O conteúdo do presente relatório é de única responsabilidade do(s) autor(es).
The contents of this report are the sole responsibility of the author(s).

On Computing a Multiple of an Elliptic Curve Point

Julio López *Ricardo Dahab*

Relatório Técnico IC–98-13

Abril de 1998

On Computing a Multiple of an Elliptic Curve Point [*]

Julio López[†] Ricardo Dahab[‡]

Email: {julioher, rdahab}@dcc.unicamp.br

Institute of Computing
State University of Campinas
Campinas, 13081-970 São Paulo, Brazil
March 6, 1998

Abstract

We describe an improved algorithm for computing a multiple of a point on elliptic curves defined over finite fields of characteristic greater than three. The proposed algorithm is based on new formulae for computing repeated doubling points with only one field inversion, and the signed-digit exponent recoding algorithm. The new algorithm is shown to be faster than Müller's method [7]. Our new formulae for repeated doubling points can also be used to speed up algorithms such as the k-ary method and the signed binary window method.

Keywords: Elliptic Curve Cryptosystem, Multiplication.

1 Introduction

Elliptic curves defined over finite fields have been proposed for Diffie-Hellman type cryptosystems [2]. The security of these cryptosytems is based on the discrete logarithm problem on elliptic curves. Because of the apparent difficulty on solving the elliptic curve discrete logarithm problem, the length of blocks and keys can be considerably smaller, about 200 bits. This is a desirable feature in restricted computing environments, such as smart cards and wireless devices.

The basic operation of elliptic curve public-key cryptosystems is the computation of a *multiple of an elliptic point.* If m is a positive integer and P is an elliptic curve point, a multiple of an elliptic curve point mP is the result of adding P to itself m times. In this paper we discuss an efficient method for multiplying points on elliptic curves defined over finite fields of characteristic greater than three. This method is based on new explicit formulae for computing 2^iP in terms of the coordinates of the point $P = (x, y)$ and the signed-digit representation for the multiplier m. This approach was suggested by Guajardo and Paar [3] for the case of non-supersingular elliptic curves defined over finite fields of characteristic 2.

[*]Research partially supported by CNPq (grant 301172/95-1) and CAPES.

[†]Departamento de Ciencias de la Computación, Universidad del Valle, Cali, Colombia.

[‡]Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP, Brasil.

For the case of elliptic curves defined over finite fields of characteristic greater than three, Müller [7], by using the theory of division polynomials, developed an algorithm to compute $4P$ which only needs one field inversion (but more multiplications and squarings). We, by repeatedly applying the doubling point formula, developed an algorithm to compute 2^iP with only one inversion (and extra multiplications and squarings). As a result, the proposed algorithm for multiplying points is faster than Müller's methods [7].

This paper is organized as follows. Section 2 presents a brief summary of elliptic curves defined over the finite field $GF(p)$, where p is a prime number greater than three. In Section 3, we introduce a new algorithm for computing 2^iP with only one field inversion. Then, in Section 4, we present the improved algorithm to compute mP, and compare them to Müller's results.

2 Elliptic Curves over $GF(p)$

Let p be prime number greater than 3. Let $a, b \in GF(p)$ be such that $4a^3 + 27b^2 \neq 0$ in $GF(p)$. An elliptic curve E over $GF(p)$ defined by the parameters a and b is the set of points $P = (x_p, y_p)$ where x_p and y_p are elements of $GF(p)$ that satisfy the following equation in $GF(p)$:

$$y_p^2 = x_p^3 + ax_p + b,$$

together with an extra point $\mathcal{O}$, the point at infinity.

It is well known that the set of points E forms a commutative finite group with $\mathcal{O}$ as the group identity under the addition given by the "tangent and chord method". Explicit rational formulae for the addition rule involve several arithmetic operations in the underlying finite field. The rules for addition, in affine coordinates, are given by: If $P = (x_1, y_1) \in E$, then $-P = (x_1, -y_1)$. If $Q = (x_2, y_2) \in E$, $Q \neq -P$, then $P + Q = (x_3, y_3)$, where

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 && (1) \\ y_3 &= \lambda \cdot (x_1 - x_3) - y_1, && (2) \end{aligned}$$

and

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1}, & \text{if} P \neq Q, \\[2ex] \dfrac{3x_1^2 + a}{2y_1}, & \text{if} P = Q. \end{cases}$$

The following lemma counts the number of (quadratic) field operations required to compute an elliptic addition.

Lemma 1 *One elliptic addition of two different points of E can be performed with one inversion, two multiplications and one squaring in $GF(p)$; doubling an elliptic point needs one inversion, two multiplications and two squarings in $GF(p)$.*

3 Improved Formulae and Algorithm

In this section we present an algorithm for computing $2^iP, i \geq 2$ where $P = (x, y)$ is a point of E. The standard algorithm computes 2^iP by repeatedly applying the doubling formulae (1) and (2) i times. Such algorithm needs i inversions in the underlying finite field. The following theorem shows how to compute 2^iP with only one field inversion.

Theorem 1 *Let $P = (x, y)$ be a point on the elliptic curve E. Then the coordinates of the point $2^iP = (x_i, y_i), i \geq 2$, are given by*

$$
\begin{aligned}
x_i &= \frac{\alpha_i}{\rho_{i-1}^2} && (3)\\
y_i &= \frac{\beta_i}{\rho_{i-1}^3} && (4)
\end{aligned}
$$

where

$$
\begin{aligned}
\delta_k &= \alpha_{k-1} \cdot \nu_{k-1}^2, \quad \alpha_0 = x, \ \rho_0 = \nu_0 = 2y && (5)\\
\alpha_k &= \omega_{k-1}^2 - 2\delta_k, && (6)\\
\beta_k &= \omega_{k-1} \cdot (\delta_k - \alpha_k) - (\nu_{k-1}^2)^2/2, && (7)\\
\omega_k &= 3\alpha_k^2 + a \cdot (\rho_{k-1}^2)^2, \quad \omega_0 = 3\alpha_0^2 + a && (8)\\
\nu_k &= 2\beta_k, && (9)\\
\rho_k &= \nu_k \cdot \rho_{k-1}, \quad k = 1 \ldots i && (10)
\end{aligned}
$$

Proof. We will prove by induction on i that $x_i = \frac{\alpha_i}{\rho_{i-1}^2}$ and $y_i = \frac{\beta_i}{\rho_{i-1}^3}$. This is easily true for $i = 1$. Now assume that the statement is true for $i = n$; we prove it for $i = n + 1$:

$$
\begin{aligned}
x_{n+1} &= \left(\frac{3x_n^2 + a}{2y_n}\right)^2 - 2x_n = \left(\frac{3\alpha_n^2 + a\rho_{n-1}^4}{\rho_{n-1}\nu_n}\right)^2 - 2\frac{\alpha_n \nu_n^2}{\rho_n^2}\\
&= \frac{\omega_n^2 - 2\alpha_n\nu_n^2}{\rho_n^2} = \frac{\alpha_{n+1}}{\rho_n^2}
\end{aligned}
$$

similarly, for y_{n+1} we obtain:

$$
\begin{aligned}
y_{n+1} &= \frac{3x_n^2 + a}{2y_n}(x_n - x_{n+1}) - y_n = \frac{w_n}{\rho_n}\left(\frac{\alpha_n}{\rho_{n-1}^2} - \frac{\alpha_{n+1}}{\rho_n^2}\right) - \frac{\beta_n}{\rho_{n-1}^3}\\
&= \frac{\omega_n(\delta_{n+1} - \alpha_{n+1}) - \frac{1}{2}\nu_n^4}{\rho_n^3}\\
&= \frac{\beta_{n+1}}{\rho_n^3}. \qquad \square
\end{aligned}
$$

Using Theorem 1, we immediately describe an algorithm for computing 2^iP in affine coordinates.

Algorithm 1: Fast Repeated Doubling Points

Input	: $P = (x, y) \in E, i \geq 2.$
Output	: $Q = 2^iP$

```
1. $A \leftarrow x,\ B \leftarrow y,\ v \leftarrow 2B,\ p \leftarrow v,\ w \leftarrow 3A^2 + a.$
2. for $k$ from 1 to $i-1$ do :
    2.1 $d \leftarrow A \cdot v^2.$
    2.2 $A \leftarrow w^2 - 2d.$
    2.3 $B \leftarrow w \cdot (d - A) - (v^2)^2/2.$
    2.4 $w \leftarrow 3A^2 + a \cdot (p^2)^2$ .
    2.5 $v \leftarrow 2B.$
    2.6 $p \leftarrow p \cdot v.$
3. $w \leftarrow p^{-1} \cdot w,\ v \leftarrow (p^{-1} \cdot v)^2,\ d \leftarrow A \cdot v.$
4. $x_i \leftarrow w^2 - 2d.$
5. $y_i \leftarrow w \cdot (d - x_i) - p \cdot v^2/2.$
6. return($Q = (x_i, y_i)$).
```

Some modifications which improve the performance of this algorithm are: (i) if a is small enough, then a multiplication by a in step 2.4 can be done by repeated additions; (ii) if $a = p - 3$, then in step 2.4, w can be computed as $3(A - p^2) \cdot (A + p^2)$.

In considering the cost of computing repeated doublings, we neglect the cost of linear time operations such as additions, subtractions and multiplications by 2 or 3 because they are much faster than multiplication and inversion field operations. Note that multiplication by 2^{-1} can be computed in linear time if we use the fact that for $r \in GF(p)$ we have $2^{-1} \cdot r = \frac{r}{2}$, if r is even, and $2^{-1} \cdot r = \frac{(r+p)}{2}$, if r is odd. The following corollary counts the number of (quadratic) field operations required by Algorithm 1 to perform 2^iP.

Corollary 1 *Consider an arbitrary point $P \in E$, with an order larger than 2^i. Then Algorithm 1 requires $4i + 1$ multiplications, $6i - 4$ squarings and one inversion in $GF(p)$.*

Another way to compute 2^iP is to use projective coordinates. We first transform P into projective coordinates, then we compute 2^iP by repeatedly applying the doubling formula, given in [6, 4], i times, and finally we convert the result to affine coordinates. Such method requires $7i - 1$ multiplications, $5i - 1$ squarings and one inversion in $GF(p)$. Therefore, Algorithm 1 is at least 20% faster than this approach.

Our algorithm reduces the number of inversions in the underlying field at the cost of multiplications and squarings. Table 1, derived from Corollary 1, shows the number of field operations needed for $4P, 8P$ and $16P$ in affine coordinates. Note that for the particular case of $4P$, Müller's algorithm requires 14 multiplications and our requires 9 multiplications. If we assume that the time to perform a field inversion is approximately equivalent to that of 25 field multiplications (see, e.g. LiDIA [7]) and that squaring takes the same time as multiplication, then we expect Algorithm 1 for $4P$ to be approximately 22% faster than the standard algorithm and 8% than Müller's algorithm.

Table 1 : Comparing the new, Müller and standard algorithm for computing repeated doubling points

Calculation	Method	#Mul.	#Sqr.	#Inv.
$4P$	New	9	8^1	1
	Müller [7]	14	7	1
	Standard	4	4	2
$8P$	New	13	14	1
	Standard	6	6	3
$16P$	New	17	20	1
	Standard	8	8	4

Now we investigate under which conditions, on a given implementation of $GF(p)$, Algorithm 1 outperforms the standard method. Let us consider an implementation of $GF(p)$ and let r_1 and r_2 be the relative costs of a field inversion (multiplication) to the cost of a field multiplication (squaring) respectively. From Corollary 1 and Lemma 1 we obtain a bound between ratios r_1 and r_2 for which our algorithm is better than the standard method. The following theorem gives this bound.

Theorem 2 *Consider an implementation of the field $GF(p)$. Let $P \in E$ be a point of order larger than 2^i. Then Algorithm 1 outperforms the standard algorithm in computing 2^iP if*

$$r_1 \geq 2 + \frac{4}{r_2} + \frac{3}{i-1}.$$

For instance, if squaring is approximately as costly as an arbitrary field multiplication, then Algorithm 1 computes $8P$ more efficiently than the standard method if r_1 is at least 7.5. The larger r_1 is, the more superior our algorithm is when compared to the standard algorithm.

4 Algorithms for Computing mP

Research on speeding up a multiple of an elliptic curve point is fundamental for cryptographic applications, since a multiple of a point, the most time consuming operation, dominates the computations involved in such applications. The naive (or binary) algorithm for computing mP requires on average $\lfloor \frac{1}{2} log_2 m \rfloor$ elliptic curve additions and $\lfloor log_2 m \rfloor$ elliptic curve doublings. Since each elliptic curve addition or doubling requires one field inversion, we need on average $\lfloor \frac{3}{2} log_2 m \rfloor$ inversions.

Various methods have been proposed [3, 4, 5] for speeding up the binary method. Such methods reduce either the number of addition points or the number of doubling points, and therefore the total number of field inversions is reduced. We observe also that most generalizations of the binary method such as the k-ary method, the sliding-window method etc., are based on recoding the exponent (multiplier) m and processing several s_i "digits"

[1] For $i = 2$, $v = p$.

(i.e. computing $2^{s_i}Q_i$) of the representation of m in each iteration. Thus, we can use our Algorithm 1 for speeding up the computation of mP. In Table 2 we summarize the expected time complexity of several algorithms for computing mP with their respective number of precomputed points (PP). A detailed analysis of the k-ary method and the signed binary window method can be found in [3, 4, 5, 7].

We compare the 4-ary method combined with Algorithm 1 to Müller's algorithms and the binary method. According to Table 2, if we assume $r_1 = 25$, $r_2 = 1$ and m about 200 bits, then we expect that the 4-ary method (with Algorithm 1) to be approximately 48% faster than the standard algorithm and 29% faster than Müller's algorithm (the improved 2-ary). For applications such as smart cards, where storage space is limited, the 2-ary method (with Algorithm 1) is slightly faster than Müller's algorithm (the improved 2-ary). Notice however we can speed up the improve 2-ary method in [7] with our Algorithm 1.

Table 2: Expected complexity of several algorithms for computing mP with Algorithm 1 (A1)

Method	Multiplication	Squaring	Inversion	PP
Binary	$3\log_2 m$	$\frac{5}{2}\log_2 m$	$\frac{3}{2}\log_2 m$	0
2-ary + A1	$\frac{21}{4}\log_2 m$	$\frac{35}{8}\log_2 m$	$\frac{7}{8}\log_2 m$	2
Improved [7]	$\frac{487}{64}\log_2 m$	$\frac{487}{128}\log_2 m$	$\frac{103}{128}\log_2 m$	2
4-ary + A1	$\frac{151}{32}\log_2 m$	$\frac{335}{64}\log_2 m$	$\frac{31}{64}\log_2 m$	14

4.1 Signed-digit Representation

In this section we consider a particular method for recoding the exponent m. Once the recoding of m is obtained, we can use the binary method to compute mP.

Definition 1 *[5] If $m = \sum_{i=0}^{t} d_t 2^i$ where $d_i \in \{0, 1, -1\}, 0 \le i \le t$, then $(d_t \ldots d_1 d_0)_{SD}$ is called a signed-digit representation with radix 2 for the integer m. A signed-digit representation of m is said to be sparse if no two non-zero entries are adjacent in the representation.*

The following algorithm, due to Reitwiesner [9], produces a sparse signed-digit representation having at most $t + 1$ digits and the smallest number of non-zero entries among all signed-digit representations for m.

Algorithm 2: Signed-digit exponent recoding

```
Input   : $m = (m_{t+1} m_t \ldots m_1 m_0)_2$ with $m_{t+1} = m_t = 0$.
Output  : A signed-digit representation $(d_t \ldots d_1 d_0)_{SD}$ for $m$
  1. $c_0 \leftarrow 0$.
  2. $S[0,0,0] \leftarrow [0,0]$,   $S[0,0,1] \leftarrow [0,0]$, $S[0,1,0] \leftarrow [0,1]$.
     $S[0,1,1] \leftarrow [1,-1]$, $S[1,0,1] \leftarrow [0,1]$, $S[1,0,1] \leftarrow [1,-1]$.
     $S[1,1,0] \leftarrow [1,0]$,   $S[1,1,1] \leftarrow [1,0]$.
  3. for $i$ from 0 to $t$ do :
     3.1 $c_{i+1} \leftarrow S[m_i, m_{i+1}, c_i][1]$.
     3.2 $d_{i+1} \leftarrow S[m_i, m_{i+1}, c_i][2]$.
  4. return $((d_t \ldots d_1 d_0)_{SD})$.
```

4.2 Proposed Algorithm

We describe two versions of the proposed algorithm. The first is a "right-to-left" multiplication algorithm, which reads the bits of the signed-digit representation of m from the low order bit to the high order bit. The second algorithm, "right-to-left", scans the bits of the signed-digit representation of m from the high order bit to the low order bit. Both versions of the proposed algorithm require no precomputation; but they use two temporary variables (one elliptic point of additional storage) more than the binary method.

4.2.1 Right-to-Left Algorithm

The proposed "right-to-left" algorithm processes the bits of m from the least significant to the most significant. At each iteration, the point 2^iP is computed , and depending on the scanned bit value, a subsequent elliptic addition is performed. Since the signed-digit representation is sparse, the elliptic addition is performed at most at every two iterations. Then, the "right-to-left" algorithm computes mP using the equation

$$m \cdot P = (\cdots(((d_0P) + d_1 2P) + d_2 2^2 P) + \cdots + d_t 2^t P). \tag{11}$$

Algorithm 3: Right-to-Left Algorithm

Input	: A signed-digit representation $(d_t \ldots d_1 d_0)_{SD}$ for m, $P \in E$
Output	: $Q = m \cdot P$

```
1. $Q \leftarrow d_0P = (x_0, y_0)$.
2. $A \leftarrow x_0,\ B \leftarrow y_0,\ v \leftarrow 2B,\ p \leftarrow v,\ w \leftarrow 3A^2 + a$.
3. for $i$ from 1 to $t$ do :
     if $d_i \neq 0$ then
         $w \leftarrow w/p$.
         $v \leftarrow (v/p)^2$.
         $d \leftarrow A \cdot v$.
         $A \leftarrow w^2 - 2d$.
         $B \leftarrow w \cdot (d - A) - p \cdot v^2/2$.
         $Q \leftarrow Q\ +\ d_i(A, B)$.
         if $i \neq t$ then $v \leftarrow 2B,\ p \leftarrow v,\ w \leftarrow 3A^2 + a$ fi.
     else
         $d \leftarrow A \cdot v^2$.
         $A \leftarrow w^2 - 2d$.
         $B \leftarrow w \cdot (d - A) - (v^2)^2/2$.
         $w \leftarrow 3A^2 + a \cdot (p^2)^2,\ v \leftarrow 2B,\ p \leftarrow p \cdot v$.
4. return($Q = mP$)
```

Now, we estimate the complexity of this algorithm. For a random exponent m, Algorithm 2 produces a signed-digit representation of m with $\frac{1}{3}\log_2 m$ nonzero coefficients on average (see [1]). Therefore Algorithm 3 has to perform $\frac{1}{3}\log_2 m$ elliptic point additions and an amount of computation equivalent to performing $\frac{1}{3}\log_2 m$ times $8P$ on average. It follows that the number of field operations required by Algorithm 3 is found to be: $\frac{1}{3}(2+13)\log_2 m$

multiplications, $\frac{1}{3}(1+14)\log_2 m$ squarings and $\frac{2}{3}\log_2 m$ inversions on average. We summarize, in Table 3, the expected number of field operations to compute mP.

4.2.2 Left-to-Right Algorithm

The proposed "left-to-right" algorithm scans the bits of m in descending order $d_t, d_{t-1}, \ldots d_0$. We use the following equation to compute mP in affine coordinates:

$$m \cdot P = 2 \cdot (\cdots 2 \cdot (2d_t \cdot P + d_{t-1} \cdot P) \cdots + d_1 \cdot P.) + d_0 \cdot P \qquad (12)$$

Algorithm 4: Left-to-Right Algorithm

Input	: A signed-digit representation $(d_t \ldots d_1 d_0)_{SD}$ for m, $P \in E$
Output	: $Q = m \cdot P$

```
1. $Q \leftarrow d_t P$, $i \leftarrow t-1$.
2. while $(i \geq 0)$ do :
    2.1 $l \leftarrow 0$.
    2.2 while $(i \geq 0$ and $d_i = 0)$ do :
        $l \leftarrow l+1$, $i \leftarrow i-1$.
    2.3 if $i \geq 0$ then $Q \leftarrow 2^{l+1}Q + d_i P$,  $i \leftarrow i-1$
        else $Q \leftarrow 2^l Q$,  $i \leftarrow i-1$.
3. return($Q$)
```

For a random multiplier m, Algorithm 4 requires on average the same number of field operations performed by Algorithm 3. If we consider the field implementation of $GF(p)$ given in [7], where $r_1 = 25$ and $r_2 = 1$, then we expect both algorithms 3 and 4 to be about 15% faster than Müller's method and 38% faster than the binary method.

We point out that the signed binary window method (SBWM), combined with Algorithm 1, presents the best trade-off between speedup and precomputed points among generalizations of the binary method, for field implementations where Theorem 1 applies. Its time complexity is shown in Table 3.

Table 3 : Expected time complexity of algorithms for computing mP in affine (A) coordinates

Method	#Mult.	#Sqr.	#Inv.	#PP
Algorithm 3	$5\log_2 m$	$5\log_2 m$	$\frac{2}{3}\log_2 m$	0
Algorithm 4	$5\log_2 m$	$5\log_2 m$	$\frac{2}{3}\log_2 m$	0
SBWM	$\frac{7}{3}\log_2 m$	$\frac{13}{6}\log_2 m$	$\frac{7}{6}\log_2 m$	7
SBWM + Alg. 1	$4.5\log_2 m$	$5.5\log_2 m$	$\frac{1}{3}\log_2 m$	7

5 Conclusion

We have presented a new algorithm for computing repeated doubling points, with only one field inversion, on elliptic curves defined over finite fields of characteristic greater than 3. This algorithm, in combination with a signed-digit exponent recoding algorithm, yields a memory efficient (no precomputation) algorithm for computing a multiple of an elliptic curve point. For the field implementation considered in [7], we expect an improvement of up to 38% in the running time of our new algorithms compared to the binary algorithm. Our new algorithm for repeated doubling points can also be used to speed up algorithms such the k-ary method and the signed binary window method.

Note added during review: when writting this report, we were not aware of a recent publication in IEEE P1363 [8] of February 9, 1998, in which new formulae for doubling a point in projective coordinates are presented. They are:

$$2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2)$$

where,

$$\begin{aligned} Z_2 &= 2Y_1 \cdot Z_1 \\ W &= 3X_1^2 + a \cdot Z_1^4 \\ D &= 4X_1 \cdot Y_1^2 \\ X_2 &= W^2 - 2D \\ Y_2 &= W \cdot (D - X_2) - 8Y_1^4 \end{aligned}$$

Using these formulae to compute $2^i P$ results in an algorithm that performs the same number of field multiplications and three more field squarings than Algorithm 1.

References

[1] Çetin K. Koç. "High-Speed RSA Implementation ". Technical Report TR 201, RSA Laboratories, 1994.

[2] W. Diffie and M. Hellman. "New directions in cryptography". *IEEE Transaction of information Theory*, 22:644–654, 1976.

[3] J. Guajardo and C. Paar. "Efficient Algorithms for Elliptic Curve Cryptosystems". In *CRYPTO '97, LNCS*. Springer-Verlag, 1997.

[4] K. Koyama and Y. Tsuruoka. "Speeding up Elliptic Cryptosystems by Using a Signed Binary Method ". In *Advances in Cryptology-CRYPTO'92*, LNCS 740, pages 345–357. Springer-Verlag, 1993.

[5] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 1997.

[6] Atsuko Miyaji. "Elliptic Curves over F_p Suitable for Cryptosystems". In *Advances in Cryptology-AUSCRYPT'92 (LNCS 718)*, pages 479–491. Springer-Verlag, Berlin, 1993.

[7] Volker Müller. "Efficient Algorithms for Multiplication on Elliptic Curves". Technical Report TI-9/97, Darmstdadt University of Technology, Germany, 1997.

[8] IEEE P1363. "Standard Specifications for Public-Key Cryptography", Annex A, draft. 1998.

[9] G. Reitwiesner. "Binary Arithmetic". *Advances in Computers*, pages 231–308, 1960.