = c_1^{p(0)} = c_1^x,$$

Dan sebagainya

$$M' \stackrel{\text{def}}{=} \frac{c_2}{\prod_{j=1}^{t} w_j^{\delta_j}} = \frac{h^y \cdot M}{c_1^x} = \frac{(g^x)^y \cdot M}{(g^y)^x} = M.$$

Perhatikan bahwa kumpulan $t - 1$ otoritas yang rusak tidak mempelajari apa pun tentang kunci privat x dari bagiannya. Selain itu, dapat ditunjukkan bahwa mereka tidak mempelajari apa pun dari proses dekripsi selain nilai yang diperoleh kembali M.

Musuh yang berbahaya (aktif). Perlakuan kami di atas mengasumsikan bahwa pihak berwenang yang mendekripsi beberapa teks tersandi semuanya berperilaku benar. (Jika tidak, akan mudah bagi salah satu dari mereka untuk menyebabkan hasil yang salah dengan menerbitkan nilai sewenang-wenang w_j.) Kami juga berasumsi bahwa pemilih berperilaku

jujur, dan mengenkripsi suara baik 0 atau 1. (Perhatikan bahwa seorang pemilih dapat mempengaruhi pemilu secara tidak adil dengan mengenkripsi nilai yang besar atau nilai negatif.) Potensi perilaku berbahaya semacam ini dapat dicegah dengan menggunakan teknik di luar cakupan buku ini.

13.4 SKEMA ENKRIPSI GOLDWASSER – MICALI

Sebelum kami menyajikan skema enkripsi Goldwasser – Micali, kita perlu mengembangkan pemahaman yang lebih baik tentang residu kuadrat. Pertama-tama kita mengeksplorasi kasus yang lebih mudah dari residu kuadrat modulo a prime p, dan kemudian melihat kasus yang sedikit lebih rumit dari residu kuadrat modulo a komposit N.

Sepanjang bagian ini, p dan q menunjukkan bilangan prima ganjil, dan $N = pq$ menunjukkan hasil kali dua bilangan prima ganjil yang berbeda.

Residu Kuadrat Modulo a Prime

Dalam golongan $\mathbb{G}$, suatu unsur $y \in \mathbb{G}$ merupakan residu kuadrat jika terdapat $x \in \mathbb{G}$ dengan $x^2 = y$. Dalam hal ini, kita menyebut x sebagai akar kuadrat dari y. Suatu unsur yang bukan merupakan residu kuadrat disebut non-residu kuadrat. Dalam grup abelian, himpunan residu kuadrat membentuk subgrup.

Dalam kasus khusus $\mathbb{Z}_p^*$, kita mendapatkan bahwa y adalah residu kuadrat jika terdapat x dengan $x^2 = y \bmod p$. Kita mulai dengan pengamatan yang mudah.

PROPOSISI 13.16 Misalkan $p > 2$ bilangan prima. Setiap residu kuadrat pada $\mathbb{Z}_p^*$ mempunyai tepat dua akar kuadrat.

BUKTI Misalkan $y \in \mathbb{Z}_p^*$ adalah residu kuadrat. Lalu ada $x \in \mathbb{Z}_p^*$ sedemikian rupa sehingga $x^2 = y \bmod p$. Jelasnya, $(-x)^2 = x^2 = y \bmod p$. Selanjutnya $-x \neq x \bmod p$: jika $-x = x \bmod p$ maka $2x = 0 \bmod p$, yang berarti $p|2x$. Karena p adalah bilangan prima, ini berarti $p|2$ (yang tidak mungkin karena $p > 2$) atau $p|x$ (yang tidak mungkin karena $0 < x < p$). Jadi, $[x \bmod p]$ dan $[-x \bmod p]$ adalah elemen berbeda dari $\mathbb{Z}*_p$, dan y memiliki setidaknya dua akar kuadrat.

Misalkan $x' \in \mathbb{Z}_p^*$ adalah akar kuadrat dari y. Maka $x^2 = y = (x')^2 \bmod p$, menyiratkan bahwa $x^2 - (x')^2 = 0 \bmod p$. Memfaktorkan ruas kiri kita peroleh

$$(x - x')(x + x') = 0 \bmod p\,,$$

sehingga (menurut Proposisi 8.3) baik $p|(x - x')$ atau $p|(x + x')$. Dalam kasus pertama, $x' = x \bmod p$ dan dalam kasus kedua $x' = -x \bmod p$, menunjukkan bahwa y memang hanya memiliki $[\pm x \bmod p]$ sebagai akar kuadrat.

Misalkan $\mathrm{sq}_p : \mathbb{Z}_p^* \to \mathbb{Z}_p^*$ adalah fungsi $\mathrm{sq}_p(x) \stackrel{\text{def}}{=} [x^2 \bmod p]$. Gambar di atas menunjukkan bahwa sq_p merupakan fungsi dua-satu jika $p > 2$ adalah bilangan prima. Hal ini

secara langsung menyiratkan bahwa separuh elemen $\mathbb{Z}_p^*$ adalah residu kuadrat. Himpunan residu kuadrat modulo p dinotasikan dengan $\mathcal{QR}_p$, dan himpunan non-residu kuadrat dengan $\mathcal{QNR}_p$. Kita baru saja melihat bahwa untuk $p > 2$ bilangan prima

$$|\mathcal{QR}_p| = |\mathcal{QNR}_p| = \frac{|\mathbb{Z}_p^*|}{2} = \frac{p-1}{2}.$$

Definisikan $\mathcal{J}_p(x)$, simbol Jacobi dari x modulo p, sebagai berikut.[4] Misalkan $p > 2$ bilangan prima, dan $x \in \mathbb{Z} *_p$. Kemudian

$$\mathcal{J}_p(x) \stackrel{\text{def}}{=} \begin{cases} +1 \text{ if } x \text{ is a quadratic residue modulo } p \\ -1 \text{ if } x \text{ is } \textit{not} \text{ a quadratic residue modulo } p. \end{cases}$$

Notasi ini dapat diperluas secara alami untuk sembarang x yang relatif prima ke p dengan menyetel $\mathcal{J}_p(x) \stackrel{\text{def}}{=} \mathcal{J}_p([x \bmod p])$.

Bisakah kita mengkarakterisasi residu kuadrat di $\mathbb{Z}_p^*$? Kita mulai dengan fakta bahwa $\mathbb{Z}_p^*$ adalah grup siklik berorde $p-1$ (lihat Teorema 8.56). Misalkan g adalah generator dari $\mathbb{Z}_p^*$. Artinya

$$\mathbb{Z}_p^* = \{g^0, g^1, g^2, \ldots, g^{\frac{p-1}{2}-1}, g^{\frac{p-1}{2}}, g^{\frac{p-1}{2}+1}, \ldots, g^{p-2}\}$$

(ingat bahwa p ganjil, jadi $p-1$ genap). Mengkuadratkan setiap elemen dalam daftar ini dan mengurangi modulo $p-1$ dalam eksponen (lih. Akibat wajar 8.15) menghasilkan daftar semua residu kuadrat dalam $\mathbb{Z}_p^*$:

$$\mathcal{QR}_p = \{g^0, g^2, g^4, \ldots, g^{p-3}, g^0, g^2, \ldots, g^{p-3}\}.$$

Setiap residu kuadrat muncul dua kali dalam daftar ini. Oleh karena itu, residu kuadrat pada $\mathbb{Z}_p^*$ adalah elemen yang dapat ditulis sebagai g^i dengan $i \in \{0, \ldots, p-2\}$ bilangan bulat genap.

Karakterisasi di atas mengarah pada cara sederhana untuk menghitung simbol Jacobi dan mengetahui apakah suatu elemen $x \in \mathbb{Z}_p^*$ merupakan residu kuadrat atau tidak.

PROPOSISI 13.17 Misalkan $p > 2$ bilangan prima. Maka $\mathcal{J}_p(x) = x^{\frac{p-1}{2}} \bmod p$.

BUKTI Misalkan g adalah generator sembarang dari $\mathbb{Z} *_p$. Jika x adalah modulo residu kuadrat p, pembahasan kita sebelumnya menunjukkan bahwa $x = g^i$ untuk bilangan bulat genap i. Menulis $i = 2j$ dengan j bilangan bulat yang kemudian kita miliki

$$x^{\frac{p-1}{2}} = \left(g^{2j}\right)^{\frac{p-1}{2}} = g^{(p-1)j} = \left(g^{p-1}\right)^{j} = 1^{j} = 1 \bmod p,$$

dan $x^{\frac{p-1}{2}} = +1 = \mathcal{J}_p(x) \bmod p$ seperti yang diklaim.
Sebaliknya, jika x bukan residu kuadrat maka $x = g^i$ untuk bilangan bulat ganjil i. Penulisan $i = 2j + 1$ dengan j merupakan bilangan bulat yang kita miliki

$$x^{\frac{p-1}{2}} = \left(g^{2j+1}\right)^{\frac{p-1}{2}} = \left(g^{2j}\right)^{\frac{p-1}{2}} \cdot g^{\frac{p-1}{2}} = 1 \cdot g^{\frac{p-1}{2}} = g^{\frac{p-1}{2}} \bmod p.$$

dan sekarang,

$$\left(g^{\frac{p-1}{2}}\right)^2 = g^{p-1} = 1 \bmod p,$$

dan jadi $g^{\frac{p-1}{2}} = \pm 1 \bmod p$ karena $[\pm 1 \ mod \ p]$ adalah dua akar kuadrat dari 1 (lih. Proposisi 13.16). Karena g adalah generator, ia mempunyai orde $p - 1$ sehingga $g^{\frac{p-1}{2}} \neq \pm 1 \bmod p$. Oleh karena itu $x^{\frac{p-1}{2}} = -1 = \mathcal{J}_p(x) \bmod p$.

Proposisi 13.17 secara langsung memberikan algoritma waktu polinomial (lih. Algoritma 13.18) untuk menguji apakah suatu elemen $x \in \mathbb{Z} *_p$ merupakan residu kuadrat.

ALGORITHM 13.18
Deciding quadratic residuosity modulo a prime

Input: A prime p; an element $x \in \mathbb{Z}_p^*$
Output: $\mathcal{J}_p(x)$ (or, equivalently, whether x is a quadratic residue or quadratic non-residue)

$b := \left[x^{\frac{p-1}{2}} \bmod p\right]$
if $b = 1$ **return** "quadratic residue"
else return "quadratic non-residue"

Kami menyimpulkan bagian ini dengan mencatat sifat perkalian yang bagus dari residu kuadrat dan non-residu modulo p.

PROPOSISI 13.19 Misalkan $p > 2$ bilangan prima, dan $x, y \in \mathbb{Z} *_p$. Kemudian

$$\mathcal{J}_p(xy) = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y).$$

BUKTI Dengan menggunakan proposisi sebelumnya,

$$\mathcal{J}_p(xy) = (xy)^{\frac{p-1}{2}} = x^{\frac{p-1}{2}} \cdot y^{\frac{p-1}{2}} = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) \bmod p.$$

Sejak $\mathcal{J}_p(xy), \mathcal{J}_p(x), \mathcal{J}_p(y) = \pm 1$, kesetaraan juga berlaku pada bilangan bulat.

KORELARI 13.20 Misalkan $p > 2$ bilangan prima, dan misalkan $x, x' \in \mathcal{QR}_p$ dan $y, y' \in \mathcal{QNR}_p$. Kemudian:

1. $[xx' \bmod p] \in \mathcal{QR}_p$.
2. $[yy' \bmod p] \in \mathcal{QR}_p$.
3. $[xy \bmod p] \in \mathcal{QNR}_p$.

Residu Kuadrat Modulo a Komposit

Sekarang kita mengalihkan perhatian kita ke residu kuadrat pada grup $\mathbb{Z}*_N$, dimana $N = pq$. Mengkarakterisasi residu kuadrat modulo N mudah dilakukan jika kita menggunakan hasil bagian sebelumnya bersama dengan teorema sisa Cina. Ingatlah bahwa teorema sisa Cina mengatakan bahwa $\mathbb{Z}*_N \simeq \mathbb{Z}*_p \times \mathbb{Z}*_q$, dan kita misalkan $y \leftrightarrow (y_p, y_q)$ menunjukkan korespondensi yang dijamin oleh teorema tersebut (yaitu, $y_p = [y \bmod p]$ dan $y_q = [y \bmod q]$). Pengamatan utamanya adalah:

PROPOSISI 13.21 Misalkan $N = pq$ dengan p, q bilangan prima berbeda, dan $y \in \mathbb{Z}_N^*$ dengan $y \leftrightarrow (y_p, y_q)$. Maka y adalah modulo residu kuadratik N jika dan hanya jika y_p adalah modulo residu kuadratik p dan y_q adalah modulo residu kuadratik q.

BUKTI Jika y adalah modulo residu kuadrat N maka, menurut definisi, terdapat $x \in \mathbb{Z}_N^*$ sehingga $x^2 = y \bmod N$. Misal $x \leftrightarrow (x_p, x_q)$. Kemudian

$$(y_p, y_q) \leftrightarrow y = x^2 \leftrightarrow (x_p, x_q)^2 = ([x_p^2 \bmod p], [x_q^2 \bmod q]),$$

dimana $(x_p, x_q)^2$ hanyalah kuadrat dari elemen (x_p, x_q) dalam grup $\mathbb{Z}_p^* \times \mathbb{Z}_q^*$. Kami telah menunjukkan hal itu

$$y_p = x_p^2 \bmod p \quad \text{and} \quad y_q = x_q^2 \bmod q \qquad (13.4)$$

dan y_p, y_q adalah residu kuadrat (sehubungan dengan modulus yang sesuai).

Sebaliknya, jika $y \leftrightarrow (y_p, y_q)$ dan y_p, y_q masing-masing merupakan residu kuadrat modulo p dan q, maka terdapat $x_p \in \mathbb{Z}_p^*$ dan $x_q \in \mathbb{Z}_q^*$ sehingga Persamaan (13.4) berlaku. Misalkan $x \in$

$\mathbb{Z}_N^*$ sedemikian hingga $x \leftrightarrow (x_p, x_q)$. Membalikkan langkah di atas menunjukkan bahwa x adalah akar kuadrat dari y modulo N.

Proposisi di atas mencirikan residu kuadrat modulo N. Pemeriksaan bukti yang cermat menghasilkan pengamatan penting lainnya: setiap residu kuadrat $y \in \mathbb{Z}_N^*$ memiliki tepat empat akar kuadrat. Untuk melihatnya, misalkan $y \leftrightarrow (y_p, y_q)$ adalah residu kuadrat modulo N dan misalkan x_p, x_q masing-masing adalah akar kuadrat dari y_p dan y_q modulo p dan q. Kemudian empat akar kuadrat dari y diberikan oleh elemen-elemen di $\mathbb{Z}_N^*$ yang bersesuaian dengan

$$(x_p, x_q), \quad (-x_p, x_q), \quad (x_p, -x_q), \quad (-x_p, -x_q). \qquad (13.5)$$

Masing-masing adalah akar kuadrat dari y sejak itu

$$\begin{aligned}(\pm x_p, \pm x_q)^2 &= \Big([(\pm x_p)^2 \bmod p], [(\pm x_q)^2 \bmod q]\Big) \\ &= ([x_p^2 \bmod p], [x_q^2 \bmod q]) = (y_p, y_q) \leftrightarrow y\end{aligned}$$

(dimana notasi $(\cdot,\cdot)^2$ mengacu pada pengkuadratan pada grup $\mathbb{Z}_p \times \mathbb{Z}_q$).
Teorema sisa Cina menjamin bahwa empat elemen dalam Persamaan (13.5) berhubungan dengan elemen berbeda dari $\mathbb{Z}_N^*$, karena x_p dan $-x_p$ adalah modulo p yang unik (dan demikian pula untuk x_q dan $-x_q$ modulo q).

Contoh 13.22

Pertimbangkan $\mathbb{Z}_{15}^*$ (korespondensi yang diberikan oleh teorema sisa Cina ditabulasikan dalam Contoh 8.25). Elemen 4 adalah residu kuadrat modulo 15 dengan akar kuadrat 2. Karena $2 \leftrightarrow (2,2)$, akar kuadrat lainnya dari 4 diberikan oleh

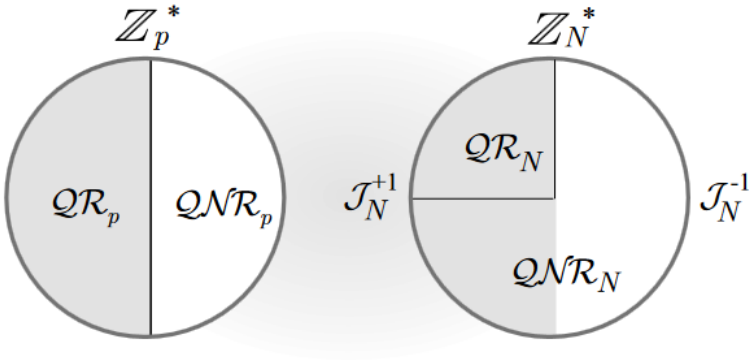
- $(2, [-2 \bmod 3]) = (2, 1) \leftrightarrow 7$;
- $([-2 \bmod 5], 2) = (3, 2) \leftrightarrow 8$; and
- $([-2 \bmod 5], [-2 \bmod 3]) = (3, 1) \leftrightarrow 13$.

Dapat dipastikan bahwa $7^2 = 8^2 = 13^2 = 4 \bmod 15$. ♦

Misalkan $\mathcal{QR}_N$ menyatakan himpunan residu kuadrat modulo N. Karena mengkuadratkan modulo N adalah fungsi empat banding satu, kita melihat bahwa tepat 1/4 elemen $\mathbb{Z}_N^*$ merupakan residu kuadrat. Sebagai alternatif, kita dapat mencatat bahwa karena $y \in \mathbb{Z}_N^*$ merupakan residu kuadrat jika dan hanya jika y_p, y_q adalah residu kuadrat, maka terdapat korespondensi satu-satu antara $\mathcal{QR}_N$ dan $\mathcal{QR}_p \times \mathcal{QR}_q$. Jadi, fraksi residu kuadrat modulo N adalah

$$\frac{|\mathcal{QR}_N|}{|\mathbb{Z}_N^*|} = \frac{|\mathcal{QR}_p| \cdot |\mathcal{QR}_q|}{|\mathbb{Z}_N^*|} = \frac{\frac{p-1}{2} \cdot \frac{q-1}{2}}{(p-1)(q-1)} = \frac{1}{4},$$

setuju dengan hal di atas.



GAMBAR 13.1: Struktur $\mathbb{Z}*_p$ dan $\mathbb{Z}*_N$.

Pada bagian sebelumnya, kita mendefinisikan simbol Jacobi $\mathcal{J}_p(x)$ untuk $p > 2$ bilangan prima. Kami memperluas definisi tersebut ke kasus N hasil kali bilangan prima ganjil p dan q yang berbeda sebagai berikut. Untuk setiap x yang relatif prima terhadap $N = pq$,

$$\begin{aligned}\mathcal{J}_N(x) &\stackrel{\text{def}}{=} \mathcal{J}_p(x) \cdot \mathcal{J}_q(x) \\ &= \mathcal{J}_p([x \bmod p]) \cdot \mathcal{J}_q([x \bmod q]).\end{aligned}$$

Kita mendefinisikan $\mathcal{J}_N^{+1}$ sebagai himpunan elemen dalam $\mathbb{Z}*_N$ yang memiliki simbol Jacobi $+1$, dan mendefinisikan $\mathcal{J}_N^{-1}$ secara analog.

Kita mengetahui dari Proposisi 13.21 bahwa jika x adalah modulo residu kuadrat N, maka $[x \bmod p]$ dan $[x \bmod q]$ masing-masing adalah modulo residu kuadrat p dan q; yaitu $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$. Jadi $\mathcal{J}_N(x) = +1$ dan kita melihat bahwa:

Jika x adalah modulo residu kuadrat N, maka $\mathcal{J}_N(x) = +1$.

Namun, $\mathcal{J}_N(x) = +1$ juga dapat terjadi ketika $\mathcal{J}_p(x) = \mathcal{J}_q(x) = -1$, yaitu ketika $[x \bmod p]$ dan $[x \bmod q]$ bukan residu kuadrat modulo p dan q (dan x bukan modulo residu kuadrat N). Hal ini ternyata berguna untuk skema enkripsi Goldwasser – Micali, dan oleh karena itu kami memperkenalkan notasi $\mathcal{QNR}_N^{+1}$ untuk himpunan elemen tipe ini. Itu adalah,

$$\mathcal{QNR}_N^{+1} \stackrel{\text{def}}{=} \left\{ x \in \mathbb{Z}_N^* \;\middle|\; \begin{array}{c} x \text{ is not a quadratic residue modulo } N, \\ \text{but } \mathcal{J}_N(x) = +1 \end{array} \right\}.$$

Sekarang mudah untuk membuktikan hal berikut (lihat Gambar 13.1):

PROPOSISI 13.23 Misalkan $N = pq$ dengan p, q bilangan prima ganjil yang berbeda. Maka:

1. Tepat separuh unsur $\mathbb{Z}*_N$ berada di $\mathcal{J}_N^{+1}$.

2. QRN terdapat pada $\mathcal{J}_N^{+1}$.
3. Tepat separuh elemen $\mathcal{J}_N^{+1}$ berada di $\mathcal{QR}_N$ (separuh lainnya berada di $\mathcal{QNR}_N^{+1}$).

BUKTI Kita mengetahui bahwa $\mathcal{J}_N(x) = +1$ jika $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$ atau $\mathcal{J}_p(x) = \mathcal{J}_q(x) = -1$. Kita juga mengetahui (dari bagian sebelumnya) bahwa separuh elemen $\mathbb{Z}_p^*$ memiliki simbol Jacobi $+1$, dan separuhnya lagi memiliki simbol Jacobi -1 (dan demikian pula untuk $\mathbb{Z}_p^*$). Mendefinisikan $\mathcal{J}_p^{+1}, \mathcal{J}_p^{-1}, \mathcal{J}_q^{+1}$, dan $\mathcal{J}_q^{-1}$ dengan cara alami, kita mendapatkan

$$\begin{aligned}\left|\mathcal{J}_N^{+1}\right| &= \left|\mathcal{J}_p^{+1} \times \mathcal{J}_q^{+1}\right| + \left|\mathcal{J}_p^{-1} \times \mathcal{J}_q^{-1}\right| \\ &= \left|\mathcal{J}_p^{+1}\right| \cdot \left|\mathcal{J}_q^{+1}\right| + \left|\mathcal{J}_p^{-1}\right| \cdot \left|\mathcal{J}_q^{-1}\right| \\ &= \frac{(p-1)}{2}\frac{(q-1)}{2} + \frac{(p-1)}{2}\frac{(q-1)}{2} = \frac{\phi(N)}{2}.\end{aligned}$$

Jadi $|\mathcal{J}_N^{+1}| = |\mathbb{Z} *_N |/2$, membuktikan bahwa separuh elemen $\mathbb{Z} *_N$ ada di $\mathcal{J}_N^{+1}$.
Kita telah mencatat sebelumnya bahwa semua residu kuadrat modulo N mempunyai simbol Jacobi $+1$, menunjukkan bahwa $\mathcal{QR}_N \subseteq \mathcal{J}_N^{+1}$.
Karena $x \in \mathcal{QR}_N$ jika dan hanya jika $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$, kita peroleh

$$\left|\mathcal{QR}_N\right| = \left|\mathcal{J}_p^{+1} \times \mathcal{J}_q^{+1}\right| = \frac{(p-1)}{2}\frac{(q-1)}{2} = \frac{\phi(N)}{4},$$

dan jadi $|\mathcal{QR}_N| = |\mathcal{J}_N^{+1}|/2$. Karena $\mathcal{QR}_N$ merupakan himpunan bagian dari $\mathcal{J}_N^{+1}$, hal ini membuktikan bahwa separuh elemen $\mathcal{J}_N^{+1}$ berada dalam $\mathcal{QR}_N$. Dua hasil berikutnya adalah analog dari Proposisi 13.19 dan Korelari 13.20.

PROPOSISI 13.24 Misalkan $N = pq$ merupakan hasil kali bilangan prima ganjil yang berbeda, dan $x, y \in \mathbb{Z} *_N$. Maka $\mathcal{J}_N(xy) = \mathcal{J}_N(x) \cdot \mathcal{J}_N(y)$.

BUKTI Menggunakan definisi $\mathcal{J}_N(\cdot)$ dan Proposisi 13.19:

$$\begin{aligned}\mathcal{J}_N(xy) = \mathcal{J}_p(xy) \cdot \mathcal{J}_q(xy) &= \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) \cdot \mathcal{J}_q(x) \cdot \mathcal{J}_q(y) \\ &= \mathcal{J}_p(x) \cdot \mathcal{J}_q(x) \cdot \mathcal{J}_p(y) \cdot \mathcal{J}_q(y) = \mathcal{J}_N(x) \cdot \mathcal{J}_N(y).\end{aligned}$$

KORELARI 13.25 Misalkan $N = pq$ adalah hasil kali bilangan prima ganjil yang berlainan, dan katakanlah $x, x' \in \mathcal{QR}_N$ dan $y, y' \in \mathcal{QNR}_N^{+1}$. Kemudian:

1. $[xx' \bmod N] \in \mathcal{QR}_N$.
2. $[yy' \bmod N] \in \mathcal{QR}_N$.
3. $[xy \bmod N] \in \mathcal{QNR}_N^{+1}$.

BUKTI Kami membuktikan klaim akhir; bukti yang lain serupa. Karena $x \in \mathcal{QR}_N$, kita mempunyai $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$. Karena $y \in \mathcal{QNR}_N^{+1}$, kita mempunyai $\mathcal{J}_p(y) = \mathcal{J}_q(y) = -1$. Menggunakan Proposisi 13.19,

$$\mathcal{J}_p(xy) = \mathcal{J}_p(x) \cdot \mathcal{J}_p(y) = -1 \quad \text{and} \quad \mathcal{J}_q(xy) = \mathcal{J}_q(x) \cdot \mathcal{J}_q(y) = -1,$$

jadi $\mathcal{J}_N(xy) = +1$. Namun xy bukan modulo residu kuadrat N, karena $\mathcal{J}_N(xy) = -1$ sehingga $[xy \bmod p]$ bukan modulo residu kuadrat p. Kita simpulkan bahwa $xy \in \mathcal{QNR}_N^{+1}$.

Berbeda dengan Korelari 13.20, tidak benar bahwa $y, y' \in \mathcal{QNR}_N$ menyiratkan $yy' \in \mathcal{QR}_N$. (Sebaliknya, seperti yang ditunjukkan dalam akibat wajar, hal ini hanya dijamin jika $y, y' \in \mathcal{QNR}_N^{+1}$.) Misalnya, kita dapat memiliki $\mathcal{J}_p(y) = +1, \mathcal{J}_q(y) = -1$ dan $\mathcal{J}_p(y') = -1, \mathcal{J}_q(y') = +1$, jadi $\mathcal{J}_p(yy') = \mathcal{J}_q(yy') = -1$ dan yy' bukan residu kuadrat meskipun $\mathcal{J}_N(yy') = +1$.

Asumsi Residuositas Kuadrat

Pada Bagian ini, kami menunjukkan algoritma yang efisien untuk memutuskan apakah input x merupakan residu kuadrat modulo a prime p. Bisakah kita mengadaptasi algoritma untuk bekerja modulo bilangan komposit N? Proposisi 13.21 memberikan solusi yang mudah untuk masalah ini asalkan faktorisasi N diketahui. Lihat Algoritma 13.26.

ALGORITHM 13.26
Deciding quadratic residuosity modulo a composite of known factorization

Input: Composite $N = pq$; the factors p and q; element $x \in \mathbb{Z}_N^*$
Output: A decision as to whether $x \in \mathcal{QR}_N$

compute $\mathcal{J}_p(x)$ and $\mathcal{J}_q(x)$
if $\mathcal{J}_p(x) = \mathcal{J}_q(x) = +1$ **return** "quadratic residue"
else return "quadratic non-residue"

(Seperti biasa, kita mengasumsikan faktor-faktor dari N adalah bilangan prima ganjil yang berbeda.) Modifikasi sederhana dari algoritma di atas memungkinkan komputasi $\mathcal{J}_N(x)$ ketika faktorisasi N diketahui.

Namun, jika faktorisasi N tidak diketahui, tidak ada algoritma waktu polinomial yang diketahui untuk memutuskan apakah suatu x merupakan residu kuadrat modulo N atau tidak. Agak mengejutkan, algoritma waktu polinomial dikenal untuk menghitung $\mathcal{J}_N(x)$ tanpa

faktorisasi N. (Meskipun algoritmenya sendiri tidak terlalu rumit, bukti kebenarannya berada di luar cakupan buku ini dan oleh karena itu kami tidak menyajikan algoritme tersebut sama sekali. Pembaca yang berminat dapat merujuk pada referensi yang tercantum di bagian akhir buku ini. bab.) Hal ini mengarah pada pengujian parsial residuositas kuadrat: jika, untuk masukan tertentu x, dinyatakan bahwa $\mathcal{J}_N(x) = -1$, maka x tidak mungkin merupakan residu kuadrat. (Lihat Proposisi 13.23.) Tes ini tidak mengatakan apa-apa ketika $\mathcal{J}_N(x) = +1$, dan tidak ada algoritma waktu polinomial yang diketahui untuk menentukan residuositas kuadrat dalam kasus tersebut (yang lebih baik daripada tebakan acak).

Kami sekarang meresmikan asumsi bahwa masalah ini sulit. Misalkan GenModulus adalah algoritma waktu polinomial yang, pada masukan 1^n, menghasilkan keluaran (N, p, q) dengan $N = pq$, dan p dan q adalah n-bit bilangan prima kecuali dengan probabilitas yang dapat diabaikan dalam n.

DEFINISI 13.27 Kita mengatakan menentukan residuositas kuadrat relatif sulit terhadap GenModulus jika untuk semua algoritme waktu polinomial probabilistik D terdapat fungsi yang dapat diabaikan sedemikian rupa negl sehingga

$$\left|\Pr[D(N, \mathsf{qr}) = 1] - \Pr[D(N, \mathsf{qnr}) = 1]\right| \leq \mathsf{negl}(n),$$

di mana dalam setiap kasus probabilitas diambil alih eksperimen di mana GenModulus(1^n) dijalankan untuk menghasilkan (N, p, q), qr dipilih secara seragam dari $\mathcal{QR}_N$, dan qnr dipilih secara seragam dari $\mathcal{QNR}_N^{+1}$.

Sangat penting dalam hal di atas bahwa qnr dipilih dari $\mathcal{QNR}_N^{+1}$ daripada $\mathcal{QNR}_N$; jika qnr dipilih dari $\mathcal{QNR}_N$ maka dengan probabilitas 2/3 $\mathcal{J}_N(x) = -1$ sehingga membedakan qnr dari residu kuadrat seragam akan mudah. (Ingat bahwa $\mathcal{J}_N(x)$ dapat dihitung secara efisien bahkan tanpa faktorisasi N.)

Asumsi residuositas kuadrat hanyalah asumsi bahwa terdapat GenModulus yang relatif sulit menentukan residuositas kuadrat. Sangat mudah untuk melihat bahwa jika menentukan residuositas kuadrat relatif sulit dibandingkan GenModulus, maka pemfaktoran juga harus sulit dibandingkan GenModulus.

Skema Enkripsi Goldwasser – Micali

Bagian sebelumnya segera menyarankan skema enkripsi kunci publik untuk pesan bit tunggal berdasarkan asumsi residuositas kuadrat:

- Kunci publik adalah modulus N, dan kunci privat adalah faktorisasinya.
- Untuk mengenkripsi '0', kirimkan residu kuadrat yang seragam; untuk mengenkripsi '1', kirimkan non-residu kuadrat seragam dengan simbol Jacobi +1.
- Penerima dapat mendekripsi ciphertext c dengan kunci privatnya dengan menggunakan faktorisasi N untuk memutuskan apakah c merupakan residu kuadrat atau tidak.

Keamanan CPA pada skema ini hampir tidak penting karena sulitnya masalah residuositas kuadrat seperti yang diformalkan dalam Definisi 13.27.

Satu hal yang hilang dari uraian di atas adalah spesifikasi bagaimana pengirim, yang tidak mengetahui faktorisasi N, dapat memilih elemen seragam $\mathcal{QR}_N$ (untuk mengenkripsi a = 0) atau elemen seragam $\mathcal{QNR}_N^{+1}$ (untuk mengenkripsi a = 1). Yang pertama mudah, sedangkan yang kedua membutuhkan kecerdikan.

Memilih residu kuadrat seragam. Memilih elemen seragam $y \in \mathcal{QR}_N$ sangatlah mudah: cukup pilih elemen seragam $x \in \mathbb{Z}*_N$ (lihat Lampiran B.2.5) dan atur $y := x^2 \bmod N$. Jelasnya $y \in \mathcal{QR}_N$. Fakta bahwa y terdistribusi secara merata di $\mathcal{QR}_N$ mengikuti fakta bahwa mengkuadratkan modulo N adalah fungsi 4 banding 1 (lihat Bagian 13.4.2) dan bahwa x dipilih secara seragam dari $\mathbb{Z}*_N$. Secara lebih rinci, perbaiki $\hat{y} \in \mathcal{QR}_N$ dan mari kita hitung probabilitas bahwa $y = \hat{y}$ setelah prosedur di atas. Nyatakan empat akar kuadrat dari $\hat{y}$ dengan $\pm\hat{x}, \pm\hat{x}'$. Kemudian:

$$\begin{aligned}\Pr[y = \hat{y}] &= \Pr[x \text{ is a square root of } \hat{y}] \\ &= \Pr[x \in \{\pm\hat{x}, \pm\hat{x}'\}] \\ &= \frac{4}{|\mathbb{Z}_N^*|} = \frac{1}{|\mathcal{QR}_N|}.\end{aligned}$$

Karena persamaan di atas berlaku untuk setiap $\hat{y} \in \mathcal{QR}_N$, kita melihat bahwa y terdistribusi secara merata di $\mathcal{QR}_N$.

Memilih elemen seragam $\mathcal{QNR}_N^{+1}$. Secara umum tidak diketahui cara memilih unsur seragam $\mathcal{QNR}_N^{+1}$ jika faktorisasi N tidak diketahui.

Apa yang menyelamatkan kita dalam konteks saat ini adalah bahwa penerima dapat membantu dengan memasukkan informasi tertentu ke dalam kunci publik. Secara khusus, kami memodifikasi skema sehingga penerima juga memilih seragam $z \in \mathcal{QNR}_N^{+1}$ dan menyertakan z sebagai bagian dari kunci publiknya. (Hal ini mudah dilakukan oleh penerima karena ia mengetahui faktorisasi N; Pengirim dapat memilih elemen seragam $y \in \mathcal{QNR}_N^{+1}$ dengan memilih elemen seragam $x \in \mathbb{Z}*_N$ (seperti di atas) dan mengaturnya $y := [z \cdot x^2 \bmod N]$. Berdasarkan Akibat wajar 13.25 bahwa $y \in \mathcal{QNR}_N^{+1}$. Kita membiarkannya sebagai latihan untuk menunjukkan bahwa y terdistribusi secara merata di $\mathcal{QNR}_N^{+1}$; kami tidak menggunakan fakta ini secara langsung dalam bukti keamanan yang diberikan di bawah ini.

Kami memberikan penjelasan lengkap tentang skema enkripsi Goldwasser – Micali, yang menerapkan ide di atas, dalam Konstruksi 13.28.

TEOREMA 13.29 Jika masalah residuositas kuadratik relatif sulit terhadap GenModulus, maka skema enkripsi Goldwasser–Micali aman terhadap CPA.

BUKTI Misalkan Π menunjukkan skema enkripsi Goldwasser – Micali. Kami membuktikan bahwa Π memiliki enkripsi yang tidak dapat dibedakan dengan adanya penyadap; berdasarkan Teorema 11.6 ini menyiratkan bahwa ini aman terhadap CPA.

Misalkan $\mathcal{A}$ adalah musuh waktu polinomial probabilistik sembarang. Pertimbangkan musuh PPT D berikut yang mencoba menyelesaikan masalah residuositas kuadrat relatif terhadap GenModulus:

Algoritma D:
Algoritme ini diberikan N dan z sebagai masukan, dan tujuannya adalah untuk menentukan apakah $z \in \mathcal{QR}_N$ atau $z \in \mathcal{QNR}_N^{+1}$.

CONSTRUCTION 13.28

Let GenModulus be as usual. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n, run GenModulus(1^n) to obtain (N, p, q), and choose a uniform $z \in \mathcal{QNR}_N^{+1}$. The public key is $pk = \langle N, z \rangle$ and the private key is $sk = \langle p, q \rangle$.
- Enc: on input a public key $pk = \langle N, z \rangle$ and a message $m \in \{0, 1\}$, choose a uniform $x \in \mathbb{Z}_N^*$ and output the ciphertext

$$c := [z^m \cdot x^2 \bmod N].$$

- Dec: on input a private key sk and a ciphertext c, determine whether c is a quadratic residue modulo N using, e.g., Algorithm 13.26. If yes, output 0; otherwise, output 1.

Skema enkripsi Goldwasser – Micali.

- Atur $pk = \langle N, z \rangle$ dan jalankan $\mathcal{A}(pk)$ untuk mendapatkan dua pesan bit tunggal m_0, m_1.
- Pilih bit seragam b dan bit seragam $x \in \mathbb{Z}_N^*$, lalu atur $c := [z^{m_b} \cdot x^2 \bmod N]$.
- Berikan ciphertext c kepada $\mathcal{A}$, yang pada gilirannya mengeluarkan bit b'. Jika $b' = b$, keluaran 1; jika tidak, keluaran 0.

Mari kita analisa perilaku D. Ada dua kasus yang perlu dipertimbangkan:

Kasus 1: Katakanlah input ke D dihasilkan dengan menjalankan GenModulus(1^n) untuk mendapatkan (N, p, q) dan kemudian memilih seragam $z \in \mathcal{QNR}_N^{+1}$. Kemudian D menjalankan $\mathcal{A}$ pada kunci publik yang dibuat persis seperti pada Π, dan kita melihat bahwa dalam kasus ini tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh D didistribusikan secara identik dengan tampilan $\mathcal{A}$ dalam eksperimen $\mathrm{PubK}^{\mathrm{eav}}_{\mathcal{A},\Pi}(n)$. Karena D mengeluarkan 1 tepat ketika keluaran b' dari $\mathcal{A}$ sama dengan b, kita punya

$$\Pr[D(N, \mathsf{qnr}) = 1] = \Pr[\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1],$$

dimana qnr mewakili elemen seragam $\mathcal{QNR}_N^{+1}$seperti pada Definisi 13.27.

Kasus 2: Katakanlah input ke D dihasilkan dengan menjalankan GenModulus(1^n) untuk mendapatkan (N, p, q) dan kemudian memilih seragam $z \in QR_N$. Kami mengklaim bahwa tampilan $\mathcal{A}$ dalam hal ini tidak bergantung pada bit b. Untuk melihat hal ini, perhatikan bahwa ciphertext c yang diberikan kepada $\mathcal{A}$ adalah residu kuadrat seragam terlepas dari apakah 0 atau 1 dienkripsi:

- Ketika a 0 dienkripsi, $c = [x^2 \bmod N]$ untuk $x \in \mathbb{Z}_N^*$yang seragam, sehingga c adalah residu kuadrat yang seragam.
- Ketika a 1 dienkripsi, $c = [z \cdot x^2 \bmod N]$untuk seragam $x \in \mathbb{Z}_N^*$. Misalkan $\hat{x} \stackrel{\text{def}}{=} [x^2 \bmod N]$, dan perhatikan bahwa $\hat{x}$ adalah elemen grup QRN yang terdistribusi merata. Karena $z \in QR_N$, kita dapat menerapkan Lemma 11.15 untuk menyimpulkan bahwa c juga terdistribusi secara merata di QR_N.

Karena pandangan $\mathcal{A}'$ tidak bergantung pada b, peluang $b' = b$ dalam kasus ini adalah tepat 1/2. Itu adalah,

$$\Pr[D(N, \text{qr}) = 1] = \frac{1}{2},$$

dimana qr mewakili elemen seragam QRN seperti pada Definisi 13.27. Dengan demikian,

$$\left|\Pr[D(N, \text{qr}) = 1] - \Pr[D(N, \text{qnr}) = 1]\right| = \left|\Pr[\text{PubK}^{\text{eav}}_{\mathcal{A},\Pi}(n) = 1] - \tfrac{1}{2}\right|.$$

Dengan asumsi bahwa masalah residuositas kuadrat relatif sulit terhadap GenModulus, terdapat pengabaian fungsi yang dapat diabaikan sehingga

$$\left|\varepsilon(n) - \tfrac{1}{2}\right| \leq \text{negl}(n);$$

Dengan demikian,

$$\varepsilon(n) \leq \tfrac{1}{2} + \text{negl}(n).$$

Ini melengkapi buktinya.

13.5 SKEMA ENKRIPSI RABIN

Seperti disebutkan di awal bab ini, skema enkripsi Rabin menarik karena keamanannya setara dengan asumsi bahwa pemfaktoran itu sulit. Hasil analogi tidak diketahui untuk enkripsi berbasis RSA, dan masalah RSA berpotensi lebih mudah daripada memfaktorkan. (Hal yang sama berlaku untuk skema enkripsi Goldwasser – Micali, dan ada kemungkinan bahwa menentukan modulo residuositas kuadratik N lebih mudah daripada memfaktorkan N)

Menariknya, skema enkripsi Rabin (setidaknya secara dangkal) sangat mirip dengan skema enkripsi RSA namun memiliki keuntungan karena didasarkan pada asumsi yang berpotensi lebih lemah. Fakta bahwa RSA lebih banyak digunakan dibandingkan RSA

tampaknya lebih disebabkan oleh faktor historis dibandingkan faktor teknis; kita membahas ini lebih lanjut di akhir bagian ini.

Kita mulai dengan beberapa pendahuluan tentang menghitung akar kuadrat modular. Kami kemudian memperkenalkan permutasi pintu jebakan yang dapat didasarkan langsung pada asumsi bahwa pemfaktoran itu sulit. Skema enkripsi Rabin (atau, setidaknya, satu contohnya) kemudian diperoleh dengan menerapkan hasil dari Bagian 13.1. Sepanjang bagian ini, kita terus misalkan p dan q menyatakan bilangan prima ganjil, dan misalkan $N = pq$ menyatakan hasil kali dua bilangan prima ganjil yang berbeda.

Menghitung Akar Kuadrat Modular

Skema enkripsi Rabin mengharuskan penerima untuk menghitung akar kuadrat modular, sehingga pada bagian ini kita mengeksplorasi kompleksitas algoritmik dari masalah ini. Pertama-tama kami menunjukkan algoritme yang efisien untuk menghitung akar kuadrat modulo a prime p, dan kemudian memperluas algoritme ini untuk memungkinkan penghitungan akar kuadrat modulo a komposit N dari faktorisasi yang diketahui. Pembaca yang bersedia menerima keberadaan algoritme ini dapat melompat ke bagian berikut, di mana kami menunjukkan bahwa menghitung akar kuadrat modulo komposit N dengan faktorisasi yang tidak diketahui setara dengan memfaktorkan N.

Misalkan p adalah bilangan prima ganjil. Menghitung modulo akar kuadrat p relatif sederhana ketika $p = 3 \, mod \, 4$, tetapi lebih rumit ketika $p = 1 \, mod \, 4$. (Kasus yang lebih mudah adalah yang kita perlukan untuk skema enkripsi Rabin seperti yang disajikan di Bagian 13.5; kami menyertakan kasus kedua untuk kelengkapan.) Dalam kedua kasus, kami menunjukkan cara menghitung salah satu akar kuadrat dari residu kuadrat $a \in \mathbb{Z}_p^*$. Perhatikan bahwa jika x adalah salah satu akar kuadrat dari a, maka $[-x \, mod \, p \,]$ adalah akar lainnya.

Kami menangani kasus yang lebih mudah terlebih dahulu. Katakanlah $p = 3 \, mod \, 4$, artinya kita dapat menulis $p = 4i + 3$ untuk suatu bilangan bulat i. Karena $a \in \mathbb{Z}_p^*$ adalah residu kuadrat, kita mempunyai $\mathcal{J}_p(a) = 1 = a^{\frac{p-1}{2}} \bmod p$ (lihat Proposisi 13.17). Mengalikan kedua ruas dengan a kita peroleh

$$a = a^{\frac{p-1}{2}+1} = a^{2i+2} = \left(a^{i+1}\right)^2 \bmod p \,,$$

jadi $a^{i+1} = a^{\frac{p+1}{4}} \bmod p$ adalah akar kuadrat dari a. Artinya, kita memperoleh akar kuadrat dari modulo p hanya dengan menghitung $x := \left[a^{\frac{p+1}{2}} \bmod p\right]$.

Sangat penting bahwa $(p + 1)/2$ adalah genap karena hal ini memastikan bahwa $(p + 1)/4$ adalah bilangan bulat (ini diperlukan agar $a^{\frac{p+1}{4}} \bmod p$ terdefinisi dengan baik; ingat bahwa eksponennya harus berupa bilangan bulat). Pendekatan ini tidak berhasil ketika $p = 1 \, mod \, 4$, dalam hal ini $p + 1$ adalah bilangan bulat yang tidak habis dibagi 4.

Ketika $p = 1 \, mod \, 4$, kita melakukannya dengan sedikit berbeda. Termotivasi oleh pendekatan di atas, kita mungkin berharap menemukan bilangan bulat ganjil r yang

menyatakan $a^r = 1 \bmod p$. Maka, seperti di atas, $a^{r+1} = a \bmod p$ dan $a^{\frac{r+1}{2}} \bmod p$ akan menjadi akar kuadrat dari a dengan $(r + 1)/2$ bilangan bulat. Meskipun kita tidak dapat melakukan hal ini, kita dapat melakukan hal yang sama baiknya: kita akan mencari bilangan bulat ganjil r beserta elemen $b \in \mathbb{Z}_p^*$ dan bilangan bulat genap r' sehingga

$$a^r \cdot b^{r'} = 1 \bmod p.$$

Maka $a^{r+1} \cdot b^{r'} = a \bmod p$ dan $a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p$ adalah akar kuadrat dari a (dengan eksponen $(r + 1)/2$ dan $r'/2$ adalah bilangan bulat).

Kami sekarang menjelaskan pendekatan umum untuk mencari r, b, dan r' dengan sifat-sifat yang disebutkan. Misal $\frac{p-1}{2} = 2^\ell$ di mana ℓ, m adalah bilangan bulat dengan $\ell' \geq 1$ dan m ganjil. Karena a adalah residu kuadrat, kita mengetahui bahwa

$$a^{2^\ell m} = a^{\frac{p-1}{2}} = 1 \bmod p. \qquad (13.6)$$

Artinya $a^{2^\ell m/2} = a^{2^{\ell-1} m}$ adalah akar kuadrat dari 1. Akar kuadrat dari 1 modulo p adalah $\pm 1 \bmod p$, jadi $a^{2^{\ell-1} m} = \pm 1 \bmod p$. Jika $a^{2^{\ell-1} m} = 1 \bmod p$, kita berada dalam situasi yang sama seperti pada Persamaan (13.6) kecuali eksponen a sekarang habis dibagi pangkat 2 yang lebih kecil. Ini adalah kemajuan ke arah yang benar: jika kita bisa mendapatkan hingga eksponen a tidak habis dibagi pangkat 2 (seperti yang terjadi di sini jika $\ell = 1$), maka eksponen a adalah ganjil dan kita dapat menghitung akar kuadrat seperti yang dibahas sebelumnya. Kita berikan sebuah contoh, dan diskusikan sebentar bagaimana menangani kasus ketika $a^{2^{\ell-1} m} = -1 \bmod p$

Contoh 13.30

Ambil p = 29 dan a = 7. Karena 7 adalah residu kuadrat modulo 29, kita mempunyai 7^{14} mod 29 = 1 dan kita tahu bahwa 7^7 mod 29 adalah akar kuadrat dari 1. Faktanya,

$$7^7 = 1 \bmod 29,$$

dan eksponen 7 ganjil. Jadi $7^{(7+1)/2} = 7^4 = 23$ mod 29 adalah akar kuadrat dari 7 modulo 29.

Untuk meringkas algoritme sejauh ini: kita mulai dengan $a^{2^\ell m} = 1 \bmod p$ dan kita mengeluarkan faktor 2 dari eksponen hingga salah satu dari dua hal terjadi: $a^m = 1 \bmod p$, atau $a^{2^{\ell'} m} = -1 \bmod p$ untuk beberapa $\ell' < \ell$. Dalam kasus pertama, karena m ganjil, kita dapat langsung menghitung akar kuadrat dari a seperti pada Contoh 13.30.

Dalam kasus kedua, kita akan "mengembalikan" +1 di ruas kanan persamaan dengan mengalikan setiap ruas persamaan dengan −1 mod p. Namun, seperti yang dimotivasi di awal diskusi ini, kita ingin mencapai hal ini dengan mengalikan ruas kiri persamaan dengan

beberapa elemen b yang dipangkatkan genap. Jika kita mempunyai non-residu kuadrat $b \in \mathbb{Z}_p^*$ ini mudah dilakukan: karena $b^{2^\ell m} = b^{\frac{p=1}{2}} = -1 \bmod p$, kita punya

$$a^{2^{\ell'} m} \cdot b^{2^\ell m} = (-1)(-1) = +1 \bmod p.$$

Dengan ini kita dapat melanjutkan seperti sebelumnya, mengambil akar kuadrat dari ruas kiri untuk mengurangi pangkat terbesar dari 2 membagi eksponen a, dan mengalikannya dengan $b^{2^\ell m}$ (sesuai kebutuhan) sehingga ruas kanan selalu +1. Perhatikan bahwa eksponen b selalu habis dibagi dengan pangkat 2 yang lebih besar daripada eksponen a (sehingga kita dapat mengambil akar kuadrat dengan membaginya dengan 2 pada kedua eksponen). Kami terus melakukan langkah-langkah ini sampai eksponen a ganjil, dan kemudian dapat menghitung akar kuadrat dari a seperti dijelaskan sebelumnya. Pseudocode untuk algoritma ini, yang memberikan cara lain untuk melihat apa yang sedang terjadi, diberikan di bawah pada Algoritma 13.31. Dapat diverifikasi bahwa algoritme berjalan dalam waktu polinomial jika diberi non-residu kuadrat b karena jumlah iterasi loop dalam adalah $\ell = \mathcal{O}(\log p)$

Satu hal yang belum kita bahas adalah cara mencari b. Faktanya, tidak ada algoritma waktu polinomial deterministik untuk menemukan modulo p non-residu kuadrat yang diketahui. Untungnya, mudah untuk menemukan non-residu kuadrat secara probabilistik: cukup pilih elemen seragam $\mathbb{Z}_p^*$ hingga non-residu kuadrat ditemukan. Hal ini berhasil karena separuh elemen $\mathbb{Z}_p^*$ merupakan non-residu kuadrat, dan karena algoritma waktu polinomial untuk menentukan modulo residuositas kuadrat bilangan prima telah diketahui (lihat Bagian 13.4 untuk bukti dari kedua pernyataan ini). Artinya algoritma yang kami tunjukkan sebenarnya diacak ketika $p = 1 \, mod \, 4$; algoritma waktu polinomial deterministik untuk menghitung akar kuadrat dalam kasus ini tidak diketahui.

ALGORITHM 13.31
Computing square roots modulo a prime

Input: Prime p; quadratic residue $a \in \mathbb{Z}_p^*$
Output: A square root of a

case $p = 3 \bmod 4$:
  return $[a^{\frac{p+1}{4}} \bmod p]$
case $p = 1 \bmod 4$:
  let b be a quadratic non-residue modulo p
  compute ℓ and m odd with $2^\ell \cdot m = \frac{p-1}{2}$
  $r := 2^\ell \cdot m$, $r' := 0$
  for $i = \ell$ to 1 {
  // maintain the invariant $a^r \cdot b^{r'} = 1 \bmod p$
    $r := r/2$, $r' := r'/2$
    if $a^r \cdot b^{r'} = -1 \bmod p$
      $r' := r' + 2^\ell \cdot m$
  }
  // now $r = m$, r' is even, and $a^r \cdot b^{r'} = 1 \bmod p$
  return $\left[a^{\frac{r+1}{2}} \cdot b^{\frac{r'}{2}} \bmod p\right]$

Contoh 13.32

Di sini kita mempertimbangkan "kasus terburuk", ketika mengambil akar kuadrat selalu menghasilkan −1. Misalkan $a \in \mathbb{Z}_p^*$ adalah elemen yang akar kuadratnya ingin kita hitung; misalkan $b \in \mathbb{Z}_p^*$ adalah non-residu kuadrat; dan misalkan $\frac{p-1}{2} = 2^3 \cdot m$ dimana m ganjil.

Pada langkah pertama, kita mempunyai $a^{2^3 m} = 1 \bmod p$. Karena $a^{2^3 m} = \left(a^{2^3 m}\right)^2$ dan akar kuadrat dari 1 adalah ±1, berarti $a^{2^2 m} = \pm 1 \bmod p$; dengan asumsi kasus terburuk, $a^{2^2 m} = -1 \bmod p$. Jadi, kita kalikan dengan $a^{2m} \cdot b^{2^2 m} = -1 \bmod p$ untuk mendapatkan

$$a^{2^2 m} \cdot b^{2^3 m} = 1 \bmod p.$$

Pada langkah kedua, kita amati bahwa $a^{2m} \cdot b^{2^2 m}$ adalah akar kuadrat dari 1; sekali lagi dengan asumsi kasus terburuk, maka kita mempunyai $a^{2m} \cdot b^{2^2 m} = -1 \bmod p$. Mengalikannya dengan $b^{2^2 m}$ untuk "memperbaiki" hasilnya

$$a^{2m} \cdot b^{2^2 m} \cdot b^{2^3 m} = 1 \bmod p.$$

Pada langkah ketiga, mengambil akar kuadrat dan mengasumsikan kasus terburuk (seperti di atas) kita memperoleh $a^m \cdot b^{2m} \cdot b^{2^2 m} = -1 \bmod p$; mengalikannya dengan "faktor koreksi" $b^{2^3 m}$kita mendapatkan

$$a^m \cdot b^{2m} \cdot b^{2^2 m} \cdot b^{2^3 m} = 1 \bmod p.$$

Kami sekarang berada di tempat yang kami inginkan. Untuk menyimpulkan algoritma, kalikan kedua ruas dengan a untuk mendapatkan

$$a^{m+1} \cdot b^{2m+2^2 m+2^3 m} = a \bmod p.$$

Karena m ganjil, $(m+1)/2$ adalah bilangan bulat dan $a^{\frac{m+1}{2}} \cdot b^{m+2m+2^2 m} \bmod p$ adalah akar kuadrat dari a.

Contoh 13.33

Di sini kita memberikan contoh konkritnya. Misalkan $p = 17, a = 2$, dan $b = 3$. Perhatikan bahwa di sini $(p-1)/2 = 2^3$ dan $m = 1$.

Kita mulai dengan $2^{2^2} = 1 \bmod 17$. Jadi 2^{2^2} harus sama dengan $\pm 1 \, mod \, 17$; dengan perhitungan, 2^{2^2} = −1 mod 17. Dikalikan dengan 3^{2^3} menghasilkan $2^{2^2} \cdot 3^{2^3} = 1 \bmod 17$.

Selanjutnya, kita mengetahui bahwa $2^2 \cdot 3^{2^3}$ adalah akar kuadrat dari 1 sehingga harus sama dengan ±1 mod 17; perhitungan menghasilkan $2^2 \cdot 3^{2^3}$ = 1 mod 17. Jadi tidak diperlukan istilah koreksi di sini.

Dengan membagi dua eksponennya lagi, kita mendapatkan bahwa $2 \cdot 3^2$ = 1 mod 17. Sekarang kita hampir selesai: mengalikan kedua ruas dengan 2 menghasilkan $2^2 \cdot 3^2$ = 2 mod 17, sehingga 2 · 3 = 6 mod 17 adalah akar kuadrat dari 2 .

Menghitung Akar Kuadrat Modulo N

Tidak sulit untuk melihat bahwa algoritme yang telah kami tunjukkan untuk menghitung akar kuadrat modulo a bilangan prima dapat diterapkan dengan mudah pada kasus penghitungan akar kuadrat modulo a komposit $N = pq$ dari faktorisasi yang diketahui. Secara khusus, misalkan $a \in \mathbb{Z}_N^*$ adalah residu kuadrat dengan $a \leftrightarrow (a_p, a_q)$ melalui teorema sisa Cina. Menghitung akar kuadrat x_p, x_q dari a_p, a_q modulo p dan q, masing-masing, menghasilkan akar kuadrat (x_p, x_q) dari a (lihat Bagian 13.4.2). Mengingat x_p, x_q, representasi x yang terkait dengan (x_p, x_q) dapat dipulihkan seperti yang dibahas di Bagian 8.1. Artinya, untuk menghitung akar kuadrat dari modulo bilangan bulat $N = pq$ dari faktorisasi yang diketahui:

- Hitung $a_p := [a \ mod \ p\ ]$ dan $a_q := [a \ mod \ q]$.
- Dengan menggunakan Algoritma 13.31, hitung akar kuadrat x_p dari a_p modulo p dan akar kuadrat x_q dari a_q modulo q.
- Konversi representasi $(x_p, x_q) \in \mathbb{Z}_p^* \times \mathbb{Z}_p^*$ ke $x \in \mathbb{Z}_N^*$ dengan $x \leftrightarrow (x_p, x_q)$. Keluaran x, yang merupakan akar kuadrat dari modulo N .

Sangat mudah untuk memodifikasi algoritma sehingga mengembalikan keempat akar kuadrat dari a.

Permutasi Trapdoor Berdasarkan Anjak Piutang

Kita telah melihat bahwa menghitung akar kuadrat modulo N dapat dilakukan dalam waktu polinomial jika faktorisasi N diketahui. Kami tunjukkan di sini bahwa, sebaliknya, menghitung akar kuadrat modulo gabungan N dari faktorisasi yang tidak diketahui sama sulitnya dengan memfaktorkan N.

Lebih formalnya, misalkan GenModulus adalah algoritma waktu polinomial yang, pada masukan 1n, menghasilkan keluaran (N, p, q) dengan $N = pq$ dan p dan q adalah n-bit bilangan prima kecuali dengan probabilitas yang dapat diabaikan dalam n. Pertimbangkan percobaan berikut untuk algoritma tertentu $\mathcal{A}$ dan parameter n:

Eksperimen komputasi akar kuadrat SQRA,GenModulus(n):

1. Jalankan GenModulus(1^n) untuk mendapatkan keluaran N, p, q.
2. Pilih seragam $y \in \mathcal{QR}_N$.
3. Diberikan (N, y), dan keluaran $x \in \mathbb{Z}_N^*$.
4. Keluaran percobaan ditetapkan 1 jika $x^2 = y \bmod N$, dan 0 jika tidak.

DEFINISI 13.34 Kita mengatakan bahwa menghitung akar kuadrat relatif sulit dibandingkan GenModulus jika untuk semua algoritma waktu polinomial probabilistik $\mathcal{A}$ terdapat fungsi yang dapat diabaikan sehingga

$$\Pr[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n).$$

Sangat mudah untuk melihat bahwa jika menghitung akar kuadrat relatif sulit terhadap GenModulus maka pemfaktoran juga relatif sulit terhadap GenModulus: jika keluaran moduli N oleh GenModulus dapat difaktorkan dengan mudah, maka akan mudah untuk menghitung akar kuadrat modulo N dengan memfaktorkan terlebih dahulu N dan kemudian menerapkan algoritma yang dibahas di bagian sebelumnya. Tujuan kami sekarang adalah untuk menunjukkan kebalikannya: jika pemfaktoran relatif sulit dibandingkan GenModulus, maka masalah penghitungan akar kuadrat juga sulit. Kami tekankan lagi bahwa hasil analog tidak diketahui untuk masalah RSA atau masalah penentuan residuositas kuadrat.

Kuncinya adalah lemma berikut, yang mengatakan bahwa dua akar kuadrat "tidak berhubungan" dari elemen apa pun di $\mathbb{Z}_N^*$ dapat digunakan untuk memfaktorkan N .

LEMMA 13.35 Misalkan $N = pq$ dengan p, q bilangan prima ganjil yang berbeda. Diketahui x, $\hat{x}$ sedemikian sehingga $x^2 = y = \hat{x}^2 \bmod N$ tetapi $x \neq \pm\hat{x} \bmod N$, adalah mungkin untuk memfaktorkan N dalam polinomial waktu di N .

BUKTI Kita menyatakan bahwa gcd$(N, x + \hat{x})$ atau gcd$(N, x - \hat{x})$ sama dengan salah satu faktor prima dari N.Karena perhitungan gcd dapat dilakukan dalam waktu polinomial, ini membuktikan lemmanya.

Jika $x^2 = \hat{x}^2 \bmod N$ maka

$$0 = x^2 - \hat{x}^2 = (x - \hat{x}) \cdot (x + \hat{x}) \bmod N,$$

dan seterusnya $N \mid (x + \hat{x})(x - \hat{x})$. Kemudian $p \mid (x + \hat{x})(x - \hat{x})$ dan p membagi salah satu suku ini. Ucapkan $p|(x + \hat{x})$ (pembuktiannya berjalan sama jika $p|(x - \hat{x})$). Jika $q|(x + \hat{x})$ lalu $N|(x + \hat{x})$, tetapi hal ini tidak dapat terjadi karena $x \neq -\hat{x} \bmod N$. Jadi $q|(x + \hat{x})$ dan $\gcd(N, x + \hat{x}) = p$.

Cara alternatif untuk membuktikan hal di atas adalah dengan melihat apa yang terjadi pada sisa representasi Tiongkok. Ucapkan $x \leftrightarrow (x_p, x_q)$. Lalu, karena x dan $\hat{x}$ adalah akar kuadrat dengan nilai y yang sama, kita tahu bahwa $\hat{x}$ berkorespondensi dengan salah satu (x_p, x_q) atau $(-x_p, -x_q)$.. (Ini tidak bisa sesuai dengan (x_p, x_q) atau $(-x_p, -x_q)$. karena yang pertama sesuai dengan x sedangkan yang kedua sesuai dengan $[-x \bmod N]$, dan kedua kemungkinan dikesampingkan oleh asumsi lemma.) Katakanlah $\hat{x} \leftrightarrow (-x_p, x_q)$. Kemudian

$$[x + \hat{x} \bmod N] \leftrightarrow (x_p, x_q) + (-x_p, x_q) = (0, [2x_q \bmod q]),$$

dan kita melihat bahwa $x + \hat{x} = 0 \mod p$ sedangkan $x + \hat{x} \neq 0 \bmod p$. Oleh karena itu $\gcd(N, x + \hat{x}) = p$, faktor dari N .
Sekarang kita dapat membuktikan hasil utama dari bagian ini.

TEOREMA 13.36 Jika memfaktorkan relatif sulit terhadap GenModulus, maka menghitung akar kuadrat relatif sulit terhadap GenModulus.

BUKTI Misalkan algoritma $\mathcal{A}$ waktu polinomial probabilistik yang menghitung akar kuadrat (seperti pada Definisi 13.34). Pertimbangkan algoritma waktu polinomial probabilistik berikut $\mathcal{A}_{fact}$ untuk memfaktorkan keluaran moduli oleh GenModulus:

Algoritma $\mathcal{A}_{fact}$:
Algoritma diberikan modulus N sebagai masukan.

- Pilih seragam $x \in \mathbb{Z}_N^*$ dan hitung $y := [x^2 \ mod \ N \ ]$.
- Jalankan $\mathcal{A}(N, y)$ untuk memperoleh keluaran $\hat{x}$.
- Jika $\hat{x}^2 = y \bmod N$ dan $\hat{x} \neq \pm x \bmod N$ maka faktorkan N menggunakan Lemma 13.35.

Berdasarkan Lemma 13.35, kita mengetahui bahwa $\mathcal{A}_{fact}$ berhasil memfaktorkan N tepat ketika $\hat{x} \neq \pm x \bmod N$ dan $\hat{x}^2 = y \bmod N$. Itu adalah,

$$\begin{aligned}
&\Pr[\mathsf{Factor}_{\mathcal{A}_{\mathrm{fact}},\mathsf{GenModulus}}(n) = 1] \\
&= \Pr\left[\hat{x} \neq \pm x \bmod N \wedge \hat{x}^2 = y \bmod N\right] \\
&= \Pr\left[\hat{x} \neq \pm x \bmod N \mid \hat{x}^2 = y \bmod N\right] \cdot \Pr\left[\hat{x}^2 = y \bmod N\right], \quad (13.7)
\end{aligned}$$

dimana semua probabilitas di atas mengacu pada eksperimen $\mathrm{Faktor}_{\mathcal{A}_{fact},\mathrm{GenModulus}(n)}$ (lihat Bagian 8.2 untuk penjelasan eksperimen ini). Dalam percobaan, modulus N yang diberikan sebagai masukan fakta dihasilkan oleh GenModulus(1^n), dan y adalah modulo residu kuadrat seragam N karena x dipilih secara seragam dari $\mathbb{Z}_N^*$. Jadi tampilan $\mathcal{A}$ ketika dijalankan sebagai subrutin oleh $\mathcal{A}_{fact}$ didistribusikan persis seperti tampilan $\mathcal{A}$ pada eksperimen $\mathrm{SQR}_{\mathcal{A}_{fact},\mathrm{GenModulus}(n)}$. Karena itu,

$$\Pr\left[\hat{x}^2 = y \bmod N\right] = \Pr\left[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1\right]. \qquad (13.8)$$

Dikondisikan pada nilai residu kuadrat y yang digunakan dalam eksperimen $\mathrm{Faktor}_{\mathcal{A}_{fact},\mathrm{GenModulus}(n)}$, nilai x kemungkinan besar merupakan salah satu dari empat

kemungkinan akar kuadrat dari y. Ini berarti bahwa dari sudut pandang algoritma $\mathcal{A}$ (dijalankan sebagai subrutin oleh $\mathcal{A}_{fact}$), x mempunyai kemungkinan yang sama untuk masing-masing dari empat akar kuadrat dari y. Artinya, jika $\mathcal{A}$ menghasilkan akar kuadrat $\hat{x}$ dari y, probabilitas bahwa $\hat{x} \neq \pm x \bmod N$ adalah tepat 1/2. (Kami menekankan bahwa kami tidak membuat asumsi apa pun tentang bagaimana x didistribusikan di antara akar kuadrat dari y, dan khususnya di sini kami tidak berasumsi bahwa $\mathcal{A}$ menghasilkan akar kuadrat seragam dari y. Sebaliknya kami menggunakan fakta bahwa x terdistribusi secara seragam di antara akar kuadrat dari y.) Artinya,

$$\Pr\left[\hat{x} \neq \pm x \bmod N \mid \hat{x}^2 = y \bmod N\right] = \frac{1}{2}. \qquad (13.9)$$

Menggabungkan Persamaan (13.7) – (13.9), kita melihat bahwa

$$\Pr\left[\mathsf{Factor}_{\mathcal{A}_{\mathrm{fact}},\mathsf{GenModulus}}(n) = 1\right] = \frac{1}{2} \cdot \Pr\left[\mathsf{SQR}_{\mathcal{A},\mathsf{GenModulus}}(n) = 1\right].$$

Karena pemfaktoran relatif sulit terhadap GenModulus, ada fungsi yang dapat diabaikan sehingga dapat diabaikan

$$\Pr[\mathsf{Factor}_{\mathcal{A}_{\mathrm{fact}},\mathsf{GenModulus}}(n) = 1] \leq \mathsf{negl}(n),$$

yang berarti $\Pr\left[\mathrm{SQR}_{\mathcal{A}_{fact},\mathrm{GenModulus}}(n) = 1\right] \leq 2 \cdot \mathrm{negl}(n)$. Karena $\mathcal{A}$ bersifat arbitrer, maka ini melengkapi pembuktiannya.

Teorema sebelumnya mengarah langsung ke sekumpulan fungsi satu arah (lihat Definisi 8.76) berdasarkan $\mathrm{GenModulus}$ apa pun yang relatif sulit untuk difaktorkan:

- Algoritma Gen, pada input n menjalankan $\mathrm{GenModulus}(n)$ untuk mendapatkan (N, p, q) dan output $I = N$. Domain $\mathcal{D}_I$ adalah $\mathbb{Z}_N^*$ dan rentang $\mathcal{R}_I$ adalah $\mathcal{QR}_N$.
- Algoritma $Samp$, pada masukan N, memilih elemen seragam $x \in \mathbb{Z}_N^*$.
- Algoritma f, pada masukan N dan $x \in \mathbb{Z}_N^*$, keluaran $[x^2 \bmod N]$.

Teorema sebelumnya menunjukkan bahwa keluarga ini bersifat satu arah jika pemfaktoran relatif sulit terhadap $\mathrm{GenModulus}$.

Kita dapat mengubahnya menjadi rangkaian permutasi satu arah dengan menggunakan moduli N bentuk khusus dan membiarkan $\mathcal{D}_I$ menjadi subset dari $\mathbb{Z}_N^*$. Panggil $N = pq$ bilangan bulat Blum jika p dan q adalah bilangan prima berbeda dengan $p = q = 3 \bmod 4$. Kunci untuk membuat permutasi adalah proposisi berikut.

PROPOSISI 13.37 Misalkan N bilangan bulat Blum. Maka setiap modulo residu kuadrat N mempunyai tepat satu akar kuadrat yang juga merupakan residu kuadrat.

BUKTI Katakanlah $N = pq$ dengan $p = q = 3 \, mod \, 4$. Dengan menggunakan Proposisi 13.17, kita melihat bahwa −1 bukanlah modulo residu kuadrat p atau q. Hal ini karena untuk $p = 3 \, mod \, 4$ dinyatakan bahwa $p = 4i + 3$ untuk beberapa i dan seterusnya

$$(-1)^{\frac{p-1}{2}} = (-1)^{2i+1} = -1 \bmod p$$

karena $2i + 1$ ganjil). Sekarang misalkan $y \leftrightarrow (y_p, y_q)$ adalah modulo residu kuadrat sembarang N dengan empat akar kuadrat

$$(x_p, x_q), \quad (-x_p, x_q), \quad (x_p, -x_q), \quad (-x_p, -x_q).$$

Kami mengklaim bahwa salah satunya adalah modulo residu kuadratik N . Untuk melihatnya, asumsikan $J_p(x_p) = +1$ dan $J_q(x_q) = -1$ (pembuktiannya berjalan sama dalam kasus lain). Dengan menggunakan Proposisi 13.19, kita punya

$$\mathcal{J}_q(-x_q) = \mathcal{J}_q(-1) \cdot \mathcal{J}_q(x_q) = +1 \,,$$

dan $(x_p, -x_q)$ berhubungan dengan modulo residu kuadratik N (menggunakan Proposisi 13.21). Demikian pula, $J_p(-x_p) = -1$ sehingga tidak ada akar kuadrat lain dari y yang merupakan residu kuadrat modulo N .

Dinyatakan secara berbeda, proposisi di atas mengatakan bahwa ketika N adalah bilangan bulat Blum, fungsi $f_N : \mathcal{QR}_N \rightarrow \mathcal{QR}_N$ yang diberikan oleh $f_N(x) = x^2 \bmod N$ adalah permutasi atas $\mathcal{QR}_N$. Memodifikasi algoritma pengambilan sampel Samp di atas untuk memilih seragam $x \in \mathcal{QR}_N$ (yang, seperti telah kita lihat, dapat dilakukan dengan mudah dengan memilih seragam $r \in \mathbb{Z}_N^*$ dan mengatur $x := r^2 \bmod N$) menghasilkan satu keluarga -permutasi arah. Terakhir, karena modulo akar kuadrat N dapat dihitung dalam waktu polinomial dengan faktorisasi N , modifikasi langsung akan menghasilkan rangkaian permutasi pintu jebakan berdasarkan $\mathrm{GenModulus}$ apa pun yang relatif sulit untuk difaktorkan. Ini kadang-kadang disebut permutasi pintu jebakan keluarga Rabin. Kesimpulan:

TEOREMA 13.38 Misalkan $\mathrm{GenModulus}$ adalah algoritma yang, pada input 1^n, menghasilkan output (N, p, q) dimana $N = pq$ dan p dan q adalah bilangan prima yang berbeda (kecuali mungkin dengan probabilitas yang dapat diabaikan) dengan $p = q = 3 \bmod 4$. Jika memfaktorkan relatif sulit terhadap $\mathrm{GenModulus}$, maka terdapat rangkaian permutasi pintu jebakan.

Skema Enkripsi Rabin

Kita dapat menerapkan hasil Bagian 13.1 pada permutasi pintu jebakan Rabin untuk mendapatkan skema enkripsi kunci publik yang keamanannya didasarkan pada pemfaktoran.

Untuk melakukan hal ini, pertama-tama kita perlu mengidentifikasi predikat inti untuk permutasi pintu jebakan ini.

CONSTRUCTION 13.39

Let GenModulus be a polynomial-time algorithm that, on input 1^n, outputs (N, p, q) where $N = pq$ and p and q are n-bit primes (except with probability negligible in n) with $p = q = 3 \bmod 4$. Construct a public-key encryption scheme as follows:

- Gen: on input 1^n run GenModulus(1^n) to obtain (N, p, q). The public key is N, and the private key is $\langle p, q \rangle$.
- Enc: on input a public-key N and message $m \in \{0, 1\}$, choose a uniform $x \in \mathcal{QR}_N$ subject to the constraint that $\mathsf{lsb}(x) = m$. Output the ciphertext $c := [x^2 \bmod N]$.
- Dec: on input a private key $\langle p, q \rangle$ and a ciphertext c, compute the unique $x \in \mathcal{QR}_N$ such that $x^2 = c \bmod N$, and output $\mathsf{lsb}(x)$.

Meskipun kita dapat mengacu pada Teorema 13.3, yang menyatakan bahwa predikat hard-core yang sesuai selalu ada, ternyata bit paling tidak signifikan lsb adalah predikat hard-core untuk permutasi pintu jebakan Rabin seperti halnya untuk kasus RSA. Menggunakan ini sebagai predikat inti kami, kami memperoleh skema Konstruksi 13.39.

Skema enkripsi Rabin. Teorema 13.5 dan 13.38 mengimplikasikan hasil sebagai berikut.

TEOREMA 13.40 Jika pemfaktoran relatif sulit terhadap GenModulus, maka Konstruksi 13.39 aman untuk CPA.

Enkripsi Rabin vs. Enkripsi RSA

Perlu diperhatikan persamaan dan perbedaan antara kriptosistem Rabin dan RSA. (Pembahasan di sini berlaku untuk konstruksi kriptografi apa pun—tidak harus berupa skema enkripsi kunci publik—yang didasarkan pada permutasi pintu jebakan Rabin atau RSA.)

Permutasi RSA dan pintu jebakan Rabin tampak sangat mirip, dengan mengkuadratkan dalam kasus Rabin sesuai dengan pengambilan e = 2 dalam kasus RSA. (Tentu saja, 2 tidak relatif prima terhadap $\varphi(N)$ sehingga Rabin bukanlah kasus khusus RSA.) Dalam hal keamanan yang ditawarkan oleh setiap konstruksi, kekerasan komputasi akar kuadrat modular setara dengan kekerasan pemfaktoran, sedangkan kekerasan penyelesaian masalah RSA tidak diketahui tersirat oleh kekerasan pemfaktoran. Oleh karena itu, permutasi pintu jebakan Rabin didasarkan pada asumsi yang berpotensi lebih lemah: secara teoritis ada kemungkinan bahwa seseorang dapat mengembangkan algoritma yang efisien untuk memecahkan masalah RSA, namun menghitung akar kuadrat akan tetap sulit. Atau, seseorang mungkin mengembangkan suatu algoritma yang memecahkan masalah RSA lebih cepat daripada algoritma pemfaktoran yang dikenal. Lemma 13.35 memastikan, bagaimanapun, bahwa

komputasi akar kuadrat modulo N tidak akan pernah bisa lebih cepat dari algoritma terbaik yang tersedia untuk memfaktorkan N.

Dalam hal efisiensi, permutasi RSA dan Rabin pada dasarnya sama. Sebenarnya, jika eksponen e yang besar digunakan dalam kasus RSA maka komputasi pangkat eth (seperti dalam RSA) sedikit lebih lambat dibandingkan mengkuadratkan (seperti dalam Rabin). Di sisi lain, diperlukan kehati-hatian lebih saat bekerja dengan permutasi Rabin karena permutasi ini hanya merupakan permutasi pada sebagian $\mathbb{Z}_N^*$, berbeda dengan RSA, yang memberikan permutasi pada seluruh $\mathbb{Z}_N^*$.

Skema enkripsi “plain Rabin”, yang dibangun dengan cara analog dengan enkripsi RSA biasa, rentan terhadap serangan teks sandi terpilih yang memungkinkan musuh mempelajari seluruh kunci privat. Meskipun RSA biasa juga tidak aman untuk CCA, serangan teks sandi terpilih yang diketahui pada RSA biasa tidak terlalu merusak karena serangan tersebut memulihkan pesan tetapi tidak memulihkan kunci pribadinya. Mungkin adanya serangan terhadap “plain Rabin” mempengaruhi para kriptografer, sejak awal, untuk menolak penggunaan enkripsi Rabin sepenuhnya.

Ringkasnya, dalam praktiknya, permutasi RSA jauh lebih luas digunakan dibandingkan permutasi Rabin, namun mengingat hal-hal di atas, hal ini tampaknya lebih disebabkan oleh kebetulan sejarah dibandingkan dengan pembenaran teknis yang memaksa.

DAFTAR PUSTAKA

Bellare, M., & Rogaway, P. (1993). "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols." In Proceedings of the 1st ACM Conference on Computer and Communications Security (CCS '93) (pp. 62-73). New York, NY: ACM Press.

Bellare, M., & Rogaway, P. (2005). "Introduction to Modern Cryptography." Retrieved from https://web.cs.ucdavis.edu/~rogaway/papers/intro.pdf

Bernstein, D. J. (2009). "Introduction to Post-Quantum Cryptography." Retrieved from https://cr.yp.to/papers.html#pqcrypto

Boneh, D., & Boyen, X. (2004). "Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles." In Advances in Cryptology - EUROCRYPT 2004 (pp. 223-238). Berlin, Heidelberg: Springer.

Boneh, D., & Franklin, M. (2001). "Identity-Based Encryption from the Weil Pairing." SIAM Journal on Computing, 32(3), 586-615.

Boneh, D., & Lipton, R. (1995). "A Revocable-Identity-Based Cryptosystem." In Advances in Cryptology - CRYPTO '95 (pp. 427-432). Berlin, Heidelberg: Springer.

Boneh, D., & Shacham, H. (2008). "Group Signatures with Verifier-Local Revocation." In Advances in Cryptology - EUROCRYPT 2004 (pp. 167-185). Berlin, Heidelberg: Springer.

Boneh, D., & Shoup, V. (2000). "A Graduate Course in Applied Cryptography." In Advances in Cryptology - CRYPTO 2000 (pp. 179-199). Berlin, Heidelberg: Springer.

Boneh, D., & Shoup, V. (2004). "A Graduate Course in Applied Cryptography." Retrieved from https://crypto.stanford.edu/~dabo/cryptobook/

Boneh, D., & Waters, B. (2003). "Conjunctive, Subset, and Range Queries on Encrypted Data." In Advances in Cryptology - EUROCRYPT 2003 (pp. 535-554). Berlin, Heidelberg: Springer.

Boneh, D., & Waters, B. (2005). "A Collusion Resistant Broadcast Encryption Scheme." In Advances in Cryptology - CRYPTO 2005 (pp. 480-501). Berlin, Heidelberg: Springer.

Boneh, D., & Waters, B. (2007). "Constrained Pseudorandom Functions and Their Applications." In Advances in Cryptology - EUROCRYPT 2007 (pp. 112-129). Berlin, Heidelberg: Springer.

Delfs, H., & Knebl, H. (2007). Introduction to Cryptography: Principles and Applications (3rd ed.). Berlin, Germany: Springer.

Delfs, H., & Knebl, H. (2016). "Elliptic Curves and Their Applications to Cryptography: An Introduction." Berlin, Germany: Springer.

Diffie, W., & Hellman, M. (1976). "New Directions in Cryptography." IEEE Transactions on Information Theory, 22(6), 644-654.

Diffie, W., & van Oorschot, P. C. (1992). "Authentication and Authenticated Key Exchanges." Designs, Codes and Cryptography, 2(2), 107-125.

Ferguson, N., & Schneier, B. (2003). "Practical Cryptography." New York, NY: John Wiley & Sons

Ferguson, N., Schneier, B., & Kohno, T. (2010). Cryptography Engineering: Design Principles and Practical Applications. Indianapolis, IN: Wiley Publishing.

Goldreich, O. (2001). Foundations of Cryptography: Volume 1, Basic Tools. Cambridge, UK: Cambridge University Press.

Goldwasser, S., & Bellare, M. (2012). "The Complexity of Decision versus Search." In Advances in Cryptology - CRYPTO '93 (pp. 238-253). Berlin, Heidelberg: Springer.

Goldwasser, S., & Micali, S. (1984). "Probabilistic Encryption." Journal of Computer and System Sciences, 28(2), 270-299.

Goldwasser, S., & Micali, S. (1986). "Probabilistic Encryption & How to Play Mental Poker Keeping Secret All Partial Information." Journal of the ACM, 33(4), 830-840.

Hohenberger, S., & Waters, B. (2014). "Short Signatures Without Random Oracles." In Advances in Cryptology - EUROCRYPT 2009 (pp. 569-588). Berlin, Heidelberg: Springer.

Katz, J., & Lindell, Y. (2014). "Introduction to Modern Cryptography." Retrieved from https://eprint.iacr.org/2014/878

Katz, J., & Lindell, Y. (2015). Introduction to Modern Cryptography (2nd ed.). Boca Raton, FL: CRC Press.

Katz, J., & Lindell, Y. (2016). "Introduction to Modern Cryptography." Retrieved from https://eprint.iacr.org/2016/059

Katz, J., & Lindell, Y. (2017). "Secure Multiparty Computation." In Introduction to Modern Cryptography (3rd ed., pp. 399-434). Boca Raton, FL: CRC Press.

Katz, J., & Lindell, Y. (2018). "Hash Functions and Data Integrity." In Introduction to Modern Cryptography (3rd ed., pp. 237-268). Boca Raton, FL: CRC Press.

Katz, J., & Lindell, Y. (2019). "Public-Key Cryptography." In Introduction to Modern Cryptography (3rd ed., pp. 201-236). Boca Raton, FL: CRC Press.

Koblitz, N. (1987). "Elliptic Curve Cryptosystems." Mathematics of Computation, 48(177), 203-209.

Koblitz, N. (1994). A Course in Number Theory and Cryptography. Berlin, Germany: Springer-Verlag.

Koblitz, N. (1997). "Elliptic Curve Cryptosystems." Mathematics of Computation, 48(177), 203-209.

Menezes, A. J., & Vanstone, S. A. (1996). "Elliptic Curve Cryptography and Public Key Infrastructure." New York, NY: Springer.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1997). Handbook of Applied Cryptography. Boca Raton, FL: CRC Press.

Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. (1998). "Handbook of Applied Cryptography." New York, NY: CRC Press.

Naor, M., & Yung, M. (1989). "Universal One-Way Hash Functions and Their Cryptographic Applications." In Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC '89) (pp. 33-43). New York, NY: ACM Press.

Naor, M., & Yung, M. (1990). "Public-Key Cryptosystems Provably Secure Against Chosen Ciphertext Attacks." In Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC '90) (pp. 427-437). New York, NY: ACM Press.

Paar, C., & Pelzl, J. (2010). Understanding Cryptography: A Textbook for Students and Practitioners. New York, NY: Springer.

Paar, C., & Pelzl, J. (2015). "Understanding Cryptography: A Textbook for Students and Practitioners (2nd ed.)." New York, NY: Springer.

Paillier, P. (1999). "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes." In Advances in Cryptology - EUROCRYPT 1999 (pp. 223-238). Berlin, Heidelberg: Springer.

Paillier, P. (2007). "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes." In Advances in Cryptology - CRYPTO 2007 (pp. 223-238). Berlin, Heidelberg: Springer.

Rabin, M. O. (1979). "Digitalized Signatures and Public-Key Functions as Intractable as Factorization." MIT Laboratory for Computer Science Technical Report.

Rivest, R. L. (1990). "Cryptographic Computation: Secure Computations of a Public Key Cryptosystem." MIT Laboratory for Computer Science Technical Report.

Rivest, R. L., Shamir, A., & Adleman, L. M. (1978). "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems." Communications of the ACM, 21(2), 120-126.

Rogaway, P. (2003). "The Moral Character of Cryptographic Work." Cryptology ePrint Archive, Report 2003/064. Retrieved from https://eprint.iacr.org/2003/064

Rogaway, P. (2004). "Efficient Instantiations of Tweakable Blockciphers and Refinements to Modes OCB and PMAC." In Fast Software Encryption (pp. 16-35). Berlin, Heidelberg: Springer.

Rogaway, P., & Bellare, M. (2006). "Robust Computational Secret Sharing and a Unified Account of Classical Cryptographic Primitives." In Advances in Cryptology - CRYPTO 2006 (pp. 232-249). Berlin, Heidelberg: Springer.

Rogaway, P., & Bellare, M. (2015). "Introduction to Modern Cryptography." Retrieved from https://web.cs.ucdavis.edu/~rogaway/papers/sym-intro.pdf

Rogaway, P., & Shrimpton, T. (2006). "Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance." In Fast Software Encryption (pp. 371-388). Berlin, Heidelberg: Springer.

Rogaway, P., & Shrimpton, T. (2008). "Ciphers with Arbitrary Finite Domains." In Advances in Cryptology - CRYPTO 2008 (pp. 440-457). Berlin, Heidelberg: Springer.

Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. New York, NY: John Wiley & Sons.

Shamir, A. (1979). "How to Share a Secret." Communications of the ACM, 22(11), 612-613.

Shamir, A. (1997). "Visual Cryptography." In Advances in Cryptology - EUROCRYPT 1996 (pp. 1-12). Berlin, Heidelberg: Springer.

Shor, P. W. (1994). "Algorithms for Quantum Computation: Discrete Logarithms and Factoring." In Proceedings of the 35th Annual Symposium on Foundations of Computer Science (FOCS '94) (pp. 124-134). Washington, DC: IEEE Computer Society Press.

Smart, N. P. (2003). Cryptography: An Introduction (2nd ed.). London, UK: McGraw-Hill.

Smart, N. P. (2005). "A Universal One-Way Hash Function Construction Based on Non-interactive Zero-Knowledge Proofs." In Advances in Cryptology - EUROCRYPT 2005 (pp. 505-524). Berlin, Heidelberg: Springer.

Stallings, W. (2020). Cryptography and Network Security: Principles and Practice. Boston, MA: Pearson.

van Oorschot, P. C., & Wiener, M. J. (1996). "A Known-plaintext Attack on Two-key Triple Encryption." In Advances in Cryptology - CRYPTO '90 (pp. 318-325). Berlin, Heidelberg: Springer.

Van Tilborg, H. C. A., & Jajodia, S. (2014). Encyclopedia of Cryptography and Security (2nd ed.). New York, NY: Springer.

TEKNOLOGI KRIPTOGRAFI MODERN

Dr. Joseph Teguh Santoso, S.Kom, M.Kom.

BIODATA PENULIS

Dr. Joseph Teguh Santoso, M.Kom adalah pemimpin yang visioner dan praktisi industri berpengalaman, yang menjabat sebagai Rektor Universitas Sains dan Teknologi Komputer (Universitas STEKOM), salah satu universitas terkemuka di Jawa Tengah, Indonesia. Dengan pengalaman lebih dari 13 tahun di dunia bisnis dan praktisi industri di China, beliau membawa perspektif global dan inovasi yang signifikan ke dalam dunia akademis.Sebagai seorang entrepreneur, penulis adalah pencipta TopLoker.com, sebuah platform inovatif yang merevolusi cara mencari dan menawarkan pekerjaan. TopLoker.com adalah portal lowongan bursa kerja terbesar di Indonesia, khusus untuk pendidikan SMA/SMK sederajat. TopLoker.com telah mendapatkan penghargaan sebagai juara 1 Startup4industry 2022 oleh Kementerian Perindustrian Republik Indonesia. Kontribusi Dr. Joseph dalam menyediakan akses pekerjaan yang luas bagi lulusan SMA/SMK telah membantu banyak individu menemukan peluang kerja yang sesuai dengan keahlian mereka. Selain itu, Dr. Joseph Teguh Santoso, M.Kom adalah pendiri dari dua organisasi yaitu (1) organisasi guru/pendidik PTIC (Perkumpulan Teacherpreneur Indonesia Cerdas) yang bertujuan untuk meningkatkan kualitas pendidikan dan kesejahteraan guru/pendidik dengan wawasan entrepreneurship, serta (2) organisasi industri PERKIVI (Perkumpulan Komunitas Industri dan Vokasi Indonesia) yang berfokus pada pengembangan link and match antara industri dan dunia pendidikan. Sebagai Rektor, Dr. Joseph Teguh Santoso, M.Kom memiliki kepemimpinan yang berorientasi pada hasil, dan berkomitmen untuk mendorong kemajuan Universitas Sains dan Teknologi Komputer (Universitas STEKOM). Saat ini Universitas STEKOM telah mengalami transformasi positif dalam peningkatan kualitas pendidikan, perluasan fasilitas, serta penguatan kemitraan Perguruan Tinggi Nasional dan Internasional. Beliau memprioritaskan pengembangan sumber daya manusia dan penelitian, serta memastikan bahwa universitas berada di garis depan dalam inovasi dan teknologi untuk mencapai tujuan akhir, yaitu lulusan yang mampu bekerja dan sukses setelah lulus.Dr. Joseph Teguh Santoso, M.Kom sering diundang sebagai pembicara di berbagai konferensi nasional maupun internasional dan telah menerima berbagai penghargaan atas dedikasinya dalam bidang pendidikan, industri, dan kewirausahaan.

YAYASAN PRIMA AGUS TEKNIK

PENERBIT :
YAYASAN PRIMA AGUS TEKNIK
Jl. Majapahit No. 605 Semarang
Telp. (024) 6723456. Fax. 024-6710144
Email : penerbit_ypat@stekom.ac.id

ISBN 978-623-8642-17-5 (PDF)
9 786238 642175