Regular Paper

Efficient Construction of Elliptic Curves over Optimal Extension Field

Yuichi Futa† and Atsuko Miyaji††

Recently, Bailey and Paar proposed the Optimal Extension Field (OEF) which is defined over a base field with a computer's word size. Since the arithmetic in an OEF is relatively faster than that in $\boldsymbol{F}_{2^n}$, elliptic curves over an OEF would be more attractive when applied to a smart card, a personal computer, etc. However the definition of an OEF is rather strict since it is based on a general condition sufficient for fast arithmetic. In this paper, we extend the definition of an OEF such that it includes more extension fields with efficient arithmetic. Furthermore we construct elliptic curves over an OEF including our extended OEF efficiently by applying the SEA algorithm. Our implementation can count order of elliptic curves over 155-bit extended OEF and 160-bit OEF in 10.1 and 11.6 seconds on average on PentiumII 400 MHz (Linux-2.2.5), respectively.

1. Introduction

Koblitz[15] and Miller[20] proposed a method by which public key cryptosystems can be constructed on a group of points of an elliptic curve over a finite field instead of a finite field. If elliptic curve cryptosystems satisfy MOV-conditions[13),19)] and avoid p-divisible elliptic curves over $\boldsymbol{F}_{p^r}$[26),29),31)], then the only known attacks that are possible are the Pollard ρ-method[24)] and the Pohlig-Hellman method[23)]. Hence with current knowledge, we can construct elliptic curve cryptosystems over a smaller definition field than the discrete-logarithm-problem (DLP)-based cryptosystems like the ElGamal cryptosystems[7)] or the DSA[8)] and RSA cryptosystems[25)]. Elliptic curve cryptosystems with a 160-bit key are thus believed to have the same security as both the ElGamal cryptosystems and RSA with a 1,024-bit key. This is why elliptic curve cryptosystems have been discussed in ISO/IEC CD 14888-3, ISO/IEC DIS 11770-3, ANSI ASC X.9, X.9.62, and IEEE P1363[13)].

All known attacks can be avoided only by choosing elliptic curves with appropriate order[10),19),23),26),29),31)]. Therefore, order counting of elliptic curves is the main important factor for elliptic curve cryptosystems. Schoof proposed an order counting algorithm which runs in time polynomial[27)]. Although it is not so efficient, Elkies and Atkin improve Schoof's algorithm in time $O(\log^6 p)$, which is collectively called the SEA algorithm. Some results using the SEA algorithm are reported by Couveigne and Morain in the case of $\boldsymbol{F}_p$, and Lercier in the case of $\boldsymbol{F}_{2^n}$.

Recently, Bailey and Paar proposed the Optimal Extension Field (OEF) which is defined over a base field with a computer's word size. Since the arithmetic in an OEF is relatively faster than that in $\boldsymbol{F}_{2^n}$, elliptic curves over an OEF would be more attractive when applied to a smart card, a personal computer, etc. However the definition of an OEF is rather strict since it is based on a general condition sufficient for fast arithmetic. In fact some extension fields with fast arithmetic are excluded from an OEF.

As for construction methods of elliptic curves, there are three typical methods: the lifting method[30)], Complex Multiplication (CM) method[1)] and the order counting method. The lifting method constructs elliptic curves over $E/\boldsymbol{F}_{p^n}$ by lifting elliptic curves $E/\boldsymbol{F}_p$, therefore it can be implemented fast, and exponentiation can also be implemented fast by using the Frobenius map. However, in the case of an OEF, a rather large degree n is necessary since $\#E(\boldsymbol{F}_{p^n})$ is divisible by $\#E(\boldsymbol{F}_p)$, which increases the amount of computation for elliptic curve exponentiation. On the other hand, the CM method is also not suitable for construction of elliptic curves over an OEF since the definition field $\boldsymbol{F}_{p^n}$ is fixed. Therefore the order counting method is the most suitable for constructing elliptic curves over an OEF, because the method is aimed at searching elliptic curves over a fixed field, and does not need the larger extension degree. However it has not been re-

† Matsushita Electric Industrial Co., Ltd.
†† Japan Advanced Institute of Science and Technology

ported to construct elliptic curves over an OEF.

In this paper, we extend the definition of an OEF in such a way that it includes more extension fields with fast arithmetic. We also propose an inversion algorithm suitable for our extended OEF and implement arithmetic in our extended OEF. Furthermore, we apply the SEA algorithm to elliptic curves over an OEF including our extended OEF. We discuss efficiency of an OEF in SEA and estimate the amount of computation for the most dominant step of the SEA algorithm over OEFs. We also implement the SEA algorithm over OEFs (PentiumII 400 MHz, Linux-2.2.5 and Alpha21164A 600 MHz, Linux-2.2.1). The average running times for order counting of an elliptic curve over a 155-bit extended OEF and 160-bit OEF are 10.1 and 11.6 seconds on PentiumII 400 MHz (Linux-2.2.5), respectively.

This paper is organized as follows. Section 2 summarizes the known results of OEFs, elliptic curves and the SEA algorithm. Section 3 describes how the definition of an OEF is extended. Section 4 presents a new inversion algorithm suitable for our extended OEF. Section 5 presents implementation results of arithmetic in our extended OEF and compares the original OEF with our extended OEF. Section 6 describes the SEA algorithm over OEFs. Section 7 presents implementation results of the SEA algorithm over OEFs.

2. Known Results

2.1 Optimal Extension Field

In this section, we present a summary of the Optimal Extension Field (OEF)[2]. OEF is an extension field $\boldsymbol{F}_q\,(q = p^n)$ that satisfies the following conditions:

`OEF 1` $|p|$ is at most a computer's word size, where $|p|$ is the size of p.

`OEF 2` $p = 2^m \pm c$, where c is smaller than half of computer's word size.

`OEF 3` The generator polynomial is a binomial $G(x) = x^n - \omega$.

The arithmetic in OEF is usually done by polynomial basis, where $\boldsymbol{F}_{p^n}$ is identified by $\boldsymbol{F}_p[\alpha]/G(\alpha)$. `OEF 1` makes arithmetic in the extension field $\boldsymbol{F}_q$ efficient since arithmetic of $\boldsymbol{F}_p$ is done more efficiently. `OEF 2` and `OEF 3` reduce the computation time for modular reductions in $\boldsymbol{F}_p$ and in $\boldsymbol{F}_q$, respectively. The computation time for the inversion in OEF $\boldsymbol{F}_{p^3}$ is $1\mathcal{I}$ and $12\mathcal{M}$, where $\mathcal{I}(\ \mathcal{M})$ is the computation time for an inversion (a multiplication) in $\boldsymbol{F}_p$[14].

2.2 Elliptic Curves

Here we set an elliptic curve $E/\boldsymbol{F}_q(q = p^n,\ p \geq 5)$,

$$E : y^2 = x^3 + ax + b \quad (a, b \in \boldsymbol{F}_q), \qquad (1)$$

where $4a^3+27b^2 \neq 0$[30]. The $\boldsymbol{F}_q$-rational points are denoted by $E(\boldsymbol{F}_q)$,

$$E(\boldsymbol{F}_q) = \{(x, y) \in \boldsymbol{F}_q \times \boldsymbol{F}_q | \ y^2 = x^3 + ax + b\} \cup \{O\}. \qquad (2)$$

The j-invariant j of E is given by $j = 1728 \cdot 4a^3/(4a^3 + 27b^2)$.

Definition 1

(qth-power Frobenius map 30)) The qth-power Frobenius map ϕ_q is defined

$$\phi_q : E(\overline{\boldsymbol{F}}_q) \ni (x,\, y) \mapsto (x^q,\, y^q) \in E(\overline{\boldsymbol{F}}_q), \qquad (3)$$

where $\overline{\boldsymbol{F}}_q$ is an algebraic closure of $\boldsymbol{F}_q$.

As for the number of rational points, the following Hasse's theorem holds.

Theorem 1 (Hasse 30)) Let E be an elliptic curve over $\boldsymbol{F}_q$. Then, $\#E(\boldsymbol{F}_q)$ satisfies

$$|t = q + 1 - \#E(\boldsymbol{F}_q)| \leq 2\sqrt{q}, \qquad (4)$$

where t is the trace of ϕ_q.

From Eq. (4), counting $\#E(\boldsymbol{F}_q)$ is equivalent to computing the trace t. The Frobenius map satisfies the following characteristic equation,

$$\phi_q^2 - t\phi_q + q = 0. \qquad (5)$$

Here we denote a subgroup of l-torsion points by $E[l]$. The division polynomial $f_l(X)$, which is used in computing multiplication by l of a point on the elliptic curve E, is also important for the SEA algorithm. Here we define the division polynomial.

Division polynomial 27)

$$f_{2n}(X) = f_n(f_{n+2}f_{n-1}^2 - f_{n-2}f_{n+1}^2)/2$$

$$f_{2n+1}(X) = \begin{cases} f_{n+2}f_n^3 - f(X)^2 f_{n+1}^3 f_{n-1} & (n\text{: odd}) \\ f(X)^2 f_{n+2}f_n^3 - f_{n+1}^3 f_{n-1} & (n\text{: even}) \end{cases}$$

$$f_{-1}(X) = -1,\ f_0(X) = 0,\ f_1(X) = 1,$$

$$f_2(X) = 2,$$

$$f_3(X) = 3X^4 + 6aX^2 + 12bX - a^2,$$

$$f_4(X) = 4(X^6 + 5aX^4 + 20bX^3 - 5a^2X^2 - 4abX - 8b^2 - a^3), \qquad (6)$$

where $f(X) = X^3 + aX + b$.

The condition of $P \in E[l]$ is simply represented by using the division polynomial,

$$E[l] \ni P = (x, y) \Leftrightarrow f_l(x) = 0. \qquad (7)$$

2.3 Schoof's Algorithm

In this section, we summarize Schoof's algo-

rithm briefly[27]. From Eq. (5), any l-torsion point P (l: prime) satisfies the following equation,

$$(\phi_q^2 + q)P = t\phi_q P, \quad (8)$$

for $0 \leq {}^{\exists}t \leq l-1$. This means that we can compute $t \pmod l$ by restricting (5) to $E[l]$. Therefore, we first compute $t \bmod l$ for $\prod_{l \geq 2} l > 4\sqrt{q}$, next combine these values by the Chinese Remainder Theorem, and finally determine the exact value of t.

In order to compute (8), Schoof's algorithm uses the division polynomial (6). Therefore, the computation time of finding $t \bmod l$ is $O(l^5 \log^2 q)$ for each l. In total, the computation time of Schoof's algorithm is $O(\log^8 q)$.

2.4 SEA Algorithm

Here we summarize an improvement of Schoof's algorithm by Elkies and Atkin. Elkies uses a factor of $f_l(X)$ in order to find $t \bmod l$ when the eigenvalue of ϕ_q is in $\boldsymbol{Z}/l\boldsymbol{Z}$[9]. Compared with the degree of $f_l(X)$, $(l^2-1)/2$, the degree of a factor is bounded to at most $(l-1)/2$. Therefore the computation time of finding $t \bmod l$ is $O(l^3 \log^2 q)$ for each l. Atkin improves by restricting $t \bmod l$ in several values[28]. The computation time is also $O(l^3 \log^2 q)$ for each l. Furthermore Couveigne and Morain introduce the isogeny cycle method[6].

All such improvements are incorporated together in the SEA algorithm. The total computation time is $O(\log^6 q)$. The dominant steps of the SEA algorithm are as follows.

Polynomial exponentiation X^q
: Elkies' algorithm first factorizes the modular equation $\Phi_l(X)$ over $\boldsymbol{F}_q$ to judge whether the eigenvalue of ϕ_q is in $\boldsymbol{Z}/l\boldsymbol{Z}$ or not, which is done quickly[21),28]. Then, it is needed to compute the polynomial exponentiation $X^q \bmod \Phi_l(X)$. Furthermore, $X^q \bmod g_l(X)$ must also be determined, where $g_l(X)$ is a factor of $f_l(X)$.

Elliptic curve exponentiation
: Atkin's algorithm finds candidates of $t \pmod l$, but not the exact value of $t \pmod l$. Therefore, in the final stage the exact value of $t \pmod{\prod l}$ must be determined from the candidates. It is determined only through elliptic curve exponentiations, which is called the match and sort algorithm[16].

The polynomial exponentiation is the main dominant step in the SEA algorithm.

3. Extended OEF

In this section, we investigate how an OEF can be extended. In a sense the definition of an OEF is rather strict since arithmetic in an OEF is the most efficient as we have seen in Section 2.1. For example, a field $\boldsymbol{F}_{p^5}$ ($p = 2^{31}-1$) does not satisfy the conditions required to be an OEF since a binomial generator polynomial does not exist. Obviously a field $\boldsymbol{F}_{p^5}$ can offer the most efficient arithmetic in a 32-bit CPU of roughly 160-bit fields. Therefore it is meaningful to extend the definition of an OEF when the increase in cost for an arithmetic is negligible.

3.1 OEF Conditions

We compare the arithmetic under two conditions `OEF 2` and `OEF 3`. The computation amount of arithmetic in an OEF is mainly determined by `OEF 2`. So we leave the condition `OEF 2` unchanged, and extend the definition of an OEF as follows:

`EXOEF 3` The generator polynomial is a binomial or a trinomial $G(x) = x^n - \alpha x - \beta$.

The following efficient fields become available under the definition of an extended OEF:

1. In the case of $\boldsymbol{F}_{p^5}$ ($p = 2^{31}-1$), a binomial generator does not exist, but a trinomial generator $G(x) = x^5 - x - 8$ does.
2. In the case of $\boldsymbol{F}_{p^3}$ ($p = 2^{64}-59,\ 2^{64}-83$), $\boldsymbol{F}_{p^5}$ ($p = 2^{32}-17,\ 2^{32}-99$), $\boldsymbol{F}_{p^7}$ ($p = 2^{32}-5,\ 2^{32}-65$) etc., a binomial generator does not exist, but a trinomial generator does.
3. In the case of $\boldsymbol{F}_{p^n}$ ($(p,n) = (2^8-5, 23), (2^8-15, 29)$), which is suitable for a smart card with 8-bit CPU, a binomial generator does not exist.

Presently, there has not been any report on an attack on elliptic curve cryptosystems that depend on the form of the defined field, but that depend on the number of rational points of elliptic curves[19),26),29]. Hence, an elliptic curve over an extended OEF would be as secure as one over the original OEF, and would be more efficient to implement.

3.2 A Trinomial Generator

In this section, we compare the arithmetic in an OEF with a binomial generator to that in an extended OEF with a trinomial generator. Obviously the amount of computation necessary for addition and subtraction in an OEF is the same in spite of a generator. So multiplication and inversion in an OEF are essential in order to compare the efficiency of the generators. We discuss the computation time of multiplication in OEFs by showing examples of $x^5 - 11$ and

$x^5 - x - 8$, each of which is a generator of about 160-bit field in a 32-bit base field.

For elements $\sum_{i=0}^{4} x_i x^i$, $\sum_{i=0}^{4} y_i x^i$ in the definition field, multiplication is represented as follows.

$$\sum_{i=0}^{4} x_i x^i \times \sum_{i=0}^{4} y_i x^i = \sum_{i=0}^{8} z_i x^i \left(= \sum_{i=0}^{4} z_i x^i + \sum_{i=0}^{3} z_i x^5 x_i \right). \quad (9)$$

Hereafter we discuss each modular reduction of $\sum_{i=0}^{8} z_i x^i$ to $x^5 - 11$ and $x^5 - x - 8$. The computation time for modular reduction is estimated as follows:

(1) $x^5 - 11$
4 multiplications by 11 and 4 additions in the base field[2],

(2) $x^5 - x - 8$
4 multiplications by 8 and $2 \times 4 = 8$ additions in the base field.

In view of the number of necessary multiplications and additions, Case 1 is more efficient than Case 2. However one multiplication by 11 requires 2 shifts and 2 additions in a base field and one multiplication by 8 requires only 1 shift since

$$r \times 11 = r \times 2^3 + r \times 2 + r, \quad (10)$$

$$r \times 8 = r \times 2^3. \quad (11)$$

If we consider that, necessary computations of modular reductions are the following:

(1) $x^5 - 11$
8 shifts and 12 additions in the base field,

(2) $x^5 - x - 8$
4 shifts and 8 additions in the base field.

As we have seen above, the computation time of modular reductions depends on the total hamming weight of the coefficients. Therefore if a trinomial generator with less hamming weight of coefficients can be chosen, then multiplication in an extended OEF with a trinomial generator is faster than that of a trinomial generator. In the next section, we discuss the inversions in extended OEFs with a trinomial generator.

4. Inversion in OEF

As of date, inversion algorithms suitable for OEFs with a binomial generator have been proposed[3),14)]. However these algorithms are not suitable for extended OEFs with a trinomial generator. In this section, we propose an inversion algorithm suitable for extended OEF with a trinomial generator after summarizing known inversion algorithms.

4.1 Known Methods of Inversions in OEFs

Inversion in OEFs can be implemented efficiently. For example, a method of inversion that uses Cramer's formula has been proposed[14)]. Their method is useful when an extension field has a small extension degree like 2 or 3. However, if the extension degree is larger than 3, it is rather complicated and is not useful since computation of a cofactor determinant is difficult. Other known methods are as follows.

(1) Extended Euclidean method
(2) Gaussian elimination method
(3) Bailey's and Paar's method (BP)[3)]

(1), (2) are familiar algorithms[4)]. On the other hand, **(3)** has been proposed by Bailey and Paar, recently. This method computes an inversion in a base field $\boldsymbol{F}_p$ by using an exponentiation. This method is especially efficient for an OEF with a binomial generator. Here we describe Algorithm **(3)** when applied to an OEF with a binomial generator.

In the binomial case, p-exponentiation of $x(\alpha) = \sum_{i=0}^{n}(x_i \alpha^i)$ is

$$x(\alpha)^p = \sum_{i=0}^{n}(x_i(\omega^{(p-1)/n})^i \alpha^i). \quad (12)$$

If we compute $(\omega^{(p-1)/n})^i$ in advance and keep those results, the amount of computation necessary for **(3)** is

$$((\log_2(n-1))(n^2+n-1) + n^2 + 2n - 1)\mathcal{M} + \mathcal{I}, \quad (13)$$

where $\mathcal{M}$ and $\mathcal{I}$ denote the computation amount for multiplication and inversion on a base field $\boldsymbol{F}_p$, respectively. However in the trinomial case, p-exponentiation of $x(\alpha) = \sum_{i=0}^{n}(x_i \alpha^i)$ is as follows:

$$x(\alpha)^p = \sum_{i=0}^{n}(x_i h(\alpha)^i), \quad (14)$$

where $h(\alpha) = \alpha^p$ in $\boldsymbol{F}_{p^n}$. In the case of an extended OEF with a trinomial generator, the amount of computation is estimated as follows:

$$((\log_2(n-1))(2n^2 - n) + 2n^2)\mathcal{M} + \mathcal{I}. \quad (15)$$

4.2 Inversion Algorithm Suitable for a Trinomial Generator

In this section, we propose an efficient inversion method for an extended OEF with a trinomial generator. This method reduces the term

$n\mathcal{I}$ in the equation for determining the amount of computation to $\mathcal{I}$ in the Gaussian elimination method.

Hereafter, we explain our method through an example where $\boldsymbol{F}_{p^5} = \boldsymbol{F}_p(\alpha)$, $p = 2^{31} - 1$, and α is a root of $x^5 - x - 8 = 0$. Here, we set an inverse of $x = x_0 + x_1\alpha + \cdots + x_4\alpha^4 \in \boldsymbol{F}_{p^5}$ to be $y = y_0 + y_1\alpha + \cdots + y_4\alpha^4$. Then y satisfies,

$$\begin{pmatrix} x_0 & 8x_4 & 8x_3 & 8x_2 & 8x_1 \\ x_1 & x_0 + x_4 & x_3 + 8x_4 & x_2 + 8x_3 & x_1 + 8x_2 \\ x_2 & x_1 & x_0 + x_4 & x_3 + 8x_4 & x_2 + 8x_3 \\ x_3 & x_2 & x_1 & x_0 + x_4 & x_3 + 8x_4 \\ x_4 & x_3 & x_2 & x_1 & x_0 + x_4 \end{pmatrix} \cdot \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}. \quad (16)$$

We need the following steps to compute the inverse y.

(1) Let m_{ij} and c_j be $(i,\ j)$ element of matrix and the i-th element of the vector of right side in Eq. (16) respectively, where $(i,\ j)$ denotes the i-th row and the j-th column. Then, transform the matrix in Eq. (16) to a triangle matrix. In the elimination step of the 1-st column, compute the following:

(a) Compute $\prod_{k\neq i} m_{k1}$ $(1 \le i \le 5)$ and $\prod m_{k1}$ as follows:
- (i) $s_1 \leftarrow m_{11}m_{21},\ s_2 \leftarrow s_1 m_{31},\ t_1 \leftarrow s_2 m_{41}\ (= \prod_{k\neq 5} m_{k1})$.
- (ii) $t_2 \leftarrow s_2 m_{51} (= \prod_{k\neq 4} m_{k1})$.
- (iii) $s_5 \leftarrow m_{41}m_{51},\ t_3 \leftarrow s_1 s_5\ (= \prod_{k\neq 3} m_{k1})$.
- (iv) $s_4 \leftarrow s_5 m_{31},\ t_4 := m_{11}s_4\ (= \prod_{k\neq 2} m_{k1})$.
- (v) $t_5 \leftarrow m_{21}s_4\ (= \prod_{k\neq 1} m_{k1})$.
- (vi) $t_0 \leftarrow t_1 m_{51}\ (= \prod m_{k1})$.

(b) $m_{ij} \leftarrow \prod_{k\neq i} m_{k1} \cdot m_{ij} - \prod_{k\neq 1} m_{k1} \cdot m_{1j}$ $c_i \leftarrow \prod_{k\neq i} m_{k1} \cdot c_i - \prod_{k\neq 1} m_{k1} \cdot c_1$ $(1 \le i \le 5,\ 2 \le j \le 5)$.

(c) $m_{11} \leftarrow \prod m_{k1}$ and $m_{i1} \leftarrow 0$ $(2 \le i \le 5)$.

In the elimination steps of other columns, repeat the computations in the same way and Eq. (16) is transformed to

$$\begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} & m_{15} \\ 0 & m_{22} & m_{23} & m_{24} & m_{25} \\ 0 & 0 & m_{33} & m_{34} & m_{35} \\ 0 & 0 & 0 & m_{44} & m_{45} \\ 0 & 0 & 0 & 0 & m_{55} \end{pmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \end{pmatrix}. \quad (17)$$

(2) Compute $\boldsymbol{F}_p$-inverses of diagonal elements of matrix in Eq. (17) as follows:

(a) Compute $\prod_{k\neq i} m_{kk}$ $(1 \le i \le 5)$ and $\prod m_{kk}$ in the same way as Step (1)(a).

(b) $u \leftarrow 1/(\prod m_{kk})$.

(c) $1/m_{11} = u\prod_{k\neq 1} m_{kk}$, $1/m_{22} = u\prod_{k\neq 2} m_{kk}$, $1/m_{33} = u\prod_{k\neq 3} m_{kk}$, $1/m_{44} = u\prod_{k\neq 4} m_{kk}$, $1/m_{55} = u\prod_{k\neq 5} m_{kk}$.

(3) Compute each components of the inverse y from y_4 to y_0 in Eq. (17).

Steps (1)(a) and (b) take $(3n-5)\mathcal{M}$ and $n^2\mathcal{M}$. All of Step (1) take $(\sum_{k=2}^{n}(n^2+3n-5)-n)\mathcal{M} = (\frac{1}{6}n(n+1)(2n+1)-1+\frac{3}{2}n(n+1)-3-5(n-1)-n)\mathcal{M}$. Steps (2) and (3) take $(4n-5)\mathcal{M}+\mathcal{I}$ and $\frac{1}{2}n(n+1)\mathcal{M}$. Therefore, the inverse is computed in time:

$$\left(\frac{1}{3}n^3 + \frac{5}{2}n^2 + \frac{1}{6}n - 4\right)\mathcal{M} + \mathcal{I}. \quad (18)$$

4.3 Comparison between Known Methods and Our Method

In this section, we compare our method with known methods. The computation time for an OEF over a binomial and a trinomial is estimated as follows.

(1) Extended Euclidean method (a binomial and a trinomial)

$$(2n^2 + n - 4)\mathcal{M} + n\mathcal{I} \quad (19)$$

(2) Bailey's and Paar's method (a binomial)

$$((\log_2(n-1))(n^2+n-1) \quad (20)$$
$$+ n^2 + 2n - 1)\mathcal{M} + \mathcal{I} \quad (21)$$

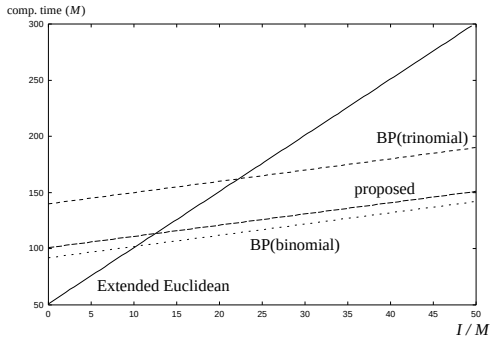
(3) Bailey's and Paar's method (a trinomial)

$$((\log_2(n-1))(2n^2-n) \quad (22)$$
$$+ 2n^2)\mathcal{M} + \mathcal{I} \quad (23)$$

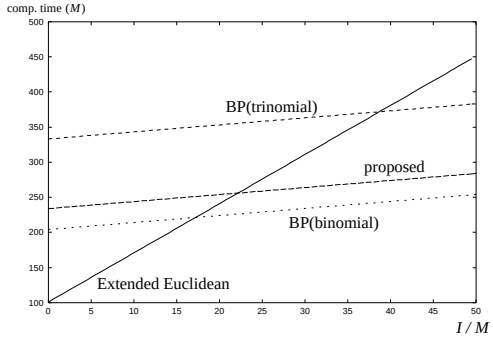
(4) Our proposed method (a binomial and a trinomial)

$$\left(\frac{1}{3}n^3 + \frac{5}{2}n^2 + \frac{1}{6}n - 4\right)\mathcal{M} + \mathcal{I} \quad (24)$$

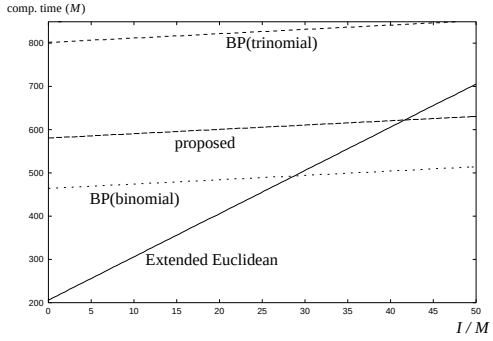
Figure 1 shows the amount of computation required in inversion methods when $n = 5,\ 7,\ 10$. We conclude that our proposed method is the most efficient in trinomial case for $5 \le n \le 7$ since $\mathcal{I}/\mathcal{M}$ ranges roughly from 20 to 60 as seen in Table 2. As the extension degree increases, the extended Euclidean method becomes the most efficient.



(i) $n = 5$



(ii) $n = 7$



(iii) $n = 10$

Fig. 1 Comparison of inversion methods.

5. Implementation of Arithmetic in OEF

5.1 Timing of Arithmetic in Extension Fields

Table 1 shows the OEF's parameters which we discuss in this paper. The platforms are a PentiumII 400 MHz (Linux-2.2.5) for **OEF32**, **EXOEF31**, **OEF31** and an Alpha21164A 600 MHz (Linux-2.2.1) for **OEF64**, **OEF61**, **EXOEF61**. We use inline assembly codes in C programs for 32, 64-bit addition, subtraction, multiplication and shift.

Table 2 shows the running times for arithmetics in the base field $\boldsymbol{F}_p$. We compute the inverse in the base field of **OEF32**, **EXOEF31**, **OEF31** by the extended Euclidean algorithm and **OEF64**, **OEF61**, **EXOEF61** by the extended binary algorithm. We see that the multiplication in **EXOEF31** is 1.62 times faster than that of **OEF32** in Table 2.

The hamming weight of c ($p = 2^{31} - c$) in **OEF31** is 3. When the hamming weight of c ($p = 2^{31} - c$) is more than 2, multiplying by c using the multiplication instruction of the CPU is faster than using shifts and additions of the CPU on PentiumIIs. Therefore, we use the multiplication instruction of the CPU for multiplications by c. We see that the multiplication in **EXOEF31** is 1.68 times faster than that in **OEF31** in Table 2.

[Comparison of time of inversions between our method and BP method]

Table 3 shows the time required for inversions that use our method and the BP method, and a comparison between the normal binomial case and the best trinomial case. Our method is more suitable than the BP method for **EXOEF31-155**☆. **EXOEF31-155**, the best trinomial case, is more efficient than **OEF31-155**, a normal binomial case. Therefore we see

Table 1 OEF's parameters.

	Size (bits)	p	Generator polynomial
EXOEF31-155	155	$2^{31} - 1$	$x^5 - x - 8$
OEF31-155	155	$2^{31} - 1057$	$x^5 - 2$
OEF31-186	186	$2^{31} - 1$	$x^6 - 5$
OEF31-217	217	$2^{31} - 1$	$x^7 - 3$
OEF32-160	160	$2^{32} - 5$	$x^5 - 2$
OEF32-192	192	$2^{32} - 387$	$x^6 - 2$
OEF32-224	224	$2^{32} - 1053$	$x^7 - 2$
OEF61-183-1	183	$2^{61} - 1$	$x^3 - 5$
EXOEF61-183	183	$2^{61} - 1$	$x^3 - x - 4$
OEF61-183-2	183	$2^{61} - 1$	$x^3 - 37$
OEF64-192	192	$2^{64} - 189$	$x^3 - 2$

Table 2 Running times for arithmetics in $\boldsymbol{F}_p$ (μsec).

	p	Add.	Mul.	Inv.
EXOEF31	$2^{31} - 1$	0.031	0.056	3.039
OEF31	$2^{31} - 1057$	0.031	0.094	3.682
OEF32	$2^{32} - 5$	0.026	0.091	3.807
OEF61 (EXOEF61)	$2^{61} - 1$	0.047	0.077	4.984
OEF64	$2^{64} - 189$	0.041	0.144	5.730

☆ An improved method for the extended Euclidean method was proposed in Ref. 22), but it is slower than our method when $5 \leq n \leq 7$ because the method in Ref. 22) uses many branch instructions.

Table 3 Running times for arithmetics in the binomial case and the trinomial case (μsec).

	Add.	Mul.	Inv.
EXOEF31-155	0.12	1.40	(proposed) 10.20
			(BP) 11.36
OEF31-155	0.13	2.05	11.31
OEF61-183-1	0.09	0.81	6.32
EXOEF61-183	0.09	0.87	6.49
OEF61-183-2	0.09	0.90	6.36

Table 4 Running times for arithmetics in OEFs (μsec).

	Add.	Mul.	Sq.	Inv.
EXOEF31-155	0.12	1.40	0.94	10.20
OEF31-186	0.13	1.88	1.31	13.69
OEF31-217	0.14	2.46	1.62	17.62
OEF32-160	0.12	2.18	1.32	11.56
OEF32-192	0.16	3.01	1.78	17.57
OEF32-224	0.17	4.63	2.53	23.67
OEF61-183	0.09	0.81	0.63	6.32
OEF64-192	0.14	1.27	0.63	7.64

that the parameter p takes a more important role at the running time than the generator polynomial. This means that a p that satisfies OEF conditions cannot be found, then it would be better to search for another p that will satisfy extended OEF conditions.

Table 4 shows the running times for arithmetics in OEFs. We compute the inverse in **OEF64-192**, **OEF61-183** by Cramer's formula, in **EXOEF31-155** by our method, and in the other fields by the BP method.

5.2 Further Discussion

Here we roughly estimate the running time of an elliptic curve exponentiation of an extended OEF on a smart card with an 8-bit CPU. We consider an elliptic curve E over $\boldsymbol{F}_{p^n}$ that satisfies the following conditions:

(1) $p = 2^8 - 5,\ q = p^{23}$
(2) a generator polynomial $x^{23} - 2^7x - 2^2$
(3) $E : y^2 = x^3 - 3x + 85$
(4) $\#E(\boldsymbol{F}_p) = 231\times$ (176-bit prime)

Then an exponentiation of the elliptic curve E is about 3 times faster than that of elliptic curves over a general 160-bit prime field on an 8-bit CPU. Therefore, OEFs and extended OEFs are efficient on an 8-bit CPU.

6. Construction Method of Elliptic Curve over OEF

6.1 SEA suitable for OEF

There are three typical construction methods for elliptic curves with appropriate order: the lifting method[30], Complex Multiplication (CM) method[1] and the order counting method. In the case of elliptic curves over OEFs, the order counting method is the most suitable since it does not place any restriction on elliptic curves. On the other hand in the lifting method order of elliptic curves should be rather larger, and in the CM method it is difficult to find an elliptic in practical time.

6.2 OEF Suitable for SEA

OEFs are also suitable for SEA algorithms under the following situation:

1. **Arithmetic:** The arithmetic in OEFs is faster than that in $\boldsymbol{F}_{2^n}$.
2. **Polynomial exponentiation X^q:** As we described in Section 2.4, the polynomial exponentiation X^q is the dominant step in the SEA algorithm. Here, we estimate the computation time for X^q.

   SEA algorithm requires the computation of $X^q \bmod \Phi_l(X)$, where the degree of $\Phi_l(X)$ is equal to $l+1$. The computation of X^q requires the following steps.

   Polynomial Exponentiation
   $[X^p \bmod \Phi_l(X)]$
   It takes $\frac{3}{2}|p|\mathcal{P}$ on the average, where $\mathcal{P}$ denotes the computation time of polynomial multiplication in $\boldsymbol{F}_q[X]/\Phi_l(X)$.

   Polynomial Multiplication
   $[X^{2p},\ X^{3p},\ \cdots,\ X^{lp}]$
   $X^{2p},\ X^{3p},\ \cdot,\ X^{lp}$ is computed in such a way that $X^{2p} = X^p \cdot X^p,\ X^{3p} = X^{2p} \cdot X^p,\ \cdots,\ X^{lp} = X^{(l-1)p} \cdot X^p$, where X^p has already been computed in Step 1. Therefore, the total amount of computation required is $(l-1)\mathcal{P}$.

   Polynomial Transformation
   $[(X^p)^p = X^{p^2}, (X^{p^2})^p = X^{p^3}, \cdots, (X^{p^{n-1}})^p = X^{p^n}]$
   A polynomial $g(X)$ over $\boldsymbol{F}_p$ satisfies $g(X)^p = g(X^p)$. Therefore, we can determine $g(X^p)$ by replacing $X,\ X^2,\ \cdots,\ X^l$ of $g(X)$ to $X^p,\ X^{2p},\ \cdots,\ X^{lp}$. The amount of computation for $g(X^q)$ is approximately $\mathcal{P}$, as the computation requires $l(l+1)$ $\boldsymbol{F}_p$-multiplications. Therefore, the total amount of computation required is $(n-1)\mathcal{P}$.

   In total, the polynomial exponentiation $X^q \bmod \Phi_l(X)$ takes $(\frac{3}{2}|p| + l + n - 2)\mathcal{P}$, where $|q| = n|p|$. For instance, we consider the amount of computation necessary when $l = 5$. When $n = 5$, i.e., $|p| = 32$, $\frac{3}{2}|p|+l+n-2 = 56$ and when $n = 1$, i.e., $\boldsymbol{F}_q$

is a prime field, $\frac{3}{2}|p|+l+n-2 = 242$. Hence, we conclude that the number of necessary polynomial multiplications for polynomial exponentiation in an OEF is smaller than that in a prime field. Moreover, the polynomial multiplication in an OEF is faster than that in a prime field, because the arithmetic in an OEF is faster. Therefore, the polynomial exponentiation $X^q \bmod \Phi_l(X)$ in an OEF is faster than that in a prime field.

Elkies' algorithm requires the polynomial exponentiation $X^q \bmod g_l(X)$ to find the eigenvalue for ϕ_q. It is also necessary to compute the polynomial exponentiation $Y^{q-1} = f(X)^{(q-1)/2} \bmod g_l(X)$. It is computed in the following way:

Step 1. Compute the polynomial exponentiation $f(X)^{(p-1)/2} \bmod g_l(X)$.

Step 2. For $i = 2, 3, \cdots, n$, compute $f(X)^{((p^2-1)/2}, f(X)^{((p^3-1)/2}, \cdots, f(X)^{(p^n-1)/2} = f(X)^{(q-1)/2}$, using the following equation,

$$f(X)^{(p^i-1)/2} = (f(X)^{(p^{i-1}-1)/2})^p f(X)^{(p-1)/2}, \quad (25)$$

where we also use the computation results of $X^p, X^{2p}, X^{3p}, \cdots$.

3. Inversion: The match and sort algorithm searches the exact point for $t \pmod{\prod l}$. When using projective or Jacobian coordinates, inversions are required for every comparison primitive, and when using affine coordinates, inversions are required for every addition or doubling primitive. Therefore, in both cases, inversions are required frequently. Since the inversions in OEFs are faster than those in prime fields, the match and sort algorithm over an OEF is more efficient than that over a prime field.

7. Implementation of Construction of Elliptic Curves

In this section, we present implementation results.

7.1 The SEA Algorithm over OEF

Let $\mathcal{M}_\mathcal{O}$, $\mathcal{S}_\mathcal{O}$, $\mathcal{I}_\mathcal{O}$ be the amount of computation necessary for multiplication, squaring and inversion over an OEF, respectively. The addition and doubling of elliptic curves take $2\mathcal{M}_\mathcal{O} + \mathcal{S}_\mathcal{O} + \mathcal{I}_\mathcal{O}$ and $2\mathcal{M}_\mathcal{O} + 2\mathcal{S}_\mathcal{O} + \mathcal{I}_\mathcal{O}$ in affine coordinates. In Jacobian coordinates[5], they become $12\mathcal{M}_\mathcal{O} + 4\mathcal{S}_\mathcal{O}$ and $4\mathcal{M}_\mathcal{O} + 6\mathcal{S}_\mathcal{O}$ respectively. According to Table 4, in **OEF32-160**, $\mathcal{S}_\mathcal{O} = 0.61\mathcal{M}_\mathcal{O}, \mathcal{I}_\mathcal{O} = 5.30\mathcal{M}_\mathcal{O}$. Therefore,

$$2\mathcal{M}_\mathcal{O} + \mathcal{S}_\mathcal{O} + \mathcal{I}_\mathcal{O} = 7.91\mathcal{M}_\mathcal{O} \quad (26)$$

$$2\mathcal{M}_\mathcal{O} + 2\mathcal{S}_\mathcal{O} + \mathcal{I}_\mathcal{O} = 8.52\mathcal{M}_\mathcal{O} \quad (27)$$

$$12\mathcal{M}_\mathcal{O} + 4\mathcal{S}_\mathcal{O} = 14.44\mathcal{M}_\mathcal{O} \quad (28)$$

$$4\mathcal{M}_\mathcal{O} + 6\mathcal{S}_\mathcal{O} = 7.66\mathcal{M}_\mathcal{O} \quad (29)$$

The match and sort part of the SEA algorithm requires to check whether two points are the same or not for every computation of elliptic curve exponentiation. In Jacobian coordinates, we need $4\mathcal{M}_\mathcal{O}$ for checking. On the other hand, affine coordinates do not require additional computation. Hence, we use affine coordinates for the match and sort part.

Table 5 Running times for the SEA algorithms over OEFs (sec).

	Total	Match & Sort	X^q
EXOEF31-155	10.14	1.17	5.06
OEF31-186	28.92	3.03	15.70
OEF31-217	74.18	3.90	45.64
OEF32-160	11.60	1.75	5.48
OEF32-192	33.09	2.62	18.38
OEF32-224	88.21	7.46	48.89
OEF61-183	20.18	3.72	9.72
OEF64-192	28.53	5.24	15.19
pf160	32.68	3.93	20.79
pf192	95.16	9.36	65.03

We select 100 curves randomly and count order of these curves. **Table 5** shows the average running times of the SEA algorithm over OEFs. We used the same computes as in Section 5.1 and we use the Karatsuba's Method for polynomial multiplication. **pf160**, **pf192** in Table 5 are the computation results of the SEA algorithm over the prime field[11], whose platform is the same as **OEF32**, **EXOEF31**, **OEF31**. In the case of a prime field (i.e., $q = p$), the computation time for polynomial multiplication X^q takes up 64–68% of the total time of the SEA algorithm over $\boldsymbol{F}_p$[18]. We see that the ratio is 50–64% in Table 5☆. As a result, the total time of the SEA algorithm is the fastest among all types of finite fields.

7.2 Construction of Elliptic Curves over OEF

We construct elliptic curves with prime order by using the early abort method[17),18]. We configured 100 random sequences and constructed 100 elliptic curves with prime order. **Table 6**

☆ In the case of $\boldsymbol{F}_{2^n}$, the ratio is very small[18]. However, the SEA algorithm over $\boldsymbol{F}_{2^n}$ is slower than that over OEF since arithmetic in $\boldsymbol{F}_{2^n}$ is slower than that in OEF.

Table 6 Running times for constructing elliptic curves.

	Total time (seconds)	Number of searched elliptic curves
OEF32-160	247.8	258.4
OEF32-192	651.5	210.9
EXOEF31-155	192.0	179.3
OEF31-186	542.2	236.2

shows the average running times for constructing elliptic curves with prime order.

8. Conclusion

We have extended the idea of OEFs in such a way to include a trinomial generator in addition to a binomial generator. As a result, our extended OEF includes another extension field $\boldsymbol{F}_{p^5}$ ($p = 2^{31}-1$) with fast arithmetic. We have also proposed a new inversion algorithm in extended OEF suitable for a trinomial generator. Our inversion algorithm has made arithmetic in extended OEFs more efficient.

Furthermore, we estimated the amount of computation necessary for the SEA algorithms over OEFs. As a result, we have confirmed that computation time decreases for the SEA algorithm over an OEF compared to both of $\boldsymbol{F}_p$ and $\boldsymbol{F}_{2^n}$.

We have also been implemented the SEA algorithm over OEF (PentiumII 400 MHz, Linux-2.2.5 and Alpha21164A 600 MHz, Linux-2.2.1). The average running times for order counting of an elliptic curve over a 155-bit extended OEF and 160-bit OEF are 10.1 and 11.6 seconds on PentiumII 400 MHz (Linux-2.2.5), respectively.

Acknowledgments The authors are grateful to Masao Kasahara, Ryuichi Sakai, Masanobu Kaneko and Keiji Horiuchi for invaluable comments.

References

1) Atkin, A.O.L. and Morain, F.: Elliptic curves and primality proving, *Math. Computation*, Vol.61, pp.29–68 (1993).
2) Bailey, D.B. and Paar, C.: Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms, *Advances in Cryptology – Proc. Crypto'98*, Lecture Notes in Compute Science, Vol.1462, pp.472–485, Springer-Verlag (1998).
3) Bailey, D.B. and Paar, C.: Inversion in Optimal Extension Fields, Conference on The Mathematics of Public-Key Cryptography, Jun. 12–17 (1999).
4) Cohen, H.: *A Course in Computational Algebraic Number Theory*, Graduate Texts in Math., Vol.138, Third corrected printing, Springer-Verlag (1996).
5) Cohen, H., Miyaji, A. and Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates, *Advances in Cryptology – Proc. Asiacrypto'98*, Lecture Notes in Computer Science Vol.1514, pp.51–65, Springer-Verlag (1998).
6) Couveignes, J.M. and Morain, F.: Schoof's algorithm and isogeny cycles, *Proc. ANTS-I*, Lecture Notes in Compute Science, Vol.877, pp.43–58, Springer-Verlag (1994).
7) ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms, *IEEE Trans. Inf. Theory*, IT-31, pp.469–472 (1985).
8) Proposed federal information processing standard for digital signature standard (DSS), *Federal Register*, Vol.56, No.169, pp.42980–42982 (1991).
9) Elkies, N.D.: Explicit isogenies, Preprint (1991).
10) Frey, G. and Rück, H.G.: A remark concerning m-divisibility and the discrete logarithm in the divisor class group of curves, *Math. Computation* Vol.62, pp.865–874 (1991).
11) Futa, Y. and Miyaji, A.: Efficient construction of prime order elliptic curves (in Japanese), *SCIS'99*, pp.857–862 (1999).
12) Horiuchi, K., Futa, Y., Sakai, R. and Kasahara, M.: Construction of Elliptic Curves with Prime Order and Estimation of Its Complexity (in Japanese), *IEICE*, Vol.J82-A, No.8, pp.1269–1277 (1999).
13) IEEE P1363 Working Draft, June 16 (1998).
14) Kobayashi, T., Morita, H., Kobayashi, K. and Hoshino, F.: Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic, *Advances in Cryptology – Proc. Eurocrypt'99*, Lecture Notes in Computer Science, Vol.1592, pp.176–189, Springer-Verlag (1999).
15) Koblitz, N.: Elliptic curve cryptosystems, *Math. Computation*, Vol.48, pp.203–209 (1987).
16) Lercier, R.: Algorithmique des courbes elliptiques dans les corps finis, Thése, *École Polytechnique-LIX* (1997).
17) Lercier, R.: Finding Good Random Elliptic Curves for Cryptosystems Defined over F_{2^n}, *Advances in Cryptology – Proc. Eurocrypt'97*, Lecture Notes in Computer Science, Vol.1233, pp.379–392, Springer-Verlag (1997).
18) Lercier, R. and Morain, F.: Counting the number of points on elliptic curve over finite fields: Strategies and performances, *Advances in Cryptology – Proc. Eurocrypt'95*, Lecture Notes in Computer Science, Vol.921, pp.79–94, Springer-Verlag (1995).

19) Menezes, A., Okamoto, T. and Vanstone, S.: Reducing elliptic curve logarithms to logarithms in a finite field, *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, pp.80–89 (1991).
20) Miller, V.S.: Use of elliptic curves in cryptography, *Advances in Cryptology – Proc. Crypto'85*, Lecture Notes in Computer Science, Vol.218, pp.417–426, Springer-Verlag (1986).
21) Morain, F.: Calcul du nombre de points sur une courbe elliptique dans un corps fini: Aspects algorithmiques, *Journal de Théorie des Nombres de Bordeux*, Vol.7, pp.255–282 (1995).
22) Nagao, K.: Some idea on arithmetics of Jacobian group of hyperelliptic curve (in Japanese), *Algebra and Computation '99* (1999).
23) Pohlig, S.C. and Hellman, M.E.: An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance, *IEEE Trans. Inf. Theory*, Vol.IT-24, pp.106–110 (1978).
24) Pollard, J.: Monte Carlo methods for index computation (mod p), *Math. Computation*, Vol.32, pp.918–924 (1978).
25) Rivest, R., Shamir, A. and Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems, *Comm. ACM*, Vol.21, No.2, pp.120–126 (1978). *IEEE Trans. Inf. Theory*, Vol.IT-24, pp.106–110 (1978).
26) Satoh, T. and Araki, K.: Fermat quotients and the polynomial time discrete log algorithm for anomalous elliptic curves, *Commentarii Math. Univ. St. Pauli.*, Vol.47, pp.81–92 (1998).
27) Schoof, R.: Elliptic Curves Over Finite Fields and the Computation of Square Roots mod p, *Math. Computation*, Vol.44, pp.483–494 (1985).
28) Schoof, R.: Counting points on elliptic curve over finite fields, *Journal de Théorie des Nombres de Bordeux*, Vol.7, pp.219–254 (1995).
29) Semaev, I.A.: Evaluation of discrete logarithms in a group of p-torsion points of an elliptic curve in characteristic p, *Math. Computation*, vol.67, pp.353–356 (1998).
30) Silverman, J.H.: *The Arithmetic of Elliptic Curves*, GTM, Vol.106, Springer-Verlag, New York (1986).
31) Smart, N.P.: The discrete logarithm problem on elliptic curves of trace one, *J. Cryptology*, to appear.

(Received November 30, 1999)
(Accepted June 1, 2000)

Yuichi Futa was born in Osaka, Japan, on September 21, 1973. He received the B.E. and the M.E. degrees in electronics and information science from Kyoto Institute of Technology, Kyoto, Japan in 1996 and 1998, respectively. Since 1998, he has been with Multimedia Development Center in Matsushita Electric Industrial Co., Ltd. and engaged in research and development for secure communication. His interests are in cryptography and information security.

Atsuko Miyaji received the B.Sc., the M.Sc., and Dr.Sci. degrees in mathematics from Osaka University, Osaka, Japan in 1988, 1990, and 1997 respectively. She joined Matsushita Electric Industrial Co., Ltd. from 1990 to 1998 and engaged in research and development for secure communication. She has been an associate professor at JAIST (Japan Advanced Institute of Science and Technology) since 1998. Her research interests include the application of projective varieties theory into cryptography and information security. She is a member of the Institute of Electronics, Information and Communication Engineers and the Information Processing Society of Japan.