PAPER Special Section on Cryptography and Information Security

A Cyclic Window Algorithm for Elliptic Curves over OEF

Tetsutaro KOBAYASHI[†a)], ***Regular Member***, **Fumitaka HOSHINO**[†], ***Nonmember***, ***and*** **Kazumaro AOKI**[†], ***Regular Member***

SUMMARY This paper presents a new sliding window algorithm that is well-suited to an elliptic curve defined over an extension field for which the Frobenius map can be computed quickly, e.g., optimal extension field. The algorithm reduces elliptic curve group operations by approximately 15% for scalar multiplications for a practically used curve in compared to Lim-Hwang's results presented at PKC2000, which was the fastest previously reported. The algorithm was implemented on computers. Scalar multiplication can be accomplished in 573 μs, 595 μs, and 254 μs on Pentium II (450 MHz), 21164A (500 MHz), and 21264 (500 MHz) computers, respectively.

key words: *cyclic window algorithm, optimal extension field, base-ϕ expansion, elliptic curve cryptosystem*

1. Introduction

Many studies have been conducted on fast exponentiation algorithms, since many public key cryptosystems require exponentiation g^e [1, Sect. 8]. If the base or exponent cannot be fixed, the sliding window exponentiation [1, 14.85 Algorithm in p.616] is one of the most efficient algorithms for computing exponentiations.

Recently, elliptic curve cryptosystems have been the focus of much attention, since they provide many advantages, for example, a short key length and fast computation speed. In particular, the use of GF(p^m), an optimal extension field (OEF) [2] for software implementation, has determined that an elliptic curve cryptosystem is faster than a public key cryptosystem based on modular exponentiations.

The algorithms for an exponentiation can be used for scalar multiplications in group operations defined over an elliptic curve. Moreover, if an extension field is used for an elliptic curve, another technique called the base-ϕ expansion [3] can be employed. Until now, several studies on base-ϕ expansion have been conducted including application to OEF [4], but these studies focused on expanding the scalar representation to a base-ϕ representation and the results of the expansion are only analogous to modular exponentiation algorithms.

Solinas presented base-ϕ expansion with a window algorithm for an elliptic curve [5]. His algorithm considered only a field with characteristic two, and he used a (sliding) window algorithm. Meanwhile, Lim and Hwang presented the Lim-Lee algorithm [6] which was applied to an elliptic curve defined over OEF [7], [8]. They used a more sophisticated algorithm to compute a scalar multiplication using the base-ϕ representation, but they did not use any special properties derived from the elliptic curve properties. Moreover, Tsuruoka and Koyama used optimal addition sequences for this scenario [9]. Their main target is GF(p^m) with midium sized p ($p \leqq 128$).

This paper improves the sliding window algorithm to accelerate elliptic curve scalar multiplication over OEF with a large sized p (e.g. $\log_2 p \approx 16, 32$ or 64). The main idea of this paper is to employ a "cycle" window rather than a "slide" window.

2. Preliminaries

2.1 Elliptic Curve

An elliptic curve cryptosystem comprises group arithmetics defined in an elliptic curve over a finite field. The number of field operations follows the representation of a point on an elliptic curve. We can roughly classify the representation into two groups: one is the projective system which does not require field inversion, and the other is the affine system which requires field inversions. For more details, Cohen et al. [10] presents a discussion on the advantages and disadvantages of these systems.

Below is an example of Jacobian coordinates, one of the projective systems. Using a and b as parameters, a non-supersingular elliptic curve with a characteristic greater than 3 is defined as

$$Y^2 = X^3 + aXZ^4 + bZ^6 \quad (4a^3 + 27b^2 \neq 0). \qquad (1)$$

We can define the group operations for the rational points on the curve.

2.2 Previous Studies on OEF

This section provides a brief overview of the history of the implementations of scalar multiplication on an elliptic curve over OEF.

Manuscript received March 23, 2002.
Manuscript revised July 4, 2002.
Final manuscript received September 20, 2002.

[†]The authors are with NTT Information Sharing Platform Laboratories, NTT Corporation, Yokosuka-shi, 239-0847 Japan.
a) E-mail: kotetsu@isl.ntt.co.jp

Bailey and Paar proposed OEFs [2]. An OEF can be represented as $\mathrm{GF}(p)[x]/(f(x))$, where $p = 2^n - c$ ($\log_2 c \leqq \frac{1}{2}n$) and $f(x) = x^m - \omega$. (Note that n is the size of p and m is the extension degree.) An OEF is very well suited to software implementation because the conditions,

$$2^n \equiv c \pmod{p}$$
$$x^m \equiv \omega \pmod{f(x)},$$

can be used to reduce the complexity of field multiplications. They showed that their implementations are significantly faster than the implementations previously reported.

Kobayashi et al. applied the base-ϕ expansion technique [3] to scalar multiplication in $E(\mathrm{GF}(p^m))/\mathrm{GF}(p)$ [4]. They use the pth-power Frobenius map,

$$\phi : E \to E$$
$$(x, y) \mapsto (x^p, y^p).$$

Similar to an elliptic curve over $\mathrm{GF}(2^m)$, (hereafter a binary field), the computational cost of Frobenius map ϕ, is very low. This condition made their implementations about twice as fast as that reported by Bailey and Paar.

Lim and Hwang presented implementations that do not use any epoch-making techniques; however, their accumulation of minor techniques increased the speed of their implementation by approximately two-fold [7], [8].

2.3 Window Algorithms

The 2^w-ary algorithm [1, 14.83 Algorithm in pp.615–616] is an algorithm that computes kP using online precomputation. A brief description of the algorithm is given below.

Step 1: Compute $Q_i \leftarrow iP$ for $(2 \leqq i < 2^w)$, and let $Q_0 = \mathcal{O}$ and $Q_1 = P$.

Step 2: Using c_i that satisfies

$$k = \sum_{i=0}^{\lfloor (\log_2 k)/w \rfloor} 2^{wi} c_i \quad (0 \leqq c_i < 2^w),$$

compute

$$kP \leftarrow \sum_{i=0}^{\lfloor (\log_2 k)/w \rfloor} 2^{wi} Q_{c_i}.$$

It accelerates scalar multiplication in comparison to the binary algorithm [1, 14.76 and 14.79 Algorithms in pp.614–615].

Moreover, the sliding window algorithm [1, 14.85 Algorithm in p.616] was proposed to improve the 2^w-ary algorithm. By using c_i and w_i satisfying $k = \sum_i 2^{w_i} c_i$, compute $kP \leftarrow \sum_i 2^{w_i} Q_{c_i}$, where $w_i \geqq w_{i-1} + w$.

This paper applies this approach to the base-ϕ expansion algorithm, and proposes an improved version of the sliding window algorithm called the *cyclic window algorithm* to compute $\sum_{i=0}^{m-1} c_i \phi^i P$.

3. Cyclic Window Algorithm

This section proposes the *cyclic window algorithm* and provides an analysis of the algorithm. The cyclic window algorithm computes scalar multiplication kP.

3.1 Notations

This section defines the notations used in the following sections that have not appeared in the previous sections.

Notation 1: Let $\overline{x}$ be the complementation of all elements in binary vector x. That is

$$\overline{[x_0, x_1, \ldots, x_{m-1}]} := [\overline{x_0}, \overline{x_1}, \ldots, \overline{x_{m-1}}]. \tag{2}$$

Notation 2: ϕ mapping for vector $[x_0, x_1, \ldots, x_{m-1}]$ is defined as

$$\phi[x_0, x_1, \ldots, x_{m-2}, x_{m-1}] := [x_{m-1}, x_0, x_1, \ldots, x_{m-2}].$$

Notation 3: "$x \sqsupseteq y$" is true if binary vector x includes binary vector y. More precisely,

$$[x_0, x_1, \ldots, x_{m-1}] \sqsupseteq [y_0, y_1, \ldots, y_{m-1}] := \begin{cases} \mathsf{true} & \text{if } x_j \vee \overline{y_j} = 1 \text{ for } \forall j \ (0 \leqq j < m) \\ \mathsf{false} & \text{otherwise} \end{cases}$$

Notation 4: $w_{\mathrm{H}}(x)$: Hamming weight of x, i.e., number of non-zero elements in vector x.

Notation 5: $v_i := [1, e_{i,0}, e_{i,1}, \ldots, e_{i,m-2}]$, where $\{e_{i,j}\}_{j=0}^{m-2}$ is the binary representation of i $(0 \leqq i < 2^{m-1})$, i.e., $e_{i,j}$ satisfies the following equation.

$$i = \sum_{j=0}^{m-2} e_{i,j} 2^j, \quad \text{where } e_{i,j} \in \{0, 1\}. \tag{3}$$

For example, we have $v_1 = [1, 1, 0, 0, 0, 0, 0]$ and $v_{13} = [1, 1, 0, 1, 1, 0, 0]$ for $m = 7$.

Notation 6:

$$([a_1, a_2, \ldots, a_n], [b_1, b_2, \ldots, b_n]) := a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

3.2 Our Approach

This section describes the key idea of the cyclic window algorithm using an example.

We assume that k is already expanded to base-ϕ representation, for example,

$$k = 4 + 11\phi + 7\phi^2 + 2\phi^3 + 3\phi^4 + 1\phi^5 + 0\phi^6,$$

and $m = 7$. The expansion can be done using Fig. 1 [4, Step 1 in Base-ϕ Scalar Multiplication Procedure]†, for example.

Let n' $(= \log_2 \max_{0 \leqq i < m} c_i)$ be the size of the coefficients of the expansion.

We use the online precomputation table as follows.

$$Q_1(v_1, [1, \phi, \ldots, \phi^{m-1}])P(1+\phi)P$$
$$Q_2(v_2, [1, \phi, \ldots, \phi^{m-1}])P(1+\phi^2)P$$
$$Q_3(v_3, [1, \phi, \ldots, \phi^{m-1}])P(1+\phi+\phi^2)P$$

Let L be the number of points in the table. In this case, $L = 3$.

First, we denote the base-ϕ representation as binary representations (see Fig. 2). Second, we reduce 1s using signed-binary forms applying the condition $2^j(1 + \phi + \cdots + \phi^{m-1})P = \mathcal{O}$ [4] (see Fig. 3).

Since we can compute $\phi^i Q_l$ from precomputed point Q_l with a low level of complexity, we can use not only Q_l but also $\phi^i Q_l$ $(0 \leqq i < m)$ referring to a precomputed point without much complexity. Figure 3 shows that $-\phi^6 Q_1$ can be used in the least significant bits of the coefficients of ϕ_is. If we use the basic sliding window algorithm, the scalar multiplication for the least significant bits is computed by

$$-P - \phi^3 P - \phi^6 P.$$

However we can "wrap" precomputed point Q_1 from ϕ^6 to ϕ^0, since $\phi^7 P = P$ holds. Thus, we can cyclically use the precomputed points as

$$-\phi^3 P - \phi^6 Q_1.$$

Following the above observations, we can compute kP using Eq. (4) with precomputed points Q_1, Q_2, and Q_3 similar to the left-to-right binary algorithm for multiple bases [1, Algorithm 14.88 in p.618]. It requires only four elliptic curve additions†† and three elliptic curve doublings after three elliptic curve additions in the online precomputation stage.

$$kP = 2(2(2(\phi P) + Q_2) - \phi^5 Q_3) - \phi^3 P - \phi^6 Q_1 \quad (4)$$

Input of Step 1: k, t, p (t, p satisfy $\phi^2 - t\phi + p = 0$)
Output or Step 1: $\{c_1, \ldots, c_{m-1}\}$
Step 1: **Base-ϕ Expansion of** k

- **Step 1-1:** $i \leftarrow 0$, $x \leftarrow k$, $y \leftarrow 0$, $c_j \leftarrow 0$ for $\forall j$.
- **Step 1-2:** if ($x = 0$ and $y = 0$) then go to **Step 2**.
- **Step 1-3:** $c_i \leftarrow c_i + (x \bmod p)$.
- **Step 1-4:** $v \leftarrow \lfloor \frac{x}{p} \rfloor$, $x \leftarrow tv + y$, $y \leftarrow -v$, $i \leftarrow i + 1 \bmod m$.
- **Step 1-5:** go to **Step 1-2**.

Fig. 1 Example of base-ϕ expansion procedure.

Coefficient		Binary Representation			
4	ϕ^0	0	1	0	0
11	ϕ^1	1	0	1	1
7	ϕ^2	0	1	1	1
2	ϕ^3	0	0	1	0
3	ϕ^4	0	0	1	1
1	ϕ^5	0	0	0	1
0	ϕ^6	0	0	0	0

Fig. 2 Binary representation of coefficients in base-ϕ expansion.

3.3 Procedure

Using the notations defined above, we propose the cyclic window algorithm in Fig. 4†††. The algorithm computes scalar multiplication kP, but we assume that scalar k is already expanded to base-ϕ representation using Fig. 1 to hold

$$k = \sum_{i=0}^{m-1} c_i \phi^i \quad (c_i \geqq 0).$$

By using [8, the explanation for (14)], we assume

†Notations are changed in this paper.

††Since an elliptic curve subtraction has the same complexity as an elliptic curve addition, we count the number of additions including subtractions.

†††We usually omit the first elliptic curve doubling and addition in the main computation stage, but this description is not included this in the figure, for the sake of simplicity of the description.

Coefficient		Binary Representation					Precomputation		
4	ϕ^0	0	1	-1	-1		1	1	1
11	ϕ^1	1	0	0	0		1	0	1
7	ϕ^2	0	1	0	0			1	1
2	ϕ^3	0	0	0	-1	$\cdots - \phi^3 P$	Q_1	Q_2	Q_3
3	ϕ^4	0	0	0	0				
1	ϕ^5	0	0	-1	0				
0	ϕ^6	0	0	-1	-1	$\cdots - \phi^6 Q_1$			

Fig. 3 How to use the precomputation table.

Input: c_i $(0 \leqq c_i < 2^{n'})$, P, m, $\{u_0, u_1, \ldots, u_L\}$ $(\subseteqq \{v_0, v_1, \ldots v_{2^{m-1}-1}\})$

Output: $\sum_{i=0}^{m-1} c_i \phi^i P$

Step 1: **[Coefficient adjustment stage]**
For all j, let $c_{i,j}$ be the j-th bit of c_i and d_j be a vector which is a collection of $c_{i,j}$s, i.e.,

$$c_i = \sum_{j=0}^{n'-1} c_{i,j} 2^j, \text{ where } c_{i,j} \in \{0, 1\}$$

and
$d_j \leftarrow [c_{0,j}, c_{1,j}, \ldots, c_{m-1,j}]$.
If $w_{\mathrm{H}}(d_j) \geqq (m+1)/2$ then
  $s_j \leftarrow -1$ and $d_j \leftarrow \overline{d_j}$,
else
  $s_j \leftarrow 1$.

Step 2: **[Online precomputation stage]**
Compute $Q_l \leftarrow (u_l, [P, \phi P, \phi^2 P, \ldots, \phi^{m-1} P])$, for $0 \leqq l \leqq L$.

Step 3: **[Main computation stage]**
$R \leftarrow \mathcal{O}$
for $j = n' - 1$ downto 0 do
  $R \leftarrow 2R$
  for $l \leftarrow L$ downto 0 do
    for $i \leftarrow 0$ to $m - 1$ do
      if $d_j \sqsupseteq \phi^i u_l$ then
        $d_j \leftarrow d_j \oplus \phi^i u_l$
        $R \leftarrow R + s_j \phi^i Q_l$
      end if
    end for i
  end for l
end for j
Output R.

Fig. 4 Cyclic window algorithm procedure.

$\log_2 c_i \leqq n' \leqq n$, where n is the size of p and n' is the maximum size of coefficient c_i. The algorithm inputs the structure of the online precomputation table, $\{u_0, u_1, \ldots, u_L\}$. The structure is defined in Sect. 4.1.

Note that the algorithm is a generalization of Solinas' algorithm [5].

4. Evaluation of Cyclic Window Algorithm

In this section, we evaluate the performance of the cyclic window algorithm applied to OEF for practically used parameters: the size of the maximum prime order subgroup is at least 160 bits and at most 256 bits, and the extension degree, m, is prime. Since usual computer word size w is 16, 32 or 64, we assume $3 \leqq m \leqq 19$. $(160 \leqq mw \leqq 256)$. Moreover, we assume $c_{i,j}$ is uniformly distributed.

4.1 Table Structure

It is difficult to decide the best structure of the online precomputation table. In this section, we propose a deterministic table construction method. It is not always the best structure, but there are only slight differences between other construction method, in our heuristic experiment for our targeted parameters.

Table 1 Maximum number of table elements.

Extension degree m	3	5	7	11	13	17	19
Maximum # of elements, $L_{\max}$	0	2	8	92	314	3854	13796

Let $S = \{u_0, u_1, \ldots, u_L\}$ be the subset of $\{v_0, v_1, \ldots, v_{2^{m-1}-1}\}$ that satisfies the following conditions:

- $u_0 = v_0$
- $\forall u_l [S \cap \{\phi^1(u_l), \ldots, \phi^{m-1}(u_l), \phi^0(\overline{u_l}), \phi^1(\overline{u_l}), \ldots, \phi^{m-1}(\overline{u_l})\} = \emptyset]$
- $w_{\mathrm{H}}(u_l) \leqq w_{\mathrm{H}}(u_{l+1})$
- $\sigma(l) < \sigma(l+1)$ if $w_{\mathrm{H}}(u_l) = w_{\mathrm{H}}(u_{l+1})$, where $\sigma(\cdot)$ is a permutation of the set $\{0, 1, \ldots, 2^{m-1} - 1\}$ and satisfies $u_l = v_{\sigma(l)}$.
- $\sigma(L)$ is as small as possible.

4.2 Number of Table Elements

In this section, we evaluate the bounds of the online precomputation table.

Since the extension degree is m and the table does not need to contain $\mathcal{O}$ and P, the number of table elements is at most $2^m - 2$. However, we need only one point in

$$\Big\{ uP \;\Big|\; u \in \{\phi^0(v_l), \phi^1(v_l), \ldots, \phi^{m-1}(v_l), \phi^0(\overline{v_l}), \phi^1(\overline{v_l}), \ldots, \phi^{m-1}(\overline{v_l})\}\Big\}, \tag{5}$$

because the other points can be easily computed in the main computation stage. Since m is an odd prime, the size of set (5) is exactly $2m$. Thus, it is sufficient to choose

$$L = \frac{2^m - 2}{2m} - 1 \tag{6}$$

at most (-1 in Eq. (6) means that we do not need precompute a point corresponding to u_0).

Table 1 gives the values derived from Eq. (6).

4.3 Optimal Number of Table Elements

In this paper, "optimal number of table elements," L_{opt}, represents the number of table elements that minimizes the average number of elliptic curve additions required by online precomputation and main computation stage. The number of elliptic curve doublings has no relation to the number of table elements. The cost of the Frobenius map is low enough to ignore [4] (Table 4 shows the cost of the Frobenius map is 5% or less than that of elliptic curve addition).

Term $T_{\mathrm{pre}}(L)$ denotes the number of elliptic curve additions required by online precomputation stage. Term $T_{\mathrm{main}}(L)$ denotes the number of elliptic curve additions required by main computation stage. Term

$T(L)$ is the total:

$$T(L) = T_{\text{pre}}(L) + T_{\text{main}}(L)$$

We note here that L_{opt} *is optimal* if and only if $T(L_{\text{opt}}) \leqq T(L)$ for any $0 \leqq L \leqq L_{\max}$.

4.4 Small Extension Degree

In this subsection, we study the case of a small m, that is, $m \leqq 7$. In this case, the maximum number of table elements, $L_{\max}$, is not high according to Table 1 and $L_{\max}$ is optimal for $m = 5$ and 7 if $n' \geqq 10$. We prove this in the rest of the subsection.

The proof of "$L_{\max} = 2$ is optimal for $m = 5$ if $n' \geqq 4$" is as follows.

$$\begin{aligned}
T(0) &= T_{\text{main}}(0) \\
&= 2\left(1 \cdot \frac{5}{2^5} + 2 \cdot \frac{10}{2^5}\right) n' \\
&= \frac{25}{16} n' \\
T(1) &= T_{\text{pre}}(1) + T_{\text{main}}(1) \\
&= 1 + 2\left(1 \cdot \frac{10}{2^5} + 2 \cdot \frac{5}{2^5}\right) n' \\
&= 1 + \frac{5}{4} n' \\
T(2) &= T_{\text{pre}}(2) + T_{\text{main}}(2) \\
&= 2 + 2\left(1 \cdot \frac{15}{2^5}\right) n' \\
&= 2 + \frac{15}{16} n'
\end{aligned}$$

Thus, $T(2) \leqq T(1)$ and $T(2) \leqq T(0)$ for $n' \geqq 4$.

Also, we can easily see that $L = 8$ is optimal for $m = 7$, $n' \geqq 10$.

4.5 Large Extension Degree

In this subsection, we consider a large m value, for example $m \geqq 11$. As we consider the previous sections, to maximize L is not effective because $L_{\max}$ is very large. We discuss the number of table elements, L, that minimizes $T(L)$.

$$\begin{aligned}
&T_{\text{pre}}(L) = L \\
&T_{\text{main}}(L) = (n' - 1)A(L, m) \\
&T(L) = T_{\text{pre}}(L) + T_{\text{main}}(L) \qquad (7)
\end{aligned}$$

$A(L, m)$ denotes the average number of matches for each bit in the coefficients of the base-ϕ extension. It shows the required average number of elliptic curve additions in the inner most loop of Step 3. Thus, we need $(n' - 1)$ elliptic curve doublings and $(n' - 1)A(L, m)$ elliptic curve additions in Step 3. We show $A(L, m)$ in Table 2. The table is constructed by exhaustive search using a computer. For example, we can see $L = 6$ is optimal for $(m, n') = (11, 16)$ and $L = 7$ is optimal for $(m, n') = (13, 14)$ by the table and Eq. (7).

Table 2 Average matches $A(L, m)$ for each bit position in coefficients of the base-ϕ expansion.

	m			
L	11	13	17	19
0	4.15	5.03	6.83	7.74
1	3.09	3.70	4.94	5.57
2	2.67	3.20	4.30	4.85
3	2.45	2.93	3.89	4.37
4	2.31	2.78	3.69	4.16
5	2.17	2.64	3.51	3.95
6	2.08	2.53	3.36	3.79
7	2.02	2.45	3.23	3.63
8	1.97	2.39	3.17	3.57
9	1.96	2.33	3.12	3.51
10	1.93	2.26	3.04	3.42
11	1.90	2.20	2.97	3.33
12	1.90	2.16	2.94	3.28
13	1.87	2.11	2.88	3.22
14	1.84	2.07	2.83	3.16
15	1.83	2.06	2.80	3.11
16	1.82	2.03	2.79	3.09
17	1.80	2.05	2.78	3.09
18	1.79	2.04	2.75	3.07
19	1.78	2.03	2.72	3.05
20	1.77	2.01	2.69	3.01
21	1.76	1.98	2.66	2.98
22	1.75	1.97	2.63	2.95
23	1.74	1.96	2.60	2.92
⋮	⋮	⋮	⋮	⋮
92	1.00	⋮	⋮	⋮
314	—	1.00	⋮	⋮
3854	—	—	1.00	⋮
13794	—	—	—	1.00

Table 2 shows that the table construction described in Sect. 4.1 is not the best. There are some numbers greater than one that use less of the precomputation table. However, the tendency of the values in Table 2 shows that the greater the number of table elements, the fewer the matches.

4.6 Comparison with the Best Previous Result

To the best of the authors' knowledge, Lim and Hwang's results [7], [8] were the fastest previously reported. Their computational complexity is evaluated in [8, Sect. 5]. They tried to evaluate their algorithm for general cases, but precise numbers were not clear, since their evaluation employed variables. However, they could derive precise numbers when m and n' were fixed. They showed numbers for $(m, n') = (7, 28)$, $(11, 16)$, and $(13, 14)$. We derive the optimal number of table elements for the above parameters in order to draw a comparison to their results. However, we only compare the number of elliptic curve additions, since the cyclic window algorithm and the Lim and Hwang algorithm

Table 3 Comparison with Lim and Hwang's results.

	Online precomputation ($T_{\mathrm{pre}}(L)$Ae)		Main computation ($T_{\mathrm{main}}(L)$Ae + $(n-1)$De)		Total ($T(L)$Ae + $(n-1)$De)	
(m,n)	Ours	[8]	Ours	[8]	Ours	[8]
$(7,28)$	8Ae	15Ae	34.6Ae+27De	41.2Ae+27De	42.6Ae+27De	56.2Ae+27De
$(11,16)$	6Ae	13Ae	37.2Ae+15De	41.7Ae+15De	43.2Ae+15De	54.7Ae+15De
$(13,14)$	7Ae	14Ae	38.9Ae+13De	44.1Ae+13De	45.9Ae+13De	58.1Ae+13De

"Ae" and "De" denote the computational cost of an elliptic curve addition and an elliptic curve doubling, respectively.

require the same number of elliptic curve doublings.

When m equals 7, $L = 8$ is the optimal choice according to Sect. 4.4[†]. Thus, the scalar multiplication requires

$$\begin{aligned} T(L) &= L + T_{\mathrm{main}}(L) \\ &= 8 + (28-1)\left(1 - \frac{2}{2^7}\right) = 34.6 \end{aligned}$$

elliptic curve additions on average.

When m equals 11 or 13, we need to find the smallest

$$T(L) = L + (n' - 1)A(L, m) \tag{8}$$

using Table 2. We tried all combinations of Eq. (8). We found that $L = 6$ for $m = 11$ and $L = 7$ for $m = 13$ are the optimal choices[††]. These cases require

$$T(L) = 6 + (16-1)2.08 = 37.2$$

and

$$T(L) = 7 + (14-1)2.45 = 38.9$$

elliptic curve additions, respectively. We summarize the above numbers and compare them to the results by Lim and Hwang in Table 3.

5. Implementation Examples

5.1 Parameters

We use the following parameters which are constructed [4] by lifting elliptic curves over GF(p) and the field parameters are shown in the upper columns in Table 4. Note that α is a root of $f(x)$ used by GF(p)$[x]/(f(x))$, and we choose the base as a generator of the maximum prime order subgroup.

5.1.1 Word Length: 16 Bits

$$y^2 = x^3 - 3x - 172$$

(1) Maximum prime order in subgroups

3 7735447064 0784663733 8580162818 7749646114 9221530761 (exceeding 168 bits)

(2) Base point

$(10869\alpha^{12} + 3898\alpha^{11} + 15358\alpha^{10} + 3782\alpha^9 + 4242\alpha^8 + 7589\alpha^7 + 5310\alpha^6 + 12599\alpha^5 + 10370\alpha^4 + 9316\alpha^3 + 8340\alpha^2 + 184\alpha + 9573, 8924\alpha^{12} + 9141\alpha^{11} + 9472\alpha^{10} + 8964\alpha^9 + 14633\alpha^8 + 4204\alpha^7 + 5379\alpha^6 + 13644\alpha^5 + 11470\alpha^4 + 15042\alpha^3 + 6518\alpha^2 + 15906\alpha + 7391)$

5.1.2 Word Length: 32 Bits

$$y^2 = x^3 - 3x - 85$$

(1) Maximum prime order in subgroups

239 4696831448 0862150279 8948628438 5174133848 4034750169 (exceeding 174 bits)

(2) Base point

$(200472906\alpha^6 + 172723217\alpha^5 + 174386879\alpha^4 + 403718784\alpha^3 + 23043362\alpha^2 + 525400877\alpha + 17252111, 523133120\alpha^6 + 178522781\alpha^5 + 357710308\alpha^4 + 10611891\alpha^3 + 423928020\alpha^2 + 2135201\alpha + 535095305)$

5.2 Timings

Based on the above discussion, we implemented scalar multiplication in the elliptic curves. The timing is summarized in Table 4. In the table, "½" means that we adopted the parallel multiplication technique described in [11]. For example, we can compute two OEF multiplications in 604 cycles on a Pentium II. Table 4 also shows Lim and Hwang's results [7], [8] as a reference. We refer to the detailed timings shown in [7]. However, we refer to the timings of the scalar multiplication shown in [8], since the timings of a scalar multiplication shown in [8] are faster than those in [7]. The results are scaled to 450 MHz for a Pentium II and 500 MHz for 21164.

Our implementations on the Pentium II, 21164A, and 21264 use the Jacobian coordinate, the affine coordinate, and the coordinate proposed in [7, Sect. 2.2], respectively. We selected a 160-bit random integer as a scalar. Even if we select a number close to the order of the subgroup generated by the base point as a

[†]This case requires 448 bytes to store the online precomputation table, when using the parameter shown in Table 4.

[††]These cases require 264 and 364 bytes to store the online precomputation table, respectively, when using the parameters shown in Table 4.

Table 4 Elliptic curve scalar multiplication (Cycles).

	Current study			[7]		
CPU	Pentium II	21164A	21264	Pentium II		21164
p	$2^{14}-3$	$2^{29}-3$	$2^{29}-3$	$2^{14}-3$	$2^{28}-57$	$2^{28}-57$
$f(x)$	$x^{13}-2$	$x^{7}-2$	$x^{7}-2$	$x^{13}-2$	$x^{7}-2$	$x^{7}-2$
Subgroup order (bits)	168	174	174	168	168	168
Scalar mult (μseconds)	573	595	254	791	687	672
Scalar mult (10^3)	258	298	127	356	309	336
EC add (A)	NA	3412	1544	6091	4628	4866
EC dbl (A)	NA	3830	1621	6863	5107	5543
EC add (P)	(M)3692	4152	1524	6171	4256	4696
EC dbl (P)	2528	3128	1164	4442	3086	3518
OEF inv	1/24824	2120	1010	4200	3259	3292
OEF mult	1/2604	323	117	543	379	383
OEF sqr	1/2525	309	99	404	301	359
OEF ϕ	111	116	70			
OEF add	26	58	28	91	42	59
OEF sub	21	58	28			
GF(p) inv	1	266	219	19	457	376

- (A): Affine, (P): Projective
- (M): The addend is represented by affine coordinate
- NA: Not Available
- Pentium II (450 MHz), Other CPU (500 MHz)

scalar, the time for the main computation stage hardly increases, but the time for converting the scalar to the base-ϕ representation will slightly increase.

Note that $a = -3$ is always used in Eq. (1) for fast implementation purposes and $n' \leqq n+2$ because the time to reduce $n'(\leqq n)$ is not negligible in a scalar multiplication.

6. Conclusion

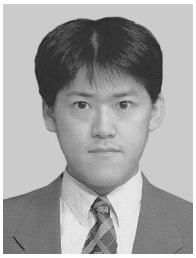
This paper presented the cyclic window algorithm, a new scalar multiplication algorithm for elliptic curves defined over OEF. The algorithm first makes an online precomputation table and then computes a scalar multiplication using the precomputation table with the Frobenius map. The condition of the Frobenius map, $\phi^m = 1$, allows us to use the precomputation table cyclically. This algorithm makes scalar multiplications about 15% faster than the best results reported to date [7], [8]. We also implemented our algorithm by software. A scalar multiplication can be computed in 573 μs, 595 μs, and 254 μs on Pentium II (450 MHz), 21164A (500 MHz), and 21264 (500 MHz) computers, respectively.

Finally, how to decide the best structure of the online precomputation table is left for future study.

References

[1] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, Handbook of applied cryptography, CRC Press, 1997.
[2] D.V. Bailey and C. Paar, "Optimal extension fields for fast arithmetic in public-key algorithms," Advances in Cryptology—CRYPTO'98, ed. H. Krawczyk, Lecture Notes in Computer Science, vol.1462, pp.472–485, Springer-Verlag, Berlin, Heidelberg, New York, 1998.
[3] N. Koblitz, "CM-curves with good cryptographic properties," Advances in Cryptology—CRYPTO'91, ed. J. Feigenbaum, Lecture Notes in Computer Science, vol.576, pp.279–287, Springer-Verlag, Berlin, Heidelberg, New York, 1992.
[4] T. Kobayashi, H. Morita, K. Kobayashi, and F. Hoshino, "Fast elliptic curve algorithm combining frobenius map and table reference to adapt to higher characteristic," Advances in Cryptology—EUROCRYPT'99, ed. J. Stern, Lecture Notes in Computer Science, vol.1592, pp.176–189, Springer-Verlag, Berlin, Heidelberg, New York, 1999. (A preliminary version was written in Japanese and presented at SCIS'99-W4-1.4.)
[5] J.A. Solinas, "An improved algorithm for arithmetic on a family of elliptic curves," Advances in Cryptology—CRYPTO'97, ed. B.S. Kaliski, Jr., Lecture Notes in Computer Science, vol.1294, pp.357–371, Springer-Verlag, Berlin, Heidelberg, New York, 1997.
[6] C.H. Lim and P.J. Lee, "More flexible exponentiation with precomputation," Advances in Cryptology—CRYPTO'94, ed. Y.G. Desmedt, Lecture Notes in Computer Science, vol.839, pp.95–107, Springer-Verlag, Berlin, Heidelberg, New York, 1994.
[7] C.H. Lim and H.S. Hwang, "Fast implementation of elliptic curve arithmetic in GF(p^n)," Public Key Cryptography—Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000, ed. H. Imai and Y. Zheng, Lecture Notes in Computer Science, vol.1751, pp.405–421, Springer-Verlag, Berlin, Heidelberg, New York, 2000.
[8] C.H. Lim and H.S. Hwang, "Speeding up elliptic scalar multiplication with precomputation," Information Security and Cryptology—ICISC'99, ed. J.S. Song, Lecture Notes in Computer Science, vol.1787, pp.102–119, Springer-Verlag, Berlin, Heidelberg, New York, 2000.
[9] Y. Tsuruoka and K. Koyama, "Fast computation over elliptic curves $E(\mathbf{F}_{q^n})$ based on optimal addition sequences," IEICE Trans. Fundamentals, vol.E84-A, no.1, pp.114–119, Jan. 2001.
[10] H. Cohen, A. Miyaji, and T. Ono, "Efficient elliptic curve exponentiation using mixed coordinates," Advances in Cryptology—ASIACRYPT'98, ed. K. Ohta and D. Pei, Lecture Notes in Computer Science, vol.1514, pp.51–65, Springer-Verlag, Berlin, Heidelberg, New York, 1998.

[11] K. Aoki, F. Hoshino, T. Kobayashi, and H. Oguro, "Elliptic curve arithmetic using SIMD," Information Security Conference—ISC'01, Lecture Notes in Computer Science, vol.2200, pp.235–247, Springer-Verlag, Berlin, Heidelberg, New York, 2001.



Tetsutaro Kobayashi received his B.Eng. and M.Eng. degrees from Tokyo Institute of Technology, Tokyo, Japan, in 1993 and 1995, respectively. He is a Researcher in the NTT Information Sharing Platform Laboratories. He is presently engaged in research on Information Security. He is interests lie in elliptic curve cryptosystems. He was awarded the SCIS'00 paper prize.

Fumitaka Hoshino received his B.Eng. and M.Eng. degrees from Tokyo University, Tokyo, Japan, in 1996 and 1998, respectively. He is a Researcher in the NTT Information Sharing Platform Laboratories.

Kazumaro Aoki received his B.S., and M.S., and D.S. degrees from Waseda University, Tokyo, Japan, in 1993, 1995, and 2001 respectively. He is a Researcher of NTT Information Sharing Platform Laboratories, NTT Corporation since 1995. He was awarded the SCIS'95 and SCIS'96 paper prizes, and 1997 IEICE Young Engineer Award.