Efficient Exponentiation in Extensions of Finite Fields without Fast Frobenius Mappings

Yasuyuki Nogami, Hidehiro Kato, Kenta Nekado, and Yoshitaka Morikawa

This paper proposes an exponentiation method with Frobenius mappings. The main target is an exponentiation in an extension field. This idea can be applied for scalar multiplication of a rational point of an elliptic curve defined over an extension field. The proposed method is closely related to so-called interleaving exponentiation. Unlike interleaving exponentiation methods, it can carry out several exponentiations of the same base at once. This happens in some pairing-based applications. The efficiency of using Frobenius mappings for exponentiation in an extension field was well demonstrated by Avanzi and Mihailescu. Their exponentiation method efficiently decreases the number of multiplications by inversely using many Frobenius mappings. Compared to their method, although the number of multiplications needed for the proposed method increases about 20%, the number of Frobenius mappings becomes small. The proposed method is efficient for cases in which Frobenius mapping cannot be carried out quickly.

Keywords: Exponentiation, Frobenius mapping, extension field, prime field, modular polynomial, window method.

Manuscript received Mar. 21, 2008; revised Sept. 10, 2008; accepted Oct. 21, 2008.

Yasuyuki Nogami (email: nogami@trans.cne.okayama-u.ac.jp), Hidehiro Kato (email: kato@trans.cne.okayama-u.ac.jp), Kenta Nekado (email: nekado@trans.cne.okayama-u.ac.jp), and Yoshitaka Morikawa (email: morikawa@cne.okayama-u.ac.jp) are with the Graduate School of Natural Science and Technology, Okayama University, Okayama, Japan.

I. Introduction

Recently, pairing-based cryptographic applications such as ID-based cryptography [1], group signature authentication [2], and broadcast encryption [3] have received much attention. Pairings such as the Weil, Tate, Eta, and Ate pairings are bilinear mappings from two subgroups in a certain elliptic curve to a subgroup in a certain extension field [4]-[6]. A pairing calculation needs an exponentiation in an extension field. The computation time of the exponentiation is about half of the total computation time of a pairing calculation. In addition, those pairing-based cryptographic applications need many exponentiations in an extension field. Such exponentiations in an extension field are the subject of this paper. This paper proposes a new exponentiation method with Frobenius mappings.

The most widely used binary method calculates an exponentiation a^n by using the binary representation of exponent n. It iterates squaring and multiplying. The binary method needs $\lfloor \log_2 n \rfloor$ squares and $\lfloor \log_2 n \rfloor / 2$ multiplications on average. The window method [7] calculates exponentiations more efficiently; however, it still needs $\lfloor \log_2 n \rfloor - w$ squares, where w is the window size. Moreover, it is not very efficient when the base a changes often. The proposed exponentiation method is closely related to the so-called interleaving exponentiation method [8]. In contrast to interleaving exponentiation methods, the proposed method can compute several exponentiations of the same base at once. For example, when $A \in F_{p^m}$ and x, y, and z are large positive integers, our exponentiation method can compute A^x, A^y, and A^z at once. Note that the proposed method is efficiently applied when the extension degree m is equal to 1. The use of Frobenius mappings for efficient exponentiation in an extension field was

well demonstrated by Avanzi and Mihailescu [9]. When the exponent n is sufficiently larger than the characteristic p and the Frobenius mapping is carried out fast in the concerned extension field, using Frobenius mappings accelerates exponentiations because the number of squares required for an exponentiation decreases to $\lfloor \log_2(p-1) \rfloor$. On the other hand, $\lfloor \log_2 n \rfloor / 2$ multiplications on average are still needed. To decrease the number of multiplications, the window method can be applied in addition to Frobenius mappings. Based on this idea, Avanzi and Mihailescu [9] proposed a fast exponentiation method called the Frobenius abusing exponentiation method. This method decreases the number of multiplications; however, the number of Frobenius mappings inversely increases. Thus, it still has two problems. First, it is not very efficient when the base often changes, and secondly, it needs fast Frobenius mapping in the concerned extension field.

It is well-known that an optimal extension field (OEF) whose modular polynomial is an irreducible binomial has a fast Frobenius mapping [10]. In order to construct the degree-m extension field F_{p^m} over F_p as an OEF, each prime factor of m needs to divide p–1. This sometimes becomes a critical condition. For example, consider Freeman's curve [11] whose embedding degree is 10. The characteristic p of the underlying finite field must satisfy

$$p = 25\chi^4 + 25\chi^3 + 25\chi^2 + 10\chi + 3 \tag{1}$$

for a certain integer χ. From this condition, it is found that the field of definition of the curve $F_{p^{10}}$ cannot be prepared as an OEF. In detail, it is because p–1 and p+1 are not divisible by 5. In this case, one would adopt an irreducible trinomial as its modular polynomial, so a Frobenius mapping in $F_{p^{10}}$ will need almost the same calculation cost as a multiplication in $F_{p^{10}}$. This is an example of a curve whose field of definition does not have fast Frobenius mapping.

To overcome these problems, this paper proposes an exponentiation method for extension fields that efficiently uses Frobenius mappings but is not based on the window method. Compared to Avanzi's method [9], the number of multiplications needed for the proposed method increases by about 20%; however, the number of Frobenius mappings is reduced. Evaluating the calculation costs, it is shown that the proposed exponentiation method is sufficiently practical for cases in which the base is often changed and a Frobenius mapping cannot be carried out quickly in the concerned extension field. In this paper, we deal with the example of an extension field F_{p^m} over prime field F_p; however, the proposed method can easily be extended for an extension field F_{q^m} over some extension field F_q, $q = p^i$. The proposed method is also applied for a scalar multiplication of a rational point of an elliptic curve defined over an extension field. The basic idea is efficiently applied in solving the plurality of general exponentiations of the same base at once. Moreover, the proposed method is efficient for fixed base scalar multiplication of elliptic curve cryptography.

Throughout this paper, p and m denote the characteristic and extension degree, respectively, where p is a prime number, and $F^*_{p^m}$ denotes the multiplicative group in F_{p^m}. Unless otherwise noted, lower and upper case letters show elements in the prime field and extension field, respectively. Greek characters denote zeroes of modular polynomials.

II. Preparation

In this section, let us briefly go over bases of extension fields, Frobenius mappings, the binary method, the window method, and previous work.

1. Bases of Extension Fields

To construct arithmetic operations in F_{p^m}, we usually need an irreducible polynomial $f(x)$ of degree m over F_p. Let ω be a zero of $f(x)$ in F_p^m, then the following set forms a basis of F_{p^m} over F_p:

$$\{1, \omega, \omega^2, \cdots, \omega^{m-1}\}, \tag{2}$$

which is called a *polynomial basis*. An arbitrary element A in F_{p^m} is written as

$$A = a_0 + a_1\omega + \cdots + a_{m-1}\omega^{m-1}. \tag{3}$$

Its vector representation A is $\boldsymbol{v}_A$ = (a_0, $a_1, \cdots$, $a_{m\text{-}1}$). A multiplication and an inversion in F_{p^m} are carried out by using $f(\omega) = 0$; thus, $f(x)$ is called the modular polynomial of F_{p^m}. When the following conjugates of ω with respect to F_p are linearly independent:

$$\{\omega, \omega^p, \omega^{p^2}, \cdots, \omega^{p^{m-1}}\}, \tag{4}$$

this is called a *normal basis*, which allows us to use an efficient *Frobenius mapping*.

Consider the normal basis (4) in F_p^m and an arbitrary element A in F_{p^m} as follows:

$$\begin{aligned} A &= a_0\omega + a_1\omega^p + \cdots + a_{m-1}\omega^{p^{m-1}} \\ &= (a_0, a_1, \cdots, a_{m-1}). \end{aligned} \tag{5}$$

The Frobenius mapping is defined by

$$\begin{aligned} &A \to A^p : \\ &A^p = a_0\omega^p + a_1\omega^{p^2} + \cdots + a_{m-2}\omega^{p^{m-1}} + a_{m-1}\omega \\ &\quad = (a_{m-1}, a_0, \cdots, a_{m-2}). \end{aligned} \tag{6}$$

Thus, using a normal basis, we can carry out a Frobenius mapping without any arithmetic operations.

An optimal extension field (OEF) [10] whose modular polynomial is an irreducible binomial has a fast Frobenius mapping. Consider the OEF F_p^m with the following irreducible binomial over F_p as the modular polynomial:

$$f(x) = x^m - s, \quad s \in F_p. \tag{7}$$

Note that the OEF uses a polynomial basis given by

$$\{1, \omega, \omega^2, \cdots, \omega^{m-1}\}, \tag{8}$$

where ω is a zero of $f(x)$. When m divides p–1, for instance, the Frobenius map of $A \in F_{p^m}$ is given as

$$\begin{aligned} A^p &= a_0 + a_1\omega^p + a_2\omega^{2p} + \cdots + a_{m-1}\omega^{(m-1)p} \\ &= a_0 + a_1 l\omega + a_2 l^2\omega^2 + \cdots + a_{m-1} l^{m-1}\omega^{m-1}, \end{aligned} \tag{9}$$

where $l = s^{(p-1)/m}$ because ω^p is given by

$$\omega^p = \omega \cdot \omega^{p-1} = \omega(\omega^m)^{(p-1)/m} = \omega \cdot s^{(p-1)/m}. \tag{10}$$

Thus, previously computing l^i for i = 1, 2,⋯, m–1, the Frobenius mapping in the OEF carries out only m–1 F_p-multiplications. On the other hand, when the modular polynomial is a degree m irreducible trinomial over F_p, the Frobenius mapping in F_{p^m} will need to multiply an (m×m) matrix over F_p. In appendix A, some Frobenius matrices are shown. In what follows, we use the following notation: $\varphi_i(A) = A^{p^i}$.

2. Binary Method

The well-known binary method quickly calculates an exponentiation with a large integer exponent. Let n be a positive integer. The binary method calculates A^n as follows.

Algorithm 1. (right-to-left binary method)

Input: A and exponent n
Output: $B = A^n$

Main procedure:
1. $X \leftarrow 1$, $B \leftarrow A$.
2. If $n = 0$, output X.
3. Otherwise,
4. if $(n \,\&\, 1) = 1$, $X \leftarrow X \cdot B$.
5. $B \leftarrow B \cdot B$.
6. $n \leftarrow n >> 1$, then go to step 2.
7. Output B.

(end of algorithm)

In what follows, & and >> in algorithms denote the bit-and and bit-shift operators, respectively. The binary method needs only $\lfloor \log_2 n \rfloor$ squares and $\lfloor \log_2 n \rfloor / 2$ multiplications on average.

3. Window Method

Here, we give an example of how to use the window method (with window size 3) to calculate A^n. First, precompute the values

$$A^2, A^3, A^4, \cdots, A^7, \tag{11}$$

whose exponents correspond to the following binary representations, respectively:

$$2=(010)_2, \quad 3=(011)_2, \quad 4=(100)_2, \cdots, \quad 7=(111)_2. \tag{12}$$

Then, we calculate A^n by combining and squaring these previously calculated components. For example, when exponent n is 318, A^n is calculated as

$$A^{318} = A^{(100111110)_2} = \{(A^{(100)_2})^{2^3}\,(A^{(111)_2})\}^{2^3}\,A^{(110)_2}. \tag{13}$$

The window size w corresponds to the bit length of each segment. The window method is efficient for repeatedly calculating exponentiations with the same base. Note that the window method still needs $\lfloor \log_2 n \rfloor - w$ squares as shown in (13). When the base is often changed, we cannot efficiently precompute the components as in (11). We must precompute new values for every exponentiation base.

4. Previous Works

Let us briefly go over the Frobenius abusing exponentiation method proposed by Avanzi and others [9]. Note that, as introduced in [9], this method requires very fast Frobenius mappings. Write the exponent n as

$$n = \sum_{i=0}^{s} n_i p^i, \quad 0 \le n_i \le p-1, \quad s = \lfloor \log_p n \rfloor, \tag{14}$$

and for some

$$w << u = \lfloor \log_2 p \rfloor, \quad K = \lfloor u / w \rfloor, \tag{15a}$$

$$n_i = \sum_{j=0}^{K-1} n_{ij} 2^{jw}. \tag{15b}$$

Then, calculate A^n in F_{p^m} as

$$\begin{aligned} A^n &= \prod_{i=0}^{s} \varphi_i(A^{n_i}) = \prod_{i=0}^{s}\prod_{j=0}^{K-1} \varphi_i(A^{n_{ij}2^{jw}}) \\ &= \prod_{j=0}^{K-1}\left(\prod_{i=0}^{s} \varphi_i(A^{n_{ij}})\right)^{2^{jw}}. \end{aligned} \tag{16}$$

This method makes efficient use of the window method to calculate $A^{n_{ij}}$. Compared to just applying the window method in addition to Frobenius mappings, (16) decreases the number of multiplications but increases the number of Frobenius mappings. This algorithm needs u+1 squarings, (s+1)K+2w–2 multiplications, and

sK Frobenius mappings. Avanzi and others [9] introduced several efficient exponentiation methods such as baby-window giant-window exponentiation. The Frobenius-abusing exponentiation previously described is the most efficient of these methods.

III. Exponentiation in an Extension Field

In what follows, let us consider an exponentiation A^n, $A \in F_{p^m}$, and let m be larger than or equal to 2.

1. Main Idea

Consider the p-adic representation of the exponent n as (14). Then, using the Frobenius mapping φ, the exponentiation A^n is calculated by

$$A^n = \prod_{i=0}^{s} \varphi_i(A^{n_i}). \tag{17}$$

The component A^{n_i} is denoted by $A[i]$ in algorithm 2. Then, exponentiation A^n is calculated by the binary method with Frobenius mappings as follows. Equation (17) leads to the well-known p-adic expansion method, which is given here.

Algorithm 2

Input: An element $A \in F_{p^m}$ and an exponent n
Output: $C = A^n$
Main procedure:
1. $B \leftarrow A$, $s \leftarrow \lfloor \log_p n \rfloor$, $t \leftarrow \lfloor \log_2(p-1) \rfloor$.
2. For $0 \le i \le s$, $A[i] = 1$.
3. If $n = 0$, output 1.
4. Otherwise,
5. Calculate the p-adic representation of n as (14),
6. for $0 \le j < t$,
7. for $0 \le i \le s$,
8. if $(n_i \,\&\, 1) = 1$, $A[i] \leftarrow A[i] \cdot B$.
9. $n_i \leftarrow n_i >> 1$.
10. $B \leftarrow B \cdot B$.
11. $C = A[s]$.
12. for $s - 1 \ge j \ge 0$,
13. $C \leftarrow \varphi_1(C)$, $C \leftarrow C \cdot A[j]$.
14. Output C.

(end of algorithm)

As described section II.4, Avanzi's method extended algorithm 2 by efficiently using the window method.

This algorithm needs t squares and on average $st/2$ multiplications in F_{p^m}. After that, (17) needs $(s-1)$ Frobenius mappings and $(s-1)$ multiplications.

We improve algorithm 2. In what follows, using a concrete example and Fig. 1, the main idea is explained. For instance, let $s = \lfloor \log_p n \rfloor$ and $t = \lfloor \log_2(p-1) \rfloor$ be 5 and 4, respectively. Suppose the binary representations of $n_0, n_1, \cdots, n_5$ are

$$n_1 = (1001)_2, \quad n_0 = (1110)_2, \tag{18a}$$
$$n_3 = (1101)_2, \quad n_2 = (1110)_2, \tag{18b}$$
$$n_5 = (1111)_2, \quad n_4 = (0101)_2. \tag{18c}$$

Let us separate exponent n into three parts as follows (see Fig. 1):

$$n = (n_5 p + n_4)p^4 + (n_3 p + n_2)p^2 + (n_1 p + n_0). \tag{19}$$

For this p-adic expansion, the following procedure is our proposal. Consider two sets $G_1 = \{A^{n_5}, A^{n_3}, A^{n_1}\}$ and $G_2 = \{A^{n_4}, A^{n_2}, A^{n_0}\}$. They are given as

$$A^{n_1} = A^8 A^1, \qquad A^{n_0} = A^8 A^4 A^2, \tag{20a}$$
$$A^{n_3} = A^8 A^4 A^1, \qquad A^{n_2} = A^8 A^4 A^2, \tag{20b}$$
$$A^{n_5} = A^8 A^4 A^2 A^1, \qquad A^{n_4} = A^4 A^1, \tag{20c}$$

As shown in (18) and (20), component A^2 is needed for A^{n_5} in G_1 and A^{n_0}, A^{n_2} in G_2. Component A^4 is needed for A^{n_3}, A^{n_5} in G_1 and A^{n_0}, A^{n_2}, A^{n_4} in G_2 (see Fig. 1), for example. Note that $A^p = \varphi_1(A)$. Then, calculate $C_{001}, C_{010}, \cdots, C_{111}$ as (see Fig. 1)

$$C_{100} = \varphi_1(A^2)A^1, \quad C_{010} = 1, \quad C_{001} = 1, \tag{21a}$$
$$C_{011} = A^8 A^2, \quad C_{101} = 1, \tag{21b}$$
$$C_{110} = \varphi_1(A^4), \quad C_{111} = \varphi_1(A^8 A^1)A^4. \tag{21c}$$

Then, A^n is calculated as

$$R_0 = C_{100}\, C_{101}\, C_{110}\, C_{111}, \tag{22a}$$
$$R_1 = C_{010}\, C_{011}\, C_{110}\, C_{111}, \tag{22b}$$
$$R_2 = C_{001}\, C_{011}\, C_{101}\, C_{111}, \tag{22c}$$
$$A^n = \varphi_4(R_2)\, \varphi_2(R_1)\, R_0. \tag{22d}$$

Equations (17) and (20) need 3 squares and 16 multiplications without using the sliding window method; however, (21) and (22) need 3 squares and only 14 multiplications. Thus, the p-adic representation and Frobenius mappings shown in (17) help to decrease the number of squares, and calculation based on (21) helps to decrease the number of multiplications. Figure 1 shows the calculations of (18)-(22). From another viewpoint, (22) just calculates three exponentiations of the same base at once.

To decrease the number of multiplications, Avanzi's method additionally applies the window method to calculate A^{n_i}. However, this increases the number of Frobenius mappings required; therefore, it requires fast Frobenius mapping. In the proposed method we take a different approach. The number of multiplications increases slightly, but the number of Frobenius mappings is much smaller than in Avanzi's method.

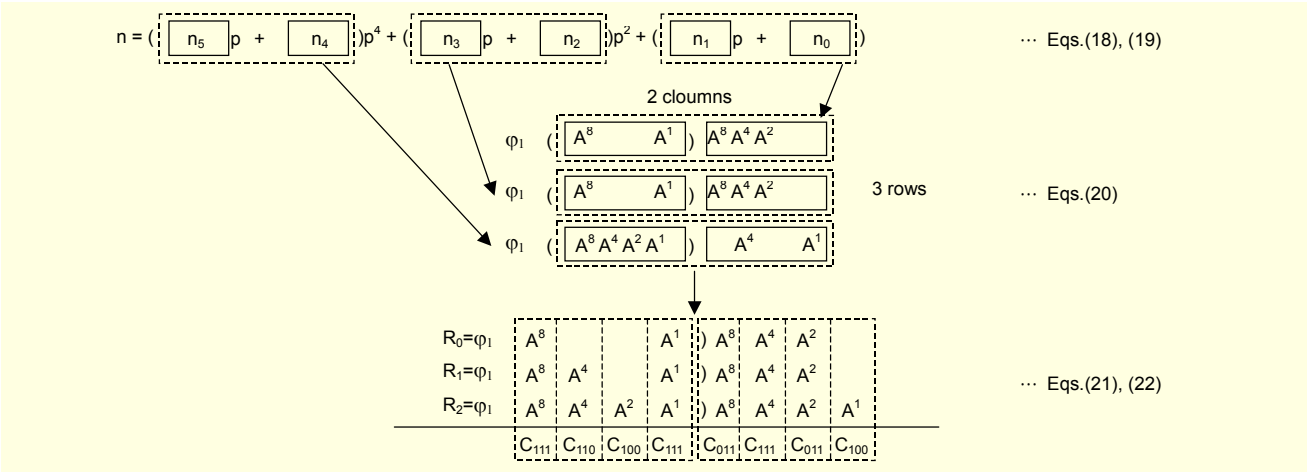

Fig. 1. Image of the calculating (18) to (22).

Accordingly, the proposed method is more efficient when the target extension field does not have fast Frobenius mapping.

2. Proposed Algorithm

As shown in Fig. 1, let the numbers of rows and columns be r and c, respectively. Write the exponent n in the form

$$n = \sum_{i=0}^{r-1} \sum_{j=0}^{c-1} n_{ij} p^{ci+j}. \tag{23}$$

The number of columns c is automatically determined by the bit-size of n and the number of rows r. The proposed method calculates A^n as

$$S_{jl} = \left\{ x \mid \sum_{i=0}^{r-1} 2^i (n_{ij} \,\&\, 2^x) = l,\ 0 \le x < t \right\}, \tag{24a}$$

where $0 \le j < c$ and $1 \le l < 2^r$,

$$T_i = \{ y \mid (2^i \,\&\, y) = 2^i,\ 1 \le y < 2^r - 1 \}, \tag{24b}$$

where $0 \le i < r$,

$$C_l = \begin{cases} 1 & \text{when } S_{jl} = 0, \\ \prod_{j=0}^{c-1} \varphi_j \left(\prod_{k \in S_{jl}} A^{2^k} \right) & \text{otherwise,} \end{cases} \tag{24c}$$

where $1 \le l < 2^r$,

$$R_i = \prod_{j=0}^{c-1} \varphi_j \left(A^{n_{ij}} \right) = \prod_{u \in T_i} C_u,\ 0 \le i < r. \tag{24d}$$

$$A^n = \prod_{i=0}^{r-1} \varphi_{ci} \left(R_i \right). \tag{24e}$$

It is found that $|S_{jl}| \le t$ and $|T_i| = 2^{r-1}$. Equation (24a) gives the set of binary representations related to column bits such as the subscript 100 of C_{100} shown in Fig. 1. Equation (24b) gives the set of integers from 1 to 2^r-1 whose i-th bit is equal to 1. Equation (24c) shows the temporary data C_l that is calculated as the product of A^{2^k}, $k \in S_{jl}$, $0 \le j < c$-1 by appropriately using their Frobenius mappings. When S_{jl} is empty, that is denoted by 0. Set C_1=1 for the following calculations. Equation (24d) calculates R_i for the i-th row by multiplying C_u such that u belongs to T_i. Finally, (24e) calculates the exponentiation A^n by using previously calculated R_i, their Frobenius mappings, and $(r-1)$ multiplications in F_{p^n}.

Algorithm 3 shows the proposed exponentiation. First, the lines 1 to 4 initialize the computational buffers. If the exponent $n = 0$, it outputs 1. Otherwise, it calculates the p-adic representation of n as (23). Then, it calculates the C_l, $1 \le l < 2^r$ as (24c) in lines 8 to 17. In detail, lines 10 to 13 calculate the S_{jl}, $0 \le j < c$, $1 \le l < 2^r$ as (24a). When $S_{jl} \ne 0$, by using the k's in S_{jl}, lines 10 to 14 calculate the product $\prod_{k \in S_{jl}} A^{2^k}$ shown in (24c). Their Frobenius mappings construct the C_l in lines 15 to 17. Then, lines 18 to 20 calculate the R_i, $0 \le i < r$ for $u \in T_i$ as (24d). Finally, lines 21 to 23 construct the result A^n as (24e).

In the proposed method, the number of temporary variables C_l and R_i is $(2^r - 1) + r$ as shown in (24). The preparation of C_l, $1 \le l < 2^r$ needs at most $c \cdot t$ multiplications, where $t = \lfloor \log_2 (p-1) \rfloor$. Using them, A^n is calculated with less than $r(2^{r-1} - 1) + (r - 1)$ multiplications as shown in (24d) and (24e). In addition, the proposed algorithm needs $(c - 1)(2^r - 1) + (r - 1)$ Frobenius mappings.

IV. Cost Evaluation

In evaluating the cost of the calculation, a subtraction in F_p is counted as an addition in F_p. As introduced in the preceding

Table 1. Comparison of the calculation cost for an exponentiation in F_{p^m}.

Degree m	3	4	5	6	7
Proposal Eqs.(24)	$r=3, c=1$ (159, 150, 2)	$r=4, c=1$ (159, 180, 3)	$r=5, c=1$ (159, 234, 4)	$r=3, c=2$ (159, 290, 9)	$r=4, c=2$ (159, 320, 18)
Avanzi Eq.(16)	$w=5$ (160, 125, 61)	$w=5$ (160, 156, 92)	$w=5$ (160, 187, 123)	$w=5$ (160, 218, 154)	$w=6$ (156, 249, 159)
Degree m	8	9	10	11	12
Proposal Eqs.(24)	$r=4, c=2$ (159, 330, 18)	$r=5, c=2$ (159, 383, 35)	$r=5, c=2$ (159, 388, 35)	$r=4, c=3$ (159, 470, 33)	$r=4, c=3$ (159, 480, 33)
Avanzi Eq.(16)	$w=6$ (156, 275, 185)	$w=6$ (156, 302, 212)	$w=6$ (156, 328, 238)	$w=6$ (156, 354, 264)	$w=6$ (156, 381, 291)

Remark : (#*S*, #*M*, #*F*) means that #*S* squares, #*M* multiplications, and #*F* Frobenius mappings in F_{p^m} are respectively needed on average.

sections, one of targets of the proposed exponentiation method is the case in which a Frobenius mapping cannot be carried out fast in the concerned extension field.

Algorithm 3. (proposed algorithm)

Input: An element $A \in F_{p^m}$ and an exponent n
Output: $D = A^n$
Main procedure:

1. $B[0] \leftarrow A$, $t \leftarrow \lfloor \log_2(p-1) \rfloor$.
2. For $1 \le i < t$, $B[i] \leftarrow B[i-1] \cdot B[i-1]$.
3. For $0 < i \le 2^r$, $C[i] = 1$.
4. For $0 \le i < r$, $R[i] = 1$.
5. If $n = 0$, output 1.
6. Otherwise,
7. calculate the p-adic representation of n as (23),
8. for $c > j \ge 0$,
9. for $0 \le k < t$,
10. $l = 0$,
11. for $0 \le i < r$,
12. if $(n_{ij}$ & $1) = 1$, $l \leftarrow l + 2^i$.
13. $n_{ij} \leftarrow n_{ij} >> 1$.
14. if $l \ne 0$, $C[l] \leftarrow C[l] \cdot B[k]$.
15. if $j \ne 0$,
16. for $1 \le i < 2^r$,
17. $C[i] \leftarrow \varphi_1(C[i])$.
18. for $0 \le i < r$,
19. for $1 \le u < 2^r$,
20. if $(2^i$ & $u) \ne 0$, $R[i] \leftarrow R[i] \cdot C[u]$.
21. $D \leftarrow R[r-1]$.
22. for $r - 2 \ge i \ge 0$,
23. $D \leftarrow \varphi_c(D)$, $D \leftarrow D \cdot R[i]$.
24. output D

(end of algorithm)

In this section, we simulate exponentiation over an extension field with a random160-bit characteristic p and extension degree $m = \{3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$. Algorithm 3 and Avanzi's method were compared in F_{p^m}. Inputting 10,000 random elements in F_{p^m} as the bases and ($m \log_2 p$)-bit random integers as an exponent, the calculation costs for the proposed method and Avanzi's method were computed. Table 1 shows a comparison of the average calculation cost for an exponentiation in F_{p^m} using the proposed method and Avanzi's method [9]. Row size r and column size c as in (24) in the proposed method and the window size w of (15) of Avanzi's method were set to the experimentally optimal values corresponding to each pair of p and m.

Table 2. Comparison of the proposed method baby-window giant-window method [9] in the case of 32-bit characteristic.

Degree m	4	6	8
Proposed method	$r=4, c=1$ (31, 58, 3)	$r=3, c=2$ (31, 65, 9)	$r=4, c=2$ (31, 88, 18)
Baby-window giant-window method [9]	$w=1$ (31, 63, 3)	$w=2$ (31, 87, 5)	$w=2$ (31, 111, 7)

Remark : (#*S*, #*M*, #*F*) means that #*S* squares, #*M* multiplications, and #*F* Frobenius mappings in F_{p^m} are respectively needed on average.

Table 1 shows the numbers of F_p-squarings, F_p-multiplications, and Frobenius mappings. For example, when $m = 3$, the proposed method with $r = 3$ and $c = 1$ needs 159 F_p-squarings, 150 F_p-multiplications, and 2 Frobenius mappings. Compared to Avanzi's method, the proposed exponentiation method needs more F_p-multiplications but fewer Frobenius mappings. As introduced in section II.1, in some cases, Frobenius mapping becomes complicated. Thus, in such cases, the proposed method will be more efficient than Avanzi's method [9].

Avanzi and others [9] also introduced the baby-window giant-window exponentiation method in which p is a 32-bit prime. Table 2 shows a comparison of the proposed method and the baby-window giant-window method [9]. In this case, the proposed exponentiation method is also more efficient.

V. Conclusion

This paper has proposed an exponentiation method in an extension field that efficiently uses Frobenius mappings. Compared to Avanzi's method [9], about 20% more multiplications are needed for the proposed method; however, the number of Frobenius mappings is low enough. Evaluating the calculation costs, it was shown that the proposed exponentiation method is practical for cases in which the base of exponentiation is often changed and a Frobenius mapping cannot be carried out fast in the concerned extension field. As a future work, the number of F_p-multiplications for the proposed method should be further decreased.

Appendix A. Frobenius Matrices

For example, consider the following parameters:

$$m = 10, \tag{A1a}$$

$$p = 4212134911 \text{ (32-bit)}, \tag{A1b}$$

$$\text{irreducible binomial: } x^{10} + 2, \tag{A1c}$$

$$\text{irreducible trinomial: } x^{10} + x + 13. \tag{A1d}$$

Equation (A2) shows Frobenius matrix M_n for a normal basis.

This means that Frobenius mapping is simply carried out by cyclic-shift operation. Thus, using an extension field inversely becomes complicated. Equation (A3) shows Frobenius matrix M_b for a polynomial basis given by using irreducible binomial (A1c) as the modular polynomial. The main diagonal coefficients of M_b denote l^i of (9) introduced in section II.1. Thus, previously computing M_b, Frobenius mapping is carried out only by (m–1) F_p-multiplications. Equation (A4) shows Frobenius matrix M_t for a polynomial basis given by using irreducible trinomial (A1d) as the modular polynomial. The first row of M_t is always (1, 0,⋯, 0) because the first entry of the polynomial basis is 1; however, other rows depend on the modular trinomial. Thus, in this case, Frobenius mapping requires about m^2 F_p-multiplications. In this case, the proposed exponentiation method works efficiently.

In the above introduction, polynomial bases are considered for irreducible binomials and trinomials because such special irreducible polynomials are usually adopted to efficiently carry out multiplication in an extension field. In detail, they are efficient for polynomial modulo operation since the number of non-zero coefficients is small.

$$M_n = \begin{bmatrix} 0&1&0&0&0&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0 \\ 0&0&0&1&0&0&0&0&0&0 \\ 0&0&0&0&1&0&0&0&0&0 \\ 0&0&0&0&0&1&0&0&0&0 \\ 0&0&0&0&0&0&1&0&0&0 \\ 0&0&0&0&0&0&0&1&0&0 \\ 0&0&0&0&0&0&0&0&1&0 \\ 0&0&0&0&0&0&0&0&0&1 \\ 1&0&0&0&0&0&0&0&0&0 \end{bmatrix} \tag{A2}$$

$$M_b = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0 \\ 0&3362631751&0&0&0&0&0&0&0&0 \\ 0&0&791230795&0&0&0&0&0&0&0 \\ 0&0&0&2089358268&0&0&0&0&0&0 \\ 0&0&0&0&448624312&0&0&0&0&0 \\ 0&0&0&0&0&4212134910&0&0&0&0 \\ 0&0&0&0&0&0&849503160&0&0&0 \\ 0&0&0&0&0&0&0&3420904116&0&0 \\ 0&0&0&0&0&0&0&0&2122776643&0 \\ 0&0&0&0&0&0&0&0&0&3763510599 \end{bmatrix} \tag{A3}$$

$$M_t = \begin{bmatrix} 1&0&0&0&0&0&0&0&0&0 \\ 2062726829&1124979754&1823274627&323064958&3683581484&481514977&3261931157&1741598628&1735563347&2759933689 \\ 2548988990&1234937071&1646966899&3517048746&1805854166&712516681&1948759334&801547702&3803521501&3300224979 \\ 822289555&2336381105&628081742&2613575938&1697555978&1400889739&3507562462&66029264&3518360137&445640071 \\ 3875561743&3184290976&1350830842&3389663137&2762346776&212922801&3524266107&1201427243&23986191&2434119754 \\ 3932326008&2663484293&1678173095&307236222&2331927710&1614732131&331017308&3761918956&1343004355&157116209 \\ 2073858903&2838260129&2415144807&3740081012&3309231865&3904317011&1824638989&2990697270&2417820003&2304287670 \\ 1326445396&1497631197&178323180&4092446731&2639138006&2656810428&3438409067&785686163&763867726&2409858198 \\ 704368424&2620212155&2198840331&2444682151&2569489181&707098076&1975073542&1696282037&1600673803&4058736513 \\ 2166231779&3239609735&568286949&3879112940&1339365838&673421108&3927183875&1157563823&922128729&2874939190 \end{bmatrix} \tag{A4}$$

References

[1] D. Boneh, B. Lynn, and H. Shacham, "Short Signatures from the Weil Pairing," *Proc. of Asiacrypt*, LNCS 2248, 2001, pp. 514-532.

[2] T. Nakanishi and N. Funabiki, "Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps," *Proc. of Asiacrypt*, LNCS 3788, 2005, pp. 443-454.

[3] D. Boneh, C. Gentry, and B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys," *Proc. of CRYPTO*, LNCS 3621, 2005, pp. 258-275.

[4] J. Silverman, *The Arithmetic of Elliptic Curve*, Springer-Verlag, 1986.

[5] H. Cohen and G. Frey, *Handbook of Elliptic and Hyperelliptic Curve Cryptography*, *Discrete Mathematics and Its Applications*, Chapman & Hall CRC, 2005, pp. 280-285, p. 458.

[6] F. Hess, N. Smart, and F. Vercauteren, "The Eta Pairing Revisited," *IEEE Transactions on Information Theory*, vol. 52, no. 10, 2006, pp. 4595-4602.

[7] A. Brauer, "On Addition Chains," *Bull. Amer. Math. Soc.*, vol. 45, 1939, pp. 736-739.

[8] B. Moller, "Algorithms for Multi-exponentiation," *Proc. of SAC*, LNCS 2259, Springer-Verlag, 2001, pp. 165-180.

[9] R. Avanzi and P. Mihailescu, "Generic Efficient Arithmetic Algorithms for PAFFs (Processor Adequate Finite Fields) and Related Algebraic Structures," *Proc. of SAC*, LNCS 3006, Springer-Verlag, 2003, pp. 320-334.

[10] D. Bailey and C. Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms," *Proc. of Asiacrypt*, LNCS 1976, 2000, pp. 248-258.

[11] D. Freeman, "Constructing Pairing-Friendly Elliptic Curves with Embedding Degree 10," *Proc. of ANTS-VII*, LNCS 4076, Springer-Verlag, 2006, pp. 452-465.

Yasuyuki Nogami graduated from Shinshu University in 1994 and received the PhD degree in 1999 from Shinshu University. He is now a research associate of Okayama University. His main fields of research are finite field theory and its applications. He is a member of IEICE and IEEE.

Hidehiro Kato graduated from Okayama University in 2005 and obtained the MS degree in 2006. He is now a doctoral candidate of the Graduate School of Natural Science and Technology, Okayama University. He is studying finite field theory, especially the implementation of fast arithmetic operations in a finite field. He is a member of IEICE.

Kenta Nekado graduated from the Department of Communication Network Engineering, the Faculty of Engineering, Okayama University, in 2007. He is now with the Graduate School of Natural Science and Technology, Okayama University, where he is studying finite field theory.

Yoshitaka Morikawa graduated from the Department of Electronic Engineering, Osaka University in 1969, and obtained the MS degree in 1971. He then joined Matsushita Electric, where he engaged in research on data transmission. In 1972, he became a research associate at Okayama University, and subsequently became an associate professor in 1985. He is now a professor of the Graduate School of Natural Science and Technology. He has been engaged in research on image information processing. He holds a Deng degree.