Bab 16

Kriptografi Public Key

Dalam kriptografi klasik (simetris), jika seseorang mengetahui cara mengenkripsi naskah asli menjadi naskah acak, maka orang tersebut juga mengetahui cara mendekripsi naskah acak yang dihasilkan. Demikian juga jika seseorang mengetahui cara mendekripsi naskah acak, maka orang tersebut juga mengetahui cara mengenkripsi naskah asli untuk menghasilkan naskah acak.

Sekitar pertengahan tahun 1970an, muncul konsep baru dalam kriptografi yaitu kriptografi *public key* (asimetris). Seseorang yang mengetahui cara mengenkripsi naskah asli belum tentu mengetahui juga cara mendekripsi naskah acak yang dihasilkan. Demikian juga seseorang yang mengetahui cara mendekripsi naskah acak belum tentu mengetahui juga cara mengenkripsi naskah asli untuk menghasilkan naskah acak tersebut. Enkripsi dan dekripsi dalam kriptografi *public key* menggunakan sepasang kunci yaitu kunci publik (*public key*) dan kunci privat (*private key*). Naskah yang telah dienkripsi menggunakan kunci privat hanya dapat didekripsi menggunakan kunci publik dan naskah yang dapat didekripsi menggunakan kunci publik dapat dipastikan telah dienkripsi menggunakan kunci privat. Sebaliknya, naskah yang telah dienkripsi menggunakan kunci publik hanya dapat didekripsi menggunakan kunci privat. Mekanisme ini memungkinkan berbagai aplikasi, dua yang terpenting diantaranya adalah distribusi kunci sesi dan tanda tangan digital (*digital signature*).

Jika A ingin mengirim kunci sesi atau rahasia lainnya ke B dan hanya ingin B yang dapat membacanya, maka A mengenkripsi rahasia tersebut menggunakan kunci publik milik B. Dengan asumsi hanya B yang memiliki kunci privat B, maka hanya B yang dapat mendekripsi rahasia yang telah dienkripsi tersebut. Cara inilah yang kerap digunakan untuk mendistribusikan kunci sesi menggunakan kriptografi *public key*.

Jika A ingin menanda-tangan suatu naskah secara digital, maka A mengen-

kripsi naskah tersebut menggunakan kunci privat miliknya dan hasil enkripsi merupakan "tanda tangan," Jika seseorang (sebut saja B) ingin memeriksa apakah naskah tersebut telah ditanda-tangan oleh A, maka B mendekripsi "tanda tangan" tersebut dengan kunci publik milik A dan membandingkan hasil dekripsi dengan naskah yang ditanda-tangan. Jika sama maka B dapat meyakinkan dirinya sendiri bahwa A telah menanda-tangan naskah tersebut karena hanya A yang memiliki kunci privat yang digunakan untuk mengenkripsi naskah untuk menghasilkan "tanda-tangan." Dalam prakteknya, yang dienkripsi bukan naskah penuh melainkan *digest* dari naskah tersebut (lihat bab 9).

Jadi dalam kriptografi *public key*, kunci publik dapat disebar-luaskan kepada umum dan sebaiknya disebar luaskan. Sebaliknya, kunci privat harus dirahasiakan oleh pemiliknya.

Dalam bab ini akan dibahas empat sistem kriptografi *public key* yang di buku ini dianggap sebagai empat yang terpenting yaitu RSA, Diffie-Hellman, DSA dan ElGamal. Keamanan RSA mengandalkan sukarnya menguraikan bilangan yang sangat besar, sedangkan keamanan tiga sistem lainnya mengandalkan sukarnya mencari logaritma diskrit. Selain itu kriptografi *knapsack* juga dibahas karena merupakan bagian dari sejarah. Protokol *zero-knowledge* dibahas karena potensinya untuk aplikasi identifikasi.

16.1 RSA

Tahun 1978, Len Adleman, Ron Rivest dan Adi Shamir mempublikasikan sistem RSA (lihat [adl78]). Semula sistem ini dipatenkan di Amerika Serikat dan seharusnya masa paten habis tahun 2003, akan tetapi RSA Security melepaskan hak paten setelah 20 September 2000. Sebetulnya sistem serupa telah dilaporkan oleh Clifford Cocks tahun 1973 meskipun informasi mengenai ini baru dipublikasi tahun 1997 karena merupakan hasil riset yang diklasifikasikan sangat rahasia oleh pemerintah Britania Raya (Clifford Cocks bekerja untuk GCHQ, suatu badan di Britania Raya yang fungsinya serupa dengan fungsi NSA di Amerika Serikat), jadi validitas paten patut dipertanyakan karena adanya *prior art*.

Kita jelaskan secara garis besar bagaimana cara kerja RSA. Setiap pengguna memilih, menggunakan *random number generator*, dua bilangan prima yang sangat besar p dan q (masing-masing lebih dari 200 digit). Untuk produk $n = pq$, jika p dan q diketahui, fungsi Euler dapat dengan mudah dikomputasi yaitu

$$\phi(n) = (p-1)(q-1).$$

Kemudian pengguna memilih, menggunakan *random number generator*, suatu bilangan e antara 1 dan $\phi(n)$ yang koprima dengan $\phi(n)$. Berikutnya pengguna

mengkomputasi *inverse* dari e modulo $\phi(n)$:

$$d \equiv e^{-1} \pmod{\phi(n)}.$$

Ini dapat dilakukan menggunakan *extended Euclidean algorithm* (lihat pembahasan kalkulasi *inverse* modulo bilangan yang koprima menggunakan *extended Euclidean algorithm* di bagian 3.5). Pengguna kemudian mempublikasi kunci publiknya

$$K_E = (n, e),$$

dan merahasiakan kunci privatnya

$$K_D = (n, d).$$

Rumus untuk mengenkripsi atau mendekripsi menggunakan kunci publik adalah

$$M^e \bmod n$$

dimana M adalah representasi naskah asli (menggunakan bilangan bulat) jika mengenkripsi, atau representasi naskah acak jika mendekripsi. Rumus untuk mengenkripsi atau mendekripsi menggunakan kunci privat adalah

$$M^d \bmod n$$

dimana M adalah representasi naskah asli jika mengenkripsi, atau representasi naskah acak jika mendekripsi. Tidak terlalu sulit untuk melihat bahwa naskah yang dienkripsi menggunakan kunci publik dapat didekripsi menggunakan kunci privat:

$$(M^e)^d \equiv M^{ed} \equiv M \pmod{n}.$$

Jika $\gcd(M, n) = 1$, maka menggunakan teorema 32 kita dapatkan

$$\begin{aligned} (M^e)^d &\equiv M^{ed} \\ &\equiv M^{\phi(n)+1} \\ &\equiv M^{\phi(n)} M \\ &\equiv M \pmod{n}. \end{aligned}$$

Untuk $\gcd(M, n) > 1$ dimana M bukan kelipatan n, ini hanya bisa terjadi jika M merupakan kelipatan p atau kelipatan q, tetapi bukan kelipatan keduanya. Jika M merupakan kelipatan p, maka

$$M^{ed} \equiv 0 \pmod{p}$$

dan

$$M^{ed} \equiv M^{\phi(p)\phi(q)+1} \equiv M^{\phi(p)\phi(q)} M \equiv M \pmod{q}.$$

Menggunakan *Chinese Remainder Theorem*, dengan $n = pq$, kita dapatkan

$$M^{ed} \equiv M \pmod{n}.$$

Untuk M kelipatan q, hal serupa dapat ditunjukkan. Untuk M kelipatan n, kita dapatkan

$$(M^e)^d \equiv 0 \equiv M \pmod{n}$$

Jadi

$$(M^e)^d \equiv M \pmod{n}$$

untuk sembarang M.

Naskah yang dienkripsi menggunakan kunci privat dapat didekripsi menggunakan kunci publik:

$$(M^d)^e \equiv M^{de} \equiv M \pmod{n}.$$

Secara umum, jika $f \not\equiv d \pmod{\phi(n)}$ maka

$$(M^e)^f \not\equiv M \pmod{n},$$

yang berarti sesuatu yang dienkripsi menggunakan kunci publik tidak dapat didekripsi selain menggunakan kunci privat. Juga, jika $f \not\equiv e \pmod{\phi(n)}$ maka secara umum

$$(M^d)^f \not\equiv M \pmod{n},$$

yang berarti sesuatu yang dienkripsi menggunakan kunci privat tidak dapat didekripsi selain menggunakan kunci publik.

Keamanan dari RSA bersandar pada fakta bahwa mengetahui n dan d secara umum tidak membantu untuk mencari e yaitu *inverse* modulo $\phi(n)$ dari d. Hal ini karena mengetahui n tidak membantu mencari $\phi(n)$ jika n tidak bisa diuraikan menjadi

$$n = pq.$$

Untuk menjaga keamanan tersebut, ada beberapa hal yang perlu diperhatikan dalam memilih p dan q:

- Nilai p harus cukup jauh dari nilai q. Sebaiknya panjang dari p harus berbeda beberapa digit dari q. Jika nilai p terlalu dekat dengan nilai q, maka *Fermat factorization* dapat digunakan untuk menguraikan $n = pq$ (lihat bagian 14.2).

- Sebaiknya $\gcd(p-1, q-1)$ tidak terlalu besar.

- Sebaiknya $p-1$ dan $q-1$ mempunyai faktor prima yang besar.

Karena perkembangan yang pesat dalam teknik penguraian, kunci RSA sebaiknya minimal 2048 bit.

RSA dapat digunakan, baik untuk *key distribution* (termasuk *key exchange*), maupun untuk *digital signature*. Karena merupakan sistem pertama yang dapat digunakan untuk *key distribution* dan *digital signature*, RSA menjadi sistem kriptografi *public key* yang terpopuler. Boleh dikatakan semua standard protokol kriptografi memperbolehkan penggunaan RSA, termasuk SSL/TLS (untuk pengamanan http) dan SSH (*secure shell*).

Pembahasan RSA diatas tidak menjelaskan standard implementasi secara rinci, termasuk format data dan kunci. Untuk yang ingin mengetahui standard RSA dengan lebih rinci dipersilahkan membaca berbagai publikasi RSA Laboratories (bagian dari RSA Security) dalam seri yang berjudul *Public-Key Cryptography Standards* (PKCS).

16.2 Diffie-Hellman

Walaupun Diffie-Hellman adalah sistem kriptografi *public key* yang pertama, Diffie-Hellman tidak sepopuler RSA karena hanya dapat digunakan untuk *key agreement*. Menggunakan Diffie-Hellman, dua pengguna, sebut saja A dan B, dapat membuat kunci rahasia yang hanya diketahui oleh A dan B, meskipun komunikasi antara A dan B dapat dilihat semua orang.

Diffie-Hellman menggunakan *finite field* $\mathbf{GF}(q)$ yang sangat besar. A dan B keduanya mengetahui $\mathbf{GF}(q)$ dan elemen $g \in \mathbf{GF}(q)^*$. $\mathbf{GF}(q)$ dan g tidak perlu dirahasiakan, jadi boleh saja diketahui semua orang. Meskipun tidak harus, g sebaiknya merupakan *generator* untuk $\mathbf{GF}(q)^*$, atau setidaknya memiliki *order* yang besar agar *range* untuk pembuatan kunci cukup besar. Diffie-Hellman bekerja sebagai berikut:

- A memilih, menggunakan *random number generator*, a, mengkomputasi $g^a \in \mathbf{GF}(q)^*$, dan mengirim g^a ke B.
- B melakukan hal yang serupa, yaitu memilih, menggunakan *random number generator*, b, mengkomputasi $g^b \in \mathbf{GF}(q)^*$, dan mengirim g^b ke A.
- Setelah menerima g^b, A mengkomputasi kunci rahasia $k = (g^b)^a = g^{ab} \in \mathbf{GF}(q)^*$.
- B, setelah menerima g^a, mengkomputasi kunci rahasia $k = (g^a)^b = g^{ab} \in \mathbf{GF}(q)^*$.

Setelah selesai, A dan B mengetahui kunci rahasia $k = g^{ab}$, akan tetapi orang lain tidak bisa mendapatkan $k = g^{ab}$ meskipun mengetahui g, g^a, dan g^b. Ini didasarkan pada asumsi bahwa untuk mendapatkan g^{ab}, orang lain harus

mengkomputasi logaritma diskrit dari g^a atau g^b untuk mendapatkan a atau b terlebih dahulu, dan komputasi logaritma diskrit terlalu sukar.

Satu catatan yang perlu diperhatikan adalah jika *finite field* yang digunakan adalah $\mathbf{GF}(p^n)$ dimana p tentunya adalah bilangan prima dan $n > 1$. Setiap bilangan bulat x dimana $0 \leq x < p^n$ perlu dipetakan ke $\mathbf{GF}(p^n)$. Karena setiap elemen dalam $\mathbf{GF}(p^n)$ dapat dianggap sebagai n-tuple, maka ada pemetaan yang *bijective* antara $\mathbf{GF}(p^n)$ dengan $\mathbf{Z}/p^n\mathbf{Z}$ karena setiap n-tuple dapat diinterpretasikan sebagai suatu bilangan dimana setiap koordinat merepresentasikan suatu digit dengan basis p. Akan tetapi $\mathbf{GF}(p^n)$ dan $\mathbf{Z}/p^n\mathbf{Z}$ mempunyai struktur yang berbeda karena yang pertama adalah suatu *finite field* sedangkan yang kedua adalah suatu *ring* dimana setiap kelipatan p tidak mempunyai *inverse*. Jadi kita tidak bisa begitu saja menggunakan aritmatika modular bilangan bulat untuk aritmatika $\mathbf{GF}(p^n)$.

$\mathbf{GF}(2^n)$ banyak digunakan sebagai *finite field* untuk Diffie-Hellman karena komputasi dalam $\mathbf{GF}(2^n)$ sangat elegan dan mudah diprogram. Akan tetapi, karena logaritma diskrit lebih mudah dikomputasi untuk $\mathbf{GF}(2^n)^*$ dibandingkan $\mathbf{GF}(p)^*$, dimana p merupakan bilangan prima ganjil yang besarnya hampir sama dengan 2^n, maka n harus dipilih sangat besar (lebih dari 2000). Buku ini merekomendasikan penggunaan $\mathbf{GF}(p)$.

16.3 DSA

DSA (*Digital Signature Algorithm*) adalah salah satu algoritma yang digunakan dalam DSS (*Digital Signature Standard*), standard untuk *digital signature* yang dibuat oleh FIPS. DSS juga memperbolehkan penggunaan RSA. Karena DSS mewajibkan penggunaan SHA-1 (lihat bagian 9.2), maka DSA atau RSA digunakan untuk mengenkripsi *digest* sebesar 160 bit.

Parameter yang digunakan oleh DSA adalah sebagai berikut:

- Suatu bilangan prima p yang dipilih menggunakan *random number generator* minimum 512 bit, sebaiknya 1024 bit.

- Suatu bilangan prima q yang dipilih menggunakan *random number generator* sebesar 160 bit dimana q membagi $p - 1$. Untuk implementasi, mungkin lebih mudah untuk memilih q terlebih dahulu kemudian memilih p dimana $p \equiv 1 \pmod{q}$.

- Suatu bilangan $g \in \mathbf{GF}(p)^*$ yang mempunyai *order* q yang dipilih sebagai berikut: Pilih bilangan g_0 menggunakan *random number generator* dimana $1 < g_0 < p - 1$ lalu komputasi $g = g_0^{(p-1)/q} \bmod p$. Jika $g > 1$ maka itulah g yang dipilih. Jika tidak maka pilih g_0 yang lain sampai kita dapatkan $g > 1$.

- Suatu bilangan x yang dipilih menggunakan *random number generator* dimana $0 < x < q$.
- Bilangan $y = g^x \bmod p$.
- Suatu bilangan k yang dipilih menggunakan *random number generator* dimana $0 < k < q$. Setiap kali menanda-tangan, k yang baru harus dipilih.

Parameter p, q, g, x dan y merupakan parameter semi-permanen, sedangkan k yang baru harus dipilih setiap kali menanda-tangan. Parameter p, q, g dan y merupakan parameter publik, jadi harus disebar-luaskan, sedangkan parameter x dan k harus dirahasiakan.

Berikut adalah cara DSA membuat *digital signature* untuk naskah M menggunakan parameter diatas. Hasil pembuatan *digital signature* adalah sepasang bilangan bulat (r, s) dimana $0 \leq r, s < q$. Rumus untuk r dan s adalah sebagai berikut:

$$\begin{aligned} r &= (g^k \bmod p) \bmod q, \\ s &= (k^{-1}(\text{SHA-1}(M) + xr) \bmod q, \end{aligned}$$

dimana SHA-1(M) adalah hasil komputasi *digest* menggunakan SHA-1 (lihat bagian 9.2). Jika ternyata $r = 0$ atau $s = 0$ (kemungkinannya sangat kecil), maka pembuatan *digital signature* dapat diulang menggunakan nilai k yang lain.

Seseorang yang ingin memeriksa *digital signature* akan mendapatkan M, r dan s. Pertama r dan s diperiksa nilainya apakah $0 < r < q$ dan $0 < s < q$. Jika tidak maka *digital signature* ditolak. Pemeriksa kemudian mengkomputasi:

$$\begin{aligned} w &= s^{-1} \bmod q, \\ u_1 &= \text{SHA-1}(M)w \bmod q, \\ u_2 &= rw \bmod q, \text{ dan} \\ v &= (g^{u_1} y^{u_2} \bmod p) \bmod q. \end{aligned}$$

Jika $v = r$ maka pemeriksa dapat cukup yakin bahwa M telah ditanda-tangan menggunakan kunci pasangan dari y. Mari kita buktikan bahwa jika semua komputasi dilakukan sesuai dengan aturan DSA maka $v = r$. Untuk itu kita buat $M' = \text{SHA-1}(M)$, jadi

$$\begin{aligned} s &= (k^{-1}(M' + xr) \bmod q, \\ u_1 &= M'w \bmod q. \end{aligned}$$

Semua *inverse* dalam pembuktian adalah dalam $\mathbf{GF}(q)$. Menggunakan rumus komputasi kita dapatkan

$$v = (g^{u_1} y^{u_2} \bmod p) \bmod q$$

$$\begin{aligned} &= (g^{M'w} y^{rw} \bmod p) \bmod q \\ &= (g^{M'w} g^{xrw} \bmod p) \bmod q \\ &= (g^{(M'+xr)w} \bmod p) \bmod q. \end{aligned}$$

Karena

$$s = k^{-1}(M' + xr) \bmod q,$$

maka

$$w = k(M' + xr)^{-1} \bmod q,$$

jadi

$$(M' + xr)w \bmod q = k \bmod q.$$

Kembali ke v:

$$\begin{aligned} v &= (g^k \bmod p) \bmod q \\ &= r. \end{aligned}$$

Selesailah pembuktian kita.

Suatu hal yang menarik dengan DSA adalah fungsi dari parameter k. Setiap kali menanda-tangan, nilai k yang baru dipilih secara acak. Walaupun naskah yang ditanda-tangan sama, jika ditanda-tangan dua kali oleh pemilik a, tanda tangannya berbeda. Jadi k berfungsi seperti *initialization vector*.

16.4 ElGamal

Sistem kriptografi ElGamal menjadi populer ketika pengembang *open source software* untuk kriptografi mencari alternatif dari RSA yang ketika itu patennya masih berlaku. Seperti halnya dengan Diffie-Hellman dan DSA, keamanan ElGamal didasarkan atas sukarnya mengkomputasi logaritma diskrit. Akan tetapi berbeda dengan Diffie-Hellman yang khusus dirancang untuk *key agreement* dan DSA yang khusus dirancang untuk *digital signature*, ElGamal lebih seperti RSA karena fungsinya untuk enkripsi umum. Kita akan bahas ElGamal untuk enkripsi terlebih dahulu, kemudian dilanjutkan dengan pembahasan versi ElGamal untuk *digital signature*.

ElGamal menggunakan *finite field* $\mathbf{GF}(q)$ yang besar. Untuk memudahkan pembahasan, $q = p$, suatu bilangan prima yang sangat besar. Jika $q = p^n$ dimana $n > 1$ maka dalam pembahasan p harus diganti dengan q dan aritmatika yang digunakan bukan sekedar aritmatika modular bilangan bulat. Jadi disini *finite field* yang digunakan adalah $\mathbf{GF}(p)$.

Suatu $g \in \mathbf{GF}(p)^*$ dipilih yang sebaiknya adalah suatu *generator* untuk $\mathbf{GF}(p)^*$. Jika bukan *generator* maka sebaiknya g mempunyai *order* yang sangat besar dalam $\mathbf{GF}(p)^*$. Untuk membuat kunci privat, pengguna memilih,

menggunakan *random number generator*, a, dimana $0 < a < p-1$. Kunci publik pasangan a adalah $g^a \in \mathbf{GF}(p)^*$. Tentunya kunci publik perlu disebar-luaskan. Seseorang yang mengetahui kunci publik tidak bisa mendapatkan kunci privat (a) tanpa mengkomputasi logaritma diskrit dari g^a.

Seseorang yang ingin mengenkripsi suatu naskah P (representasi P menggunakan bilangan bulat harus lebih kecil dari p) untuk hanya dapat didekripsi oleh pemilik a memilih, menggunakan suatu *random number generator*, k, dan mengirim pasangan

$$(g^k, Pg^{ak})$$

kepada pemilik a. Ini dapat dilakukan karena pengirim mengetahui g, g^a, P dan k. Seperti halnya dengan DSA, k disini berfungsi sebagai *initialization vector*. Pemilik a dapat mendekripsi kiriman sebagai berikut:

- Mengkomputasi $(g^k)^{(p-1-a)} = g^{-ak}$. Ini dapat dilakukan karena pemilik a mengetahui g^k, p dan a.
- Mengalikan Pg^{ak} dengan g^{-ak} untuk mendapatkan P.

Jadi Pg^{ak} seolah naskah P menggunakan topeng atau *mask*, dan mengalikannya dengan g^{-ak} membuka topeng tersebut.

Satu hal yang menarik dengan ElGamal adalah kemampuan untuk *reencryption*. Jika g dan g^a diketahui, seseorang dapat membuat r secara acak dan mentransformasi pasangan (g^k, Pg^{ak}) menjadi $(g^{(k+r)}, Pg^{a(k+r)})$ (kalikan g^k dengan g^r dan Pg^{ak} dengan g^{ar}). Jadi *initialization vector* telah diganti dari k menjadi $k+r$, tanpa harus mengetahui k. Ini dapat digunakan untuk anonimitas, karena tanpa mengetahui a atau r, pasangan $(g^{(k+r)}, Pg^{a(k+r)})$ tidak dapat dikaitkan dengan (g^k, Pg^{ak}). Aplikasi anonimitas contohnya untuk *electronic voting*.

Seperti halnya dengan RSA, ElGamal dapat digunakan untuk *digital signature*. Berikut adalah cara pemilik a menanda-tangan suatu naskah S dimana representasi S menggunakan bilangan bulat lebih kecil dari $p-1$:

- Pilih, menggunakan *random number generator*, k yang koprima dengan $p-1$ ($\gcd(k, p-1) = 1$). Lakukan komputasi $r = g^k \bmod p$.
- Cari solusi untuk x dimana $g^S \equiv g^{ar} r^x \pmod{p}$.
- *Digital signature* untuk S adalah pasangan (r, x).

Solusi untuk x dapat dicari oleh pemilik a karena

$$\begin{aligned} g^S &\equiv g^{ar} r^x \pmod{p} \\ &\equiv g^{ar} g^{kx} \pmod{p} \\ &\equiv g^{ar+kx} \pmod{p}. \end{aligned}$$

Ini berarti

$$S \equiv ar + kx \pmod{p-1},$$

jadi

$$x = k^{-1}(S - ar) \bmod p - 1,$$

dimana *inverse* dikalkulasi dalam aritmatika modulo $p-1$. Seseorang yang ingin memeriksa tanda tangan diatas cukup memeriksa bahwa

$$g^S \equiv g^{ar} r^x \pmod{p}.$$

Ini dapat dilakukan oleh pemeriksa karena ia mengetahui g, S, g^a, r dan x.

16.5 Knapsack

Tahun 1970an banyak riset yang dilakukan guna menemukan mekanisme untuk kriptografi *public key*. Selain kriptografi yang berbasis pada sukarnya penguraian dan kriptografi yang berbasis pada sukarnya logaritma diskrit, ada juga yang mencoba membuat sistem kriptografi *public key* yang berbasis pada sukarnya mencari solusi untuk *knapsack problem*, dipelopori oleh Merkle dan Hellman.

Secara informal, *knapsack problem* adalah masalah bagaimana mengisi suatu *knapsack* yang mempunyai kapasitas tertentu dengan benda-benda dari sekumpulan yang mempunyai ukuran berbeda-beda sehingga *knapsack* terisi penuh sesuai kapasitas. Tidak semua benda dari kumpulan perlu dimasukkan kedalam *knapsack*. *Knapsack problem* dapat didefinisikan secara formal sebagai berikut.

Definisi 50 (Knapsack Problem) *Jika* $\{v_0, v_1, \ldots, v_{k-1}\}$ *adalah himpunan dengan* k *elemen, dimana setiap elemen* $v_i \in \mathbf{Z}, v_i > 0$*, dan* $V \in \mathbf{Z}, V > 0$*, cari solusi* $\{d_0, d_1, \ldots, d_{k-1}\}$ *dimana setiap* $d_i \in \{0, 1\}$ *dan*

$$\sum_{i=0}^{k-1} d_i v_i = V.$$

Bisa saja suatu *knapsack problem* tidak mempunyai solusi, atau ada solusi yang unik, atau ada lebih dari satu solusi. Secara umum *knapsack problem* adalah masalah yang tergolong *NP-complete* (bersama dengan *travelling salesman problem*), jadi terlalu sukar untuk dikomputasi. Namun ada jenis *knapsack problem* yang dapat dipecahkan dengan efisien yaitu *super-increasing knapsack problem* dimana, deretan $v_0, v_1, \ldots, v_{k-1}$ dapat diurutkan sehingga untuk setiap v_i:

$$v_i > \sum_{j=0}^{i-1} v_j.$$

Sebagai contoh, $2, 3, 7, 15, 31$ adalah deretan yang *super-increasing*. Untuk deretan yang *super-increasing* ada algoritma berikut untuk memecahkan *knapsack problem* dengan efisien.

1. $W \leftarrow V, j \leftarrow k$.
2. $j \leftarrow j - 1, d_j \leftarrow 0$.
3. Jika $v_j \leq W$ maka $d_j \leftarrow 1, W \leftarrow W - v_j$.
4. Jika $j > 0$ dan $W > 0$ kembali ke langkah 2.
5. Sukses jika $W = 0$ dan gagal jika $W > 0$.

Untuk deretan $2, 3, 7, 15, 31$ dan $V = 24$ kita dapatkan sukses dengan $d_0 = 1, d_1 = 0, d_2 = 1, d_3 = 1, d_4 = 0$.

Dasar dari kriptografi *knapsack* seperti Merkle-Hellman adalah

- untuk yang mengetahui kunci privat, esensi dekripsi adalah memecahkan *super-increasing knapsack problem*, sesuatu yang mudah; sedangkan
- untuk yang tidak mengetahui kunci privat, esensi dekripsi adalah memecahkan *knapsack problem* secara umum, sesuatu yang sangat sukar.

Berikut dijelaskan sistem Merkle-Hellman (lihat juga [hel78]). Dalam sistem Merkle-Hellman, nilai parameter k sama dengan banyaknya bit dalam unit naskah.

Untuk keperluan pasangan kunci, dipilih secara acak suatu deretan yang *super-increasing* $v_0, v_1, \ldots, v_{k-1}$ dan suatu bilangan bulat m dimana

$$\sum_{i=0}^{k-1} v_i < m.$$

Ini dapat dilakukan misalnya dengan memilih, menggunakan *random number generator*, sederetan bilangan bulat positif $z_0, z_1, \ldots, z_k$, lalu membuat

$$\begin{aligned} v_0 &\leftarrow z_0, \\ v_i &\leftarrow z_i + \sum_{j=0}^{i-1} v_j, \text{ untuk } 1 \leq i < k, \\ m &\leftarrow z_k + \sum_{i=0}^{k-1} v_i. \end{aligned}$$

Langkah selanjutnya adalah memilih suatu bilangan bulat a secara acak, dimana $0 < a < m$ dan $\gcd(a, m) = 1$. Ini dapat dilakukan dengan memilih,

menggunakan *random number generator*, suatu bilangan bulat a', dimana $0 < a' < m$ dan kemudian membuat a sebagai bilangan terkecil yang mematuhi $a' \leq a$ dan $\gcd(a, m) = 1$. Setelah itu dibuat b:

$$b = a^{-1} \bmod m$$

dimana *inverse* adalah dalam aritmatika modulo m. Kunci publik K_E adalah $(w_0, w_1, \ldots, w_{k-1})$ dimana

$$w_i = av_i \bmod m.$$

Kunci privat K_D adalah $(b, m, v_0, v_1, \ldots, v_{k-1})$.

Untuk mengenkripsi naskah $P = (p_0, p_1, \ldots, p_{k-1})$ dimana $p_i \in \{0, 1\}$, seseorang yang mengetahui kunci publik mengkomputasi

$$C = \sum_{i=0}^{k-1} p_i w_i.$$

Untuk mendekripsi C, seseorang yang mengetahui kunci privat melakukan komputasi

$$V = bC \bmod m.$$

Kita dapatkan

$$\begin{aligned} V &= bC \bmod m \\ &= (b \sum_{i=0}^{k-1} p_i w_i) \bmod m \\ &= (\sum_{i=0}^{k-1} p_i b_i w_i) \bmod m \\ &= (\sum_{i=0}^{k-1} p_i v_i) \bmod m \text{ (karena } b_i w_i \equiv v_i \pmod m) \\ &= \sum_{i=0}^{k-1} p_i v_i \text{ (karena } \sum_{i=0}^{k-1} p_i v_i < m). \end{aligned}$$

Karena $V = \sum_{i=0}^{k-1} p_i v_i$ maka $(p_0, p_1, \ldots, p_{k-1})$ dapat dicari secara efisien menggunakan algoritma untuk *super-increasing knapsack problem*.

Untuk yang hanya mengetahui kunci publik dan tidak mengetahui kunci privat, mencari $(p_0, p_1, \ldots, p_{k-1})$ dari $C = \sum_{i=0}^{k-1} p_i w_i$ adalah memecahkan secara umum *knapsack problem*, sesuatu yang sangat sukar. Itulah asumsi semula. Namun karena *knapsack problem* ini merupakan transformasi yang relatif sederhana dari suatu *super-increasing knapsack problem*, Adi Shamir berhasil

menemukan cara memecahkan Merkle-Hellman dalam *polynomial-time* (lihat [sha82]). Meskipun ada usaha memperbaiki Merkle-Hellman untuk mendapatkan sesuatu yang benar-benar *knapsack problem* umum, minat pada kriptografi *knapsack* surut setelah itu.

16.6 Zero-Knowledge Protocol

Zero-knowledge protocol agak berbeda dengan jenis-jenis kriptografi yang sudah dibahas. Meskipun semua jenis kriptografi mengandung unsur probabilitas, untuk *zero-knowledge protocol*, probabilitas berperan langsung dalam mekanismenya.

Secara garis besar, *zero-knowledge protocol* adalah mekanisme dimana seorang dengan rahasia tertentu, sebut saja Peggy, dapat meyakinkan pengujinya, sebut saja Victor, bahwa Peggy mengetahui rahasia itu, tanpa membuka rahasia tersebut kepada Victor atau orang lain. Tentu saja rahasia itu tidak bisa sembarang rahasia, tetapi rahasia yang mempunyai konsekuensi yang dapat diperiksa oleh Victor.

Contoh yang sering digunakan untuk menjelaskan konsep *zero-knowledge protocol* biasanya menggunakan gua atau terowongan yang bercabang. Versi disini, panjang terowongan sekitar 300m, dan 100m setelah pintu masuk, terowongan bercabang dua. Kedua cabang berjalan paralel dan pintu keluar keduanya berdekatan. Sekitar 5m dari pintu keluar terdapat suatu pintu yang menghubungkan kedua cabang yang hanya dapat dibuka oleh seseorang yang mengetahui suatu kode rahasia. Peggy ingin meyakinkan Victor bahwa ia (Peggy) mengetahui kode rahasia tersebut, tanpa memberi petunjuk kepada Victor apa kode rahasia tersebut. Victor sendiri tidak mengetahui kode rahasia tersebut. Peggy masuk kedalam terowongan lalu memilih satu cabang. Saat Peggy tiba di pintu antara kedua cabang, yaitu 5m sebelum pintu keluar, ia memberi kabar kepada Victor menggunakan telpon genggam. Victor kemudian menyatakan dari cabang mana Peggy harus keluar. Karena Peggy hanya diberi waktu 10 detik untuk keluar, jika ia berada di cabang yang salah maka tidak mungkin ia kembali ketempat dimana terowongan bercabang lalu masuk cabang yang benar. Ia harus membuka pintu dengan kode rahasia dan melewatinya. Sebelum 10 detik habis, Peggy keluar dari cabang yang diminta oleh Victor. Tentunya jika ini hanya dilakukan satu kali Victor belum tentu puas karena ada kemungkinan Peggy sebetulnya tidak mengetahui kode rahasia tetapi beruntung memilih cabang yang benar. Victor dapat meminta eksperimen ini diulang beberapa kali, setiap kali ia pastikan sebelumnya bahwa pintu antara cabang benar-benar tertutup, dan memilih secara acak lewat cabang mana Peggy harus keluar. Jika eksperimen ini diulang n kali dan Peggy selalu keluar dari cabang yang diminta, maka probabilitas bahwa Peggy tidak mengetahui kode rahasia adalah $(1/2)^n$. Contohnya, jika $n = 10$ probabilitas ini adalah

1/1024, jadi Victor dapat 99.9 persen yakin bahwa Peggy mengetahui kode rahasia, dan Victor sendiri tetap tidak mengetahui kode rahasia.

Konsep *zero-knowledge protocol* digunakan dalam beberapa protokol untuk identifikasi (*zero-knowledge identification protocol*). Protokol pertama jenis ini adalah protokol Fiat-Shamir. Kita akan bahas protokol Fiat-Shamir dan dua protokol yang merupakan derivatif dari Fiat-Shamir yaitu protokol Feige-Fiat-Shamir dan protokol Guillou-Quisquater. Ketiga protokol yang akan dibahas tergolong apa yang dinamakan *challenge-response protocol*.

Ada tiga aktor yang berperan dalam protokol Fiat-Shamir yaitu *trusted center* (sebut saja Tim), *prover* (Peggy) dan *verifier* (Victor). Tim membuat suatu modulus seperti RSA $n = pq$, mengumumkan n tetapi merahasiakan p dan q. Peggy membuat secara acak (menggunakan *random number generator*) kunci privat s, dimana $0 < s < n$ dan $\gcd(s, n) = 1$. Kunci publik Peggy adalah $v = s^2 \bmod n$ dan v diregistrasi ke Tim. Victor dapat memperoleh kunci publik Peggy v yang telah diregistrasi, dari Tim. Langkah-langkah berikut diulang t kali, setiap kali dengan nilai-nilai acak yang baru, agar Peggy dapat diidentifikasi oleh Victor.

1. Peggy memilih secara acak, menggunakan *random number generator*, r, $0 < r < n$, dan mengirim $x = r^2 \bmod n$ kepada Victor.

2. Victor memilih secara acak, menggunakan *random number generator*, e, $e \in \{0, 1\}$, dan mengirimnya ke Peggy.

3. Peggy mengkomputasi $y = rs^e \bmod n$ dan mengirim y ke Victor.

4. Jika $y = 0$ atau $y^2 \not\equiv xv^e \pmod n$ maka Victor menolak dan proses identifikasi gagal.

Jika langkah-langkah diatas telah diulang t kali tanpa penolakan maka identifikasi Peggy diterima oleh Victor. Probabilitas bahwa Peggy telah berhasil menipu Victor adalah 1 dalam 2^t.

Keamanan dari Fiat-Shamir berdasarkan pada sukarnya mengkalkulasi akar kuadrat modulo pq jika p dan q tidak diketahui (hanya produknya $n = pq$ yang diketahui). Jika p dan q diketahui, kita dapat menggunakan teknik diahir bagian 11.2 untuk mengkalkulasi akar kuadrat modulo pq, jadi p dan q harus dirahasiakan oleh Tim. Fungsi pengacakan menggunakan parameter e adalah agar Peggy tidak curang. Jika Victor selalu meminta $y = r$ ($e = 0$) maka jelas Peggy tidak perlu mengetahui s untuk menjawabnya. Jika Victor selalu meminta $y = rs \bmod n$, Peggy juga dapat mengelabui Victor tanpa mengetahui s sebagai berikut. Pada langkah 1 Peggy mengirim

$$x = r^2 v^{-1} \bmod n$$

kepada Victor. Ketika diminta untuk mengirim $y = rs \bmod n$ maka Peggy mengirim $y = r$. Jadi Victor terkelabui karena

$$\begin{aligned} xv &\equiv r^2 v^{-1} v \pmod{n} \\ &\equiv r^2 \pmod{n} \\ &\equiv y^2 \pmod{n}. \end{aligned}$$

Dengan pengacakan, jika Peggy mengirim $x = r^2 \bmod n$ maka ia kadang harus menggunakan s, sedangkan jika ia mengirim $x = r^2 v^{-1} \bmod n$ maka ia kadang harus mencari akar kuadrat modulo n karena ia harus mengirim

$$y = (r^2 v^{-1})^{-2} \bmod n.$$

Untuk menunjukkan bahwa tidak ada rahasia yang bocor ke Victor, kita gunakan cara standard yaitu dengan simulasi. Seseorang yang tidak mengetahui p, q dan s akan tetapi mengetahui apa yang akan diminta Victor untuk e tentunya akan dapat berperan sebagai Peggy dengan mengirim $x = r^2 \bmod n$ atau $x = r^2 v^{-1} \bmod n$ tergantung pada nilai e. Informasi yang dikeluarkan oleh Peggy dapat dikeluarkan oleh siapa saja tanpa mengetahui p, q dan s, jadi tidak memberi tahu nilai p, q dan s.

Sebetulnya Fiat-Shamir membocorkan 1 bit dari nilai s, yaitu *sign* ($+$ atau $-$) dari s. Protokol Feige-Fiat-Shamir menutup kebocoran ini. Selain itu Feige-Fiat-Shamir melakukan k "pembuktian" secara paralel yang mengurangi interaksi antara Peggy dan Victor karena langkah-langkah tidak perlu diulang sebanyak pada Fiat-Shamir, bahkan langkah-langkah tidak perlu diulang jika k cukup besar.

Dalam Feige-Fiat-Shamir, Peggy membuat k kunci privat $s_1, s_2, \ldots, s_k$ dimana $\gcd(s_i, n) = 1$ untuk setiap i, dan mempublikasikan k kunci publik $v_1, v_2, \ldots, v_k$ dimana

$$v_i = s_i^2 \bmod n$$

untuk setiap i. Langkah-langkah Feige-Fiat-Shamir adalah sebagai berikut.

1. Peggy memilih secara acak, menggunakan *random number generator*, r, $0 < r < n$, dan $s \in \{-1, 1\}$, dan mengirim $x = sr^2 \bmod n$ kepada Victor.

2. Victor memilih secara acak $e_1, e_2, \ldots, e_k$, $e_i \in \{0, 1\}$ untuk setiap i, dan mengirimnya ke Peggy.

3. Peggy mengkomputasi $y = rs_1^{e_1} s_2^{e_2} \cdots s_k^{e_k} \bmod n$ dan mengirim y ke Victor.

4. Jika $y^2 \not\equiv \pm x v_1^{e_1} v_2^{e_2} \cdots v_k^{e_k} \pmod{n}$ maka Victor menolak dan proses identifikasi gagal.

Jika $k = 20$, maka probabilitas bahwa Peggy berhasil mengelabui Victor kurang dari 1 dalam sejuta, dengan hanya 1 putaran langkah-langkah diatas.

Untuk Feige-Fiat-Shamir ada yang menggunakan $v_i = 1/s_i^2 \bmod n$ untuk kunci publik. Jika demikian, maka pada langkah 4, Victor harus memeriksa

$$x \equiv \pm y^2 v_1^{e_1} v_2^{e_2} \cdots v_k^{e_k} \pmod{n}.$$

Protokol Guillou-Quisquater kita bahas disini karena sangat populer untuk aplikasi identifikasi smartcard. Untuk Guillou-Quisquater, Peggy berperan sebagai smartcard, Victor adalah komputer yang sedang mengidentifikasi smartcard, dan Tim adalah pengelola smartcard. Seperti Fiat-Shamir, $n = pq$ dipilih oleh Tim, n dipublikasikan, sedangkan p dan q dirahasiakan. Selain n, parameter v yang merupakan eksponen dengan $\gcd(v, \phi(n)) = 1$ juga dipublikasikan. Untuk Peggy, terdapat parameter B yang merupakan bilangan yang merepresentasikan *credentials* dari smartcard (terdiri misalnya dari *card issuer*, *serial number* dan *expiry date*). Peggy juga diberi Tim kunci privat K, dimana

$$B \cdot K^v \equiv 1 \pmod{n}.$$

Peggy mengirim *credentials* miliknya (yang direpresentasikan menggunakan B) kepada Victor. Langkah-langkah *challenge-response* kemudian berlangsung mirip dengan Fiat-Shamir dan Feige-Fiat-Shamir:

1. Peggy memilih secara acak, menggunakan *random number generator*, r, $0 < r < n$, dan mengirim $T = r^v \bmod n$ kepada Victor.

2. Victor memilih secara acak, menggunakan *random number generator*, d, dimana $0 \leq d < v$, dan mengirimnya ke Peggy.

3. Peggy mengkomputasi $D = rK^d \bmod n$ dan mengirim D ke Victor.

4. Victor memeriksa $D^v B^d \equiv T \pmod{n}$. Jika tidak cocok maka Victor menolak dan proses identifikasi gagal.

Pada langkah 4, Victor memeriksa $D^v B^d \equiv T \pmod{n}$ karena menggunakan fakta bahwa $B \cdot K^v \equiv 1 \pmod{n}$ kita dapatkan:

$$\begin{aligned} D^v B^d &\equiv (rK^d)^v B^d \pmod{n} \\ &\equiv r^v K^{dv} B^d \pmod{n} \\ &\equiv r^v (BK^v)^d \pmod{n} \\ &\equiv r^v \pmod{n} \\ &\equiv T \pmod{n}. \end{aligned}$$

Karena komunikasi antara Peggy dan Victor tidak diamankan, ketiga protokol identifikasi yang telah dibahas mempunyai kelemahan yaitu rentan terhadap *man-in-middle attack*. Jadi dalam aplikasinya, peluang untuk seseorang menempatkan dirinya diantara Peggy dan Victor dalam media komunikasi

harus dihilangkan atau dibuat sekecil mungkin. Tentunya jika komunikasi diamankan menggunakan, misalnya RSA, maka tidak ada kelemahan ini. Namun jika sudah menggunakan RSA maka protokol identifikasi seperti diatas tidak diperlukan.

16.7 Penggunaan Kriptografi Public Key

Hampir semua sistem kriptografi *public key* yang telah dibahas dirancang untuk keperluan khusus. Diffie-Hellman dirancang untuk keperluan *key agreement*, DSA untuk *digital signature*, dan sistem yang berbasis pada *zero-knowledge protocol* untuk identifikasi. RSA dan ElGamal adalah dua sistem kriptografi *public key* yang dapat digunakan untuk enkripsi. Namun jika dibandingkan dengan sistem enkripsi klasik seperti AES, 3DES dan CAST, maka RSA dan ElGamal sangat lambat. Jadi penggunaan RSA dan ElGamal dalam enkripsi juga agak khusus yaitu mengenkripsi kunci enkripsi klasik atau mengenkripsi *digest* untuk keperluan *digital signature*. Enkripsi kunci enkripsi klasik contohnya dalam suatu sesi SSL atau SSH (akan dibahas di bab 20), kunci sesi yang biasanya merupakan kunci AES, 3DES atau CAST, dienkripsi menggunakan kunci publik jenis RSA atau ElGamal. Demikian juga jika mengenkripsi *file* yang ukurannya bisa cukup besar, kriptografi klasik digunakan untuk mengenkripsi *file*, lalu kunci klasik dienkripsi menggunakan kunci publik.

Kriptografi *public key* tentunya juga berperan sangat besar dalam suatu *public key infrastructure* (akan dibahas di bab 23). Fungsi utama *public key infrastructure* adalah manajemen *digital signature* dan kunci publik untuk dekripsi, termasuk manajemen *certificate*, untuk kunci publik *digital signature* dan untuk kunci publik enkripsi. Aplikasi untuk kunci publik yang dikelolapun beragam, dari *certificate* untuk *website*, kunci publik untuk keperluan *secure email*, sampai dengan *certificate* untuk keperluan IPsec (akan dibahas di bagian 20.3).

16.8 Ringkasan

Berbagai sistem kriptografi *public key* telah dibahas di bab ini, yaitu RSA, Diffie-Hellman, DSA, ElGamal, *knapsack*, Fiat-Shamir, Feige-Fiat-Shamir dan Guillou-Quisquater. Kecuali *knapsack*, mekanisme transformasi untuk semua sistem yang telah dibahas berandalkan pada rumus

$$a^{\phi(q)} \equiv 1 \pmod{q}$$

dimana $a \in \mathbf{N}$ dan $\gcd(a, q) = 1$, atau

$$a^{q-1} = 1$$

dimana $a \in \mathbf{GF}(q)^*$. Keamanan dari RSA, Fiat-Shamir, Feige-Fiat-Shamir dan Guillou-Quisquater berbasis pada sukarnya menguraikan q, sedangkan keamanan dari Diffie-Hellman, DSA dan ElGamal berbasis pada sukarnya mengkomputasi logaritma diskrit. Sistem kriptografi yang berbasis pada *knapsack problem* tidak populer karena hilangnya kepercayaan pada keamanannya sejak sistem pertama yang berbasis pada *knapsack problem*, yaitu Merkle-Hellman, dapat dipecahkan.

Dibandingkan kriptografi klasik (simetris), kriptografi *public key* sangat tidak efisien untuk enkripsi umum. Namun banyak keperluan enkripsi khusus yang dapat dipenuhi oleh kriptografi *public key* dan tidak dapat dipenuhi oleh kriptografi klasik. Oleh sebab itu kriptografi *public key* digunakan untuk berbagai keperluan khusus seperti *key agreement*, *key distribution*, *digital signature* dan *identification protocol*.