Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation

Julio López[1]⋆ and Ricardo Dahab[2]⋆⋆

[1] Department of Combinatorics & Optimization
University of Waterloo,
Waterloo, Ontario N2L 3G1, Canada (jclherna@cacr.math.uwaterloo.ca)
[2] Institute of Computing
State University of Campinas,
Campinas, C.P. 6176, 13083-970 SP, Brazil (rdahab@dcc.unicamp.br).

Abstract. This paper describes an algorithm for computing elliptic scalar multiplications on non-supersingular elliptic curves defined over $GF(2^m)$. The algorithm is an optimized version of a method described in [1], which is based on Montgomery's method [8]. Our algorithm is easy to implement in both hardware and software, works for any elliptic curve over $GF(2^m)$, requires no precomputed multiples of a point, and is faster on average than the addition-subtraction method described in draft standard IEEE P1363. In addition, the method requires less memory than projective schemes and the amount of computation needed for a scalar multiplication is fixed for all multipliers of the same binary length. Therefore, the improved method possesses many desirable features for implementing elliptic curves in restricted environments.

Key words. Elliptic Curves over $GF(2^m)$, Point multiplication.

1 Introduction

Elliptic curve cryptography first suggested by Koblitz [5] and Miller [12] is becoming increasingly common for implementing public-key protocols as the Diffie-Hellman key agreement. The security of these cryptosystems relies on the presumed intractability of the discrete logarithm problem on elliptic curves. Since there is no known sub-exponential type algorithm for elliptic curves over finite fields, the sizes of the fields, keys, and other parameters can be considered shorter than other public key cryptosystems such as RSA with the same level of security. This can be especially an advantage for applications where resources such as memory and/or computing power are limited.

Elliptic curves over $GF(2^m)$ are particularly attractive, because the finite field operations can be implemented very efficiently in hardware and software.

⋆ Dept. of Computer Science, University of Valle, A.A. 25130 Cali, Colombia.
Research supported by a CAPES-Brasil scholarship
⋆⋆ Partially supported by a PRONEX-FINEP research grant no. 107/97

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 316–327, 1999.

See for example [1] for a hardware implementation of $GF(2^{155})$, and [19] for a software implementation of $GF(2^{191})$.

Given an elliptic point P and a large integer k of about the size of the underlying field, the operation *elliptic scalar multiplication*, kP, is defined to be the elliptic point resulting from adding P to itself k times. This operation, analogous to exponentiation in multiplicative groups, is the most time consuming operation of the elliptic curve cryptosystems.

In this paper, the calculation of kP for a random integer k and a random point P is considered. An efficient scalar multiplication algorithm, which is an optimized version of an algorithm described in [1], is presented. The proposed algorithm is suitable for hardware and software implementation of random elliptic curves over $GF(2^m)$.

2 Previous Work

The basic method for computing kP is the addition-subtraction method described in draft standard IEEE P1363 [14]. This method is an improved version over the well known "add-and-double" (or binary) method, which requires no precomputations. For a random multiplier k, this algorithm performs on average $\frac{8}{3}\log_2 k$ field multiplications and $\frac{4}{3}\log_2 k$ field inversions in affine coordinates, and $8\frac{1}{3}\log_2 k$ field multiplications in projective coordinates.

Several proposed generalizations of the binary method (for exponentiation in a multiplicative group), such as the k-ary method, the signed window method, can be extended to compute elliptic scalar multiplications over a finite field [11]. These algorithms are based on the use of precomputation and methods for recoding the multiplier. In [3], several algorithms are analyzed under various conditions. However, most of the proposed optimizations may not be worthwhile when memory is at a premium.

Some special classes of elliptic curves defined over $GF(2^m)$ allow efficient implementations. For anomalous curves, the fastest known algorithm to compute kP is given in [17]; for curves defined over small subfields, efficient algorithms are presented in [13].

In [4,16,7] some techniques are presented for accelerating methods such as k-ary and window based methods. These methods are suitable for software implementation of random elliptic curves over $GF(2^m)$.

A different approach for computing kP was introduced by Montgomery [8]. This approach is based on the binary method and the observation that the x-coordinate of the sum of two points whose difference is known can be computed in terms of the x-coordinates of the involved points. This method uses the following variant of the binary method:

INPUT: An integer $k > 0$ and a point P.
OUTPUT: $Q = kP$.

1. Set $k \leftarrow (k_{l-1} \ldots k_1 k_0)_2$.
2. Set $P_1 \leftarrow P, \quad P_2 \leftarrow 2P$.
3. **for** i **from** $l-2$ **downto** 0 **do**
   if $k_i = 1$ **then**
   Set $P_1 \leftarrow P_1 + P_2, \quad P_2 \leftarrow 2P_2$.
   else
   Set $P_2 \leftarrow P_2 + P_1, \quad P_1 \leftarrow 2P_1$.
4. **return**$(Q = P_1)$.

Fig. 1. Algorithm 1: Binary Method

Note that this method maintains the invariant relationship $P_2 - P_1 = P$, and performs an addition and a doubling in each iteration. In [9], Montgomery's method was applied for reducing the number of registers needed to add points in supersingular curves over $GF(2^m)$. However, the authors observed that the benefits in storage provided by Montgomery's method is at a considerable expense of speed.

From the point of view of hardware implementation of elliptic curves over $GF(2^m)$, few papers have discussed efficient methods for computing kP. In [1], Montgomery's method was adapted for non-supersingular elliptic curves over $GF(2^m)$. However, the formulas given for implementing each iteration are not efficient in terms of field multiplications.

In this paper we will present an efficient implementation of Montgomery's method for computing kP on non-supersingular elliptic curves over $GF(2^m)$.

The remainder of the paper is organized as follows. In Section 3 we present a short introduction to elliptic curves over $GF(2^m)$. The proposed algorithm is described and analyzed in Section 4. Some running times of the proposed algorithm based on LiDIA are presented in Section 5. An implementation of the proposed algorithm is given in the appendix.

3 Elliptic Curves over $GF(2^m)$

Here we present a brief introduction to elliptic curves; more information on elliptic curves over finite fields of characteristic two can be found in [10,14]. Let $GF(2^m)$ be a finite field of characteristic two. A non-supersingular elliptic curve E over $GF(2^m)$ is defined to be the set of solutions $(x, y) \in GF(2^m) \times GF(2^m)$ to the equation,

$$y^2 + xy = x^3 + ax^2 + b \ ,$$

where a and $b \in GF(2^m), b \neq 0$, together with the point at infinity denoted by $\mathcal{O}$.

It is well known that E forms a commutative finite group, with $\mathcal{O}$ as the group identity, under the addition operation known as the "tangent and chord

method". Explicit rational formulas for the addition rule involve several arithmetic operations (adding, squaring, multiplication and inversion) in the underlying finite field. Formulas for adding two points in projective coordinates can be found in [10,7]. In affine coordinates, the elliptic group operation is given by the following. Let $P = (x_1, y_1) \in E$; then $-P = (x_1, x_1 + y_1)$. For all $P \in E$, $\mathcal{O} + P = P + \mathcal{O} = P$. If $Q = (x_2, y_2) \in E$ and $Q \neq -P$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \begin{cases} (\frac{y_1 + y_2}{x_1 + x_2})^2 + \frac{y_1 + y_2}{x_1 + x_2} + x_1 + x_2 + a \ , & P \neq Q \\ x_1^2 + \frac{b}{x_1^2} \ , & P = Q. \end{cases} \tag{1}$$

and

$$y_3 = \begin{cases} (\frac{y_1 + y_2}{x_1 + x_2})(x_1 + x_3) + x_3 + y_1 \ , & P \neq Q \\ x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3 \ , & P = Q. \end{cases} \tag{2}$$

Notice that the x-coordinate of $2P$ does not involve the y-coordinate of P. This observation will be used in the derivation of the improved method.

4 Improved Method

This section describes the improved method for computing kP. We first develop an algorithm in affine coordinates which requires two field inversions in each iteration. Next a "projective" version is presented with more field multiplications, but with only one field inversion at the end of the computation.

4.1 Affine Version

The extension of Montgomery's method [8] to elliptic curves over $GF(2^m)$ requires formulas for implementing Step 3 of Algorithm 1. In what follows we give efficient formulas that use only the x-coordinates of P_1, P_2 and P for performing the arithmetic operations needed in Algorithm 1. At the end of the lth iteration of Algorithm 1, we obtain the x-coordinates of kP and $(k+1)P$. We also provide a simple formula for recovering the y-coordinate of kP.

The following lemma gives another formula for computing the x-coordinate of the addition of two different points.

Lemma 1 *Let $P_1 = (x_1, y_1)$, and $P_2 = (x_2, y_2)$ be elliptic points. Then the x-coordinate of $P_1 + P_2$, x_3, can be computed as follows.*

$$x_3 = \frac{x_1 y_2 + x_2 y_1 + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \ . \tag{3}$$

Proof. Since P_1 and P_2 are elliptic points, it follows that $y_1^2 + y_2^2 + x_1 y_1 + x_2 y_2 + x_1^3 + x_2^3 = 0$. The result then follows easily from formula (1).

The following lemma shows how to compute the x-coordinate for the addition of two points whose difference is known.

Lemma 2 *Let $P = (x, y), P_1 = (x_1, y_1)$, and $P_2 = (x_2, y_2)$ be elliptic points. Assume that $P_2 = P_1 + P$. Then the x-coordinate of $P_1 + P_2$, x_3, can be computed in terms of the x-coordinates of P, P_1 and P_2 as follows.*

$$x_3 = \begin{cases} x + (\frac{x_1}{x_1 + x_2})^2 + \frac{x_1}{x_1 + x_2} & , P_1 \neq P_2 \\ x_1^2 + \frac{b}{x_1^2} & , \quad P_1 = P_2. \end{cases} \tag{4}$$

Proof. The case $P = \mathcal{O}$ follows directly from (1). Applying formula (3), we obtain that the x-coordinate of $P_2 + P_1$ can be rewritten as

$$x_3 = \frac{x_1 y_2 + x_2 y_1 + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \ . \tag{5}$$

Similarly, the x-coordinate of $P_2 - P_1$ satisfies

$$x = \frac{x_1 y_2 + x_2 (x_1 + y_1) + x_1 x_2^2 + x_2 x_1^2}{(x_1 + x_2)^2} \ . \tag{6}$$

The result follows from adding (5) and (6).

The next lemma allows one to compute the y-coordinate of P_1 when P and the x-coordinates of P_1 and $P_1 + P$ are known.

Lemma 3 *Let $P = (x, y), P_1 = (x_1, y_1)$, and $P_2 = (x_2, y_2)$ be elliptic points. Assume that $P_2 = P_1 + P$ and $x \neq 0$. Then the y-coordinate of P_1 can be expressed in terms of P, and the x-coordinates of P_1 and P_2 as follows.*

$$y_1 = (x_1 + x)\{(x_1 + x)(x_2 + x) + x^2 + y\}/x + y \ . \tag{7}$$

Proof. Since $P_2 = P_1 + P$, we obtain from (3) that y_1 satisfies the following equation:

$$x_2 (x_1 + x)^2 = x_1 y + x y_1 + x_1 x^2 + x x_1^2 \ .$$

Therefore,

$$\begin{aligned} x y_1 &= x_2 x_1^2 + x_2 x^2 + x_1 y + x_1 x^2 + x x_1^2 \\ &= x_1 \{x_1 x_2 + x_1 x + x^2 + y\} + x\{x x_2\} \\ &= x_1 \{x_1 x_2 + x_1 x + x^2 + x x_2 + x^2 + y\} \\ &\quad + x\{x_1 x_2 + x_1 x + x x_2 + y\} + x y \\ &= (x_1 + x)\{(x_1 + x)(x_2 + x) + x^2 + y\} + x y. \end{aligned}$$

The following algorithm, based on Lemmas 2 and 3, implements Montgomery's method in affine coordinates.

INPUT: An integer $k \geq 0$ and a point $P = (x, y) \in E$.
OUTPUT: $Q = kP$.

1. **if** $k = 0$ **or** $x = 0$ **then output**$(0, 0)$ **and stop.**
2. **Set** $k \leftarrow (k_{l-1} \ldots k_1 k_0)_2$.
3. **Set** $x_1 \leftarrow x, \quad x_2 \leftarrow x^2 + b/x^2$.
4. **for** i **from** $l - 2$ **downto** 0 **do**
   Set $t \leftarrow \frac{x_1}{x_1 + x_2}$.
   if $k_i = 1$ **then**
   Set $x_1 \leftarrow x + t^2 + t, \quad x_2 \leftarrow x_2^2 + b/x_2^2$.
   else
   Set $x_1 \leftarrow x_1^2 + b/x_1^2, \quad x_2 \leftarrow x + t^2 + t$.
5. **Set** $r_1 \leftarrow x_1 + x, \quad r_2 \leftarrow x_2 + x$.
6. **Set** $y_1 \leftarrow r_1(r_1 r_2 + x^2 + y)/x + y$
7. **return**$(Q = (x_1, y_1))$.

Fig. 2. Algorithm 2A: Montgomery Scalar Multiplication

Observe that Algorithm 2A, in each iteration of Step 4, performs two field inversions, one general field multiplication, one multiplication by the constant b, two squarings, and four additions; it follows that the total number of field operations to compute kP is given in the following lemma:

Lemma 4 *For computing kP, Algorithm 2A takes exactly the following number of field operations in $GF(2^m)$:*

$$\#INV. = 2\lfloor \log_2 k \rfloor + 1 \ , \quad \#MULT. = 2\lfloor \log_2 k \rfloor + 4 \ ,$$
$$\#ADD. = 4\lfloor \log_2 k \rfloor + 6 \ , \quad \#SQR. = 2\lfloor \log_2 k \rfloor + 2.$$

Remark. A further improvement to Algorithm 2A is to use an optimized routine to multiply by the constant b. Another potential improvement is to compute in parallel x_1 and x_2 from Step 4, since these calculations are independent of each other.

4.2 Projective Version

When field inversion in $GF(2^m)$ is relatively expensive (e.g., inversion based on Fermat's theorem requires at least 7 multiplications in $GF(2^m)$ if $m \geq 128$), then it may be of computational advantage to use fractional field arithmetic to perform elliptic curve calculations.

Let P, P_1 and P_2 be points on the curve E such that $P_2 = P_1 + P$. Let the x-coordinate of P_i be represented by X_i/Z_i, for $i \in \{1, 2\}$. From Lemma 2, when the x-coordinate of $2P_i$ is converted to projective coordinates it becomes

$$\begin{cases} x(2P_i) = X_i^4 + b \cdot Z_i^4 \ , \\ z(2P_i) = Z_i^2 \cdot X_i^2. \end{cases} \tag{8}$$

Similarly, the x-coordinate of $P_1 + P_2$ in projective coordinates can be computed as the fraction X_3/Z_3, where

$$\begin{cases} Z_3 = (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2 \ , \\ X_3 = x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1). \end{cases} \tag{9}$$

The addition formula requires three general field multiplications, one multiplication by x (i.e., the x-coordinate of P, which is fixed during the computation of kP), one squaring and two additions; doubling requires one general field multiplication, one multiplication by the constant b, five squarings, and one addition. A method based on these formulas is described in the next algorithm.

Fig. 3. Algorithm 2P: Montgomery Scalar Multiplication

INPUT: An integer $k \geq 0$ and a point $P = (x, y) \in E$.
OUTPUT: $Q = kP$.

1. `if` $k = 0$ `or` $x = 0$ `then output`$(0, 0)$ `and stop.`
2. `Set` $k \leftarrow (k_{l-1} \ldots k_1 k_0)_2$.
3. `Set` $X_1 \leftarrow x, \quad Z_1 \leftarrow 1, \quad X_2 \leftarrow x^4 + b, \quad Z_2 \leftarrow x^2$.
4. **for** i **from** $l - 2$ **downto** `0` **do**
   if $k_i = 1$ **then**
   `Madd`(X_1, Z_1, X_2, Z_2), `Mdouble`(X_2, Z_2).
   else
   `Madd`(X_2, Z_2, X_1, Z_1), `Mdouble`(X_1, Z_1).
5. **return**$(Q =$ `Mxy`$(X_1, Z_1, X_2, Z_2))$.

An implementation of the procedures `Madd, Mdouble` and `Mxy` is given in the appendix.

Lemma 5 *Algorithm 2P performs exactly the following number of field operations in* $GF(2^m)$*:*

$$\#INV. = 1 \ , \qquad \#MULT. = 6\lfloor \log_2 k \rfloor + 10 \ ,$$
$$\#ADD. = 3\lfloor \log_2 k \rfloor + 7 \ , \quad \#SQR. = 5\lfloor \log_2 k \rfloor + 3.$$

Remark. Since the complexity of both versions of Algorithm 2 does not depend on the number of 1's (or 0's) in the binary representation of k, this may help to prevent timing attacks. On the other hand, the use of restricted multipliers (e.g., with small Hamming weight) does not speedup directly Algorithms 2A and 2P, and this is a disadvantage compared to methods such as the binary method. However, from a practical point of view, most protocols in cryptographic applications use random multipliers.

4.3 Complexity Comparison

In the sequel, we assume that adding and squaring in $GF(2^m)$ is relatively fast. Now we compare the complexities of the addition-subtraction method to the complexity of the proposed method. This is a fair comparison since both methods do not use precomputation. For a random multiplier k, the addition-subtraction method in projective coordinates, given in [14], performs $8.3\log_2 k$ field multiplications; it follows we expect Algorithm 2P to be about 28% faster on average. However, if we use the formulas given in [7] for implementing the group operation in projective schemes, Algorithm 2P is about 14% faster than the addition-subtraction method. In the following table we summarize the complexities of these methods.

Table 1. Complexity Comparison of Algorithm 2P with other algorithms ($a = 0, 1$).

Method	Projective Coordinates
Binary [10]	$13\log_2 k$
Add-Sub [14]	$8.3\log_2 k$
Add-sub[7]	$7\log_2 k$
Algorithm 2P	$6\log_2 k$

Now we derive the cost of the addition-subtraction method (using affine coordinates) in terms of field multiplications. As mentioned in Section 2, this method performs on average $\frac{8}{3}\log_2 k$ field multiplications and $\frac{4}{3}\log_2 k$ field inversions. Thus, the total cost is $\frac{1}{3}(4r + 8)$ multiplications, where r is the cost-ratio of inversion to multiplication. This shows that for implementations of the finite field $GF(2^m)$ where $r > 2.5$ (see for example [1,19,4]), Algorithm 2P gives a computational advantage over the addition-subtraction method.

5 Running Times

In this section we present some running times we obtained in our software implementation of the proposed algorithm over the finite fields $GF(2^m)$, where $m = 163, 191$ and 239. To represent the finite fields we used LiDIA [6], a C++ based library. This finite field implementation uses a polynomial basis representation and the irreducible modulus is chosen as sparse as possible. We used a Sun UltraSPARC 300MHz machine. For comparison, we list in Table 2 the timings for the basic arithmetic operations in $GF(2^m)$.

Notice that one field inverse costs more than 9 field multiplications; therefore, the use of LiDIA may illustrate the performance of the proposed algorithm in situations where a field inverse is relatively expensive compared to field multiplication.

In Table 3 we present average running times for computing a scalar multiplication using several methods. These values were obtained using the following

Table 2. Average running times (in microseconds) for $GF(2^m)$ using LiDIA.

Extension m	Add.	Sqr.	Mult.	Inv.
163	0.6	2.3	10.5	96.2
191	0.7	2.0	10.9	118.1
239	0.8	2.6	14.6	162.8

Table 3. Average running times (in milliseconds) for computing mP.

Extension m	Binary[10]	Add-Sub.[14]	Algorithm 2P
163	27.5	19.1	13.5
191	33.1	22.4	16.0
239	52.3	35.1	25.6

test: we select 10 random elliptic curves ($a = 0$) over $GF(2^m)$, then we multiply a random point P in each curve with 100 randomly chosen integers of size $< 2^m$. We implemented the binary method in projective coordinates (see [10]), the addition-subtraction method [14] and Algorithm 2P. From Table 3 we conclude that the proposed method on average is 27-29% faster than the addition-subtraction method and 51% faster than the binary method. These timings show that the theoretical improvement of Algorithm 2P, given in Table 1, is observed in a actual implementation.

6 Conclusion

In this paper, we have presented an efficient method for computing elliptic scalar multiplications, which is an optimized version of an algorithm presented in [1]. The method performs exactly $6\lfloor \log_2 k \rfloor + 10$ field multiplication for computing kP on elliptic curves selected at random, is easy to implement in both hardware and software, requires no precomputations, works for any implementation of $GF(2^n)$, is faster than the addition-subtraction method on average, and uses fewer registers than methods based on projective schemes. Therefore, the method appears useful for applications of elliptic curves in constraint environments such as mobile devices and smart cards.

7 Acknowledgments

The first author would like to thank Alfred Menezes for making him possible to visit the center CACR and for his encouragement and support during this work. We also thanks the referees for their helpful comments.

References

1. G. B. Agnew, R. C. Mullin and S. A. Vanstone, "An Implementation of Elliptic Curve Cryptosystems Over $F_{2^{155}}$", *IEEE journal on selected areas in communications, Vol 11. No. 5*, June 1993.
2. ANSI X9.62: "The Elliptic Curve Digital Signature Algorithm (ECDSA)", draft, July 1997.
3. D. M. Gordon, "A survey of Fast Exponentiation Methods", *Journal of Algorithms*, 27, pp. 129-146, 1998.
4. J. Guajardo and C. Paar, "Efficient Algorithms for Elliptic Curve Cryptosystems", *Advances in Cryptology, Proc. Crypto'97, LNCS 1294,* B. Kaliski, Ed., Springer-Verlag,1997,pp. 342-356.
5. N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, 48, pp. 203-209, 1987.
6. LiDIA Group **LiDIA v1.3**- A library for computational number theory. TH-Darmstadt, 1998.
7. J. Lopez and R. Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$", *SAC'98, LNCS* Springer Verlag, 1998.
8. P. Montgomery, "Speeding the Pollard and elliptic curve methods of factorization", *Mathematics of Computation*, vol 48, pp. 243-264, 1987.
9. A. Menezes and S. Vanstone, "Elliptic curve cryptosystems and their implementation", *Journal of Cryptology*, 6, 1993, pp. 209-224.
10. A. Menezes, *Elliptic curve public key cryptosystems,* Kluwer Academic Publishers, 1993.
11. A. Menezes, P. van Oorschot and S. Vanstone, *Handbook of applied cryptography*, CRC Press, 1997.
12. V. Miller, "Uses of elliptic curves in cryptography", *Advances in Cryptology: proceedings of Crypto'85, Lecture Notes in Computer Science*, vol. 218. New York: Springer-Verlag, 19986, pp. 417-426.
13. V. Müller, "Fast Multiplication on Elliptic Curves over Small Fields of Characteristic Two", *Journal of Cryptology*, 11, 1998, pp. 219-234.
14. IEEE P1363: "Editorial Contribution to Standard for Public Key Cryptography", *draft*, 1998.
15. R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, "Fast key exchange with elliptic curve systems," *Advances in Cryptology, Proc. Crypto'95, LNCS 963,* D. Coppersmith, Ed., Springer-Verlag, 1995, pp. 43-56.
16. R. Schroeppel, "Faster Elliptic Calculations in $GF(2^n)$," *preprint*, March 6, 1998.
17. J. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," *Advances in Cryptology, Proc. Crypto'97, LNCS 1294,* B. Kaliski, Ed., Spring-Verlag, 1997, pp. 357-371.
18. E. De Win, A. Bosselaers, S. Vanderberghe, P. De Gersem and J. Vandewalle, "A fast software implementation for arithmetic operations in $GF(2^n)$," *Advances in Cryptology, Proc. Asiacrypt'96, LNCS 1163,* K. Kim and T. Matsumoto, Eds., Springer-Verlag, 1996, pp. 65-76.
19. E. De Win, S. Mister, B. Prennel and M. Wiener, "On the Performance of Signature based on Elliptic Curves", *LNCS*, 1998.

8 Appendix

`Mdouble` (Doubling algorithm)

Input: the finite field $GF(2^m)$; the field elements a and $c = b^{2^{m-1}} (c^2 = b)$ defining a curve E over $GF(2^m)$; the x-coordinate X/Z for a point P.
Output: the x-coordinate X/Z for the point $2P$.

1. $T_1 \leftarrow c$
2. $X \leftarrow X^2$
3. $Z \leftarrow Z^2$
4. $T_1 \leftarrow Z \times T_1$
5. $Z \leftarrow Z \times X$
6. $T_1 \leftarrow T_1^2$
7. $X \leftarrow X^2$
8. $X \leftarrow X + T_1$

This algorithm requires one general field multiplication, one field multiplication by the constant c, four field squarings and one temporary variable.

`Madd` (Adding algorithm)

Input: the finite field $GF(2^m)$; the field elements a and b defining a curve E over $GF(2^m)$; the x-coordinate of the point P; the x-coordinates X_1/Z_1 and X_2/Z_2 for the points P_1 and P_2 on E.
Output: The x-coordinate X_1/Z_1 for the point $P_1 + P_2$.

1. $T_1 \leftarrow x$
2. $X_1 \leftarrow X_1 \times Z_2$
3. $Z_1 \leftarrow Z_1 \times X_2$
4. $T_2 \leftarrow X_1 \times Z_1$
5. $Z_1 \leftarrow Z_1 + X_1$
6. $Z_1 \leftarrow Z_1^2$
7. $X_1 \leftarrow Z_1 \times T_1$
8. $X_1 \leftarrow X_1 + T_2$

This algorithm requires three general field multiplications, one field multiplication by x, one field squaring and two temporary variables.

`Mxy` (Affine coordinates)

Input: the finite field $GF(2^m)$; the affine coordinates of the point $P = (x, y)$; the x-coordinates X_1/Z_1 and X_2/Z_2 for the points P_1 and P_2.
Output: The affine coordinates $(x_k, y_k) = (X_2, Z_2)$ for the point P_1.

1. if $Z_1 = 0$ then output (0,0) and stop.
2. if $Z_2 = 0$ then output $(x, x + y)$ and stop.
3. $T_1 \leftarrow x$
4. $T_2 \leftarrow y$
5. $T_3 \leftarrow Z_1 \times Z_2$
6. $Z_1 \leftarrow Z_1 \times T_1$
7. $Z_1 \leftarrow Z_1 + X_1$
8. $Z_2 \leftarrow Z_2 \times T_1$
9. $X_1 \leftarrow Z_2 \times X_1$
10. $Z_2 \leftarrow Z_2 + X_2$
11. $Z_2 \leftarrow Z_2 \times Z_1$
12. $T_4 \leftarrow T_1^2$
13. $T_4 \leftarrow T_4 + T_2$
14. $T_4 \leftarrow T_4 \times T_3$
15. $T_4 \leftarrow T_4 + Z_2$
16. $T_3 \leftarrow T_3 \times T_1$
17. $T_3 \leftarrow inverse(T_3)$
18. $T_4 \leftarrow T_3 \times T_4$
19. $X_2 \leftarrow X_1 \times T_3$
20. $Z_2 \leftarrow X_2 + T_1$
21. $Z_2 \leftarrow Z_2 \times T_4$
22. $Z_2 \leftarrow Z_2 + T_2$

This algorithm requires one field inversion, ten general field multiplications, one field squaring and four temporary variables.