Capítulo 9

Algoritmo paralelo de multiplicación escalar

9.1. Introducción

Dado un entero positivo n y un punto P sobre una curva elíptica E, el cálculo de nP, esto es, el resultado de sumar n veces el punto P a si mismo, denominada la *multiplicación escalar*, es la operación principal de los criptosistemas de curvas elípticas. En este capítulo presentamos un algoritmo que, utilizando p procesadores, puede calcular nP en tiempo $O(\log n + H(n)/p + \log p)$, donde $H(n)$ es el peso de Hamming de n. Más aún, si este algoritmo es aplicado a las curvas de Koblitz, el tiempo de ejecución se reduce a $O(H(n)/p + \log p)$.

9.2. Multiplicación escalar

Se han reportado varias implementaciones rápidas de criptosistemas de curvas elípticas [42, 40, 88, 20, 21]. La seguridad de los criptosistemas de curvas elípticas se basa en la dificultad del problema del logaritmo discreto sobre una curva elíptica, y por lo tanto la operación principal de estos criptosistemas es la *exponenciación*, también conocida como *multiplicación escalar*. Para una revisión general de los métodos de exponenciación ver [39]. Con el objeto de lograr multiplicación escalar rápida sobre curvas elípticas se han considerado tres factores: el campo de definición [20, 88], cadenas aditivas [20, 53, 70, 88] y sistemas de coordenadas óptimos [18, 21]. También se han considerado curvas elípticas con aritméticas eficientes [50, 99, 100]. En este trabajo proponemos la aplicación de procesamiento paralelo para

acelerar la exponenciación sobre curvas elípticas.

Existen actualmente algoritmos paralelos para la exponenciación de enteros módulo un un entero de m bits [9, 1, 59] y también para exponenciación sobre los campos $GF(2^m)$ [101] y $GF(q^m)$ [104]. En este capítulo presentamos un algoritmo paralelo para calcular multiplicación escalar sobre curvas elípticas. Este algoritmo sería especialmente eficiente para curvas binarias anómalas, también conocidas como curvas de Koblitz. La implementación de nuestro algoritmo conduciría a implementaciones eficientes de criptosistemas de curvas elípticas en computadoras de múltiples procesadores.

9.3. Método binario

Tenemos una curva elíptica E y un punto $P \in E$. Deseamos calcular nP, para algún entero no negativo n. Sea $n = (b_{N-1}b_{N-2}b_{N-3}\cdots b_2b_1b_0)_2$ la representación binaria de n, donde

$$N = \lfloor \log n \rfloor + 1. \tag{9.1}$$

Entonces

$$n = b_{N-1}2^{N-1} + b_{N-2}2^{N-2} + \cdots + b_2 2^2 + b_1 2 + b_0. \tag{9.2}$$

Utilizando la expansión de Horner, la última expresión se vuelve

$$n = (\cdots((b_{N-1}2 + b_{N-2})2 + \cdots + b_1)2 + b_0). \tag{9.3}$$

De (9.3) tenemos

$$nP = 2(\cdots 2(2(2(b_{N-1}P) + b_{N-2}P) + \cdots + b_1P) + b_0P. \tag{9.4}$$

La última expresión nos sugiere una forma de obtener nP. Definimos la sucesión de puntos $P_1, \cdots, P_N$ de la siguiente manera. Sea

$$\begin{aligned} P_1 &= P \\ P_i &= \begin{cases} 2P_{i-1} & \text{si } b_{N-i} = 0 \\ 2P_{i-1} + P & \text{si } b_{N-i} = 1 \end{cases} \end{aligned} \tag{9.5}$$

para $i = 2, \cdots, N$. Podemos observar fácilmente que $P_N = nP$.

El método arriba delineado, para obtener nP mediante el cálculo de los términos $P_1, \cdots, P_N$, es conocido como *algoritmo binario* [48].

De (9.5) podemos deducir que para obtener el término P_i necesitamos una *duplicación* y, si el correspondiente bit de la representación binaria de n es uno, una suma de puntos. Este cálculo es realizado $N-1$ veces, donde $N = \lfloor \log n \rfloor + 1$. Si $H(n)$ es el *peso binario de Hamming* de n, esto es, el número de bits en uno en la representación binaria de n, entonces necesitamos $H(n)-1$ suma de puntos además de los $N-1$ doblamientos. Por lo tanto podemos enunciar el siguiente teorema

Teorema 9.1 *Si P es un punto sobre una curva elíptica E y n es un entero positivo, para obtener nP por el método binario se requieren $\lfloor \log n \rfloor$ duplicaciones y $H(n)-1$ sumas de puntos.*

Si suponemos que el tiempo que toma sumar dos puntos de la curva elíptica es casi el mismo que se toma el hacer una duplicación, y denotamos a este tiempo como t, podemos concluir a partir del teorema 9.1 el siguiente:

Corolario 9.2 *El tiempo de ejecución del algoritmo binario es*

$$(\lfloor \log n \rfloor + H(n) - 1) \cdot t,$$

donde t es el tiempo que toma una sola operación sobre la curva elíptica.

Como hemos visto en el capítulo anterior, se puede obtener una mejora al método binario mediante el uso de sistemas numéricos redundantes Morain and Olivos [70] observaron que sobre curvas elípticas el inverso de un punto puede obtenerse sin costo adicional. Para las curvas $y^2 = x^3 + Ax + B$ sobre $GF(p)$ con $p > 3$, el inverso de (x, y) es $(x, -y)$ y, para $y^2 + xy = x^3 + Ax^2 + B$ sobre $GF(2^m)$, el inverso es $(x, x + y)$. Entonces, podemos considerar representaciones de la forma

$$n = \sum_{i=0}^{N-1} c_i 2^i \tag{9.6}$$

donde $c_i \in \{-1, 0, 1\}$ para $i = 0, \ldots, N-1$. Una *forma no adyacente(FNA)* es una representación con $c_i c_{i+1} = 0$ para $0 \leq i < N-1$. Dado que la FNA tiene en general más ceros, entonces la ventaja de usarla es un menor número de adiciones. Morain and Olivos [70] mostraron que el número de dígitos diferentes a cero en una FNA es $N/3$.

9.4. Método binario de orden p

En esta sección presentaremos una generalización del método binario que hace uso del paralelismo cuando se tienen p procesadores.

Primero, dividimos la representación binaria de n en $\lceil N/p \rceil$ bloques de p bits cada uno y entonces dividimos a n en $s_0, s_1, \ldots, s_p$ tal que la representación binaria de cada s_i esté formada por $\lceil N/p \rceil$ bloques de p bits puestos en 0 a excepción del bit i-ésimo, el cual tiene el mismo valor que el i-ésimo bit del bloque correspondiente en la representación binaria de n. Entonces, podemos definir a s_i, para $i = 0, \ldots, p-1$, como sigue:

$$\begin{aligned} s_0 &= b_0 + b_p 2^p + b_{2p} 2^{2p} + \cdots + b_{(\lceil \frac{N}{p} \rceil - 1)p} 2^{(\lceil \frac{N}{p} \rceil - 1)p} \\ s_1 &= b_1 2 + b_{p+1} 2^{p+1} + b_{2p+1} 2^{2p+1} + \cdots + b_{(\lceil \frac{N}{p} \rceil - 1)p+1} 2^{(\lceil \frac{N}{p} \rceil - 1)p+1} \\ &\vdots \\ s_{p-1} &= b_{p-1} 2^{p-1} + b_{2p-1} 2^{2p-1} + b_{3p-1} 2^{3p-1} + \cdots + b_{p\lceil \frac{N}{p} \rceil - 1} 2^{p\lceil \frac{N}{p} \rceil - 1}. \end{aligned} \tag{9.7}$$

Esto es

$$s_i = \sum_{j=0}^{\lceil \frac{N}{p} \rceil - 1} b_{jp+i} 2^{jp+i} \tag{9.8}$$

para $i = 0, 1, \ldots, p-1$, donde estamos considerando algunos *bits de relleno* $b_k = 0$ para $N - 1 < k \leq p\lceil N/p \rceil - 1$. El conjunto entero $\{s_0, s_1, \cdots, s_{p-1}\}$ puede ser obtenido fácilmente a partir de n haciendo XOR con la máscara apropiada. Es claro a partir de (9.2) y (9.7) que

$$n = s_0 + s_1 + \cdots + s_{p-1} \tag{9.9}$$

y por lo tanto, podemos calcular nP como la suma de $s_0P, s_1P, \ldots, s_{p-1}P$.

Ahora, podemos delinear nuestro *método binario de orden p* en dos etapas de la forma siguiente:

1. *Dispersión de bits:* Calcular el conjunto $\{s_0, s_1, \ldots, s_{p-1}\}$ tal como está definido en (9.7). Utilizando p procesadores, calcular en paralelo las multiplicaciones escalares

   $$s_0P, s_1P, \ldots, s_{p-1}P$$

   ejecutando el método binario en cada procesador.

2. *Abanico asociativo:* Utilizando $\lceil p/2 \rceil$ procesadores calcular en paralelo

$$nP = s_0P + s_1P + \ldots + s_{p-1}P.$$

Para hacer esto, primero separamos la suma total como

$$\sum_{i=0}^{p-1} s_i = \sum_{i=0}^{\lceil p/2 \rceil - 1} s_i + \sum_{i=\lceil p/2 \rceil}^{p-1} s_i,$$

y entonces ambas mitades pueden calcularse en paralelo. Procedemos de igual manera recursivamente hasta que solo se requiere una sola adición de dos puntos sobre la curva elíptica. Es claro que en el nivel más profundo de la recursión tenemos que realizar $\lceil p/2 \rceil$ adiciones en paralelo, y para obtener la suma total tenemos que hacer $\lceil \log p \rceil$ pasos recursivos.

Podemos observar que si $p = 1$ entonces nuestro algoritmo se reduce al algoritmo binario común. A continuación, analizaremos el tiempo de ejecución del método binario de orden p.

Teorema 9.3 *El tiempo de ejecución del algoritmo binario de orden p, en el mejor de los casos, es*

$$\lfloor \log n \rfloor + \lceil H(n)/p \rceil + \lceil \log p \rceil - 1$$

y, en el peor caso,

$$\lfloor \log n \rfloor + H(n) - 1.$$

Demostración
Primero, podemos observar que el conjunto de los enteros $\{s_0, s_1, \cdots, s_{p-1}\}$ puede obtenerse a partir de n en un tiempo despreciable. En la primera etapa de nuestro algoritmo, cada procesador ejecuta el algoritmo binario para calcular s_iP, donde i es el índice del procesador. Del corolario 9.2, requerimos $\lfloor \log s_i \rfloor + H(s_i) - 1$ pasos para hacer este cálculo. Ahora, dado que $a_{N-1} = 1$, uno de los procesadores tiene que calcular $N-1$ duplicaciones y si suponemos que para que pueda iniciar la segunda etapa del algoritmo, deben haber terminado la primera todos los procesadores, entonces el tiempo de ejecución de la primera fase será

$$T_1 = N + H^* - 2, \tag{9.10}$$

donde

$$H^* = \max_{0 \le i \le p-1} H(s_i). \tag{9.11}$$

Dado que tenemos que

$$H(n) = \sum_{i=0}^{p-1} H(s_i) \tag{9.12}$$

entonces

$$\lceil H(n)/p \rceil \leq H^* \leq H(n). \tag{9.13}$$

En el mejor caso, todos los bits en uno en la representación binaria de n están igualmente dispersos entre los enteros $\{s_i\}$, de modo que la carga está perfectamente balanceada sobre todos los procesadores, y entonces tenemos

$$H^* = \left\lceil \frac{H(n)}{p} \right\rceil. \tag{9.14}$$

Y de (9.14), (9.10) y (9.1) tenemos que, en el mejor caso, el tiempo de la primera etapa está dado por

$$T_1 = \lfloor \log n \rfloor + \lceil H(n)/p \rceil - 1. \tag{9.15}$$

En la segunda fase de nuestro algoritmo, la suma $s_0P + \cdots + s_{p-1}P$ es calculada mediante el algoritmo de abanico asociativo en

$$T_2 = \lceil \log p \rceil \tag{9.16}$$

pasos, y por lo tanto el tiempo total de ejcución para el mejor caso será

$$T = T_1 + T_2 = \lfloor \log n \rfloor + \lceil H(n)/p \rceil + \lceil \log p \rceil - 1. \tag{9.17}$$

En el peor caso $H^* = H(n)$, lo cual implica por la definición de H^* (9.11) que existe algún índice k, $0 \leq k \leq p-1$, tal que $H(s_k) = H(n)$ y, de (9.12), $H(s_i) = 0$ para $i \neq k$ y entonces $s_i = 0$ para $i \neq k$. De (9.9) tenemos que $n = s_k$ y entonces $s_kP = nP$. Entonces tenemos que nP es calculado en la primera etapa por el procesador k. Por lo tanto el tiempo de ejecución será el mismo que el establecido en el corolario 9.2. □

Se puede lograr una mejora si se utiliza la FNA como entrada a nuestro algoritmo en lugar de la representación binaria de n. Dado que la FNA tiene un mayor número de ceros, entonces podemos obtener los mismos tiempos de ejecución utilizando menos procesadores. Pero antes de eso podemos observar que en nuestro algoritmo un cuello de botella es provocado por las duplicaciones realizadas en la primera etapa, representadas por el término lineal en N en (9.10), el cual es independiente del número p de procesadores.

En la deducción que hemos hecho del tiempo de ejecución de nuestro algoritmo, fue supuesto que el tiempo requerido para hacer una duplicación

o una adición de puntos es el mismo, pero en general depende del sistema de coordenadas en el cual esté representada la curva elíptica. Los sistemas de coordenadas más conocidos son los de coordenadas afines y proyectivas [92]. Otros dos sistemas de coordenadas, las coordenadas Jacobianas y Jacobianas de Chudnovsky, han sido propuestos en [18]. La eficiencia de coordenadas Jacobianas para el cálculo de exponentes sobre curvas elípticas ha sido discutida en [20]. En [21] se ha propuesto un sistema coordenado Jacobiano modificado, el cual dá duplicaciones más rápidas que las coordenadas afines, proyectivas, Jacobianas y Jacobianas de Chudnovsky. Por lo tanto podriamos utilizar este sistema coordenado Jacobiano modificado para superar parcialmente el cuello de botella de las duplicaciones en nuestro algoritmo.

Un enfoque más fructifero podría ser el uso de curvas elípticas especiales en las cuales las duplicaciones sean innecesarias del todo. Este enfoque se expondrá en la siguiente sección.

9.5. El método τ-ario de orden p

Las *curvas binarias anómalas*, propuestas por Koblitz [50], han sido utilizadas para obtener implementaciones eficientes de criptosistemas de curvas elípticas [62, 99, 100]. Estas curvas son

$$E_1 \ : \ y^2 + xy = x^3 + x^2 + 1 \tag{9.18}$$

y

$$E_2 \ : \ y^2 + xy = x^3 + 1 \tag{9.19}$$

sobre $GF(2^m)$. Para estas puede verificarse fácilmente que si el punto (x, y) está en la curva, entonces también está el punto (x^2, y^2). Por lo tanto, podemos definir el automorfismo de Frobenius como $\varphi(x, y) = (x^2, y^2)$, el cual corresponde a la multiplicación por $\tau = (1 + \sqrt{-7})/2$ en E_1 y por $-\bar{\tau} = (-1 + \sqrt{-7})/2$ en E_2. Utilizando bases normales sobre $GF(2^m)$, la multiplicación por τ requiere solamente dos desplazamientos cíclicos y estos pueden hacerse en un tiempo des-preciable.

Dado que τ es un elemento de norma 2 en el dominio euclideano $\mathbb{Z}[\tau]$ cualquier entero n tiene una representación de la forma siguiente

$$n = \sum_{i=0}^{\infty} c_i \tau^i \tag{9.20}$$

para $c_i \in \{0, 1\}$. Para cualquier $n \in \mathbb{Z}[\tau]$ una representación como (9.20) es denominada una FNA si $c_i \in \{0, \pm 1\}$ y $c_i c_{i+1} = 0$ para toda $i \geq 0$. Solinas

[99] mostró la existencia de una FNA de peso mínimo proporcionando un algoritmo para calcular directamente la FNA. Si $\tau | n$, entonces $c_0 = 0$. De otra forma, τ^2 divide a $n+1$ o a $n-1$ y la FNA termina en $(0,-1)$ o $(0,1)$, respectivamente. Este proceso se repite sobre n/τ, $(n+1)/\tau^2$ o $(n-1)/\tau^2$.

Como $\varphi^m(x,y) = (x^{2^m}, y^{2^m}) = (x,y)$, cualquier par de representaciones que coincidan modulo $\tau^m - 1$ originaran el mismo endomorfismo en la curva. Basandose en esta propiedad, Meier and Staffelbach [62] mostraron lo siguiente:

Teorema 9.4 *Toda $n \in \mathbb{Z}[\tau]$ tiene una representación*

$$n \equiv \sum_{i=0}^{m-1} c_i \tau^i \pmod{\tau^m - 1}, \tag{9.21}$$

donde $c_i \in \{0, \pm 1\}$.

El método binario puede extenderse en una forma natural al método τ-ario basandose en la representación de n dada por (9.21). Podemos entonces definir un método τ-ario de orden p paralelo.

Utilizando (9.21) podemos definir al conjunto $\{s_0, \cdots, s_{p-1}\} \subseteq \mathbb{Z}[\tau]$, donde

$$s_i = \sum_{j=0}^{\lceil \frac{m}{p} \rceil - 1} c_{jp+i} \tau^{jp+i} \tag{9.22}$$

para $i = 0, 1, \ldots, p-1$.

El *método τ-ario de orden p* consiste de las siguientes dos etápas:

1. *Dispersión de bits:* Utilizando p procesadores calcular en paralelo

   $$s_0 P, s_1 P, \cdots, s_{p-1} P$$

   corriendo el algoritmo τ-ario en cada procesador.

2. *Abanico asociativo:* Utilizando $\lceil p/2 \rceil$ procesadores calcular

   $$\sum_{k=0}^{p-1} s_k P$$

   con el algoritmo de abanico asociativo, en $\lceil \log p \rceil$ pasos.

Si definimos $H(n)$ como el número de dígitos distintos a cero en la representación dada por (9.21), podemos establecer el siguiente teorema:

Teorema 9.5 *El tiempo de ejecución del método τ-ario de orden p, en el mejor de los casos, es*

$$\lceil H(n)/p \rceil + \lceil \log p \rceil - 1$$

y

$$H(n) - 1$$

en el peor caso.

Demostración
La demostración de este teorema es la misma que la del teorema 9.3 excepto que el tiempo para la fase uno será

$$T_1 = H^* - 1 \tag{9.23}$$

donde H^* está definido como en (9.11). □

Meier y Staffelbach [62] conjeturaron, basandose en evidencia experimental, que en promedio la mitad de los c_i serán diferentes a cero. De modo que podemos esperar que cuando $p \approx m/2$ el tiempo de ejecución sea cercano a $\log H(n)$. Pueden lograrse aún mayores mejoras a la representación τ-aria para reducir el número de dígitos diferentes a cero [100], de modo que se requieran un número menor de procesadores para alcanza iguales velocidades de procesamiento.