Capítulo 8

Algoritmos para CCE

8.1. Introducción

Los algoritmos para criptosistemas de curvas elípticas pueden clasificarse en dos categorias: los de *alto nivel*, relacionados con las operaciones de puntos sobre la curva elíptica y los de *bajo nivel*, relacionados con el campo subyacente.

8.2. Algoritmos del campo subyacente.

Las dos elecciones más comunes en aplicaciones prácticas del campo finito subyacente son $GF(2^m)$ y $GF(p)$, donde p es un primo diferente a 2 y 3. El problema de logaritmo discreto es igualmente difícil en curvas sobre $GF(p)$ como en curvas definidas sobre $GF(2^m)$, suponiendo que 2^m y p son aproximadamente iguales. Hasta la fecha no ha habido ningún descubrimiento matemático que permita suponer que el logaritmo discreto para CE sobre $GF(2^m)$ sea más fácil que para CE sobre $GF(p)$ o viceversa.

Si se elije al campo $GF(2^m)$ como campo finito subyacente, entonces existen varias posibles maneras en las cuales se pueden representar los elementos de $GF(2^m)$. Como anteriormente detallamos, las dos formas más eficientes son mediante *representación en bases normales óptimas* así como mediante la *representación en bases polinomiales* [64]. Dado que los elementos en una representación pueden ser eficientemente convertidos a elementos en la otra representación por medio de una matríz de cambio de base adecuada, la intratabilidad del logaritmo discreto sobre CE no es afectada por la elección de la representación.

Se ha descrito [2] un circuito integrado para la realización de opera-

ciones sobre curvas elípticas definidas sobre $GF(2^{155})$ el cual, trabajando a una frecuencia de 40 MHz, puede realizar aproximadamente 40,000 adiciones elípticas por segundo. Dicho circuito estaría compuesto de 12,000 compuertas, mientras que un chip similar para la multiplicación modular de enteros de 512 bits [46] consta aproximadamente de 50,000 compuertas. La implementación en software sobre el mismo campo [88] realizada en una SPARC IPC efectua 2,000 adiciones de puntos por segundo. Resultados similares se han logrado para implementaciones en software sobre una PC con procesador Pentium II [32].

La implementación de software de CE sobre el campo de Galois compuesto $GF((2^n)^m)$ fue descrita por vez primera en [42] para el campo de Galois $GF((2^8)^{13})$. Más recientemente, sistemas de CE sobre el campo compuesto $GF((2^{16})^{11})$ fueron descritos de forma independiente en [10] y [106]. En ambos casos, la multiplicación en el campo se realiza mediante la búsqueda en tablas en el subcampo $GF(2^n)$. En cambio, en [40] se propone el uso del algoritmo de Karatsuba-Ofman [47] para la multiplicación de elementos del campo. El algoritmo de Karatsuba-Ofman [47] ha sido anteriormente estudiado para la multiplicación de polinomios en general [28] [48]. Una aplicación del algoritmo de Karatsuba-Ofman [47] a polinomios sobre campos finitos en el contexto del hardware se describe en [76]. Los campos de Galois compuestos han sido aplicados en arquitecturas de hardware para algoritmos de clave pública [77]. Si bien la utilización del algoritmo de Karatsuba-Offman en campos compuestos $GF((2^n)^m)$ permite el uso de paralelismo, hay que señalar como hicimos anteriormente, que el uso de curvas elípticas sobre campos compuestos ha sido desestimado por considerarse estas menos seguras [30][37].

8.3. Algoritmos de multiplicación escalar

8.3.1. Algoritmo binario

La operación central a los CCE es el cálculo de kP, donde k es un entero y P es un punto sobre una curva elíptica. El problema de multiplicar un punto P de una CE por un entero k es análogo a elevar a la potencia k a un elemento de un grupo multiplicativo.

El algoritmo estándar para este problema es el método de exponenciación binaria, detalladamente expuesto en [48].

Sea E una curva elíptica definida sobre un campo finito $GF(q)$ y sea P un punto sobre E de orden n. Sea k un entero positivo. Deseamos calcular la potencia kP.

Sea $k = (b_{m-1}b_{m-2}b_{m-3}\cdots b_2b_1b_0)_2$ la expansión binaria de k, donde m es tal que $b_{m-1} = 1$. Entonces

$$k = b_{m-1}2^{m-1} + b_{m-2}2^{m-2} + ... + b_2 2^2 + b_1 2 + b_0. \tag{8.1}$$

De la expansión de Horner

$$k = (\cdots((b_{m-1}2 + b_{m-2})2 + \cdots + b_1)2 + b_0) \tag{8.2}$$

obtenemos

$$kP = (2(\cdots 2(2(2(b_{m-1}P) + b_{m-2}P) + \cdots + b_1P) + b_0P). \tag{8.3}$$

Construyamos entonces la siguiente sucesión de puntos sobre la curva,

$$P_0 = P \tag{8.4}$$

y

$$P_i = \begin{cases} 2P_{i-1} & \text{si } b_{m-i-1} = 0 \\ 2P_{i-1} + P & \text{si } b_{m-i-1} = 1 \end{cases} \tag{8.5}$$

para $i = 1, \cdots, m-1$. Se puede verificar fácilmente que $P_{m-1} = kP$.

Obsérvese que como en cada paso se requiere duplicar el punto anterior y, si el correspondiente bit de la expansión binaria de k es 1, hacer una suma con el punto P, el algoritmo binario requiere de $m-1$ sumas de puntos sobre la curva y tantas duplicaciones de puntos como bits en uno haya en la expansión binaria de k. Por lo tanto, el cálculo de kP por el algoritmo binario requiere de $m-1$ sumas de puntos sobre la curva y $H_2(k)-1$ duplicaciones de puntos sobre la curva, donde $H_2(k)$ es el *peso de Hamming* de k, esto es, el número de unos en la expansión binaria de k.

8.3.2. Algoritmo m-ario

Una generalización del método binario es el método m-ario [19] en el cual se procesan m bits del exponente en cada iteración. Sean P un punto sobre la curva elíptica de orden n, y e un entero tal que $0 < e \leq n$.

Sea $e = (e_{k-1}e_{k-2}\cdots e_1e_0)_2$ la expansión binaria de e. Dividiendo esta expansión en s bloques de r bits, tenemos que $s \cdot r = k$ y, si $m = 2^r$,

$$e = (\mu_{s-1}\mu_{s-2}\cdots\mu_1\mu_0)_m$$

es la expansión en la base m de e, donde $0 \leq \mu_i \leq m-1$. Por lo tanto

$$\mu_i = (e_{ir+r-1}e_{ir+r-2}\cdots e_{ir})_2 = \sum_{j=0}^{r-1} e_{ir+j}2^j$$

y

$$e = \sum_{i=0}^{s-1} \mu_i 2^{ir} = \sum_{i=0}^{s-1} \mu_i m^i.$$

La expansión de Horner del polinomio anterior es

$$e = (\cdots((\mu_{s-1}m + \mu_{s-2})m + \cdots + \mu_1)m + \mu_0 \tag{8.6}$$

y por lo tanto

$$eP = m(\cdots(m(m(\mu_{s-1}P) + \mu_{s-2}P)) + \cdots + \mu_1 P) + \mu_0 P. \tag{8.7}$$

La ecuación anterior nos sugiere una forma de evaluar la potencia eP mediante el cálculo de una sucesión de puntos sobre la curva, $P_1, P_2, \cdots, P_s$, donde

$$\begin{aligned} P_1 &= \mu_{s-1}P \\ P_i &= m \cdot P_{i-1} + \mu_{s-i}P \end{aligned} \tag{8.8}$$

para $i = 2, \cdots, s$. Puede verificarse con facilidad que $P_s = eP$.

El método m-ario requiere un procesamiento previo que puede realizarse eficientemente en paralelo [33]. En [88] se aplica una versión del método m-ario para la multiplicación de puntos sobre curvas elípticas. Una mejora al método m-ario es el método de ventana deslizante [51]. Se pueden lograr algoritmos secuenciales más rápidos utilizando cadenas aditivas [104].

8.3.3. Cadenas aditivas

Dado un entero $e \in \mathbb{N}$, una *cadena aditiva* para e es una secuencia de pares $(j_1, k_1), \ldots, (j_L, k_L)$ con $0 \leq j_i, k_i < i$ para toda $1 \leq i \leq L$ y si se define $a_0, \ldots, a_L \in \mathbb{N}$ como $a_0 = 1$ y $a_i = a_{j_i} + a_{k_i}$, para $1 \leq i \leq L$, entonces se cumple que $a_L = e$. La *longitud* de la cadena aditiva es el entero L.

Dada una cadena aditiva para el entero e, se puede calcular eP, donde P es un punto sobre una curva elíptica, mediante el cálculo sucesivo de los puntos a_iP, para $1 \leq i \leq L$. El número de operaciones a realizar será proporcional a la longitud L de la cadena aditiva. Por lo tanto, resulta de especial interés encontrar cadenas aditivas de longitud mínima. Sin embargo, se ha demostrado [104] que, dados dos enteros positivos e y k, el problema de determinar si existe una cadena aditiva para e con $L \leq k$ es NP-completo. Por lo tanto, los algoritmos existentes determinan cadenas aditivas que son razonablemente cortas [104].

Más aún, Schönhage [85] demostró que si $l(e)$ denota la L más pequeña para la cual existe una cadena aditiva de longitud L para e, entonces

$$l(e) \geq \log_2 e + \log_2 H_2(e) - 2{,}13$$

donde $H_2(e)$ es el peso de Hamming de e. En consecuencia, cualquier algoritmo del cálculo de exponentes basados en cadenas aditivas será de complejidad lineal sobre el número de bits del exponente, $\log_2 e$.

8.3.4. Método de adición y substracción

Una especialización de las cadenas aditivas a la exponenciación sobre curvas elípticas es el método de cadenas de adición y substracción [70], el cual ha sido utilizado para implementaciones eficientes de curvas de Koblitz [100]. En este método se utiliza el hecho de que en la aritmética de puntos sobre una curva elíptica realizar una sustracción es equivalente en complejidad a realizar una adición. Se ha mostrado [70] que el método binario requiere de aproximadamente 12 % más operaciones elípticas que el método de adición-sustracción.

Morain y Olivos [70] observaron que en las curvas elípticas el inverso de un punto puede ser obtenido facilmente. Para curvas $y^2 = x^3 + ax + b$ sobre $GF(p)$ con $p > 3$, el inverso de (x, y) es $(x, -y)$ y para $y + xy = x^3 + ax^2 + b$ sobre $GF(2^m)$, el inverso de $(x, x + y)$. Entonces podemos considerar representaciones de la forma

$$k = \sum_{i=0}^{m-1} c_i 2^i \tag{8.9}$$

con $c_i \in \{-1, 0, 1\}$ para $i = 0, \ldots, m$.

A la representación del entero k como la expansión binaria con signo (8.9) tal que $c_i c_{i+1} = 0$ para $i = 0, \ldots, m-1$ se le denomina *forma no adyacente* (FNA). La FNA tiene la propiedad de que no hay dos coeficientes consecutivos que sean diferentes de cero. Por ejemplo,

$$FNA(29) = (1, 0, 0, -1, 0, 1) \tag{8.10}$$

dado que $29 = 34 - 4 + 1$.

Así como cada entero positivo tiene una única expansión binaria, también posee una FNA única. Más aún, la FNA posee menos coeficientes diferentes de cero que cualquier expansión binaria con signo de k [39]. A continuación mostraremos un algoritmo para encontrar la FNA de un entero.

Algoritmo de generación de la FNA

La entrada del algoritmo es un entero positivo k y la salida es FNA(k).

```
Hacer c ← k
Hacer S ← ⟨⟩
Mientras c > 0 hacer
        Si c es impar
            u ← 2 − (c mod 4)
            c ← c − u
        sino
            u ← 0
        S ← ⟨u, S⟩
        Hacer c ← c/2
Fin
Salida S
```

Obsérvese que, si bien hemos enunciado nuestro algoritmo en términos de aritmética entera, puede ser implementado en términos de operaciones de bits sobre la expansión binaria de k.

Algoritmo de adición-substracción

El algoritmo previo puede ser modificado como se muestra a continuación para dar origen a un algoritmo de multiplicación escalar para puntos sobre curvas elípticas.

La entrada del algoritmo es un entero positivo k y un punto P sobre la curva elíptica y la salida es el punto kP.

```
Hacer c ← k
Hacer Q ← O, P0 ← P
Mientras c > 0 hacer
        Si c es impar
            u ← 2 − (c mod 4)
            c ← c − u
            Si u = 1 entonces Q ← Q + P0
            Si u = −1 entonces Q ← Q − P0
        Hacer c ← c/2
        Hacer P0 ← 2P0
Fin
```

`Salida` Q

Se ha mostrado [70] que el peso de Hamming H de una FNA(k) satisface

$$H \approx \frac{1}{3} \log_2 k. \tag{8.11}$$

Por lo tanto, el algoritmo de adición-substracción requiere de $\sim m$ duplicaciones y $\sim m/3$ sumas. Esto mejora al método binario clásico, en el cual se utiliza la expansión binaria en lugar de la FNA. Se ha encontrado que el método binario requiere aproximadamente

8.3.5. Método de ventana

El método de adición-substracción puede ser generalizado para producir aún algoritmos más eficientes suponiendo que hay memoria suficiente disponible y que se han hecho algunos cálculos previos. El método que presentaremos es el denominado *método de ventana de ancho* w.

Sea w un entero mayor a 1. Entonces cada entero positivo posee una única *FNA de ancho* w, una expansión como

$$k = \sum_{j=0}^{l-1} u_j 2^j \tag{8.12}$$

donde:

- cada u_j diferente de cero es impar y menor en valor absoluto que 2^{w-1};
- entre cualquier w coeficientes consecutivos a lo más uno es diferente de cero.

Cuando $w = 2$ tenemos las FNA ordinarias.

La FNA de ancho w se escribe como:

$$\text{FNA}_w(k) = \langle u_{l-1}, \ldots, u_0 \rangle.$$

A continuación especificaremos un algoritmo para calcular la FNA_w.

Algoritmo de generación de la FNA_w

El algoritmo toma como entrada a un entero positivo k y da como salida FNA_w.

```
Hacer c ← k
Hacer S ← ⟨⟩
Mientras c > 0 hacer
        Si c es impar
            u ← 2 − (c mod 2^w)
            c ← c − u
        sino
            u ← 0
        S ← ⟨u, S⟩
        Hacer c ← c/2
Fin
Salida S
```

Dada una FNA de ancho w, se puede cálcular la multiplicación escalar por k mediante el algoritmo que describimos a continuación.

Algoritmo de adición-substracción de ancho w

La entrada del algoritmo es un entero positivo k y un punto P sobre la curva elíptica. La salida es el punto kP. Antes de poder ejecutar el algoritmo hay que hacer los siguientes cálculos previos:

```
Hacer P_0 ← P
Hacer P_{2^{w-2}-1} ← 2P_0
Para  i ← 1 hasta 2^{w-2} − 1
    Hacer P_i ← P_{i-1} + P_{2^{w-2}-1}
```

Y el algoritmo es como sigue:

```
Calcular FNA_w(k) = ⟨u_{l-1}, ..., u_0⟩
Hacer Q ← O
Para  j ← l − 1 hasta 0
   Hacer Q ← 2Q
   Si u_j ≠ 0
          i ← (|u_j| − 1)/2
          Si u_j > 0
             Hacer Q ← Q + P_i
          sino
             Hacer Q ← Q − P_i
Salida Q
```

La densidad promedio de una FNA_w es $(w+1)^{-1}$. Por lo tanto pueden reducirse el número de adiciones elípticas en una multiplicación escalar, sin embargo el método no reduce el número de duplicaciones respecto a las que se realizan en el método binario.