Memoirs of the Faculty of Engineering, Okayama University, Vol.37, No.2, pp.73-87, March 2003

A Fast Implementation of Elliptic Curve Cryptosystem with Prime Order Defined over F_{p^8}

Yasuyuki Nogami† Yoshitaka Morikawa†

Department of Communication Network Engineering
Okayama University
Okayama 700-8530 Japan

Public key cryptosystem has many uses, such as to sign digitally, to realize electronic commerce. Especially, RSA public key cryptosystem has been the most widely used, but its key for ensuring sufficient security reaches about 2000 bits long. On the other hand, elliptic curve cryptosystem(ECC) has the same security level with about 7-fold smaller length key. Accordingly, ECC has been received much attention and implemented on various processors even with scarce computation resources.

In this paper, we deal with an elliptic curve which is defined over extension field $F_{p^{2^c}}$ and has a prime order, where p is the characteristic and c is a non negative integer. In order to realize a fast software implementation of ECC adopting such an elliptic curve, a fast implementation method of definition field $F_{p^{2^c}}$ especially F_{p^8} is proposed by using a technique called successive extension. First, five fast implementation methods of base field F_{p^2} are introduced. In each base field implementation, calculation costs of F_{p^2}-arithmetic operations are evaluated by counting the numbers of F_p-arithmetic operations. Next, a successive extension method which adopts a polynomial basis and a binomial as the modular polynomial is proposed with comparing to a conventional method. Finally, we choose two prime numbers as the characteristic, and consider several implementations for definition field F_{p^8} by using five base fields and two successive extension methods. Then, one of these implementations is especially selected and implemented on Toshiba 32-bit micro controller TMP94C251(20MHz) by using C language. By evaluating calculation times with comparing to previous works, we conclude that proposed method can achieve a fast implementation of ECC with a prime order.

I. Introduction

Recently, in the modern information-oriented society, various equipments are connected to the internet as terminals. In order to protect the equipments or some important informations from evil internet users, information security technology has played a key role. Especially, public key cryptosystem has many uses, such as to sign digitally, to realize electronic commerce[1]~[3]. RSA cryptosystem is one of public-key cryptosystems and has been the most widely used, but its key for ensuring sufficient security reaches about 2000 bits long[4]. Therefore, it is not efficient to implement RSA cryptosystem on a terminal with scarce computation resources, such as IC card and 32-bit micro controller. On the other hand, elliptic curve cryptosystem(ECC)[5],[6] has the same security level with about 7-fold smaller length key as compared to RSA cryptosystem. Accordingly, ECC has been received much attention and implemented on various processors[7],[8].

Elliptic curve adopted in ECC will be almost given by

$$E(x,y) = y^2 - x^3 - ax - b = 0. \tag{1}$$

In addition, coefficients a, b are elements in some finite field, which is called coefficient field, and the solutions (x, y) to Eq.(1) are called rational points. The rational points over an elliptic curve form an additive Abelian group, and the security of the ECC relies upon the difficulty of discrete logarithm problem on this group. This problem is so-called elliptic curve discrete logarithm problem(ECDLP)[5]. Since the additive Abelian group plays a role of key space in the ECC, the order of the group, that is the number of rational points, must be a large prime or divisible by a large prime for ensuring sufficient security. In practice, such a large prime should be 160 bits long at least[5]. Correspondingly, the order of its definition field, in which the coordinates of the rational points lie, has to be 160 bits at least[5]. Therefore, if a concerned processor has only scarce computation resources, we should especially pay attention to its software implementation so as to satisfy the following two requirements:

- Encryption and decryption can be carried out within comfortable processing time.

†E-mail: {nogami,morikawa}@cne.okayama-u.ac.jp

- Programs can be implemented even on a processor with scarce computation resources.

Previous works achieving these requirements can be classified into two subjects as follows[9],[10]:

- Fast implementation method of definition field.
- Fast scalar multiplication method for rational points.

These requirements are not separately dealt with in these previous works. And moreover, the former can be classified roughly into two types according to the definition field whether a prime field or an extension field. In this paper, fast implementation method using an extension field as the definition field is discussed. Accordingly, this paper belongs to the former subject.

In the ECC defined over extension field F_{p^m}, where p is the characteristic and m is the extension degree, the pair of p and m has to satisfy $m \log p \geq 160$ in order to ensure its security[10]. For example, we may adopt a 30 bits long prime and 6 as p and m, respectively. It yields easy implementation even on a terminal with scarce computation resources. Specifically, if the definition field has fast arithmetic operations, then the encryption/decryption will be fast carried out. From such a background, the authors have already proposed a method to generate an elliptic curve which is defined over extension field $F_{p^{2^c}}$[11], has a prime order, and can resist against Frey-Ruck(FR) attack[12], where c is constant and elliptic curve with a prime order is abbreviated to EPO throughout this paper. The paper[12] does not take Weil Descent attack into account since this attack can be applied to a certain ECC only in the case of characteristic $p = 2, 3$ at present[13],[14].

Based on these researches, this paper realizes a fast software implementation of extension field $F_{p^{2^c}}$ especially F_{p^8} by using a technique called *successive extension*. By using the extension field as the definition field of ECC in which an EPO is adopted, scalar multiplication for rational points, which is needed in the encryption/decryption processes, is programed onto 32-bits micro controller. Then, it is shown that our implementation can achieve a fast implementation of such an ECC with a prime order.

By using *successive extension*, F_{p^2} over F_p, F_{p^4} over F_{p^2}, and F_{p^8} over F_{p^4} are successively constructed. In this paper, five fast implementation methods of F_{p^2} are introduced, where each implemented field is used as the base field for the successive extension. And then, we evaluate calculation costs of F_{p^2}-arithmetic operations, such as multiplication, in each base field by counting the numbers of F_p-arithmetic operations needed for the implementation of F_{p^2}-arithmetics. After that, the efficiency of each implementation is discussed from a view point of a fast implementation of the base field.

Next, an efficient successive extension method for a fast implementation of the definition field is discussed. At first, a general extension of base field F_q to extension field F_{q^2}, in which a binomial and a polynomial basis are adopted as the modular polynomial and the basis respectively, is shown with comparing to a conventional method[10], in which a trinomial and a normal basis are adopted. In addition, calculation costs of the arithmetic operations of implemented F_{q^2} are evaluated by counting the numbers of F_q-arithmetic operations. After that, we discuss the efficiency of these two extension methods for a fast implementation of the definition field.

Finally, we choose two prime numbers as the characteristic and then especially consider several definition fields with using the preceding two successive extension methods and five base fields. For each definition field F_{p^8}, calculation costs of F_{p^8}-arithmetic operations are evaluated by counting the numbers of F_p-arithmetic operations. Based on the evaluation, some definition fields that are especially expected to carry out their arithmetic operations fast on generally-used processor are selected. After that, those definition fields are explicitly implemented on Intel Celeron(400MHz) processor by using C language, then calculation times of those F_{p^8}-arithmetic operations are measured. Based on the calculation times and also the preceding discussions, one definition field which is expected to be the best for a fast implementation of ECC is selected and then implemented on Toshiba 32-bit micro controller TMP94C251(20MHz) by using C language, after that the average of the calculation time of a scalar multiplication is measured. Concludingly, by comparing these calculation times to previous works, it is shown that our proposed method can achieve a fast implementation of ECC with a prime order.

Notations: Throughout this paper, capital letters and not capital letters, such as "A" and " a ", denote elements in extension field and its base field, respectively. In addition, p denotes a prime number, and Greek letters, such as ω, denotes a zero of irreducible polynomial. Abbreviations ADD_m, SUB_m, MUL_m, SQR_m, FRO_m, and INV_m means addition, subtraction, multiplication, square operation, Frobenius mapping, and inversion in extension field F_{p^m}, respectively, in other words lower suffix denotes extension degree over a prime field F_p.

II. Fundamentals

In this section, we deal with fundamentals of elliptic curve, elliptic curve cryptosystem(ECC), and extension field with fast arithmetic.

A. Elliptic curve

A.1 Coefficient field and definition field

An elliptic curve over finite field F_q is defined as the set of solutions to the equation

$$E(x, y) = y^2 - x^3 - ax - b = 0, \tag{2}$$

with $a, b \in F_q$ and $char(F_q) \neq 2, 3$. The notation $char(F_q)$ shows the characteristic of F_q and it must be a prime. The solutions (x, y) to Eq.(2) are called F_q-rational points when the coordinates of x and y lie in F_q. Previous work[11] has dealt with elliptic curves that the coordinates lie in some extension field but the coefficients a, b is contained in its proper subfield. In order to describe the difference clearly, we call the field in which a, b is contained coefficient field and that in which coordinates lie definition field.

A.2 Group of rational points

For F_q-rational points $P(x_1, y_1), Q(x_2, y_2)$ and defining equation(2), addition $P+Q = (x_3, y_3)$ is defined as follows:

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} & P \neq Q \\ \dfrac{3x_1^2 + a}{2y_1} & P = Q \end{cases}, \quad (3a)$$

$$x_3 = \lambda^2 - x_1 - x_2, \quad y_3 = (x_1 - x_3)\lambda - y_1, \quad (3b)$$

In the following, if $P = Q$ this addition is especially called EC doubling(ECD), and else EC addition(ECA).

The elliptic curve has exactly one *point at infinity*, which is denoted by $\mathcal{O}$. Together with the point $\mathcal{O}$, the set of F_q-rational points forms an Abelian group under the additive law of Eq.(3a),(3b). The *point at infinity* $\mathcal{O}$ plays a role of identity element in this group. In this paper, the set of the F_q-rational points together with $\mathcal{O}$ is denoted by $E(F_q)$, and the number of the rational points, which is refered to as the order of elliptic curve, is denoted by $\#E(F_q)$. In addition, order $\#E(F_q)$ can be calculated by using SEA algorithm[5].

Now, let us consider how to discover a F_q-rational point on the curve Eq.(2). At first, for a random element $\alpha \in F_q$, calculate the LHS of Eq.(4) with $\beta = -E(\alpha, 0)$.

$$\beta^{\frac{q-1}{2}} = \begin{cases} 1 \text{ or } 0 & ; \text{QPR} \\ -1 & ; \text{QPNR} \end{cases}, \quad (4)$$

where QPR and QPNR are abbreviations of quadratic power residue and quadratic power non residue, respectively. If β is a QPR, then calculate its square roots $\pm\sqrt{\beta} \in F_q$ by using the algorithm[15], and both of $(\alpha, \pm\sqrt{\beta})$ become F_q-rational points.

A.3 Elliptic curve cryptosystem(ECC)

Elliptic curve cryptosystem[5] is based on the difficulty of elliptic curve discrete logarithm probrem(ECDLP) in the additive Abelian group which is introduced in the previous subsection. ECDLP can be easily understood by using the following equation,

$$Q = \underbrace{P + P + \cdots + P}_{k \text{ times}} = kP. \quad (5)$$

For a certain F_q-rational point P, we can calculate k times of P by using a certain algorithm even if the order $\#E(F_q)$ is very large, such as 160 bits. But, it is too hard to inversely calculate coefficient k from only the coordinates of P and Q, this problem is ECDLP.

The calculation defined by Eq.(5) is usually refered to as scalar multiplication, and ECC needs several scalar multiplications in the encryption/decryption processes[5]. In addition, if order $\#E(F_q)$ is very large, it is not efficient to recursively calculate the scalar multiplication as seen in the center of Eq.(5). In order to carry out scalar multiplication fast, some efficient methods has been proposed, such as Binary method[5], Sliding Window method[5], and Frobenius method[10]. Non-Adjacent Form signed binary method(NAF method) is one of such methods and adopted in this paper[16].

B. *Extension field with fast arithmetics*

The length of encryption/decryption key in ECC, that is the size of the order of an elliptic curve, can be roughly estimated from the order of the definition field[5]. For example, let us consider an extension field F_{p^m} as its definition field, order $\#E(F_{p^m})$ becomes about 160 bits when $m \log p$ is about 160. For fast software implementation of ECC, the extension field must have fast arithmetics, such as multiplication and inversion in the extension field. As conventional methods satisfying such requirement, optimal extension field(OEF)[17] and all-one polynomial field(AOPF)[9] are well known. Table I shows possible extension degrees of OEF and AOPF, respectively.

TABLE I

POSSIBLE EXTENSION DEGREES OF OEF AND AOPF

	2	4	5	6	7	8	9	10	16
OEF	○	○	○	○	○	○	○	○	○
AOPF	○	○	×	○	×	×	×	○	○

○···possible, ×···impossible

In these fields, we restrict their characteristic and the modular polynomial as follows.

1. Characteristic p is a pseudo Mersenne prime of computer's *word size*, where we call a prime in the form of $2^n \pm c$ ($\log_2 c \leq n/2$) pseudo Mersenne prime.
2. Modular polynomial is an irreducible binomial(OEF) or an irreducible all-one polynomial(AOPF).

III. FAST IMPLEMENTATION OF F_{p^2}

For a fast software implementation of ECC which is defined over extension field F_{p^m}, we should choose its characteristic p and entension degree m versus to the processor's *word size* as seen in Table II, in which p and m should satisfy $m \log p \geq 160$. In addition, since this paper adopts

TABLE II

CHARACTERISTIC AND EXTENSION DEGREE VERSUS TO PROCESSOR'S WORD SIZE

word size[bits]	characteristic[bits]	extension degree
8	$4 \sim 8$	$20 \sim 40$
16	$10 \sim 16$	$10 \sim 16$
32	$20 \sim 32$	$5 \sim 8$
64	$40 \sim 64$	3, 4

EPO generation algorithm[11], we must restrict the extension degree to a certain power of 2. And moreover, since we deal with an implementation on a 32-bit micro processor, it is desirable that the length of characteristic p is less than 32 bits. Under these conditions, a case that extension degree m equals to 8 is especailly suitable. In this paper, a technique called *successive extension*[18] is introduced in order to realize a fast implementation of extension field F_{p^8} as the definition field.

This section deals with a fast implementation of F_{p^2} which is used as the base field for successive extension. First, we discuss five fast implementation methods for F_{p^2} which are specified by the modular polynomial as shown in Table III, and these methods are refered to by the notation with number, such as $F_{p^2}(1)$. Then, we evaluate

TABLE III

CORRESPONDENCE BETWEEN FIVE IMPLEMENTATION METHODS OF F_{p^2} AND THEIR MODULAR POLYNOMIALS

Method	Modular polynomial	Notation
OEF	x^2+1	$F_{p^2}(1)$
	x^2-2	$F_{p^2}(2)$
AOPF	x^2+x+1	$F_{p^2}(3)$
NEF	x^2-x-1	$F_{p^2}(4)$
	x^2-x+1	$F_{p^2}(5)$

calculation costs of MUL_2, SQR_2, and INV_2 in each implementation by counting the numbers of F_p-arithmetic operations, such as ADD_1 and MUL_1. It should be noted that $F_{p^2}(4)$ and $F_{p^2}(5)$ are both implemented by NTT method[18], which is called NEF in the followings. In this paper, there are no discussions of addition and subtraction in the extension field except for ADD_2 and SUB_2, and details of a fast implementation of F_p-arithmetics can be seen in Bailey et al.[17].

A. Conditions for the modular polynomial to be irreducible

Modular polynomial has to be irreducible[19]. Table IV shows the necessary and sufficient condition for each modular polynomial tabulated in Table III to be irreducible.

TABLE IV

CONDITIONS FOR THE MODULAR POLYNOMIAL TO BE IRREDUCIBLE

	Condition	Reference
$F_{p^2}(1)$	$p \not\equiv 1 \pmod 4$	Section4
$F_{p^2}(2)$	$2^{(p-1)/2} \equiv -1 \pmod p$	Section4
$F_{p^2}(3)$	$p \equiv 2 \pmod 3$	[9]
$F_{p^2}(4)$	$5^{(p-1)/2} \equiv -1 \pmod p$	Section4
$F_{p^2}(5)$	$(-3)^{(p-1)/2} \equiv -1 \pmod p$	Section4

For example, let us consider a Mersenne prime $2^{31}-1$ as characteristic p. Then, we can fast perform the basic arithmetics in prime field F_p[17], however, only the modular polynomials of $F_{p^2}(1)$ and (4) tabulated in Table III become irreducible. Therefore, it is possible to implement $F_{p^2}(1)$ and (4) but not $F_{p^2}(2)$, (3) and (5).

B. Basis of extension field F_{p^2}

The oerformance of basic arithmetic operations in extension field is closely related to the choise of basis. If the choise is wrong, arithmetic operations in the extension field will become complicated. The basis adopted in each F_{p^2} implementation is shown in Table V. Accordingly, the implementations of arithmetic operations become simplified. Since pseudo polynomial basis $\{\omega, \omega^2\}$ of $F_{p^2}(3)$ is equal to $\{\omega, \omega^p\}$[9], it is also called optimal normal basis(ONB).

TABLE V

BASIS OF EACH IMPLEMENTATION METHOD

	Basis	Basis type
$F_{p^2}(1)$	$\{1, \omega\}$	polynomial basis
$F_{p^2}(2)$	$\{1, \omega\}$	polynomial basis
$F_{p^2}(3)$	$\{\omega, \omega^2\}$	pseudo polynomial basis
$F_{p^2}(4)$	$\{\omega, \omega^p\}$	normal basis
$F_{p^2}(5)$	$\{\omega, \omega^p\}$	normal basis

* ω is a zero of the modular polynomial.

C. Implementation of MUL_2, SQR_2, and INV_2

In this subsection, how to implement MUL_2, SQR_2, and INV_2 fast are explicitly discussed by using $F_{p^2}(1)$ arithmetics as an example. For the others $F_{p^2}(2)\sim(5)$, refer to Appendix.A. And then, calculation costs of these arithmetic operations in each F_{p^2} are concluded in Table VI.

C.1 MUL_2

In the case of $F_{p^2}(1)$, arbitrary elements $A, B \in F_{p^2}$ are represented by using the basis of $F_{p^2}(1)$, which is tabulated in Table V, as follows:

$$A = a_0 + a_1\omega, \quad a_0, a_1 \in F_p, \tag{6a}$$
$$B = b_0 + b_1\omega, \quad b_0, b_1 \in F_p. \tag{6b}$$

For example, ADD_2 and SUB_2 are calculated by

$$A \pm B = (a_0 \pm b_0) + (a_1 \pm b_1)\omega. \tag{7}$$

Multiplication of A and B, that is MUL_2, can be calculated by the following equation, where a relation $\omega^2 = -1$ is used.

$$\begin{aligned} AB &= a_0b_0 + (a_0b_1 + a_1b_0)\omega + a_1b_1\omega^2 \\ &= (a_0b_0 - a_1b_1) + (a_0b_1 + a_1b_0)\omega. \end{aligned} \tag{8a}$$

Calculation of Eq.(8a) can be implemented by using an ADD_1, a SUB_1, and 4 MUL_1's. Now, if we calculate the second term of RHS of Eq.(8a) by using Karatsuba algorithm(KA)[20], then it follows that

$$a_0b_1 + a_1b_0 = (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1, \tag{8b}$$

where it is noted that terms a_0b_0 and a_1b_1 have been already calculated in the first parenthesis of Eq.(8a). As seen in Eq.(8b), the calculation by using KA increases the number of ADD_1's and SUB_1's, however, that of MUL_1's can be decreased, and then which leads to substantial savings of the calculation time, accordingly it is quite effective for a fast implementaion of the definition field.

C.2 SQR_2

Let us consider substitutions $b_0 = a_0$ and $b_1 = a_1$ into Eq.(8a), then A^2 can be calculated as follows.

$$\begin{aligned} A^2 &= (a_0^2 - a_1^2) + 2a_0a_1\omega \\ &= (a_0 + a_1)(a_0 - a_1) + (a_0a_1 + a_0a_1)\omega. \end{aligned} \tag{9}$$

C.3 INV_2

In each field $F_{p^2}(1)\sim(5)$, Itoh-Tsujii algorithm(ITA)[21], which is an inversion algorithm using Frobenius mapping effectively, is adopted for a fast implementation of INV_2. In this case, ITA can be expressed by

$$n = AA^p, \quad n \in F_p, \tag{10a}$$
$$A^{-1} = n^{-1}A^p, \tag{10b}$$

where A is an arbitrary non-zero element in F_{p^2} and n is its *norm*[19]. Inversion n^{-1}, that is INV_1, can be implemented by using extended Euclid algorithm(EEA)[20].

Now, let us consider ITA in the case of $F_{p^2}(1)$. At first, Frobenius mapping $\phi(A) = A^p$ is given by Eq.(11). It is noted that $\omega + \omega^p = 0$ is hold from a relation between the coefficient and the zeros of the modular polynomial.

$$\phi(A) = A^p = a_0^p + a_1^p\omega^p = a_0 - a_1\omega. \tag{11}$$

Substituting Eq.(6a) and Eq.(11) into Eqs.(10), respectively, we can calculate A^{-1} as follows.

$$n = a_0^2 - a_1^2\omega^2 = a_0^2 + a_1^2, \tag{12a}$$
$$A^{-1} = n^{-1}(a_0 - a_1\omega). \tag{12b}$$

D. *Efficient base field F_{p^2} for fast implementation*

Table VI shows calculation costs of MUL_2, SQR_2, and INV_2 in each base field of $F_{p^2}(1)\sim(5)$. It is noted that the number of SUB_1's is counted into that of ADD_1's because their calculation times are almost the same to each other on a generally-used processor, such as Celeron. In addition, twice of $a \in F_p$, that is $2a$, is implemented by using an addition like $a + a$, which can be seen in Eq.(9).

As described in Section2-1.3, ECC needs several scalar multiplications. Accordingly, a lot of arithmetic operations in the definition field are needed. Since this paper deals with extension field F_{p^8} especially, its base field F_{p^2} for successive extension should have fast basic arithmetic operations, especially MUL_2, SQR_2, and INV_2. Since a MUL_1 needs more calculation time than an ADD_1 on a generally-used processor as mentioned in Section5, it can be said that $F_{p^2}(1)$, (3), and (5) are superior to $F_{p^2}(2)$ and (4), which is found by comparing the number of MUL_1's needed for each SQR_2 implementation on Table VI.

IV. Efficient successive extension for fast implementation of definition field

Based on the calculation costs tabulated in Table VI, in this section we discuss efficient successive extension for a fast implementation of the definition field of ECC, which will lead to a fast implementation of ECC.

First, let us consider a general extension of base field F_q to extension field F_{q^2}. Then, a fast implementation method which adopts a polynomial basis and a binomial as the modular polynomial is proposed with comparing to the previous work[10] which adopts a normal basis and a trinomial. In addition, calculation costs of F_{q^2}-arithmetic operations are evaluated by counting the numbers of arithmetic operations in the base field needed for the implementation of arithmetic operations in the extension field. After that, we discuss the efficiency of these two extension methods. Throughout this section, let q be p^m.

TABLE VI

THE NUMBERS OF ADD_1'S, MUL_1'S, AND INV_1'S NEEDED FOR THE IMPLEMENTATIONS OF MUL_2, SQR_2, AND INV_2

	MUL_2		SQR_2		INV_2		
	ADD_1	MUL_1	ADD_1	MUL_1	ADD_1	MUL_1	INV_1
$F_{p^2}(1)$	5	3	3	2	2	4	1
$F_{p^2}(2)$	6	3	2	3	3	4	1
$F_{p^2}(3)$	4	3	4	2	2	4	1
$F_{p^2}(4)$	4	3	3	3	2	4	1
$F_{p^2}(5)$	4	3	4	2	2	4	1

A. Modular polynomial, irreducibility, and basis

Irreducible polynomials of degree 2 over F_q are classified roughly into two types as follows, where $u, v_1, v_0 \in F_q$.

$$x^2 + u, \tag{13a}$$

$$x^2 + v_1 x + v_0, \ v_1 \neq 0. \tag{13b}$$

Irreducibility of polynomial of degree 2 can be tested by using its discriminant D as follows:

$$D^{\frac{q-1}{2}} = \begin{cases} 0, 1 & ; \text{reducible} \\ -1 & ; \text{irreducible} \end{cases}. \tag{14}$$

Discriminant D for the polynomials shown in Eqs.(13) are respectively given by

$$D = -4u, \tag{15a}$$

$$D = v_1^2 - 4v_0. \tag{15b}$$

Let us call extension methods using binomial Eq.(13a) and trinomial Eq.(13b) as the modular polynomial OEX and NEX [18], respectively. From Eq.(14) and Eqs.(15), the correspondence between the modular polynomial and its irreducible condition is given in Table VII. In addition, the bases adopted in OEX and NEX are shown in Table VIII, where τ is a zero of each modular polynomial.

TABLE VII

CORRESPONDENCE BETWEEN THE MODULAR POLYNOMIAL AND ITS IRREDUCIBLE CONDITION

Method	Modular polynomial	Irreducible condition
OEX	$x^2 + u$	$(-u)^{(q-1)/2} = -1$
NEX	$x^2 + v_1 x + v_0$	$(v_1^2 - 4v_0)^{(q-1)/2} = -1$

TABLE VIII

BASIS ADOPTED IN OEX AND NEX

Method	Basis	Basis type
OEX	$\{1, \tau\}$	polynomial basis
NEX	$\{\tau, \tau^q\}$	normal basis

B. MUL_{2m}

Let us consider arbitrary elements $A, B \in F_q$ for each extension method OEX and NEX as follows:

$$A = a_0 + a_1\tau, \ \ a_0, a_1 \in F_q \tag{16a}$$

$$B = b_0 + b_1\tau, \ \ b_0, b_1 \in F_q \tag{16b}$$

$$A = a_1\tau + a_q\tau^q, \ \ a_1, a_q \in F_q \tag{17a}$$

$$B = b_1\tau + b_q\tau^q, \ \ b_1, b_q \in F_q \tag{17b}$$

In the case of OEX, multiplication of A and B, that is MUL_{2m}, can be calculated in the same manner of Eq.(8b),

$$\begin{aligned} AB &= a_0b_0 + (a_0b_1 + a_1b_0)\tau + a_1b_1\tau^2 \\ &= (a_0b_0 - ua_1b_1) + (a_0b_1 + a_1b_0)\tau, \end{aligned} \tag{18a}$$

$$a_0b_1 + a_1b_0 = (a_0 + a_1)(b_0 + b_1) - a_0b_0 - a_1b_1. \tag{18b}$$

On the other hand, in the case of NEX, MUL_{2m} can be calculated as follows. We can see its details in NTT[10].

$$t = (a_1 - a_q)(b_1 - b_q)\frac{v_0}{v_1}, \tag{19a}$$

$$AB = (t - a_1b_1v_1)\tau + (t - a_qb_qv_1)\tau^q. \tag{19b}$$

C. SQR_{2m}

In the case of OEX, substituting $b_0 = a_0$ and $b_1 = a_1$ into Eqs.(18), SQR_{2m} can be calculated by

$$A^2 = (a_0^2 - ua_1^2) + 2a_0a_1\tau. \tag{20a}$$

It seems that it is the fastest to calculate A^2 by Eq.(20a), however, let us consider to calculate $2a_0a_1$, which is second term of the RHS of Eq.(20a), by using KA as follows:

$$2a_0a_1 = (a_0 + a_1)^2 - a_0^2 - a_1^2. \tag{20b}$$

From Table VI, we can easily find that a square operation can be implemented faster than a multiplication. Therefore, $2a_0a_1$ should be calculated by using square operation as seen in Eq.(20b). It is noted that terms a_0^2 and a_1^2 have been already calculated in the parenthesis of Eq.(20a).

In the case of NEX, substituting $b_1 = a_1$ and $b_2 = a_2$ into Eqs.(19), SQR_{2m} can be calculated by

$$t = (a_1 - a_q)^2 \frac{v_0}{v_1}, \quad (21a)$$

$$A^2 = (t - a_1^2 v_1)\tau + (t - a_q^2 v_1)\tau^q. \quad (21b)$$

D. Frobenius mapping with respect to F_q

Eq.(11) shows Frobenius mapping of arbitrary element in F_{p^2} with respect to subfield F_p. In this section's case, we must consider Frobenius mapping of arbitrary element $A \in F_{q^2}$ with respect to base field F_q. In the case of OEX, this mapping is given as follows:

$$\phi(A) = A^q = a_0^q + a_1^q \tau^q = a_0 - a_1\tau, \quad (22)$$

in which a relation $\tau + \tau^q = 0$ is used. In the case of NEX, ϕ needs no arithmetic operations because the adopted basis is a normal basis, and this mapping is given as follows.

$$\phi(A) = A^q = a_1^q \tau^q + a_q^q \tau^{q^2} = a_q\tau + a_1\tau^q \quad (23)$$

E. INV_{2m}

As the same of INV_2 introduced in Section3-3.3, both OEX and NEX adopt ITA for the inversion, that is INV_{2m}. If we denote a non-zero element of F_{q^2} by A, then ITA can be expressed as follows:

$$n = AA^q, \quad (24a)$$

$$A^{-1} = n^{-1}A^q. \quad (24b)$$

In the case of OEX, the above equations can be developed by substituting Eq.(16a) and Eq.(22) into Eqs.(24) and using relation $\tau^2 = -u$,

$$n = a_0^2 - a_1^2\tau^2 = a_0^2 + ua_1^2, \quad (25a)$$

$$A^{-1} = n^{-1}(a_0 - a_1\tau), \quad (25b)$$

where n becomes a non-zero element of F_q.

In the case of NEX, Eqs.(24) can be developed by substituting Eq.(17a) and Eq.(23) into Eqs.(24) and using relation $\tau + \tau^q = -v_1$,

$$n = (\tau + \tau^q)\{-(a_1 - a_q)^2 \frac{v_0}{v_1} - a_1 a_q v_1\}$$

$$= a_1 a_q v_1^2 + (a_1 - a_q)^2 v_0, \quad (26a)$$

$$A^{-1} = n^{-1}(a_q\tau + a_1\tau^q). \quad (26b)$$

F. Comparison between OEX and NEX

The calculation costs of F_{q^2}-arithmetic operations, such as MUL_{2m}, are shown in Table IX corresponding to each implementation method. Every data is evaluated by counting the numbers of F_q-arithmetic operations and the others, such as u times, v_1 times, and so on. The reason why the numbers of u times, v_1 times, and such operations are individually counted is that the choice of coefficients u, v_1, v_0 of modular polynomial Eqs.(13) affects the performance of F_{q^2}-arithmetics. To choose them from the basis of F_q, such as a zero of the modular polynomial of F_q, is the best for a fast implementation of F_{q^2}-arithmetic operations, where it is noted that modular polynomials Eqs.(13) must be defined over F_q. In order to see its efficiency, let us consider a case that base field F_q is $F_{p^2}(1)$ introduced in Section3 and coefficient u is ω chosen from basis of $F_{p^2}(1)$ shown in Table V. In this case, u times, that is ω times, for arbitrary element A represented by Eq.(6a) can be calculated as follows:

$$\omega A = a_0\omega + a_1\omega^2 = -a_1 + a_0\omega, \quad (27)$$

where it is noted that $\omega^2 = -1$ is hold. Therefore, the heavy arithmetic operations, such as a multiplication, are not required. Especially for NEX, coefficient v_1 should be chosen to ± 1 because the number of v_1 times is the largest among those of such operations as seen in Table IX.

As described in the previous paragraph, if we can choose a zero of the modular polynomial of F_q as coefficients u and v_2, supposing $v_1 = \pm 1$, u times and such operations can be fast implemented as seen in Eq.(27). For such choices, conditions shown in Table VII must be satisfied in each implementation method. In the case of OEX, let τ be a zero of modular polynomial $x^2 + u$, accordingly τ is contained in F_{q^2}, then $x^2 - \tau$ is irreducible if $q \equiv 1 \pmod 4$. And then, let θ be a zero of $x^2 - \tau$, $x^2 - \theta$ is irreducible over F_{q^4} unconditionally. In the same way, let γ be a zero of $x^2 - \theta$, then $x^2 - \gamma$ is irreducible over F_{q^8}. And so forth, we can choose a zero of each modular polynomial as coefficient u. On the other hand, since NEX does not have such a property, a zero of the modular polynomial cannot be always chosen as coefficient v_2 even whether $v_1 = \pm 1$ or not. If a zero of the modular polynomial cannot be chosen as the coefficient, let ω be a zero of the modular polynomial, it is desirable that $\omega \pm 1$ or $\omega \pm 2$ are chosen as the coefficient, because $\omega \pm 1$ times and the others for arbitrary element A can be also fast implemented as follows:

$$(\omega \pm 1)A = (a_0\omega + a_1\omega^2) \pm (a_0 + a_1\omega)$$

$$= (\pm a_0 - a_1) + (a_0 \pm a_1)\omega. \quad (28)$$

Consequently, ω times, $\omega \pm 1$ times, and $\omega \pm 2$ times can be implemented by using only additions or subtractions. By evaluating calculation costs of these operations with the

TABLE IX

THE NUMBERS OF ADD_m'S, MUL_m'S, SQR_m'S, INV_m'S, u TIMES, v_1 TIMES, v_0 TIMES, v_0/v_1 TIMES, AND v_1^2 TIMES, NEEDED FOR THE IMPLEMENTATIONS OF MUL_{2m}, SQR_{2m}, FRO_{2m}, AND INV_{2m}

Operation	Method	ADD_m	MUL_m	SQR_m	INV_m	u	v_1	v_0	v_0/v_1	v_1^2
MUL_{2m}	OEX	5	3	-	-	1	-	-	-	-
	NEX	4	3	-	-	-	2	-	1	-
SQR_{2m}	OEX	4	-	3	-	1	-	-	-	-
	NEX	3	-	3	-	-	2	-	1	-
FRO_{2m}	OEX	1	-	-	-	-	-	-	-	-
	NEX	-	-	-	-	-	-	-	-	-
INV_{2m}	OEX	2	2	2	1	1	-	-	-	-
	NEX	2	3	1	1	-	-	1	-	1

number of ADD_1's, the results are given in Table X, where ω is a zero of each modular polynomial seen in Table III. In any base field F_{p^2}, a few times of ADD_1 are needed, however, such operations will scarcely affect a fast implementation.

TABLE X

THE NUMBER OF ADD_1'S NEEDED FOR THE IMPLEMENTATIONS OF ω TIMES, $\omega \pm 1$ TIMES, AND $\omega \pm 2$ TIMES IN EACH F_{p^2} (1)~(5)

	$\omega-2$	$\omega-1$	ω	$\omega+1$	$\omega+2$
$F_{p^2}(1)$	5	3	1	**2**	**4**
$F_{p^2}(2)$	4	2	1	2	3
$F_{p^2}(3)$	6	4	**2**	1	**3**
$F_{p^2}(4)$	4	2	**2**	3	4
$F_{p^2}(5)$	4	2	1	**3**	5

* Bold faces are used in Table XI.

Based on the above discussions, the following remarks are derived with comparing OEX and NEX on Table IX.

- With respect to MUL_{2m} and SQR_{2m}, NEX is superior.
- With respect to INV_{2m}, OEX is superior.

The former can be easily found by comparing the numbers of ADD_m's, and the latter by comparing on MUL_m and SQR_m with noting that a square operation can be implemented faster than a multiplication in the extension field, where such a property can be seen in Table VI.

V. Fast implementation of ECC using elliptic curve with prime order on micro controller

In this section, let us choose two prime numbers as the characteristic and then consider definition field F_{p^8} by using successive extension methods OEX and NEX on each base field of F_{p^2}(1)~(5). For each definition field, the calculation costs of F_{p^8}-arithmetic operations, such as MUL_8, are evaluated by counting the numbers of F_p-arithmetic operations. Based on this evaluation, some definition fields that are especially expected to carry out arithmetic operations fast on a generally-used processor are selected. After that, these definition fields are explicitly implemented on Intel Celeron(400MHz) processor by using C language, then the calculation times of F_{p^8}-arithmetic operations are timed. Finally, based on this calculation times, two definition fields which will be especially suitable for a fast implementation of ECC are selected and then implemented on Toshiba micro controller TMP94C251(20MHz), which is 32-bit micro controller, by using C language. After that, the calculation times of ECA, ECD, and scalar multiplication are timed, then a comparison between the proposed method and the previous works is given.

A. Fast implementation of definition field F_{p^8}

Let us consider two prime numbers $2^{31}-1$ and $2^{29}-3$ as the characteristic. Accordingly, we can consider 20 varieties of definition field F_{p^8} in combination with successive extension methods OEX, NEX and base fields F_{p^2}(1)~(5). In this paper, 10 definition fields were selected from them, where the parameters of these fields were chosen as shown in Table XI with paying attention to a fast implementation. In Table XI, polynomials $f_{2\to4}(x)$ and $f_{4\to8}(x)$ denote the modular polynomial of F_{p^4} over F_{p^2} and that of F_{p^8} over F_{p^4}, and which are used for successive extensions of $F_{p^2} \to F_{p^4}$ and $F_{p^4} \to F_{p^8}$, respectively. In addition, ω and τ are a zero of the modular polynomial of F_{p^2} and that of $f_{2\to4}(x)$, respectively. For example, in the case of F_{p^8}(1-A), ω and τ are zeros of x^2+1 and $x^2-(\omega+2)$.

According to Section4-6, coefficients of $f_{4\to8}(x)$ shown in Table XI are chosen to $v_1=-1$ and $u, v_0 = \pm\tau, -(\tau\pm1)$, where u, v_1, v_0 are defined in Eqs.(13). Table XII shows the

TABLE XI

ENTRIES OF DEFINITION FIELD F_{p^8} CORRESPONDING TO BASE FIELD F_{p^2} AND SUCCESSIVE EXTENSION METHOD WITH THE MODULAR POLYNOMIAL

Characteristic p	Base field	Extension method	$f_{2\to4}(x)$ †	$f_{4\to8}(x)$ †	Entry No.
$2^{31}-1$	$F_{p^2}(1)$	OEX, OEX	$x^2-(\omega+2)$ ††	$x^2-\tau$ ††	F_{p^8}(1-A)
	$F_{p^2}(1)$	NEX, NEX	$x^2-x-(\omega+1)$	$x^2-x+\tau$	F_{p^8}(1-B)
	$F_{p^2}(4)$	OEX, OEX	$x^2-\omega$	$x^2-\tau$	F_{p^8}(4-A)
	$F_{p^2}(4)$	NEX, NEX	$x^2-x+\omega$	$x^2-x-(\tau+1)$	F_{p^8}(4-B)
$2^{29}-3$	$F_{p^2}(2)$	OEX, OEX	$x^2-\omega$	$x^2-\tau$	F_{p^8}(2-A)
	$F_{p^2}(2)$	NEX, NEX	$x^2-x-\omega$	$x^2-x-(\tau+1)$	F_{p^8}(2-B)
	$F_{p^2}(3)$	OEX, OEX	$x^2-(\omega+2)$	$x^2-\tau$	F_{p^8}(3-A)
	$F_{p^2}(3)$	NEX, NEX	$x^2-x+\omega$	$x^2-x-(\tau+1)$	F_{p^8}(3-B)
	$F_{p^2}(5)$	OEX, OEX	$x^2-(\omega+1)$	$x^2-\tau$	F_{p^8}(5-A)
	$F_{p^2}(5)$	NEX, NEX	$x^2-x-\omega$	$x^2-x-(\tau+1)$	F_{p^8}(5-B)

† $f_{2\to4}(x)$ and $f_{4\to8}(x)$ denote modular polynomial of F_{p^4} over F_{p^2} and that of F_{p^8} over F_{p^4}, respectively.

†† ω and τ are a zero of modular polynomial of base field F_{p^2} and a zero of $f_{2\to4}(x)$, respectively.

calculation costs of τ times, $\tau\pm1$ times, and $\tau\pm2$ times, which are evaluated in the same manner of Table X and the details are shown in Appendix.B. The calculation cost

TABLE XII

THE NUMBER OF ADD_1'S NEEDED FOR THE IMPLEMENTATION OF τ TIMES, $\tau\pm1$ TIMES, AND $\tau\pm2$ TIMES IN EACH F_{p^8}(1-A)~(5-B)

	$\tau-2$	$\tau-1$	τ	$\tau+1$	$\tau+2$
F_{p^8}(1-A)	9	6	**4**	6	9
F_{p^8}(1-B)	10	6	**6**	9	13
F_{p^8}(4-A)	10	6	**2**	6	10
F_{p^8}(4-B)	10	6	6	**10**	14
F_{p^8}(2-A)	9	5	**1**	5	9
F_{p^8}(2-B)	9	5	5	**9**	13
F_{p^8}(3-A)	9	5	**3**	5	9
F_{p^8}(3-B)	10	6	6	**10**	14
F_{p^8}(5-A)	9	5	**3**	5	9
F_{p^8}(5-B)	9	5	5	**9**	13

* **Bold faces are used in Table XI.**

of $-\tau$ times should be also evaluated, however, difference of sign of τ makes a trifling change between addition and subtraction as seen in Eq.(18a). Since it is supposed that the calculation costs of addition and subtraction are almost the same to each other as described in Section3-4, it is also considered that the calculation costs of $\pm\tau$ times are the same to each other. By the way, it is noted that all of coefficients of $f_{2\to4}(x)$ and $f_{4\to8}(x)$ tabulated in Table XI are chosen not only so as to be implemented fast but also so as to be irreducible as the modular polynomial.

The calculation costs of MUL_8, SQR_8, and INV_8 in each definition field F_{p^8}(1-A) ~ (5-B) can be evaluated from Table VI, IX, X, and XII by counting the numbers of F_p-arithmetic operations as shown in Table XIII. The numbers in parenthesis are of ADD_1's, MUL_1's, and INV_1's from the left, respectively. From the table, we can easily find

TABLE XIII

THE NUMBERS OF ADD_1'S, MUL_1'S, AND INV_1'S NEEDED FOR THE IMPLEMENTATION OF MUL_8, SQR_8, AND INV_8

	MUL_8	SQR_8	INV_8
F_{p^8}(1-A)	(111, 27, 0)††	(83, 18, 0)	(138, 44, 1)
F_{p^8}(1-B)	(97, 27, 0)	**(69,18,0)**†	(132, 48, 1)
F_{p^8}(4-A)	(94, 27, 0)	(75, 27, 0)	(118, 52, 1)
F_{p^8}(4-B)	(92, 27, 0)	(73, 27, 0)	(124, 52, 1)
F_{p^8}(2-A)	(108, 27, 0)	(62, 27, 0)	(121, 52, 1)
F_{p^8}(2-B)	(106, 27, 0)	(60, 27, 0)	(139, 52, 1)
F_{p^8}(3-A)	(98, 27, 0)	(88, 18, 0)	**(132,44,1)**
F_{p^8}(3-B)	(92, 27, 0)	(82, 18, 0)	(128, 48, 1)
F_{p^8}(5-A)	(98, 27, 0)	(88, 18, 0)	**(132,44,1)**
F_{p^8}(5-B)	**(88,27,0)**	(78, 18, 0)	(122, 48, 1)

† **Bold faces are especially expected to be fast implemented.**

†† The numbers in parenthesis are of ADD_1's, MUL_1's, and INV_1's from the left, respectively.

that NEX is superior than OEX with respect to MUL_8 and SQR_8, on the other hand, OEX is superior than NEX with respect to INV_8, which has been also concluded in

Section4-6. By the way, though the numbers of ADD_1's are widely distributed as compared to those of MUL_1's, it is because the calculation costs tabulated in Table X and XII are distributed. Let us compare the number of ADD_1's needed for the implementation of MUL_8 in each F_{p^8}(1-A) and F_{p^8}(5-B) on Table XIII, for example. The former needs 111 ADD_1's but the latter needs 88 ADD_1's only. As seen in Table X, the choise of the coefficients of $f_{2\to 4}(x)$ is mainly causing this difference. To be more precise, the latter selected $\omega+2$ as the coefficient, that is not the best, but the former selected ω, that is the best. It depends on the irreducibility whether we can select the best coefficients for the modular polynomial or not.

Since our purpose is to achieve a fast implementation of ECC, now let us consider ECA and ECD implementations. These implementations need several F_{p^8}-arithmetics as shown in Eqs.(3), and which are concluded in Table XIV.

TABLE XIV

THE NUMBERS OF ADD_8'S, MUL_8'S, SQR_8'S, AND INV_8'S NEEDED FOR THE IMPLEMENTATION OF ECA AND ECD

	ADD_8	MUL_8	SQR_8	INV_8
ECA	6	2	1	1
ECD	8	2	2	1

It is noted that $3x_1^2$ and $2y_1$ in Eq.(3a) are implemented by using $x_1^2+x_1^2+x_1^2$ and y_1+y_1, respectively. From Table XIII and Table XIV, the calculation costs of ECA and ECD can be evaluated by using the numbers of F_p-arithmetic operations as shown in Table XV. Based on the results shown in Table XV, let us select five definition fields as follows, which are especially suitable for a fast implementation of ECC.

$$F_{p^8}(1\text{-A}),\ F_{p^8}(1\text{-B}),\ F_{p^8}(3\text{-A}),\ F_{p^8}(3\text{-B}),\ F_{p^8}(5\text{-B}).$$

For the above definition fields, F_{p^8}-arithmetic operatations, ECA, and ECD are explicitly implemented on Intel Celeron processor by using C language, after that, the averages of the calculation times of these operations are timed. The results are shown in Table XVI. On this table, we can find a question why ECA calculation over F_{p^8}(1-A) is carried out faster than over F_{p^8}(5-B), though the number of ADD_1's needed for the implementation of ECA over F_{p^8}(1-A) is much smaller than that over F_{p^8}(5-B) as shown in Table XV. Its ansewr lies in the difference between the numbers of MUL_1's, that is 4 MUL_1's. To be more precise, one is the difference between the characteristics adopted in F_{p^8}(1-A) and F_{p^8}(5-B) and the other is the calculation time ratio of a MUL_1 to an ADD_1 on Intel Celeron processor, which is denoted by $R_{M/A}$ in the following.

TABLE XV

THE NUMBERS OF ADD_1'S, MUL_1'S, AND INV_1'S NEEDED FOR THE IMPLEMENTATION OF ECA AND ECD

	ECA	ECD
F_{p^8}(1-A)	(491, 116, 1)	(590, 134, 1)
F_{p^8}(1-B)	(443, 120, 1)	(528, 138, 1)
F_{p^8}(4-A)	(429, 133, 1)	(520, 160, 1)
F_{p^8}(4-B)	(429, 133, 1)	(518, 160, 1)
F_{p^8}(2-A)	(447, 133, 1)	(525, 160, 1)
F_{p^8}(2-B)	(459, 133, 1)	(535, 160, 1)
F_{p^8}(3-A)	(464, 116, 1)	(568, 134, 1)
F_{p^8}(3-B)	(442, 120, 1)	(540, 138, 1)
F_{p^8}(5-A)	(464, 116, 1)	(568, 134, 1)
F_{p^8}(5-B)	(424, 120, 1)	(518, 138, 1)

* The numbers in parenthesis are of ADD_1's, MUL_1's, and INV_1's from the left, respectively.

TABLE XVI

AVERAGE CALCULATION TIMES OF F_{p^8}-ARITHMETIC OPERATATIONS AND ECA, ECD

unit: μs

	MUL_8	SQR_8	INV_8	ECA	ECD
F_{p^8}(1-A)	1.51	1.08	3.42	8.00	9.05
F_{p^8}(1-B)	1.30	0.97	3.46	7.74	8.66
F_{p^8}(3-A)	2.04	1.56	4.44	10.9	12.5
F_{p^8}(3-B)	1.89	1.50	4.50	10.9	12.6
F_{p^8}(5-B)	1.95	1.51	4.55	10.5	12.2

* By using Intel Celeron(400MHz) processor.

In order to make sure of the former, that is the difference of the characteristic, let us consider the calculation flow of MUL_1 of two elements $x, y \in F_p$ as schematically shown in Fig.1. In this flow, n bits prime $2^n - s$ is considered as the characteristic. First, calculate product z of x and y as shown at (A), where $(x_0,\cdots,x_{n-1})$,$(y_0,\cdots,y_{n-1})$, and $(z_0,\cdots,z_{2n-2})$ in Fig.1 show the binary representations of x, y, and z, respectively. And then, modulo p operation can be carried out as shown at (B), that is, calculate upper digits $z_n,\cdots,z_{2n-2}$ times s, and then add it to lower digits $z_0,\cdots,z_{n-1}$. If the result is larger than p, then modulo p operation must be carried out again as shown at (C). If we consider the probability that modulo p operation must be carried out twice in a MUL_1 operation like at (C), then in the case of Mersenne prime $2^{31}-1$ as the characteristic the second modulo p operation is not needed but in the

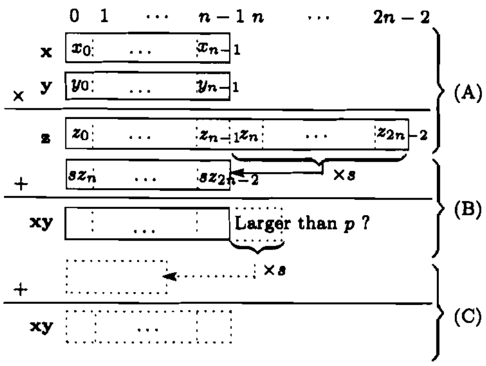

Fig. 1. Calculation flow of $\mathrm{MUL_1}$

case of pseudo Mersenne prime $2^{29}-3$ as characteristic it is sometimes needed. Next, in order to make sure of the latter, calculation time ratio $R_{M/A}$ was measured on Celeron and also TMP94C251, then the results were both about 7. Accordingly 4 $\mathrm{MUL_1}$'s are almost equivalent to 28 $\mathrm{ADD_1}$'s. Concludingly, we should choose a Mersenne prime as characteristic and the number of $\mathrm{MUL_1}$'s is prefered not to be large for a fast implementation of the definition field.

From the above discussion, the choise of characteristic and the number of $\mathrm{MUL_1}$'s have keys for a fast implementation. To decrease the number of $\mathrm{MUL_1}$'s, we should use OEX for successive extension from the discussion in Section4. On the other hand, to choose a Mersenne prime as the characteristic, we must adopt $F_{p^2}(1)$ or $F_{p^2}(4)$ as the base field from the discussion in Section3. Based on these remarks, it can be considered that F_{p^8}(1-A) is the best of all entries tabulated in Table XI. In the next subsection, scalar multiplication is implemented over F_{p^8}(1-A) on processors in particular.

B. Fast implementation of ECC using EPO

At first, EPO can be generated by using Danno, et al. algorithm[11] as follows.

EPO generation algorithm

Input: Characteristic p and extension degree 2^c.

Output: Elliptic curve with prime order $E'(F_{p^{2^c}})$.

Step1: Choose coefficients $a, b \in F_p$ of $E(x,0)$, which is defined by Eq.(2), at random. Then, test the irreducibility of $E(x,0)$ by using Hiramoto, et al.[22].

$$\begin{cases} \text{irreducible} & \rightarrow \text{ go to Step2,} \\ \text{reducible} & \rightarrow \text{ go to Step1.} \end{cases}$$

Step2: Compute order $\#E(F_p)$ of $E(x,y)$ over prime field F_p by using SEA algorithm[5].

Step3: Determine $\#E'(F_{p^{2^c}})$ by

$$\#E'(F_{p^{2^c}}) = p^{2^c} + 1 + D_{2^c}(t_1, p), \tag{29}$$

where t_1 and $D_{2^c}(t_1,p)$ are respectively given by

$$t_1 = p + 1 - \#E(F_p), \tag{30}$$

$$D_{2^c}(t_1,p) = \sum_{i=0}^{2^{c-1}} \frac{2^c}{2^c - i}\binom{2^c - i}{i}(-p)^i {t_1}^{2^c-2i}. \tag{31}$$

After that, test whether $\#E'(F_{p^{2^c}})$ is prime or not.

$$\begin{cases} \text{prime} & \rightarrow \text{ go to Step4,} \\ \text{composite} & \rightarrow \text{ go to Step1.} \end{cases}$$

Step4: Determine $E'(x,y) = y^2 - x^3 - aA^2x - bA^3$ by using a quadratic power non residue $A \in F_{p^{2^c}}$, consequently, we obtaine an EPO $E'(F_{p^{2^c}})$.

(End of algorithm)

Since this paper especially deals with a case of extension degree $m = 8$, exponent c is determined to be 3. In addition, let F_{p^8}(1-A) be the definition field and let a Mersenne prime $2^{31}-1$ be the characteristic, then we can obtain an EPO $E'(F_{p^8})$ by using the preceding algorithm. And, its defining equation and the order are given as follows:

$$E'(x,y) = y^2 - x^3 - 3\theta^2 x - 96\theta^3 = 0, \tag{32}$$

$$\begin{aligned}\#E'(F_{p^8}) = {} & 452312846898269724422641 1\backslash \\ & 796975436674620907503225 8\backslash \\ & 3598346095036305012010449, \end{aligned} \tag{33}$$

where the above order $\#E'(F_{p^8})$ is a 248 bits prime and θ is a zero of $f_{4\to 8}(x)$ of F_{p^8}(1-A), that is $x^2 - \tau$ as seen in Table XI. The reason why we used θ as a quadratic power non residue as seen in Eq.(32) is that such a zero θ is always a quadratic power non residue, which is one of useful properties of OEX, accordingly we need no calculation to obtain a quadratic power non residue. But, NEX does not have such a property, in other words, a zero of the modular polynomial in NEX is not always a quadratic power non residue. Accordingly, precomputation is needed to obtain a quadratic power non residue.

Next, according to Section2-1.2, we can obtain the following rational point P on elliptic curve $E'(F_{p^8})$.

$$P.x = [1,1,1,1,1,1,1,2], \tag{34a}$$

$$\begin{aligned}P.y = {} & [775160346, 1044745940, 1474766701, \\ & 1301969604, 532588157, 311746614, \\ & 1343273066, 581896508],\end{aligned} \tag{34b}$$

where $P.x$ and $P.y$ denote x-coordinate and y-coordinate, respectively, and let the basis for their vector representation be explicitly ordered as follows:

$$\{1, \omega, \tau, \omega\tau, \theta, \omega\theta, \tau\theta, \omega\tau\theta\}. \tag{35}$$

TABLE XVII

COMPARISON OF THE CALCULATION TIMES OF ECA, ECD, AND SCALAR MULTPLICATION

Processor (clock[MHz])	Extension degree m	Order $\#E$[bits]	ECA[μs]	ECD[μs]	Method and cf.	Scalar multi.[ms]
Celeron (400)	**8**	**248**	**8.00**	**9.05**	**NAF**	**2.12**
Celeron (400)	**8††**	**248**	**7.74**	**8.66**	**NAF**	**1.85**
Pentium II (400)	8	248†	9.66	11.14	–, –	–
Pentium II (400)	7	186	13.2	19.7	Frobenius, cf. [10]	1.95
Pentium II (400)	7	186	13.2	19.7	NAF, cf. [10]	3.89
Pentium II (400)	1	160	37.3	58.2	SlidingWindow, cf. [23]	9.10
Pentium II (400)	1	224	63.0	96.3	SlidingWindow, cf. [23]	20.6
Alpha 21164 (533)***	7	168	9.13	10.4	–, cf. [25]	2.02
Pentium/MMX (233)	6	186	44.8	52.4	–, cf. [26]	11.4
TMP94C251 (20)	**8**	**248**	**1005**	**1172**	**NAF**	**214**
TMP94C251 (20)	**8††**	**248**	**955**	**1127**	**NAF**	**209**
M16C (10)*	1	160	–	–	Binary, cf. [7]	480
MSP430 (1)**	1	128	15100	20700	Binary, cf. [8]	3400

Notice: **Bold faces** are the results of this paper. *: M16C, Mitsubishi 16-bit micro controller. With in-line assembly. **: MSP430, TI 16-bit micro controller. ***: With in-line assembly. $\dagger$: This is not prime order. $\dagger\dagger$: This is F_{p^8}(1-B).

It should be noted that $P.x$ and $P.y$ are elements in definition field F_{p^8}(1-A) constructed by successive extensions. Since order $\#E'(F_{p^8})$ is prime, we can see that $E'(F_{p^8})$ forms cyclic group by using P as the base point[19]. For this base point P, let us measure calculation time of a scalar multiplication under the following conditions:

- Let scalar k be chosen at random.
- Use C language and in-line assembler for programming.
- Use NAF method for scalar multiplication[16].

The programs are implemented on two processors: Intel Celeron(400MHz) and Toshiba TMP94C251(20MHz). And then, the results are concluded in Table XVII, where the calculation times of ECA, ECD, and the previous works are tabulated together for comparison.

As seen in Table XVII, order $\#E$ of the elliptic curve dealt with in this paper is not only prime but also the largest among all of entries, however, the calculation times of ECA and ECD are the fastest among them. The calculation time of a scalar multiplication of our implementation is not faster than the second entry which is using Frobenius method but about 30% faster than the other entries. Therefore, we can conclude that our proposed method has achieved fast implementation of ECC with a prime order. For future works, scalar multiplication using Frobenius method can be also applied to our proposed implementation method according to Iijima et al. research report[24], therefore, it is expected that our method can become much faster. The followings should be noticed:

- $\#E$ of the second entry is a 248-bit composite number.
- Implementations[7],[25] used in-line assembly only.
- Alpha 21164 is a 64-bit processor.
- The implementation on MSP430 is not enough secure because the order is smaller than 160 bits.

At last, the memory size of our implementaion which consists of F_{p^8}(1-A) arithmetics and various operations of ECC are concluded in Table XVIII.

TABLE XVIII

SIZE OF RAM/ROM NEEDED FOR THE IMPLEMENTAION OF F_{p^8}(1-A) ARITHMETICS AND VARIOUS OPERATIONS FOR ECC ON TMP94C251

unit:byte

Library	RAM	ROM
F_{p^8}(1-A) arithmetics	64	14K (10K)†
ECA, ECD	128	0.9K
scalar multiplication	656	4K
Total amount	848	18.9K (14.9K)†

$\dagger$: In parenthesis, the case that memory size is top priority.

Notice: K means $\times 10^3$.

As compared to a previous work[7], a little more memory is required because our programs are almost written in C language, partially using in-line assembly.

By the way, one of 8 ADD_8's needed for the implementation of ECD which can be seen in Table XIV is an addition $+a$ as seen in Eq.(3a). In our implementation using Eq.(32) as defining equation, since coefficient $a = 3\theta^2$, such an addition $+a$ is implemented by an addition $+3\tau$, where it is noted that $\theta^2 = \tau$. Moreover, this addition $+3\tau$ can be carried out by only adding 3 to the third coefficient of vector representation of addend, which can be easily understood from adopted basis Eq.(35). OEX has such a useful property but NEX does not.

VI. Conclusion

In this paper, we dealt with an elliptic curve which has a prime order and is defined over extension field $F_{p^{2c}}$. Then, in order to realize a fast software implementation of ECC adopting such an elliptic curve, a fast implementation method of definition field $F_{p^{2c}}$ especially F_{p^8} was proposed by using a technique called *successive extension.* First, five fast implementation methods for base field F_{p^2} were introduced. For each implemented base field, the calculation costs of F_{p^2}-arithmetic operations were evaluated by counting the numbers of F_p-arithmetic operations. Next, a successive extension method which adopts a polynomial basis and a binomial as the modular polynomial was proposed with comparing to a conventional method. Finally, we chose two prime numbers as characteristic p, and then considered definition field F_{p^8} with using five base fields and two successive extension methods. Among such definition fields, one definition field was selected and then implemented on Toshiba 32-bit micro controller TMP94C251(20MHz) by using C language. By evaluating the calculation times as compared to previous works, it was concluded that the proposed method could achieve a fast implementation of ECC with a prime order.

Appendix

A. Implementation of MUL_2, SQR_2, FRO_2, and INV_2

In this section, how to implement MUL_2, SQR_2, FRO_2, and INV_2 in each extension field $F_{p^2}(2)\sim(5)$ is shown.

A.1 MUL_2

$F_{p^2}(2)$: Let us consider arbitrary elements A, B shown in Eqs.(6), multiplication of A and B can be calculated by

$$AB = (a_0b_0 + 2a_1b_1) + (a_0b_1 + a_1b_0)\omega, \qquad (A.36)$$

where we used $\omega^2 = 2$. And then, the second term of the RHS of Eq.(A.36) must be calculated by Eq.(8b).

$F_{p^2}(3)$: Arbitrary elements A, B are represented by

$$A = a_1\omega + a_2\omega^2, \quad a_1, a_2 \in F_p, \qquad (A.37a)$$
$$B = b_1\omega + b_2\omega^2, \quad b_1, b_2 \in F_p. \qquad (A.37b)$$

In AOPF of extension degree 2, that is $F_{p^2}(3)$, multiplication of A and B is calculated as follows[9]:

$$t = (a_1 - a_2)(b_1 - b_2), \qquad (A.38a)$$
$$AB = (t - a_1b_1)\omega + (t - a_2b_2)\omega^2. \qquad (A.38b)$$

$F_{p^2}(4)$: Arbitrary elements A, B are represented by

$$A = a_1\omega + a_p\omega^p, \quad a_1, a_p \in F_p, \qquad (A.39a)$$
$$B = b_1\omega + b_p\omega^p, \quad b_1, b_p \in F_p. \qquad (A.39b)$$

In this case of NEF, multiplication of A and B is calculated as follows, and the details can be seen in NTT[10].

$$t = (a_1 - a_p)(b_1 - b_p), \qquad (A.40a)$$
$$AB = (t + a_1b_1)\omega + (t + a_pb_p)\omega^p. \qquad (A.40b)$$

The above equations can be written by Eqs.(19) in general, accordingly these are given by substituting $v_1 = v_0 = -1$, and $\tau = \omega$ into Eqs.(19), respectively. In addition, with comparing Eqs.(A.38) and Eqs.(A.40), we can easily find that only AOPF of extension degree 2 can also be considered as a special case of NEF.

$F_{p^2}(5)$: For arbitrary elements A, B represented by Eqs.(A.39), multiplication of A and B is calculated by

$$t = (a_1 - a_p)(b_1 - b_p), \qquad (A.41a)$$
$$AB = (a_1b_1 - t)\omega + (a_pb_p - t)\omega^p, \qquad (A.41b)$$

and these are given in the same way of previous $F_{p^2}(4)$.

A.2 SQR_2

$F_{p^2}(2)$: A^2 is calculated by the following equation, where $2a_1$ is once calculated and then used repeatedly.

$$A^2 = (a_0^2 + 2a_1^2) + 2a_0a_1\omega,$$
$$= \{a_0^2 + a_1(2a_1)\} + a_0(2a_1)\omega. \qquad (A.42)$$

$F_{p^2}(3)$: By substituting $b_1 = a_1, b_p = a_2$ into Eqs.(A.38), A^2 is calculated as follows:

$$t = (a_1 - a_2)^2, \qquad (A.43a)$$
$$A^2 = (t - a_1^2)\omega + (t - a_2^2)\omega^2$$
$$= \{a_2(a_2 - 2a_1)\}\omega + \{a_1(a_1 - 2a_2)\}\omega^2. \qquad (A.43b)$$

$F_{p^2}(4)$: By substituting $b_1 = a_1, b_p = a_p$ into Eqs.(A.40), A^2 is calculated as follows:

$$t = (a_1 - a_p)^2, \qquad (A.44a)$$
$$A^2 = (t + a_1^2)\omega + (t + a_p^2)\omega^p. \qquad (A.44b)$$

$F_{p^2}(5)$: By substituting $b_1 = a_1, b_p = a_p$ into Eqs.(A.41), A^2 is calculated as follows:

$$t = (a_1 - a_p)^2, \qquad (A.45a)$$
$$A^2 = (a_1^2 - t)\omega + (a_p^2 - t)\omega^p$$
$$= \{a_p(2a_1 - a_p)\}\omega + \{a_1(2a_p - a_1)\}\omega^p. \qquad (A.45b)$$

A.3 FRO_2

$F_{p^2}(2)$: It is the same of Eq.(11).

$F_{p^2}(3)$: Since pseudo polynomial basis $\{\omega, \omega^2\}$ is equal to $\{\omega, \omega^p\}$, that is normal basis, FRO_2 is given as follows:

$$\begin{aligned} A^p &= a_1^p\omega^p + a_2^p(\omega^2)^p \\ &= a_1\omega^p + a_2(\omega^p)^p \\ &= a_2\omega + a_1\omega^p \\ &= a_2\omega + a_1\omega^2. \end{aligned} \quad \text{(A.46)}$$

$F_{p^2}(4)$: Because of normal basis, by substituting $a_2 = a_p$ and $\omega^2 = \omega^p$ into Eq.(A.46), FRO_2 is given by

$$A^p = a_p\omega + a_1\omega^p. \quad \text{(A.47)}$$

$F_{p^2}(5)$: It is the same of Eq.(A.47).

A.4 INV_2

$F_{p^2}(2)$: As the same of Eqs.(12), inverse of non-zero element A represented by Eq.(6a) can be calculated by

$$n = a_0^2 - 2a_1^2, \quad \text{(A.48a)}$$
$$A^{-1} = n^{-1}(a_0 - a_1\omega). \quad \text{(A.48b)}$$

$F_{p^2}(3)$: By substituting Eq.(A.37a) and Eq.(A.46) into Eqs.(10), A^{-1} can be calculated as follows:

$$\begin{aligned} n &= \{-(a_1 - a_2)^2 - a_1a_2\}(\omega + \omega^2), \\ &= (a_1 - a_2)^2 + a_1a_2 \end{aligned} \quad \text{(A.49a)}$$
$$A^{-1} = n^{-1}(a_2\omega + a_1\omega^2). \quad \text{(A.49b)}$$

$F_{p^2}(4)$: By substituting Eq.(A.39a) and Eq.(A.47) into Eqs.(10), A^{-1} can be calculated as follows:

$$\begin{aligned} n &= \{-(a_1 - a_p)^2 - a_1a_p\}(\omega + \omega^p), \\ &= a_1a_p - (a_1 - a_p)^2 \end{aligned} \quad \text{(A.50a)}$$
$$A^{-1} = n^{-1}(a_p\omega + a_1\omega^p). \quad \text{(A.50b)}$$

$F_{p^2}(5)$: In the same manner of Eqs.(A.50), inverse can be calculated as follows:

$$n = a_1a_p + (a_1 - a_p)^2, \quad \text{(A.51a)}$$
$$A^{-1} = n^{-1}(a_p\omega + a_1\omega^p). \quad \text{(A.51b)}$$

B. The number of ADD_1's needed for the implementations of τ times and $\tau \pm 1$ times

In this section, it is introduced how to evaluate the calculation costs of τ times and $\tau \pm 1$ times in F_{p^8}(1-A), for example. For an arbitrary element $A \in F_{p^4}$ represented by Eq.(16a), τ times and $\tau \pm 1$ times are given as follows:

$$\tau A = a_0\tau + a_1\tau^2 = (\omega + 2)a_1 + a_0\tau, \quad \text{(A.52)}$$
$$\begin{aligned} (\tau \pm 1)A &= \{(\omega + 2)a_1 + a_0\tau\} \pm (a_0 + a_1\tau) \\ &= \{(\omega + 1)a_1 \pm (a_0 \pm a_1)\} + (a_0 \pm a_1)\tau, \end{aligned} \quad \text{(A.53)}$$

where τ is a zero of $f_{2\to4}(x) = x^2 - (\omega + 2)$ and then a relation $\tau^2 = \omega + 2$ is used. In the case of Eq.(A.52), it is enough to calculate one $\omega + 2$ times, therefore, its calculation cost is evaluated to 4 ADD_1's from Table X. In the same way, it is enough to calculate one $\omega + 1$ times and two ADD_2's in the case of Eq.(A.53), therefore, its calculation cost is evaluated to 6 ADD_1's.

C. Calculation costs of F_{p^8}-arithmetic operations

Consider the case of MUL_8 in F_{p^8}(1-A), for example. Since the adopted extension method is OEX, the calculation cost C of MUL_8 is given from Table IX as follows:

$$C = 5ADD_4 + 3MUL_4 + 4ADD_1, \quad \text{(A.54)}$$

where $4ADD_1$, for example, means 4 ADD_1's. The third term of the RHS of Eq.(A.54) is corresponding to one τ times, and which one τ times costs 4 ADD_1's from Table XII. And, the calculation cost of a MUL_4 is given as follows:

$$MUL_4 = 5ADD_2 + 3MUL_2 + 4ADD_1, \quad \text{(A.55)}$$

where the third term of the RHS of Eq.(A.55) is corresponding to one $\omega + 2$ times, see Table X. From Table VI, the calculation cost of a MUL_2 is given by

$$MUL_2 = 5ADD_1 + 3MUL_1. \quad \text{(A.56)}$$

Finally, with noting that $ADD_4 = 2ADD_2 = 4ADD_1$, cost C is obtained as follows:

$$C = 111ADD_1 + 27MUL_1. \quad \text{(A.57)}$$

REFERENCES

[1] A.Miyaji and M.Tatebayashi, "Method for Generating and Verifying Electronic Signatures and Privacy Communication Using Elliptic Curves," U.S. Patent 5442707(1995).

[2] A.Menezes and S.Vanstone, "The Implementation of Elliptic Curve Cryptosystems," LNCS 453, pp. 2-13(1990).

[3] D.Johnson and A.Menezes, "The Elliptic Curve Digital Signature Algorithm(ECDSA)," Technica Report CORR, vol.99-34, University of Waterloo(1999).

[4] R.Rivest, A.Shamir, and L.Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol.21, no.2(1978), pp.120-126.

[5] I.Blake, G.Seroussi, and N.Smart, Elliptic Curves in Cryptography, LNS 265, Cambridge University Press(1999).

[6] A.Menezes, Elliptic Curve Public Key Cryptosystems, Kluwer Academic Publishers(1993).

[7] T.Hasegawa, J.Nakajima, and M.Matsui, "A Practical Implementation of Elliptic Curve Cryptosystems over $GF(p)$ on a 16-bit Microcomputer," PKC'98, LNCS 1431(1998), pp.182-194.

[8] J.Guajardo, R.Blumel, U.Kritieger, and C.Paar, "Efficient Implementation of Elliptic Curve Cryptosystems on the TI MSP430x33x Family of Microcontrollers," PKC2001, LNCS 1992(2001), pp.365-382.

[9] A.Saito, T.Hiramoto, T.Danno, Y.Nogami, and Y.Morikawa, "Extension Fields by Using $(x^{m+1}-1)(x-1)$ as the Modulus for High-Speed Arithmetic," IEICE Technical Report, ISEC2000-119(2000), pp.129-134.

[10] T.Kobayashi, H.Morita, K.Kobayashi, and F.Hoshino, "Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adopt to Higher Characteristic," EUROCRYPT'99, LNCS 1592(1999), pp.176-189.

[11] T.Danno, Y.Nogami, and Y.Morikawa, "Conditions of Characteristic and Trace for Rank One Elliptic Curve Twisted over $F_{p^{2m}}$," Proc. The 24th Symposium on Information Theory and Its Applications(2001), pp.355-358.

[12] Y.Nogami and Y.Morikawa, "A Consideration on Elliptic Curve Cryptosystem Using Irreducible Polynomial of Degree 3 ," IEICE Technical Report, IT2001-44(2001), pp.7-12.

[13] P.Gaudry, F.Hess, and N.Smart,"Constructive and destructive facets of Weil descent on elliptic curves," Hewllet Packard Lab. Technical Report, HPL-2000-10(2000).

[14] S.Arita,"Weil descent of Elliptic Curves over Finite Fields of Characteristic Three," Proc. Crypto'98, LNCS 1462(1998), pp.472-485.

[15] T.Danno, A.Saito, Y,Nogami, and Y.Morikawa, "High-Speed Algorithm for Taking Square Root in an Extension Field by Using $(x^{m+1}-1)/(x-1)$ as the Modulus," IEICE Technical Report, IT2001-30(2001), pp.31-36.

[16] http://www.ieee.org/p1363

[17] D.B.Bailey and C.Paar, "Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms," Proc. Asiacrypt2000, LNCS 1976(2000), pp.248-258.

[18] T.Kobayashi, K.Aoki, and F.Hoshino, "OEF Using a Successive Extension," Proc. The 2000 Symposium on Cryptography and Information Security, no.B02(2000).

[19] R.Lidl and H.Niederreiter, Finite Field, Encyclopedia of Mathematics and Its Applications, Cambridge University Press(1984).

[20] D.E.Knuth, The Art of Computer Programming, Volume2: Seminumerical Algorithms, Addison-Wesley(1981).

[21] T.Itoh ans S.Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases," *Inf. and Comp.*, vol.78(1988), pp. 171-177.

[22] T.Hiramoto, Y.Nogami, and Y.Morikawa, "A Fast Algorithm to Test Irreducibility of Cubic Polynomial over $GF(P)$," IEICE Trans., vol.J84-A, no.5(2000), pp.633-641.

[23] Y.Sakai and K.Sakurai,"Efficient Scalar Multiplications on Elliptic Curves with Direct Computations of Several Doublings," IEICE Trans., vol.E84-4-A, no.1(2001), pp.120-129.

[24] T.Iijima, K.Matsuo, J.Chao, and S.Tsujii,"Construction of Frobenius Maps of Twists Elliptic Curves and Its Application to Elliptic Scalar Multiplication," Proc. The 2002 Symposium on Cryptography and Information Security(2002), pp.699-702.

[25] C.H.Lim and H.S.Hwang, "Fast Implementation of Elliptic Curve Arithmetic in $GF(p^n)$," *PKC 2000*, LNCS 1751(2000), pp. 405-421.

[26] D.B.Bailey and C.Paar, "Efficient Algorithm in Finite Field Extensions with Application in Elliptic Curve Cryptography," Journal of Cryptology, vol.14(2001), pp. 153-176.