Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic

Tetsutaro Kobayashi, Hikaru Morita, Kunio Kobayashi, and Fumitaka Hoshino

NTT Laboratories
Nippon Telegraph and Telephone Corporation
1-1 Hikari-no-oka, Yokosuka-shi, Kanagawa-ken, 239-0847 Japan
kotetsu@isl.ntt.co.jp

Abstract. A new elliptic curve scalar multiplication algorithm is proposed. The algorithm offers about twice the troughput of some conventional OEF-base algorithms because it combines the Frobenius map with the table reference method based on base-ϕ expansion. Furthermore, since this algorithm suits conventional computational units such as 16, 32 and 64 bits, its base field $\mathbf{F}_{p^m}$ is expected to enhance elliptic curve operation efficiency more than $\mathbf{F}_q$ (q is a prime) or $\mathbf{F}_{2^n}$.

Keywords: Elliptic curve cryptosystem, Scalar multiplication, OEF, Finite field, Frobenius map, Table reference method.

1 Introduction

While speeding up modular exponentiation has been a prime approach to speeding up the RSA scheme, scalar multiplication of an elliptic curve point can speed up elliptic curve schemes such as EC-DSA and EC-ElGamal. In particular, elliptic curves over $\mathbf{F}_q$ (q is a prime) or $\mathbf{F}_{2^n}$ have been implemented by many companies and standardized by several organizations such as IEEE P1363 and ISO/IEC JTC1/SC27.

For the $\mathbf{F}_{2^n}$ type, many efficient computational algorithms have been proposed. Koblitz introduced a base-ϕ expansion method that uses a Frobenius map to multiply $\mathbf{F}_{2^n}$-rational points over the elliptic curve defined over $\mathbf{F}_2, \mathbf{F}_4, \mathbf{F}_8$ or $\mathbf{F}_{16}$ in [2]. Müller [7] and Cheon et. al. [5] extended the base-ϕ expansion method to elliptic curves defined over $\mathbf{F}_{2^r}$, where r is a small integer. Koblitz also expanded the base-ϕ expansion method to $\mathbf{F}_3, \mathbf{F}_7$ in [3].

However, since the calculation over small characteristic fields does not offer adequate speed on general purpose machines, very high-capacity tables or special-purpose machines are needed. If you select $\lceil \log_2 p \rceil$ (the bit size of a prime number p) to match the operation unit of an individual computer, the scalar multiplication of $\mathbf{F}_{p^m}$ could be calculated faster than that of $\mathbf{F}_q$ or $\mathbf{F}_{2^n}$ where $\lceil \log_2 p^m \rceil$ should be close to $\lceil \log_2 q \rceil$ or $\lceil \log_2 2^n \rceil (= n)$ under the condition of the same security level. Bailey and Paar newly proposed an elliptic curve scheme on OEF (Optimal Extension Fields), or an $\mathbf{F}_{p^m}$ type, at Crypto'98 [1].

J. Stern (Ed.): EUROCRYPT'99, LNCS 1592, pp. 176–189, 1999.

Their method represents the elliptic curve points using a polynomial basis. They showed that multiplication as well as addition and subtraction can be efficiently computed by introducing a binomial as a minimal polynomial.

Though the original OEF method simply indicated how to compute addition and multiplication on $\mathbf{F}_{p^m}$, efficient computational techniques similar to those developed for the $\mathbf{F}_{2^n}$ type have not been introduced to the OEF world.

This paper extends the base-ϕ extension method from $\mathbf{F}_{2^n}$ to the general finite field $\mathbf{F}_{p^m}$ by using a table reference method. Several table reference methods have been developed for schemes using fixed primitive points – base points – such as the DSA scheme [6]. Ours is the first to combine the Frobenius map and the table reference method and so does not need any pre-computation. It can be applied to any higher-characteristic elliptic curve as well as the small-characteristic. When p equals two, this method is reduced to Koblitz's method. Different from Cheon's method, this method isn't limited to an elliptic curve defined on $\mathbf{F}_{2^r}$. The method works over OEF-type elliptic curves because the table reference method is effective even if p is large. If you select p close to 2^{16}, 2^{32} or 2^{64}, that are suitable operation units for computers, our method is about twice as fast as ordinary OEF methods.

Section 2 describes the idea of our proposed method. Its procedure is given in Sect. 3. Section 4 shows how to construct the proposed OEF parameters. Its efficiency and further techniques are given in Sects. 5 and 6. Section 7 concludes this paper.

2 Approach

2.1 Frobenius Map

In this section, we define the Frobenius map. Let $E/\mathbf{F}_p$ denote a non-supersingular elliptic curve defined over a finite field $\mathbf{F}_p$ where p is a prime or any power of a prime. $P = (\boldsymbol{x}, \boldsymbol{y})$ is an $\mathbf{F}_{p^m}$-rational point of elliptic curve E defined over $\mathbf{F}_p$. The Frobenius map ϕ is defined as

$$\phi : (\boldsymbol{x}, \boldsymbol{y}) \to (\boldsymbol{x}^p, \boldsymbol{y}^p).$$

The Frobenius map is an endomorphism over $E(\mathbf{F}_{p^m})$. It satisfies the equation

$$\phi^2 - t\phi + p = 0, \qquad -2\sqrt{p} \le t \le 2\sqrt{p}. \tag{1}$$

Since E is non-supersingular, the endomorphism ring of E is an order of the imaginary quadratic field $\mathbf{Q}(\sqrt{t^2 - 4p})$ [8]. The ring $\mathbf{Z}[\phi]$ is a subring of the endomorphism ring.

To compute the Frobenius map ϕ takes negligible time, provided that element $\boldsymbol{a} \in \mathbf{F}_{p^m}$ is represented using a normal basis of $\mathbf{F}_{p^m}$ over $\mathbf{F}_p$.

2.2 Normal Basis and Polynomial Basis

The elements of the field $\mathbf{F}_{p^m}$ can be represented in several different ways; for example, "polynomial basis" and "normal basis." In polynomial basis, element $\boldsymbol{a} \in \mathbf{F}_{p^m}$ is represented as

$$\boldsymbol{a} = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0. \tag{2}$$

where $a_i \in \mathbf{F}_p$ and α is a defining element of $\mathbf{F}_{p^m}$ over $\mathbf{F}_p$.

In normal basis, $\boldsymbol{a} \in \mathbf{F}_{p^m}$ is represented as

$$\boldsymbol{a} = a_{m-1}\alpha^{p^{m-1}} + \cdots + a_1\alpha^p + a_0\alpha \tag{3}$$

where $a_i \in \mathbf{F}_p$ and α is a generator of normal basis.

Addition and subtraction in $\mathbf{F}_{p^m}$ are quite fast in both representation forms. When you choose polynomial basis, multiplication and squaring can be done with reasonable speed.

When you choose normal basis, the p-th power operation, which is equal to the Frobenius map, is quite fast. Though multiplication isn't fast in general normal basis, there are several techniques for fast multiplication in $\mathbf{F}_{2^m}$ such as the optimal normal basis [9]. Thus, fast algorithms for scalar multiplication using the Frobenius map [2] have been developed using $\mathbf{F}_2$ or its extension field represented by normal basis.

On the other hand, we developed a fast Frobenius map algorithm for OEF [1] which has a special polynomial basis.

2.3 Frobenius Map for OEF

Let OEF be the finite field $\mathbf{F}_{p^m}$ that satisfies the following:

- p is a prime less than but close to the word size of the processor,
- $p = 2^n \pm c$, where $\log_2 c \leq n/2$ and
- An irreducible binomial $f(x) = x^m - \omega$ exists.

Although the paper [1] showed that OEF has an efficient algorithm for multiplication and squaring, there was no discussion of the Frobenius map. In this section, we present a new algorithm to compute the Frobenius map in OEF.

We consider the following polynomial basis representation of an element $\boldsymbol{a} \in \mathbf{F}_{p^m}$:

$$\boldsymbol{a} = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0$$

where $a_i \in \mathbf{F}_p$, $\alpha \in \mathbf{F}_{p^m}$ is a root of $f(x)$. Since we choose $\lceil \log_2 p \rceil$ to be less than the processor's word size, we can represent $\boldsymbol{a}$ using m registers.

The Frobenius map moves $\boldsymbol{a}$ to $\boldsymbol{a}^p$;

$$\phi(\boldsymbol{a}) = \boldsymbol{a}^p = a_{m-1}\alpha^{(m-1)p} + \cdots + a_1\alpha^p + a_0. \tag{4}$$

Since α is a root of $f(x) = 0$, $\alpha^m = \omega$,

$$\alpha^{ip} = \alpha^{(ip \bmod m)}\omega^{\lfloor ip/m \rfloor}$$

$$\text{where } \lfloor x \rfloor \text{ is the maximum integer not exceeding } x.$$

Assuming $\gcd(m, p) = 1$, $(i_1p \bmod m) = (i_2p \bmod m)$ is equivalent to $i_1 = i_2$. Thus, the map $\pi(i) \overset{\triangle}{=} ip \bmod m$ is bijective.

We rewrite Equation (4) using $\pi(i)$ as follows:

$$\boldsymbol{a}^p = a'_{m-1}\alpha^{m-1} + \cdots + a'_1\alpha + a'_0,$$
$$\text{where } a'_{\pi(i)} \overset{\triangle}{=} a_i\omega^{\lfloor \frac{ip}{m} \rfloor}.$$

Since p, m and ω are independent of an element $\boldsymbol{a}$, we can pre-compute $\omega_i = \omega^{\lfloor \frac{ip}{m} \rfloor}$ before computing the Frobenius map. Accordingly, the complete procedure to compute the Frobenius map to an element on OEF is as follows;

[Frobenius Map Procedure for OEF]

```
Input: $[a_0, \ldots, a_{m-1}]$   $(= \boldsymbol{a})$
Output: $[a'_0, \ldots, a'_{m-1}]$   $(= \phi(\boldsymbol{a}))$
Step 1: compute $b_i = a_i\omega_i$,      for $i = 1$ to $m - 1$.
Step 2: compute $a'_{\pi(i)} = b_i$,      for $i = 1$ to $m - 1$.
Step 3: $a'_0 = a_0$.
```

This procedure needs only $m-1$ multiplications on $\mathbf{F}_p$. This takes negligible time compared to multiplication on $\mathbf{F}_{p^m}$, which needs m^2 multiplications [1] on $\mathbf{F}_p$.

2.4 Base-ϕ Scalar Multiplication Method

This section describes the basic idea of base-ϕ scalar multiplication given by Koblitz[2].

Consider scalar multiplication, kP where k and P represent a scalar multiplier and an elliptic curve point P, respectively. Consider $k = 15$ as an example. By using the binary method, $15P$ is calculated as $2(2(2P + P) + P) + P$ by three elliptic curve doublings and three elliptic curve additions. If you use the signed-binary method, $15P$ is calculated as $2(2(2(2P))) - P$ by four elliptic curve doublings and one elliptic curve subtraction. General computational times are given by Table 1 where $n = \lceil \log_2 p^m \rceil$.

Base-ϕ expansion is generally calculated as follows: If an intermediate multiplier k_i is represented by $k_i = x_i + y_i\phi$ where x_i and y_i are integers, $x_0 = k$ and $y_0 = 0$, the equation is modified to $k_i = u_i + k_{i+1}\phi$, where u_i is defined as an integer such that $u_i = x_i \pmod{p}$ and $-\frac{p}{2} < u_i \leq \frac{p}{2}$.

[1] This is the straightforward method.

Table 1. Computational Times for Binary Method

		EC Doubling	EC Addition	Total
Binary	(maximum)	n	n	$2n$
	(avarage)	n	$\frac{n}{2}$	$\frac{3n}{2}$
Signed Binary	(maximum)	n	$\frac{n}{2}$	$\frac{3n}{2}$
	(avarage)	n	$\frac{3n}{8}$	$\frac{11n}{8}$

$k_{i+1} = x_{i+1} + y_{i+1}\phi$, $x_{i+1} = y_i + t\frac{x_i - u_i}{p}$ and $y_{i+1} = -\frac{x_i - u_i}{p}$ by using $\phi^2 - t\phi + p = 0$. Iterating this operation, k is expanded to

$$k = \sum_{i=0}^{l} u_i \phi^i, \qquad \text{where } -\frac{p}{2} \le u_i \le \frac{p}{2}. \tag{5}$$

l is an integer and is discussed in Sect. 3.

In the case of $k = 15, p = 2$, the elliptic curve is $E/\mathbf{F}_2 : y^2 + xy = x^3 + 1$ (trace t is 1 as an example), $15 = -1 + \phi^4 - \phi^8$. Accordingly, $15P$ is calculated by two elliptic curve additions, which is much faster than the signed or unsigned binary method.

Koblitz[2] presented the scalar multiplication algorithm for $\mathbf{F}_{2^m}$-rational points over $E/\mathbf{F}_2$. Solinas[4] improved it. In those papers $u_i \in \{-1, 0, 1\}$. Thus it needs at most l elliptic curve additions and computation of the Frobenius map to calculate kP.

On the other hand, we must limit the elliptic curve defined over $E/\mathbf{F}_p$ for $\mathbf{F}_{p^m}$-rational points to utilize a base-ϕ scalar multiplication method. Since there is only an exponential time attack such as [10], it is not obstacle to use the elliptic curves for elliptic curve cryptosystems.

2.5 Generalized Base-ϕ Scalar Multiplication

We consider the fact that the cost of $\phi^i P$ can be reduced very much for OEF as shown at Section 2.3. Though traditional base-ϕ scalar multiplication has been applied to finite fields with small characteristics, it can be applied to more general cases such as OEF.

This, however, makes each coefficient u_i in Equation (6) large, because $0 \le |u_i| \le p/2$. When u_i is large, the delay time to calculate $u_i\phi^i P$ from $\phi^i P$ becomes a bottle neck. Thus, the traditional base-ϕ method is not always faster than the binary method. This is one reason why the base-ϕ scalar multiplication method was applied only to fields with small characteristics.

To solve this problem, we introduce the idea of the table reference scalar multiplication method. After each value of $\phi^i P$ is stored in a memory table, we should perform addition on the elliptic curve. There are two different ways to look up the table and add.

One method uses only addition. If $15P + 13\phi^2 P + 2\phi^3 P$ is to be calculated, $X \leftarrow P$ and $Y \leftarrow O$ then $Y \leftarrow Y + X$ is computed twice. $X \leftarrow X + \phi^2 P$ then $Y \leftarrow Y + X$ is computed eleven times. $X \leftarrow X + \phi^3 P$ then $Y \leftarrow Y + X$ is computed twice. Generally speaking, when you compute

$$k = \sum_{i=0}^{l} u_i \phi^i \tag{6}$$

where $-\dfrac{p}{2} \leq u_i \leq \dfrac{p}{2}$, the elliptic curve addition should be computed roughly as $l + \dfrac{p}{2}$. This idea of the original table reference method was created for the non-elliptic curve scheme by Brickell et al. [6]. By introducing the base-ϕ method, this method can be enhanced to handle any primitive. Thus, our method supports not only signature generation by EC-DSA but also verification which involves multiplication of unpredictable elliptic curve points.

The second method uses both doubling and addition. If $15P + 13\phi^2 P + 2\phi^3 P = (1111)_2 P + (1101)_2 \phi^2 P + (0010)_2 \phi^3 P$ is calculated, $X \leftarrow P + \phi^2 P$ is computed then doubled. $X \leftarrow X + P + \phi^2 P$ is computed then doubled. $X \leftarrow X + P + \phi^3 P$ is computed then doubled. Finally, $X \leftarrow X + P + \phi^2 P$ is computed. In general, when you compute $k = \sum_{i=0}^{l} u_i \phi^i$ where $-\dfrac{p}{2} \leq u_i \leq \dfrac{p}{2}$, the elliptic curve addition and doubling should be computed roughly $(l+1)(\lceil \log_2 p \rceil - 1)/2$ and $\lceil \log_2 p \rceil - 2$ times, respectively. If the reference table contains not only P and $\phi^i P$ but also their combinations such as $P + \phi^2 P$, $P + \phi^3 P$ and so on, the addition and doubling times could be reduced by at least $\lceil \log_2 p \rceil - 2$.

We found that there are some trade offs. Which of the two methods is better? How many values should the memory table store? It depends on the case.

3 Procedure

3.1 Base-ϕ Scalar Multiplication

In this section, we describe the base-ϕ scalar multiplication method. The following procedure computes $Q = kP$ for inputs P and k, where $0 < k < N_m$ and P is an $\mathbf{F}_{p^m}$-rational point and is not an $\mathbf{F}_p$-rational point on E. We use $w_H(x)$ to represent the Hamming weight of x expressed in signed binary digit, N_m denotes the number of $\mathbf{F}_{p^m}$-points on E, and t denotes a trace of E.

[**Base-ϕ Scalar Multiplication Procedure**]

```
Input: $k, P, E, t, p$
Output: $Q \quad (= kP)$
Step 1: Base-$\phi$ Expansion of $k$
  Step 1-1: $i \leftarrow 0, \quad x \leftarrow k, \quad y \leftarrow 0, \quad u_j \leftarrow 0 \quad$ for $\forall j$.
  Step 1-2: if $(x = 0$ and $y = 0)$ then go to Step 2:.
  Step 1-3: $u_i \leftarrow x \bmod p$.
  Step 1-4: $v \leftarrow (x - u_i)/p, \quad x \leftarrow tv + y, \quad y \leftarrow -v, \quad i \leftarrow i + 1$.
  Step 1-5: go to Step 1-2:.
Step 2: Optimization of Base-$\phi$ Expansion
  Step 2-1: $d_i \leftarrow u_i + u_{i+m} + u_{i+2m} \quad$ for $0 \leq i < m$.
  Step 2-2: $c_i \leftarrow d_i - z \quad$ for $0 \leq i \leq m - 1$,
      where $z$ is an integer that minimizes $\sum_i w_H(c_i)$.
Step 3: Table Reference Multiplication
  Step 3-1: $P_i \leftarrow \phi^i P \quad$ for $0 \leq i < m$.
  Step 3-2: $Q \leftarrow \mathcal{O}, \quad j \leftarrow \lceil \log_2 p \rceil + 1$.
  Step 3-3: $Q \leftarrow 2Q$.
  Step 3-4: for $(i = 0$ to $m - 1)$ {
          if $(c_{ij} = 1)$ then $Q \leftarrow Q + P_i$.
    }
  Step 3-5: $j \leftarrow j - 1$.
  Step 3-6: if $(j \geq 0)$ then go to Step 3-3.
```

First, the procedure finds u_i such that $k = \sum_{i=0}^{l} u_i \phi^i$ in **Step 1** by using $\phi^2 - t\phi + p = 0$. This part of the procedure is nearly equal to the procedure in [7] and integer l is nearly equal to $2m + 3$. This is discussed in Sect. 3.2.

Next, it reduces the series of base-ϕ expansion $\{u_0, \ldots, u_l\}$ into $\{c_0, \ldots, c_{m-1}\}$ in **Step 2**. Detailed explanation is given in Sect. 3.3.

Finally, it calculates kP using $\{c_0, \ldots, c_{m-1}\}$ in **Step 3**. **Step 3** requires $\lceil \log_2 p \rceil$ elliptic curve doublings and $\frac{m \lceil \log_2 p \rceil}{2}$ elliptic curve additions at most. On the other hand, we can use the following **Step 3′** in stead of **Step 3** to compute kP using the Frobenius map. **Step 3′** requires $p + m + 2$ elliptic curve additions at most. We can choose the method which has lower computation cost.

[**Another Table Reference Multiplication Procedure**]

```
Step 3′: Table Reference Multiplication

  Step 3′-1: $Q \leftarrow O, \quad S \leftarrow O, \quad d \leftarrow \max_i \{c_i\}$.
  Step 3′-2: for $(i = 0$ to $m - 1)$ {
          if $d = c_i$ then $S = S + \phi^i P$.
    }
  Step 3′-3: $Q \leftarrow Q + S, \quad d \leftarrow d - 1$.
  Step 3′-4: if $d \neq 0$ then go to Step 3′-2.
```

3.2 Loop Number of Step 1

In this section, we discuss l in **Step 1**.

Theorem 1. [7] *Let $p \geq 4$ and let $k \in \mathbf{Z}[\phi]$. If we set $l = \lceil 2\log_p ||k|| \rceil + 3$, then there exist rational integers $-p/2 \leq u_i \leq p/2$, $0 \leq i \leq l$, such that*

$$k = \sum_{i=0}^{l} u_i \phi^i \tag{7}$$

where $||k|| := \sqrt{k\bar{k}}$ and $\bar{k}$ is the complex conjugate of k and $\lceil x \rceil$ is the minimum integer greater than or equal to x.

Since the proof of Theorem 1 in [7] does not assume p to be a small power of two, the loop in **Step 1** ends at most in $i \leq \lceil 2\log_p ||k|| \rceil + 3$ for general p.

3.3 Optimization of Base-ϕ Expansion

This section explains the background of the procedure in **Step 2**.

Step 2-1 If k is randomly chosen from $0 < k < N_m$, we can assume $k \simeq p^m$ and $l = \lceil 2\log_p k \rceil + 3 \simeq 2m + 3$. However, the series of base-ϕ expansion $\{u_0, \ldots, u_{2m+3}\}$ can be easily reduced to $\{d_0, \ldots, d_{m-1}\}$ by using the following equation;

$$\phi^m = 1 \quad \text{in } \mathrm{End}_E. \tag{8}$$

This is because $x^{p^m} = x$ for $\forall x \in \mathbf{F}_{p^m}$. Thus,

$$\begin{aligned} \sum_{i=0}^{\lceil 2\log_p k \rceil + 3} u_i \phi^i &= \sum_{i=0}^{m-1} (u_i + u_{i+m} + u_{i+2m}) \phi^i \\ &= \sum_{i=0}^{m-1} d_i \phi^i. \end{aligned}$$

Step 2-2 We can accelerate **Step 3** by decreasing the density of '1's in the bit expression of d_i by using Equation (9).

$$\sum_{i=0}^{m-1} \phi^i = 0 \tag{9}$$

Since P is not an $\mathbf{F}_p$-point over E, $\phi P \neq P$. Equation (9) is derived from Equation (8) and $\phi \neq 1$.

The theoretical required time for scalar multiplication for the case of $m = 7$ and $A = \lceil \log_2 p \rceil$ is shown in Table 2. "Type I expansion" denotes the proposed procedure using d_i instead of c_i at **Step 3** and "Type II expansion" denotes the full proposed procedure.

Table 2. Required Time for Scalar Multiplication ($m = 7$)

Algorithm	EC Addition	EC Doubling	Total
binary	$\frac{7}{2}A$	$7A$	$\frac{21}{2}A \quad (\simeq 10.5A)$
signed binary	$\frac{21}{8}A$	$7A$	$\frac{77}{8}A \quad (\simeq 9.6A)$
Type I expansion	$\frac{7}{2}A$	A	$\frac{9}{2}A \quad (\simeq 4.5A)$
Type II expansion	$\frac{77}{32}A$	A	$\frac{109}{32}A \quad (\simeq 3.4A)$

4 Elliptic Curve Generation

In this section, we discuss how to generate elliptic curves for the base-ϕ expansion method.

Let p be a prime, where $p > 3$ and let E be the elliptic curve

$$Y^2 = X^3 + aX + b \tag{10}$$

over $\mathbf{F}_p(a, b \in \mathbf{F}_p)$. We should define elliptic curve E over $\mathbf{F}_p$ to use base-ϕ expansion. In such a case, we can easily compute N_m by using Theorem 2.

Theorem 2 (Weil Conjecture [8, pp.132-137]). *Suppose E is an elliptic curve over $\mathbf{F}_p$ and $t := p + 1 - N_1$. The number of $\mathbf{F}_{p^m}$-points on E is*

$$N_m = p^m + 1 - (\alpha^m + \beta^m),$$

where α, β are the roots of $x^2 - tx + p$.

From the view point of cryptography, E is a "good" elliptic curve if N_m has a large prime factor. Since $N_n \simeq p^n$ and N_n divides N_m if n divides m, we have the best chance of getting a large prime factor of N_m when m is a prime. We can generate elliptic curve $E/\mathbf{F}_p$ with N_m that has a large prime factor by using the following procedure.

[Elliptic Curve Generation Procedure for Base-ϕ Expansion]

```
Input: $p, m$
Output: $E/\mathbf{F}_p, N_m$
Step 1: Generate $E/\mathbf{F}_p$ randomly and find its order $N_1 = p + 1 - t$.
Step 2: Find $N_m$ using the Weil conjecture.
Step 3: If $N_m$ doesn't have a large enough prime factor, go to Step 1.
```

For example, let $p = 2^{31} - 1$, $m = 7$ and $M := \dfrac{N_m}{N_1}$ $(N_1 \simeq 2^{31}, N_m \simeq 2^{186})$. We can find some parameters such that M becomes prime. One example is $a = -3, b = -212, \lceil \log_2 M \rceil = 186$.

5 Further Speed Up Techniques

5.1 Affine Coordinates

Points on an elliptic curve can be represented by some different coordinate systems: for example, affine coordinates or Jacobian coordinates, as shown in [11] and the Appendix. The number of operations on a finite field differ with the system. If you choose Jacobian coordinates, no inversion on the finite field is needed, but "elliptic curve addition" needs ten or more multiplications on the field. On the other hand, if you choose affine coordinates, "elliptic curve addition" needs one inversion and two multiplications. Thus, if inversion is faster than 8 multiplications, affine coordinates are faster than Jacobian coordinates. The implementation in [1] used the Jacobian coordinates because no efficient inversion algorithm for OEF has been proposed. Therefore, if we have a fast enough inversion algorithm, affine coordinates can accelerate elliptic curve operation.

In this section, we present a fast inversion algorithm for OEF. We consider the polynomial basis representation of a field element $\boldsymbol{a} \in \mathbf{F}_{p^m}$:

$$\boldsymbol{a} = a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0$$

where $a_i \in \mathbf{F}_p$, $\alpha \in \mathbf{F}_{p^m}$ is a primitive root of $x^m - \omega$.

The inversion $\boldsymbol{c}$ of $\boldsymbol{a}$ is defined as

$$\boldsymbol{ac} = (a_{m-1}\alpha^{m-1} + \cdots + a_1\alpha + a_0)(c_{m-1}\alpha^{m-1} + \cdots + c_1\alpha + c_0) = 1.$$

Since α is a root of $x^m - \omega$,

$$\boldsymbol{ac} = \left(\left(\sum_{0 \le u \le m-1} (a_u c_{m-1-u})\right)\alpha^{m-1}\right) + \sum_{0 \le t \le m-2}\left(\left(\sum_{0 \le u \le t} (a_u c_{t-u}) + \left(\sum_{t+1 \le u \le m-1} \left(a_u c_{t+m-u}\right)\right)\omega\right)\alpha^t\right) = 1. \quad (11)$$

We introduce c_i from Equation(11).

$$\begin{pmatrix} c_0 \\ c-1 \\ \vdots \\ c_{m-2} \\ c_{m-1} \end{pmatrix} = \begin{pmatrix} a_0 & a_{m-1}\omega & a_{m-2}\omega & \cdots & a_2\omega & a_1\omega \\ a_1 & a_0 & a_{m-1}\omega & \cdots & \cdots & a_2\omega \\ a_2 & a_1 & a_0 & \ddots & & \vdots \\ \vdots & & & & a_0 & a_{m-1}\omega \\ a_{m-1} & a_{m-2} & \cdots & \cdots & a_1 & a_0 \end{pmatrix}^{-1} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}. \quad (12)$$

Table 3. Calculation Cost for Each Coordinate System over OEF($m = 3$)

Coordinates	EC Doubling	EC Addition
Affine	57M	51M
Chudnovsky Jacobian	81M	117M
Modified Jacobian	72M	153M

For example, if $m = 3$ then

$$\begin{pmatrix} c_0 \\ c_1 \\ c_2 \end{pmatrix} = (a_0^3 - 3a_0a_1a_2\omega + a_1^3\omega + a_2^3\omega^2)^{-1} \begin{pmatrix} a_0^2 - a_1a_2\omega \\ a_2^2\omega - a_0a_1 \\ a_1^2 - a_0a_2 \end{pmatrix}.$$

[**Inverse Procedure for OEF,** $m = 3$]

```
Input: $[a_0, a_1, a_2]$   $(= \boldsymbol{a})$
Output: $[c_0, c_1, c_2]$   $(= \boldsymbol{c} = \boldsymbol{a}^{-1})$
Step 1: Compute $b_0 \leftarrow a_0^2$,   $b_1 \leftarrow a_1^2$,   $b_2 \leftarrow a_2^2\omega$,
        $e_0 \leftarrow a_0a_1$,   $e_1 \leftarrow a_1a_2\omega$,   $e_2 \leftarrow a_0a_2$,
        $e_3 \leftarrow a_0b_0$,   $e_4 \leftarrow a_1b_1\omega$,   $e_5 \leftarrow a_2(b_2 - 3e_0)\omega$.
Step 2: Compute $d \leftarrow (e_3 + e_4 + e_5)^{-1}$.
Step 3: Compute $c_0 \leftarrow d(b_0 - e_1)$,   $c_1 \leftarrow d(b_2 - e_0)$,   $c_2 \leftarrow d(b_1 - e_2)$
```

Since we can normally use ω as a small integer such as 2 or 3, we ignore multiplication by ω and 3 to count computing cost. Using this procedure, inversion over $\mathbf{F}_{p^3}$ needs 12 multiplications and one inversion over $\mathbf{F}_p$.

Let M denote the cost of multiplication over $\mathbf{F}_p$ and let the cost of inversion over $\mathbf{F}_p$ be $15M$. Then, the costs of multiplication, squaring, and inversion over $\mathbf{F}_{p^m}$ are $9M$, $6M$, and $27M$, respectively. In this case, the elliptic curve operation costs in each coodinate are as shown in Table 3. The operations over affine coordinates are about twice as fast as those over Jacobian coordinates.

Though the proposed inversion algorithm needs $O(m^3)$ computing cost, it is efficient enough for small m.

6 Total Efficiency

We show the total efficiency of the base-ϕ scalar multiplication method.

Table 4 shows the current results of our elliptic curve implementation. We implemented our algorithms on a 500 MHz DEC Alpha workstation which has a 64-bit architecture and a 400 MHz Intel Pentium II PC which has a 32-bit architecture.

We executed the elliptic curve generation algorithm shown in the procedure described in Sect. 4 for the word sizes of 16 and 32.

Table 4. Scalar Multiplication Speed

Base-ϕ Expansion Method

Platform	Order (bit)	Size of Base Field	EC-Add	EC-Double	Scalar Mult.	
			(μ**sec**)		**(msec)**	
P II 400	186	$2^{31}-1$	19.7	13.2	**1.95**	**Base-ϕ**
P II 400	186	$2^{31}-1$	19.7	13.2	3.89	Signed Binary
P II 400	156	$2^{13}-1$	32.1	22.3	**2.66**	**Base-ϕ**
P II 400	156	$2^{13}-1$	32.1	22.3	5.50	Signed Binary

"P II 400" denotes 400 MHz Pentium II PC.

Affine Coordinates

Platform	Order (bit)	Size of Base Field	EC-Add	EC-Double	Scalar Mult.	
			(μ**sec**)		**(msec)**	
Alpha 500	183	$2^{61}-1$	4.64	5.25	**0.994**	**Affine**
Alpha 500	183	$2^{61}-1$	7.8	6.24	1.58	Jacobian(Bailey[1])

"Alpha 500" denotes 500 MHz DEC Alpha workstation.

Speed in Previous Works

Platform	Order (bit)	Size of Base Field	EC-Add	EC-Double	Scalar Mult.	
			(μ**sec**)		**(msec)**	
Sparc4	180	2^5	$*^a$	$*^a$	59.2	Muller[7]
P 133	177	2	306	309	72	De Win[12]
Alpha 500	160	$2^{32}-5$	20	16.2	3.62	Bailey[1]
Alpha 500	160	$2^{16}-165$	207	166	37.1	Bailey[1]

"Sparc4" denotes SparcStation4.
"P 133" denotes 133 MHz Pentium PC.

[a] No information in [7].

The parameters used in the implementation are as follows:

[64-bit OEF]
$p=2^{61}-1, \quad m=3, \quad f(x)=x^3-37,$
$E: y^2=x^3-ax-a,$
where $a=1798615821903599087\alpha^2+257902442738591772\alpha$
$+1373279171338599842,$
α is a root of $f(x)$,

[32-bit OEF]
$p=2^{31}-1, \quad m=7, \quad f(x)=x^7-3,$
$E: y^2=x^3-3x-212,$

[16-bit OEF]
$p=2^{13}-1, \quad m=13, \quad f(x)=x^{13}-2,$
$E: y^2=x^3-3x+30,$

where $f(x)$ is a minimal polynomial.

We implemented 16-bit and 32-bit cases on the Pentium II ("P II" in Table 4) to examine the effectiveness of the base-ϕ scalar multiplication method. We implemented the 64-bit case on the DEC Alpha ("Alpha" in Table 4) to examine effectiveness of affine coordinates. "Speed in Previous Works" in Table 4 is shown as reference.

The results clarify that the proposed base-ϕ expansion method speeds up scalar multiplication by a factor of two over the traditional signed binary method in the 16-bit and 32-bit OEF cases. In the case of 64-bit OEF, the new inversion algorithm is about 60% faster for scalar multiplication.

7 Conclusions

This paper proposed a new algorithm that computes the Frobenius map and inversion over OEF-type finite field $\mathbf{F}_{p^m}$. We need only $m-1$ multiplications over $\mathbf{F}_p$ to compute the Frobenius map. The inversion algorithm needs one inversion and $O(m^3)$ multiplications over $\mathbf{F}_p$, and it is quite efficient for small m.

Consequently, we expanded the base-ϕ scalar multiplication method to suit finite fields with higher characteristic (such as OEF) by introducing the table reference method. When the proposed algorithm is applied to OEF-type elliptic curves, the algorithm is about twice as fast as some conventional OEF-base algorithms.

We proved the total efficiency of the proposed algorithm by implementation. In the case of 16-bit and 32-bit OEF, the base-ϕ expansion method is twice as fast as traditional techniques. In the case of 64-bit OEF, the calculation time is 1.6 times shorter due to use of the new inversion algorithm.

Acknowledgments

We are very grateful to Kazumaro Aoki of NTT for implementing OEF primitives on the Pentium II architecture, and Eisaku Teranishi of NTT Advanced Technology for implementing our base-ϕ method.

References

1. D. V. Bailey and C. Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms," Advances in Cryptology – CRYPTO '98, Lecture Notes in Computer Science 1462, pp.472-485, Springer, 1998.
2. N. Koblitz, "CM-Curves with Good Cryptographic Properties," Advances in Cryptology – CRYPTO'91, Lecture Notes in Computer Science 576, pp.279-287, Springer-Verlag, 1992.
3. N. Koblitz, "An Elliptic Curve Implementation of the Finite Field Digital Signature Algorithm," Advances in Cryptology – CRYPTO'98, Lecture Notes in Computer Science 1462, pp.327-337, Springer-Verlag, 1998.
4. J. A. Solinas "An Improved Algorithm for Arithmetic on a Family of Elliptic Curves," Advances in Cryptology – CRYPTO'97, Lecture Notes in Computer Science 1294, pp.357-371, Springer, 1997.

5. J. H. Cheon, S. Park and S. Park, D. Kim, "Two Efficient Algorithms for Arithmetic of Elliptic Curves Using Frobenius Map," Public Key Cryptography: Proceedings of the First international workshop, PKC '98, Lecture Notes in Computer Science 1431, pp.195-202, Springer, 1998.
6. E. F. Brickell, D. M. Gordon, K. S. McCurley and D. B. Wilson, "Fast Exponentiation with Precomputation," Advances in Cryptology – EUROCRYPT'92, Lecture Notes in Computer Science 658, pp.200-207, Springer, 1993.
7. V. Müller, "Fast Multiplication on Elliptic Curves over Small Fields of Characteristic Two," Journal of Cryptology(1998) 11, pp.219-234, 1998.
8. J. H. Silverman, *The Arithmetic of Elliptic Curves*, Springer-Verlag, New York, 1986.
9. R. Mullin, I. Onyszchuk, S. Vanstone, and R. Wilson, "Optimal Normal Basis in $GF(p^n)$," Discrete Applied Mathematics, 22:149-161, 1988.
10. M. J. Wiener and R. J. Zuccherato, "Faster Attacks on Elliptic Curve Cryptosystems," Fifth Annual Workshop on Selected Areas in Cryptography – SAC'98 pp.196-207, Workshop Record, 1998.
11. H. Cohen, A. Miyaji and T. Ono, "Efficient Elliptic Curve Exponentiation Using Mixed Coordinates," Advances in Cryptology – ASIACRYPT'98, Lecture Notes in Computer Science 1514, pp.51-65, Springer-Verlag, 1998.
12. E. De Win, A. Bosselaers and S. Vandenberghe, "A Fast Software Implementation for Arithmetic Operations in GF(2^n)," Advances in Cryptology – ASIACRYPT'96, Lecture Notes in Computer Science 1163, pp.65-76, Springer-Verlag, 1996.

Appendix: Coordinates

Let

$$E : y^2 = x^3 + ax + b \qquad (a, b \in \mathbf{F}_p, 4a^3 + 27b^2 \neq 0)$$

be the equation of an elliptic curve E over $\mathbf{F}_p$.

For Jacobian coordinates, with $x = X/Z^2$ and $y = Y/Z^3$, a point on elliptic curve P is represented as $P = (X, Y, Z)$. In order to make addition faster, the Chudnovsky Jacobian coordinates represents a Jacobian point as the quintuple (X, Y, Z, Z^2, Z^3). On the other hand, in order to make doubling faster, the modified Jacobian coordinates represents a Jacobian point as the quadruple (X, Y, Z, aZ^4).

The number of operations needed to compute elliptic curve doubling and addition is shown in Table 5.

Table 5. Operations for Each Coordinate

Coordinates	Elliptic Curve Doubling	Elliptic Curve Addition
Affine	2 M + 2 S + 1 I	2 M + 1 S + 1 I
Chudnovsky Jacobian	5 M + 6 S	11 M + 3 S
Modified Jacobian	4 M + 4 S	13 M + 6 S

M: Multiplication, S: Squaring, I: Inversion.