Bab 17

Kriptografi Elliptic Curve

Sistem kriptografi *public key* yang berbasis pada sukarnya mengkomputasi logaritma diskrit seperti Diffie-Hellman, DSA dan ElGamal bekerja menggunakan suatu *multiplicative group* $\mathbf{GF}(q)^*$. Suatu *elliptic curve over a finite field* juga memberikan *Abelian group* yang dapat digunakan untuk mekanisme kriptografi yang serupa dengan sistem berbasis logaritma diskrit. Lebih menarik lagi, *elliptic curve over a finite field* memberikan lebih banyak fleksibilitas dibandingkan *finite field* yang terbatas pada $\mathbf{GF}(p)$ dan $\mathbf{GF}(p^n)$.

Sebelum membahas penggunaan *elliptic curve* untuk kriptografi, tentunya kita perlu mengetahui apa itu *elliptic curve*, khususnya *elliptic curve over a finite field.* Suatu *elliptic curve* bukan merupakan suatu elips, tetapi merupakan suatu "kurva" untuk persamaan sebagai berikut:

$$Ax^3 + Bx^2y + Cxy^2 + Dy^3 + Ex^2 + Fxy + Gx + Iy = 0.$$

Field apa saja dapat digunakan untuk membuat *elliptic curve*, termasuk $\mathbf{R}$, $\mathbf{Q}$ (masing-masing mempunyai *characteristic* 0), dan *finite field* (yang tentunya mempunyai *characteristic* suatu bilangan prima). Jika *field* yang digunakan mempunyai *characteristic* k dimana $k \neq 2$ dan $k \neq 3$, maka persamaan diatas dapat disederhanakan menjadi:

$$y^2 = x^3 + ax + b. \tag{17.1}$$

Jika $k = 2$ maka persamaan hanya dapat disederhanakan menjadi

$$y^2 + cy = x^3 + ax + b \tag{17.2}$$

atau

$$y^2 + xy = x^3 + ax + b, \tag{17.3}$$

sedangkan jika $k = 3$ persamaan hanya dapat disederhanakan menjadi

$$y^2 = x^3 ax^2 + bx + c. \tag{17.4}$$

Suatu *elliptic curve over field* F adalah titik-titik (a, b) yang merupakan solusi untuk x dan y dalam persamaan, dimana $a, b \in F$, ditambah dengan titik di ∞ (*point at infinity*) yang diberi simbol 0. Suatu *Abelian group* dapat dibentuk menggunakan titik-titik tersebut dan operasi $+$. Untuk $F = \mathbf{R}$ (jadi $k = 0$), kita definisikan $+$ dan $-$ (*inverse*) sebagai berikut:

1. $-0 = 0$ dan $0 + Q = Q$ (jadi 0 merupakan *identity*).
2. Untuk $P \neq 0$, $-P$ adalah titik dengan koordinat x yang sama dan negatif koordinat y. Jadi $-(x, y) = (x, -y)$.
3. Jika P dan Q adalah dua titik yang berbeda, maka tidak terlalu sulit untuk melihat bahwa garis $l = \overline{PQ}$ melewati titik ketiga R. Kita definisikan $P + Q = -R$ (jadi $P + Q + R = 0$).
4. $P + (-P) = 0$.
5. Yang terahir adalah untuk $P + P$. Jika l adalah garis tangen untuk P, maka l akan bertemu dengan satu titik lagi yaitu R, kecuali jika P merupakan titik infleksi. Kita definisikan $P + P = -R$, kecuali jika P adalah titik infleksi dimana kita definisikan $P + P = -P$.

Untuk bagian 3, kita bisa dapatkan rumus yang lebih rinci lagi berdasarkan persamaan 17.1. Kita gunakan koordinat (x_P, y_P) untuk P, (x_Q, y_Q) untuk Q dan (x_R, y_R) untuk R. Jika

$$y = \alpha x + \beta$$

adalah rumus untuk garis l yang melalui P dan Q, maka

$$\alpha = (y_Q - y_P)/(x_Q - x_P)$$

dan

$$\beta = y_P - \alpha x_P.$$

Suatu titik pertemuan antara garis l dan kurva akan mematuhi persamaan

$$(\alpha x + \beta)^2 = x^3 + ax + b$$

yang, dalam bentuk *polynomial* menjadi

$$x^3 - \alpha^2 x^2 + (a - 2\beta)x + (b - \beta^2) = 0.$$

Tentunya P, Q dan R merupakan akar dari persamaan ini. Karena penjumlahan akar merupakan negatif koefisien x^2 dalam persamaan (lihat pembahasan *trace* di bagian 12.3), maka $x_R = \alpha^2 - x_P - x_Q$, jadi

$$\begin{aligned} x_R &= \left(\tfrac{y_Q - y_P}{x_Q - x_P}\right)^2 - x_P - x_Q, \\ y_R &= -y_P + \left(\tfrac{y_Q - y_P}{x_Q - x_P}\right)(x_P - x_R). \end{aligned} \tag{17.5}$$

Untuk bagian 5, kita juga bisa dapatkan rumus yang rinci, tetapi α merupakan derivatif dy/dx di P. Diferensiasi implisit persamaan 17.1 menghasilkan $\alpha = (3x_P^2 + a)/2y_P$, jadi

$$\begin{aligned} x_R &= \left(\tfrac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P, \\ y_R &= -y_P + \left(\tfrac{3x_P^2 + a}{2y_P}\right)(x_P - x_R). \end{aligned} \tag{17.6}$$

Rumus 17.5 dan 17.6 dapat digunakan untuk menunjukkan bahwa definisi diatas membentuk suatu *Abelian group*.

Rumus 17.5 dan 17.6 juga berlaku untuk *elliptical curve over a finite field* jika *finite field* mempunyai *characteristic* $k > 3$. Jika $k = 2$, maka rumus 17.5 dan 17.6 harus disesuaikan dengan persamaan 17.2 atau 17.3, sedangkan untuk $k = 3$, rumus 17.5 dan 17.6 harus disesuaikan dengan persamaan 17.4.

Tentunya jumlah titik-titik pada *elliptical curve* dengan *finite field* $\mathbf{GF}(q)$, n, tidak melebihi $2q + 1$, karena untuk setiap $x \in \mathbf{GF}(q)$ (ada q nilai x yang berbeda) terdapat maksimal dua nilai untuk $y \in \mathbf{GF}(q)$ dimana (x, y) merupakan titik pada kurva, ditambah 1 titik di ∞. Akan tetapi, karena hanya setengah dari elemen $\mathbf{GF}(q)^*$ mempunyai akar kuadrat, maka jumlah titik-titik pada kurva kira-kira hanya setengah dari itu. Untuk tepatnya, kita gunakan fungsi *quadratic character* χ. Fungsi χ memetakan setiap $x \in \mathbf{GF}(q)^*$ ke ± 1 tergantung apakah x merupakan kuadrat dalam $\mathbf{GF}(q)$ (1 jika kuadrat dan -1 jika bukan kuadrat). Jika $q = p$, suatu bilangan prima, tentunya $\chi(x) = \left(\frac{x}{p}\right)$ (lihat bagian 11.1). Untuk setiap $x \in \mathbf{GF}(q)$, banyaknya solusi y yang menjadikan (x, y) suatu titik pada kurva (jika $k > 3$) adalah

$$1 + \chi(x^3 + ax + b)$$

yang menghasilkan 0 jika $\chi(x^3 + ax + b) = -1$ dan 2 jika $\chi(x^3 + ax + b) = 1$. Jadi banyaknya titik, termasuk titik di ∞ adalah

$$1 + \sum_{x \in \mathbf{GF}(q)} (1 + \chi(x^3 + ax + b)) = 1 + q + \sum_{x \in \mathbf{GF}(q)} \chi(x^3 + ax + b).$$

Karena peluang $\chi(x^3 + ax + b)$ untuk menghasilkan 1 sama dengan peluangnya untuk menghasilkan -1, maka penjumlahan diatas ibarat melakukan *random* *walk* dan dibatasi oleh $2\sqrt{q}$. Inilah yang menjadi dasar dari teorema Hasse sebagai berikut:

Teorema 112 (Hasse's Theorem) *Jika n adalah banyaknya titik-titik pada elliptic curve yang didefinisikan menggunakan* $\mathbf{GF}(q)$, *maka*

$$|n - (q + 1)| \leq 2\sqrt{q}.$$

Jika ingin mengetahui dengan tepat banyaknya titik-titik pada *elliptic curve* yang didefinisikan menggunakan $\mathbf{GF}(q)$, ada algoritma yang pertama ditemukan oleh Schoof (lihat [sch85]) dan dikembangkan lebih lanjut oleh peneliti lainnya. Nilai n ini perlu diketahui untuk mengetahui keamanan enkripsi, karena jika n dapat diuraikan menjadi produk dari bilangan-bilangan prima yang kecil, maka metode Pohlig-Silver-Hellman dapat digunakan untuk mengkomputasi logaritma diskrit.

Sistem kriptografi *public key* yang berdasarkan pada logaritma diskrit mempunyai versi yang menggunakan *elliptic curve*. Disini kita akan bahas dua diantaranya yaitu Diffie-Hellman dan ElGamal. Untuk yang ingin mempelajari versi *elliptic curve* dari DSA disarankan untuk membaca [nis00].

Untuk Diffie-Hellman, suatu *finite field* $\mathbf{GF}(q)$ digunakan untuk membuat suatu *elliptic curve*, E. Jika dalam versi asli suatu g, yang sebaiknya merupakan *generator*, dipilih sebagai basis, maka di versi *elliptic curve*, dipilih suatu titik $B \in E$. Alice dan Bob melakukan *key agreement* menggunakan Diffie-Hellman versi *elliptic curve* sebagai berikut:

- Alice memilih, secara acak, suatu bilangan a yang besarnya sekitar q, mengkomputasi aB, dan mengirimkan aB ke Bob.
- Bob memilih, secara acak, suatu bilangan b yang besarnya sekitar q, mengkomputasi bB, dan mengirimkan bB ke Alice.
- Setelah menerima bB dari Bob, Alice mengkomputasi abB yang menjadi kunci bersama dengan Bob.
- Bob, setelah menerima aB dari Alice, mengkomputasi $abB = baB$.

Proses *key agreement* menghasilkan kunci bersama abB antara Alice dan Bob. Seseorang yang tidak mengetahui a dan tidak mengetahui b tidak dapat mengkomputasi abB dari aB dan bB, tanpa menemukan cara efisien untuk mengkomputasi logaritma diskrit (asumsi Diffie-Hellman). Tabel 17.1 membandingkan Diffie-Hellman versi *finite field* (DH) dengan Diffie-Hellman versi *elliptic curve* (ECDH).

Serupa dengan Diffie-Hellman, untuk ElGamal, suatu *finite field* $\mathbf{GF}(q)$ digunakan untuk membuat suatu *elliptic curve*, E. Suatu titik $B \in E$ digunakan sebagai basis. Untuk membuat pasangan kunci, Alice memilih secara acak suatu bilangan a yang ia jadikan kunci privat. Alice kemudian mengkomputasi

Komponen	**DH**	**ECDH**
Basis	$g \in \mathbf{GF}(q)$	$B \in E$
Potongan kunci A	g^a	aB
Potongan kunci B	g^b	bB
Kunci bersama	g^{ab}	abB

Tabel 17.1: Perbedaan Diffie-Hellman dengan versi *elliptic curve*

aB dan mempublikasikannya sebagai kunci publiknya. Untuk mengenkripsi suatu naskah yang telah dikodifikasi sebagai M dalam E, agar hanya dapat dibaca oleh Alice, seseorang melakukan langkah-langkah berikut:

- Pilih bilangan bulat k secara acak.
- Kirim pasangan $(kB, M + k(aB))$ ke Alice.

Alice dapat mendekripsi $(kB, M + k(aB))$ dengan, pertama, mengkomputasi $a(kB)$, lalu mengkomputasi

$$M + k(aB) - a(kB) = M.$$

Seseorang yang tidak mengetahui k atau kunci privat a tentunya tidak dapat mengkomputasi $k(aB)$ tanpa menemukan cara efisien untuk mengkomputasi logaritma diskrit. Tabel 17.2 membandingkan ElGamal versi *finite field* (EG) dengan ElGamal versi *elliptic curve* (ECEG).

Komponen	**EG**	**ECEG**
Basis	$g \in \mathbf{GF}(q)$	$B \in E$
Kunci privat	a	a
Kunci publik	g^a	aB
IV	k	k
Naskah	M	M
Enkripsi	(g^k, Mg^{ak})	$(kB, M + k(aB))$

Tabel 17.2: Perbedaan ElGamal dengan versi *elliptic curve*

Ada dua hal mengenai implementasi *elliptic curve* suatu sistem kriptografi berbasis logaritma diskrit yang akan kita bahas disini:

- Pemetaan antara bilangan-bilangan yang merepresentasikan naskah dengan titik-titik dalam *elliptic curve*.
- Operasi perkalian dengan bilangan bulat dalam *elliptic curve*.

Sayangnya belum ada algoritma *polynomial time* yang dapat secara deterministik memetakan bilangan-bilangan menjadi titik-titik dalam *elliptic curve*. Akan tetapi terdapat algoritma probabilisitik yang cukup efisien dan kemungkinan untuk gagal dapat dibuat sangat kecil. Kita beri contoh untuk $q = p^r$ sangat besar dan ganjil. Kita pilih suatu bilangan bulat κ berdasarkan seberapa kecil kita ingin probabilitas kegagalan, yaitu

$$1/2^{\kappa}.$$

Biasanya nilai κ antara 30 sampai dengan 50. Kita harus membatasi suatu unit naskah sehingga mempunyai nilai bilangan bulat m dimana $0 \leq m < M$ dan $M\kappa < q$. Kita dapat menulis setiap bilangan bulat dari 1 sampai dengan $M\kappa$ dalam bentuk

$$m\kappa + j$$

dimana $1 \leq j \leq \kappa$. Maka terdapat suatu *injection* dari bilangan-bilangan tersebut ke $\mathbf{GF}(q)$ sebagai berikut:

- Representasikan setiap bilangan sebagai bilangan dengan basis p, jadi maksimal terdapat r digit.
- Bilangan dengan basis p tersebut dapat diinterpretasikan sebagai elemen dari $\mathbf{GF}(q)$.

Setiap m kita petakan ke elemen dari E sebagai berikut:

1. $j \leftarrow 1$.
2. Dapatkan $x \in \mathbf{GF}(q)$ sebagai nilai $m\kappa + j$ berdasarkan *injection* diatas.
3. Komputasi $f(x)$ sebagai sisi kanan dari persamaan $y^2 = x^3 + ax + b$.
4. Coba cari akar kuadrat dari $f(x)$ menggunakan metode di ahir bagian 11.2.
5. Jika akar kuadrat $f(x)$ ditemukan, maka kita gunakan akar tersebut sebagai nilai y, dan kita selesai dengan memetakan m ke (x, y).
6. Jika belum berhasil maka $j \leftarrow j + 1$ dan kembali ke langkah 2.

Algoritma diatas akan gagal jika kita tidak menemukan (x, y) sebelum j melebihi κ, dan probabilitas kegagalan adalah $1/2^{\kappa}$.

Jika dalam aritmatika $\mathbf{GF}(q)$ untuk kriptografi operasi pemangkatan elemen dengan bilangan bulat diperlukan, maka dalam kriptografi *elliptic curve* operasi perkalian elemen dengan bilangan bulat diperlukan. Jika pemangkatan dalam aritmatika $\mathbf{GF}(q)$ dapat dilakukan secara efisien menggunakan teknik *repeated squaring* (pengkuadratan berulang), maka dalam aritmatika *elliptic* *curve* perkalian elemen dengan bilangan bulat dapat dilakukan secara efisien menggunakan *repeated doubling* (penggandaan berulang). Sebagai contoh, $100P$ dapat dikomputasi secara efisien sebagai

$$100P = 2(2(P + 2(2(2(P + 2P))))).$$

Ini dapat dilakukan menggunakan langkah-langkah berikut secara berulang, dimana $M > 0$ adalah pengali:

1. Jika $M = 1$ kita selesai dengan hasil P.
2. Jika M genap maka $M \leftarrow M/2$, kita cari hasil dengan M yang baru, lalu kalikan 2 ke hasil tersebut.
3. Jika M ganjil maka $M \leftarrow M - 1$, kita cari hasil dengan M yang baru, lalu tambahkan P ke hasil tersebut.

Kita ahiri bab ini dengan pembahasan secara ringkas mengapa kriptografi *public key* menggunakan *elliptic curve* diminati. Logaritma diskrit untuk *elliptic curve* jauh lebih sukar dibandingkan logaritma diskrit untuk $\mathbf{GF}(q)$ (kecuali untuk *supersingular elliptic curve* yang sangat jarang). Untuk *elliptic curve*, secara umum logaritma diskrit hanya dapat dikomputasi menggunakan algoritma Shanks atau algoritma Pollard, dan kedua algoritma mempunyai kompleksitas *full exponential*. Untuk *finite field*, kompleksitas logaritma diskrit mirip dengan kompleksitas penguraian yaitu *sub-exponential*. Jadi kompleksitas untuk *elliptic curve* tumbuh lebih cepat dibandingkan kompleksitas untuk *finite field*. Berdasarkan perkiraan oleh Alfred Menezes (lihat [men95]), penggunaan kriptografi *finite field* seperti DSA atau RSA dengan kunci sebesar 1024 bit sama kuatnya dengan penggunaan kriptografi *elliptic curve* dengan kunci sebesar 160 bit. Jadi cukup menyolok perbedaan besar kunci untuk kekuatan yang sama. Untuk kekuatan yang lebih besar, perbedaan semakin menyolok karena kekuatan *elliptic curve* tumbuh lebih cepat (*exponential* dibandingkan *sub-exponential* untuk *finite field*), seperti terlihat pada tabel 17.3 (data berdasarkan [rob97]). ECDSA adalah DSA versi *elliptic curve* sedang-

DSA/ElGamal	**RSA**	**ECDSA/ECES**
1024	1024	160
2048	2048	224
3072	3072	256
7680	7680	384
15360	15360	512

Tabel 17.3: Besar kunci untuk kekuatan yang sama

kan ECES adalah ElGamal versi *elliptic curve*. Tabel 17.4 menunjukkan relatif waktu yang dibutuhkan untuk berbagai operasi (data berdasarkan [rob97]). Untuk *key generation*, RSA jauh lebih lambat dibandingkan DSA/ElGamal

	DSA/ElGamal	**RSA**	**ECDSA/ECES**
	1024 bit	1024 bit	160 bit
encryption	480	17	120
decryption	240	384	60
signing	240	384	60
verification	480	17	120

Tabel 17.4: Waktu untuk berbagai operasi

dan ECDSA/ECES. Jadi cukup jelas mengapa sistem kriptografi *public key* dengan *elliptic curve* sangat menarik, terutama untuk aplikasi di perangkat kecil dengan kemampuan terbatas seperti *smartcard* atau perangkat *Bluetooth*. Kriptografi *elliptic curve* juga semakin menarik untuk penggunaan masa depan karena pertumbuhan kunci yang dibutuhkan tidak sebesar kriptografi *finite field*. Sebetulnya versi *elliptic curve* untuk RSA juga ada, namun berbeda dengan logaritma diskrit dimana versi *elliptic curve* lebih sukar untuk dipecahkan dibandingkan versi *finite field*, penguraian bilangan bulat tetap merupakan penguraian bilangan bulat, jadi tidak ada keuntungan dengan menggunakan *elliptic curve* untuk RSA.

17.1 Ringkasan

Di bab ini telah didiskusikan secara garis besar konsep *elliptic curve*, bagaimana sistem kriptografi yang berbasis pada logaritma diskrit seperti Diffie-Hellman, DSA dan ElGamal dapat diadaptasi untuk menggunakan *elliptic curve*, dan beberapa masalah implementasi kriptografi *elliptic curve*. Pembahasan ringkas mengenai mengapa kriptografi *elliptic curve* diminati terdapat pada ahir bab ini, termasuk mengapa versi *elliptic curve* untuk RSA tidak diminati.