BASIC ALGORITHMS FOR THE MULTIPLICATION OF THE POINTS IN AN ELLIPTICAL CURVE

ALGORITMOS BÁSICOS PARA LA MULTIPLICACIÓN DE PUNTOS EN UNA CURVA ELÍPTICA

Recibido: 15 de julio 2013- aceptado: 04 de diciembre 2013

Ricardo Villanueva Polanco.[1]
Universidad Simón Bolívar

keywords:
Algorithms, Elliptic Curves, Fields, Point Multiplication.

Abstract
Elliptic Curve Cryptography was introduced by Neal Koblitz and Victor Miller in 1985. The reason why is so attractive is that there is no known efficient algorithm to solve the logarithm problem. Is very important to improve the running time of the algorithms used for elliptic curve implementation. Therefore, in this article are described the basic algorithms for point multiplication in an elliptic curve. As the reading progresses, improved techniques are introduced and information is provided to know how to efficiently implement these methods with real parameter.

Palabras clave:
Algoritmos, Curvas Elípticas, Campos, Multiplicación de Puntos.

Resumen
La criptografía de curva elíptica fue introducida por Neal Koblitz y Víctor Miller en el año de 1985. La razón por la cual es atractiva, es que no se conocen algoritmos eficientes para resolver el problema del logaritmo discreto. Es muy importante además, que se mejoren los tiempos de ejecución de los algoritmos usados para la implementación de las curvas elípticas. Por consiguiente, en este artículo se describen algoritmos básicos para la multiplicación de puntos en una curva elíptica. A medida que se avanza en la lectura, se detallan técnicas más eficientes y se brindan ciertas recomendaciones para la implementación eficiente de estos métodos con parámetros reales.

1. magister en ingeniería de sistemas y computación. Universidad Simón Bolívar. E-mail: rvillanueva1@unisimonbolivar.edu.co
* Los Análisis de algoritmos criptográficos de llave pública con curvas elípticas sobre campos finitos

I. INTRODUCCIÓN

Através de los años, la criptografía de llave pública se ha convertido en una herramienta necesaria para proveer seguridad a los sistemas informáticos. Cripto-sistemas tales como ElGamal y RSA, necesitan de algoritmos eficientes para la exponenciación en un grupo determinado. Actualmente existen un sin número de técnicas para realizar esta operación pero debido a la importancia de este proceso, siempre se está a la búsqueda de nuevas versiones de implementación. El grupo definido sobre una curva elíptica no es la excepción, y gran parte de los algoritmos que funcionan en un grupo genérico, se pueden trasladar sin dificultad al grupo de las curvas elípticas.

En este artículo se explorarán algunos de estos métodos, su adecuación al grupo definido sobre las curvas elípticas y algunas recomendaciones para su implementación. En la primera sección de este artículo se detallarán los conceptos básicos de curvas elípticas definidas sobre campos de la forma F_P y curvas no supersingulares sobre F_{2^n} .

II. PRELIMINARES

A. Aritmética de curvas elípticas definidas sobre F_p

La curvas elípticas definidas sobre F_p son de la forma $y^2 = x^3 + ax + b$ donde $a, b \in F_p$ y el discriminante $4a^3 + 27b^2 \bmod p \neq 0$. El conjunto $E(F_p)$ definido como sigue:

$$E(F_p) = \{(x, y) \,|\, y^2 = x^3 + ax + b; x, y, a, b \in F_p\} \cup \{\infty\}$$

Forma un grupo abeliano bajo las siguientes operaciones:

1. Identidad: $P + \infty = \infty + P = P \;\forall\; P \in E(F_p)$.
2. Negativo: Si $P = (x, y) \in E(F_p)$, entonces $(x, y) + (x, -y) = \infty$. El punto $(x, -y)$ se denota como $-P$ y se llama negativo de P . Cabe agregar que $-\infty = \infty$.
3. Suma de puntos: Sean $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2) \in E(F_p)$ con $P_1 \neq \pm P_2$ .Entonces $P_1 + P_2 = (x_3, y_3)$ donde

$$x_3 = \frac{(y_2 + y_1)^2}{(x_2 + x_1)^2} - x_1 - x_2 \quad y \quad y_3 = \frac{y_2 + y_1}{x_2 + x_1}(x_1 - x_3) - y_1$$

4. Doble de un punto: Sea $P = (x_1, y_1) \in E(F_p)$ con $P \neq -P$. Entonces $2P = (x_2, y_2)$ donde

$$x_2 = \frac{(3x_1^2 + a)^2}{(2y_1)^2} - 2x_1 \qquad y$$

$$y_2 = \frac{3x_1^2 + a}{2y_1}(x_1 - x_2) - y_1$$

Ejemplo. Curvas elípticas sobre F_{29} .

Sea $E : y^2 = x^3 + 4x + 20$ una curva elíptica sobre F_{29} con $4a^3 + 27b^2 \bmod 29 = 7 \neq 0.$

Es fácil ver qué $P_1 = (5, 22)$ y $P_2 = (16, 27) \in E(F_{29})$. Entonces, $(5, 22) + (16, 27) = (13, 6)$ y $2P_1 = 2(5, 22) = (14, 6)$. Se puede ver que los puntos resultantes pertenecen a $E(F_{29})$.

Si el proceso de suma de puntos se realiza siguiendo los pasos descritos a continuación,

1) $\lambda = \dfrac{y_2 - y_1}{x_2 - x_1}$.
2) $x_3 = \lambda^2 - x_1 - x_2$.
3) $y_3 = \lambda(x_1 - x_3)$.
4) $y_3 = y_3 - y_1$.

Las operaciones requeridas son una división (inversión), dos multiplicaciones y 5 sumas.

Por lo general la operación más costosa es la división. Cuando la razón de los tiempos de ejecución y la multiplicación es baja (1, 2, 3, 4), se recomienda utilizar el proceso anterior para calcular la suma o el doble de un punto. En caso contrario, se recomienda utilizar coordenadas proyectivas[1]. Ver [1], [2], [3] y [4]

B. Aritmética de curvas elípticas definidas sobre F_{2^n}

Las curvas elípticas definidas sobre F_{2^n} son de dos tipos: supersingulares y no supersingulares[2].

Las curvas no supersingulares sobre F_{2^n} se expresan de la forma $y^2 + xy = x^3 + ax^2 + b$ donde $a, b \in F_{2^n}$ y $b \neq 0$. El conjunto $E(F_{2^n})$ definido como sigue:

[1]Esta distinción se hace porque las curvas elípticas pueden ser implementadas en software, hardware y/o firmware.

[2]Las curvas usadas en criptografía son las no súper-singulares debido a que las primeras sin fáciles de vulnerar con ciertas técnicas específicas. Ver [1], [2], [3] y [4].

$$E(F_{2^n}) = \{(x,y) \mid y^2 + xy = x^3 + ax^2 + b; x, y, a, b \in F_{2^n}\} \cup \{\infty\}$$

Forma un grupo abeliano bajo las siguientes condiciones:

1. Identidad: $P + \infty = \infty + P = P \ \forall \ P \in E(F_{2^n})$.
2. Negativo: Si $P = (x,y) \in E(F_{2^n})$, entonces $(x,y) + (x, x+y) = \infty$. El punto $(x, x+y)$ se denota como $-P$ y se llama negativo de P. Cabe agregar que $-\infty = \infty$.
3. Suma de puntos: Sean $P_1 = (x_1, y_1)$ y $P_2 = (x_2, y_2) \in E(F_{2^n})$ con $P_1 \neq \pm P_2$. Entonces $P_1 + P_2 = (x_3, y_3)$ donde

$$x_3 = \frac{(y_2 + y_1)^2}{(x_2 + x_1)^2} + \frac{y_2 + y_1}{x_2 + x_1} + x_1 + x_2 + a \quad y$$

$$y_3 = \frac{y_2 + y_1}{x_2 + x_1}(x_1 + x_3)x_3 + y_1$$

4. Doble de un punto: Sea $P = (x_1, y_1) \in E(F_{2^n})$ con $P \neq -P$. Entonces $2P = (x_2, y_2)$ donde

$$x_2 = \frac{(x_1 + y_1)^2}{(x_1)^2} + \frac{x_1 + y_1}{x_1} + a \quad y$$

$$y_2 = x_1^2 + \frac{x_1 + y_1}{x_1}(x_3) + x_3$$

Curvas elípticas sobre F_{2^4}.

Sea:

$$E : y^2 + xy = x^3 + z^3x^2 + (z^3 + 1)$$

Una curva no súper-singular definida sobre F_{2^4}, el cual está representado por el polinomio $f(z) = z^4 + zx + 1$ y cuyos elementos se representan por una cadena de bits de la siguiente manera $(a_3a_2a_1a_0)$. Por ejemplo, (0110) representa el polinomio $z^2 + z$.

Puede verificarse fácilmente que $P_1 = (0010,1111)$ y $P_2 = (1100,1100) \in F_{2^4}$. Entonces, es claro que $\mathrm{P1} + \mathrm{P2} = (0010,1111) + (1100,1100) = (0001,0001)$ y $2\mathrm{P1} = 2(0010,1111) = (1011,0010)$. Se puede ver que los puntos resultantes pertenecen a $E(F_{2^n})$.[3]

En forma análoga a las curvas sobre F_p, si se utilizan las definiciones de suma y del doble de un punto descritas anteriormente, el proceso utiliza una división, dos multiplicaciones y cinco sumas.

Dependiendo de la razón entre los tiempos de ejecución de la división y la multiplicación se puede optar por otras alternativas para representar los puntos, por ejemplo las coordenadas proyectivas. Las más utilizadas son las coordenadas de López - Dahab. Ver[5]

III. MULTIPLICACIÓN DE PUNTOS

La operación central de los esquemas criptográficos que se basan en curvas elípticas, es la multiplicación de un entero k por un punto P. Por eso, en esta sección se describirán algoritmos para calcular esta operación.

A. Método binario

Este método se basa en la representación binaria de un entero k. Si $k = \sum_{i=0}^{l-1} k_j 2^j$, donde cada $k_j \in \{0,1\}$, entonces kP puede calcularse así:

$$kP = \sum_{i=0}^{l-1} k_j 2^j P = 2(\ldots 2(2k_{l-1}P + k_{l-2}P) + \ldots) + k_0P$$

1) Algoritmo de método binario No 1.

A continuación se presenta el algoritmo de método binario de izquierda a derecha.

Entrada : $k = (k_{t-1}, k_{t-2}, \ldots, k_1, k_0)_2$ *y* $P \in E(K)$
donde K es un campo.
Salida : *kP*.

Inicio
1. $Q = \infty$
2. for (i=t-1 to 0)
   2.1. $Q = 2Q$
   2.2. if ($k_i = 1$) then $Q = Q + P$
3. return Q
Fin

2) Algoritmo de método binario No 2.

A continuación se presenta el algoritmo de método binario de izquierda a derecha.

Entrada : $k = (k_{t-1}, k_{t-2}, \ldots, k_1, k_0)_2$ *y* $P \in E(K)$
donde K es un campo.
Salida : *kP*.

Inicio
1. $Q = \infty$
2. for (i=0 to $t-1$)
   2.1. if ($k_i = 1$) then $Q = Q + P$
   2.2. $P = 2P$
3. return Q
Fin

[3] Las operaciones se realizan con aritmética modular sobre el polinomio de reducción.

Los métodos anteriormente descritos requieren de t dobles y w_k [4] sumas, donde w_k es el peso[5]. Por lo tanto, en tiempo esperado de ejecución es $t*D+w_k*A$, donde A y D representan los tiempos de ejecución de la suma y el doble de un punto respectivamente.

3) *Ejemplo.*

Sea $n=51=(110011)_2$ y P un punto de $E(K)$ donde K es un campo.

El algoritmo No 1 se ejecutaría como sigue:

$Q=\infty$

Primera Iteracción : $i=5\ Q=P.$

Segunda Iteracción : $i=4\ Q=3P.$

Tercera Iteracción : $i=3\ Q=6P.$

Cuarta Iteracción : $i=2\ Q=12P.$

Quinta Iteracción : $i=1\ Q=24P+1P=25P.$

Sexta Iteracción : $i=0\ Q=50P+1P=51P$

B. Forma no adyacente (NAF, por sus siglas en inglés)

Si $P=(x,y)\in E(F_q)$, entonces $-P=(x,-y)$ si F_q tiene característica mayor que 3. Lo anterior implica que restar es tan eficiente como sumar, lo que motiva la siguiente definición.

1) *Definición*

Una representación con signo de un entero n en base b está dada por:

$$n=\sum_{t=0}^{l-1} n_i b^i \ con\ |n_i|<b$$

Cuando $b=2$, a la representación del entero n se le conoce con el nombre de representación binaria con signo con $n_i\in\{-1,0,1\}$ [6].

Cabe anotar que las representaciones con signo no son únicas, por ejemplo, si $b=2$, $n=7$ se puede expresar como $100\bar{1}$ ó $1\bar{1}00\bar{1}$.

2) *Definición*

Una representación NAF de un entero k es una expresión $k=\sum_{i=0}^{l-1} k_i 2^i$, donde $k_i\in\{-1,0,1\}, k_{l-1}\neq 0$ y $k_i k_{i+1}=0 (i\geq 0)$. La longitud del NAF es l.

3) *Teorema.*

Sea k un entero positivo.

1) k tiene una única representación NAF denotada $NAF(k)$.
2) La representación $NAF(k)$ tiene menor número de dígitos diferentes de cero que cualquier otra representación con signo de k.
3) La longitud de $NAF(k)$ es a lo más $m+1$ donde m es la longitud de la representación binaria de k.
4) Si la longitud de $NAF(k)$ es l, entonces $\frac{2^l}{3}\leq k\leq \frac{2^{l+1}}{3}$
5) El peso esperado de $NAF(k)$ con longitud l es $l/3$.

4) *Algoritmo para el cálculo del NAF.*

El siguiente algoritmo calcula el $NAF(k)$ eficientemente.

Entrada : *Entero positivo k*.

Salida : $NAF(k)$.

Inicio

1. i=0
2. *while*$(k\geq 1)$
   - 2.1. if (k *es impar*) then $k_i=2-(k(\mathrm{mod}\,4)), k=k-k_i$
   - 2.2. *else* $k_i=0$
   - 2.3. $k=k/2, i=i+1$
3. return $(k_{i-1},k_{i-2},\ldots,k_1,k_0)$

Fin

El algoritmo anterior calcula correctamente la representación NAF de un entero k porque si k es impar, $k \bmod 4=1$ ó -1 y $k-k_i$ es divisible entre 4, asegurando en la próxima iteración un cero.

Del teorema anterior, se observa que si se utiliza esta representación, el número de sumas de los algoritmos anteriores podría reducirse, ya que el peso esperado de la representación binaria de un entero k de longitud t es $t/2$.

5) *Ejemplo*

Sea $k=15$, entonces el algoritmo inmediatamente anterior se ejecutaría así:

$i=0$

Primera Iteracción : $i=0\ k=15, k_0=\bar{1}, k=16, k=8.$

Segunda Iteracción : $i=1\ k_1=0, k=4$

Tercera Iteracción : $i=2\ k_2=0, k=2.$

Cuarta Iteracción : $i=3\ k_3=0, k=1.$

Quinta Iteracción : $i=4\ k_4=1, k=0, k=0$

Entonces, $NAF(15)=1000\bar{1}$

6) *Algoritmo binario con representación NAF*

A continuación se detalla el algoritmo.

[4] w_k es aproximadamente $t/2$

[5] El peso es el número de entradas diferentes de cero en la representación en base de un entero positivo k

[6] -1 se denota $\bar{1}$

Entrada : *Entero positivo* k, *un punto* $P \in E(F_q)$
Salida : kP.

Inicio
1. *Computar* $NAF(k)$
2. $Q = \infty$
3. for($j = l-1$ *to* 0)
   3.1. $Q = 2Q$
   3.2. if($u_j = 1$) *then* $Q = Q + P$
   3.2. if($u_j = \bar{1}$) *then* $Q = Q - P$
4. return Q
Fin

El algoritmo anterior es una modificación del algoritmo de método binario No 1, utilizando $NAF(k)$ en lugar de la representación binaria de k. Se sigue de los incisos 3 y 5 del teorema que en que se encuentra en líneas anteriores que el tiempo esperado del algoritmo es $\frac{l}{3}A + ID$, donde l es la longitud de $NAF(k)$ y A y D representan le tiempo de ejecución para sumar y doblar respectivamente.

C. Métodos de ventanas.

A continuación se introducirán unas generalizaciones de la representación NAF.

1) *Definición.*

Sea $w \geq 2$ un entero positivo. Una representación NAF de longitud w de un entero positivo k es de la forma $k = \sum_{i=0}^{l-1} k_i 2^i$ donde cada coeficiente diferente de cero k_i es impar, $|k_i| < 2^{w-1}, k_{l-1} \neq 0$ y para cualquier w dígito consecutivo hay a lo más un dígito distinto de cero. La longitud de la representación NAF de longitud w es l.

2) *Teorema.*

Se cumplen las siguientes propiedades:

1) k tiene una única representación NAF de longitud w denotada $NAF_w(k)$.
2) $NAF_2(k) = NAF(k)$
3) La longitud de $NAF_w(k)$ es $l+1$, donde l es la longitud de la representación binaria de k.
4) El peso esperado de la representación NAF de longitud w con longitud l es $\frac{l}{w+1}$.

3) *Algoritmo de representación NAF de longitud w*

A continuación se detalla el algoritmo.

Entrada : *Entero positivo* k, *vent* w
Salida : $NAF_w(k)$.

Inicio
1. $i = 0$
2. *while*($k \geq 1$)
   2.1. for(k *es impar*) *then* $k_i = k \bmod 2^w, k = k - k_i$
   2.2. *else* k_i
   2.3. $k = k/2, i = i+1$
4. return $(k_{i-1}, k_{i-2}, \ldots, k_1, k_0)$
Fin

En el algoritmo anterior, $k \bmod 2^w$ denota un entero u tal que $u = k \bmod 2^w$ y $-2^{w-1} \leq u \leq 2^{w-1}$. El algoritmo funciona correctamente porque cuando k es impar, $\frac{k-r}{2}$ es divisible por 2^w, donde $r = k \bmod 2^w$, asegurando que los $w-1$ dígitos siguientes sean cero.

4) *Algoritmo* NAF_w *para multiplicación de puntos,* $NAF_w(k)$.

Este algoritmo es una aplicación de NAF_w para la multiplicación de puntos en una curva elíptica.

Entrada : *Entero positivo* k, *vent* w, *un punto* $P \in E(F_q)$
Salida : kP

Inicio
1. *Computar* $NAF_w(K)$
2. *Computar* $P_i = iP$ *para* $i \in \{1, 3, 5, \ldots, 2^{w-1} - 1\}$
3. $Q = \infty$
4. for(i=l-1 to 0)
   4.1. $Q = 2Q$
   4.2. if($k_i \neq 0$) *then*
      4.2.1. if($k_i > 0$) *then* $Q = Q + P_{k_i}$
      4.2.2. *else* $Q = Q - P_{-k_i}$
5. return Q
Fin

Si la pre-computación se realiza de la siguiente manera: $Q = 2P, V_0 = P$ y $V_i = Q + V_{i-1}$ para todo $i \geq 1$, se tiene que el tiempo esperado de ejecución:

$$ID + (2^{w-2} - 1)A + \frac{l}{w+1}A + ID$$

Donde l es la longitud de $NAF_w(k)$.

Existen técnicas más avanzadas para la multiplicación de puntos pero son aplicables a cierto tipo de curvas, como por ejemplo, el método de Montgomery, método de reducción a la mitad (Point Halving), los cuales se aplican a curvas sobre campos binarios. También se definen métodos para las llamadas curvas de Kolbliz,

curvas binarias donde $A \in \{0,1\}$ y $B = 1$. Si se desea profundizar sobre estos temas ver [4]

IV. Multiplicación paralela de puntos

En muchos protocolos criptográficos se requiere calcular $nQ + kP$, donde $Q, P \in E(K)$ y n, k son enteros. Se podrían computar por separado y luego sumarse, pero es más ventajoso calcularlo simultáneamente. En esa sección, se describirán algoritmos que permiten realizar esta operación.

A. La estrategia de Shamir.

1) Algoritmo de la estrategia de Shamir.

A continuación se detalla el algoritmo que calcula simultáneamente $nQ + kP$.

Entrada : $k = (k_{s-1}, \ldots, k_0)_2, vent\ w, n = (n_{s-1}, \ldots, n_0)_2, P, Q \in E(F_q)$

Salida : $kP + nQ$

Inicio

1. *Computar* $iP + jQ$ *para todo* $i, j \in [0, 2^w - 1]$
2. *Escriba* $k = (K^{d-1}, \ldots, k^1, k^0)$ *y* $n = (N^{d-1}, \ldots, k^1, N^0)$, *donde cada* K^i, N^i *es una cadena de bits con longitud* w *y* $d = \lceil t/w \rceil$
3. $R = \infty$
4. for(i=d-1 to 0)
   - 4.1. $R = 2^w R$
   - 4.2. $R = R + (K^i P + N^i Q)$
5. return R

Fin

A continuación se mostrará un ejemplo en donde se muestra como trabaja el algoritmo anterior.

2) Ejemplo.

Sean $w = 2, n = (101101)_2 = 45$ y $k = (1100100)_2 = 100$ y $P, Q \in E(K)$.

a) Pre-computación

i	j	iP + jQ
0	0	∞
0	1	Q
0	2	2Q
0	3	3Q

i	j	iP + jQ
1	0	P
1	1	P + Q
1	2	P + 2Q
1	3	P + 3Q

i	j	iP + jQ
2	0	2P
2	1	2P + Q
2	2	2P + 2Q
2	3	2P + 3Q

i	j	iP + jQ
3	0	3P
3	1	3P + Q
3	2	3P + 2Q
3	3	3P + 3Q

Como $w = 2$, entonces las ventanas quedan escogidas así:

$$n = 00101101 \quad y \quad k = 01100100$$

$R = \infty$

Primera Iteracción : $R = P$.

Segunda Iteracción : $R = 4P + (2P + 2Q) = 6P + 2Q$.

Tercera Iteracción : $R = (24P + 8Q) + (P + 3Q) = 25P + 11Q$.

Cuarta Iteracción : $R = (100P + 44Q) + Q = 100P + 45Q$.

En el algoritmo anterior, los enteros n y k se pueden escribir en su representación $NAF(K)$ y así se reducen el número de sumas porque hay mayor número de cero simultáneos.

V. Recomendaciones

Dentro de las técnicas mencionadas a lo largo del artículo, las más utilizadas para implementaciones eficientes son las variantes de las representaciones con signo en un contexto general.

Si se desea aplicar sobre un conjunto de curvas específicas, existen métodos que aprovechan las características de dichas curvas optimizando los tiempos de respuesta de los algoritmos. En las curvas definidas sobre campos binarios, métodos como el de reducción a la mitad o Point Halving, ofrecen tiempos de respuesta muy reducidos. Para profundizar en este aspecto ver[6]

Para obtener mejores tiempos de respuesta, se deben utilizar operaciones con bits, tales como and, xor, shift, etc. Por ejemplo si se va a dividir un valor k entre 2^w, utilizar la instrucción $k \gg w$[7] o si se desea multiplicar, usar $k \ll w$[8].

Como recomendaciones adicionales se tienen las siguientes:

1) Si se está programando en lenguajes como C y C++, utilizar macros tanto como sea posible.
2) Disminuir el número de sentencias if en el código.
3) Evitar el uso de ciclos cortos.
4) La manipulación de bits pueden ser realizadas de manera eficiente con tablas pre-calculadas.
5) Trabajar con curvas definidas y recomendadas en estándares internacionales, por ejemplo, las curvas propuestas por la NIST (National Institute of Standards and Technology). Si se desea un análisis detallado de las curvas de la NIST y su implementación eficiente, ver.
6) Dependiendo del lenguaje se pueden utilizar herramientas para que la compilación se mejore. Por ejemplo, si se está trabajando en C, un buen compilador es gcc.

Referencias

[1] R. Avansi, et al., *Handbook of Elliptic and Hyperelliptic Curve Cryptography.*, 1st ed.: Chapman & Hall/CRC, 2005.

[2] M. Brown, D. Hankerson, J. Lopez, and A. Meneses, "A Software Implementation of the NIST Elliptic Curves

[7] Esta expresión hace referencia a correr w bits a la derecha y los w bits más significativos se llenan con cero.

[8] Esta expresión hace referencia a correr w bits a la izquierda.

Over Prime Fields.," in *Topics in Cryptology -CT - RSA 2001 (LNCS 2020*, 2001.

[3] D. Hankerson, J. Lopez, and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields.," in *Cryptographic Hardware and Embedded Systems - CHES 2000, (LNCS 1965)*, Berlín, 2000, pp. 1-24.

[4] D. Hankerson, A. Menezes, and A. Vanstone, *Guide to Elliptic Curve Cryptography*. Berlín: Springer-Verlag, 2003.

[5] R. Dahab and J. Lopez, "Improved Algorithms for Elliptic Curve Arithmetic in GF(2n)," Technical Report IC - 98 - 39 1998.

[6] K. Fong, D. Hankerson, J. Lopez, and A. Menezes, "Field Inversion and Point Halving Revisited, "Department of Combinatorics and Optimization, Canada, Technical Report CORR 2003 - 18 2003.