Efficient Implementation of ηT Pairing on Supersingular Elliptic Curves in Characteristic 3

川原, 祐人
九州大学大学院数理学府

https://doi.org/10.15017/21704

Efficient Implementation of η_T Pairing on Supersingular Elliptic Curves in Characteristic 3

Yuto KAWAHARA

A dissertation submitted in fulfillment of the requirements for the degree of

Doctor of Philosophy in Functional Mathematics

Supervisor: Tsuyoshi TAKAGI

Graduate School of Mathematics
Kyushu University

Abstract

Pairing-based cryptosystems can provide cryptographic schemes which have novel and useful properties, such as Identity-based encryption schemes, and they have been attracted in cryptography. These schemes are constructed by using pairings, such as the Tate and Weil pairings, hash functions, and group computations. Miller proposed the first polynomial-time algorithm for computing the Weil pairing on algebraic curves, and various pairings such as η_T, Ate, Ate$_i$, R-ate, Optimal pairings and their variants are indicated. The η_T pairing over $\mathbb{F}_{3^m}$ is one of the fastest pairing now.

We propose efficient algorithms of addition and subtraction in $\mathbb{F}_3$ and MapToPoint, which is a hash function to compute a point on elliptic curves, for efficient implementation of the η_T pairing over $\mathbb{F}_{3^m}$. Firstly, we construct instruction sequences of addition and subtraction in $\mathbb{F}_3$ with the minimum number of logical instructions, since all functions of the η_T pairing over $\mathbb{F}_{3^m}$ are based on them. Every $\mathbb{F}_3$-element is assigned to two bits, and the $\mathbb{F}_3$-addition and subtraction are considered as a map $(\mathbb{F}_2)^2 \times (\mathbb{F}_2)^2 \to (\mathbb{F}_2)^2$. We perform an exhaustive search for finding the instruction sequences of the $\mathbb{F}_3$-addition that use seven or fewer logical instructions. Indeed, we find many implementations of the $\mathbb{F}_3$-addition and subtraction with only six logical instructions, and no instruction sequence that can compute them with less than six logical instructions for any assignment of elements or logical instructions. In other words, we have proven that the minimum number of logical instructions for computing the $\mathbb{F}_3$-addition and subtraction in $\mathbb{F}_3$ is six by two-bit encoding.

MapToPoint algorithm is used to compute a point of an elliptic curve from identity. There exists two conventional algorithms for supersingular elliptic curves over $\mathbb{F}_{3^m}$: one is computed by using a square root computation in $\mathbb{F}_{3^m}$, whose computational cost is $O(\log m)$ multiplications and $O(m)$ cubings in $\mathbb{F}_{3^m}$; another is computed by using an $(m-1) \times (m-1)$ matrix over $\mathbb{F}_3$, that is stored in the off-line memory. We construct an efficient MapToPoint algorithm on the supersingular

elliptic curves by using *1/3-trace* over $\mathbb{F}_{3^m}$. The *1/3-trace* over $\mathbb{F}_{3^m}$ can compute a solution x of $x^3 - x = c$ using no multiplication in $\mathbb{F}_{3^m}$. The proposed algorithm is computed by $O(1)$ multiplications and $O(m)$ cubings in $\mathbb{F}_{3^m}$, and it stores less than m $\mathbb{F}_3$-elements in the off-line memory to efficiently compute *trace* over $\mathbb{F}_{3^m}$.

Finally, we implement the main components, the η_T pairing, arithmetic over the supersingular elliptic curves $E(\mathbb{F}_{3^m})$ and the finite field $\mathbb{F}_{3^m}$ for constructing the pairing-based cryptosystems. We implement them in C and Java as programming languages, and measure the running time of these functions on an Intel Core i7-950 processor.

Acknowledgements

Firstly, I would like to thank Professor Tsuyoshi Takagi for insightful suggestions, advices and comments. If he did not invite me to the cryptographic research 6 years ago, I had not studied this field and this thesis could not be done. I would also like to thank Professor Shigenori Uchiyama for nice comments, and I would be thankful to Professor Masaaki Shirase for their helpful advices. I would be very grateful to all members and former members in Takagi laboratory for their support.

I give a special thanks to Professor Eiji Okamoto who was indebted by joint research. Then I give a great thanks to Kazumaro Aoki, Tetsutaro Kobayashi, Gen Takahashi, Go Yamamoto and the members of NTT Information Sharing Platform Laboratories for their fruitful comments and discussions at joint research and intern. I have received support from the Japan Society for the Promotion of Science during Ph.D. student.

Finally, I would like to thank everyone in my family for their support, advices and encouragement.

Contents

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Public Key Cryptography

Recently, there exists many convenient network services such as Cloud computing that is a model for using shared pool of computing resources on-demand. In order to use the network services, information need to be communicated securely because much personal information (e.g., name, address, password, message) may be included in the information. Cryptography is a base factor of information security in information and communication technologies.

Symmetric key cryptography is one of the algorithms for cryptography. It uses a common single key in both encryption and decryption. In order to use the symmetric key cryptography, both users who communicate some information initialize the same key for encryption and decryption. In addition, users must have many keys corresponding to each user's key. AES is generally used as the symmetric key encryption now, and it can calculate much quickly encryption and decryption of a message.

Diffie and Hellman gave a solution to key agreement of the symmetric key cryptography in 1976 [35], and this paper introduced a concept of public key cryptography. In this solution, two users compute g^a and g^b for random integers a, b and a cyclic multiplicative group generated by g, and send them mutually. Then each user computes $(g^b)^a$ and $(g^a)^b$ respectively, and uses it as the symmetric key of the symmetric key cryptography.

In 1978, Rivest, Shamir and Adleman proposed RSA cryptography that is the first practical public key encryption [90]. It is the revolutionary idea to communicate information securely, since the problem of the key agreement solved using public key cryptography. For example, Alice shows her public key correspond-

ing to her private key in public. Bob encrypts a message with her public key and sends the ciphertext to her. Alice decrypts the ciphertext with own private key. The security of RSA cryptography is based on an RSA assumption, it believed to be equivalent to an integer factorization problem. The fastest algorithm to solve it is number field sieve now.

ElGamal showed a new public key cryptosystem, called ElGamal cryptosystem, that encrypt and decrypt using exponentiation over finite fields [38]. Its security is based on a discrete logarithm problem over finite fields, which is the problem to compute x satisfied with $a = g^x$ from a and g, where a generator g of a finite field, and an element a in the field. In 1985, Koblitz and Miller independently suggested elliptic curve cryptography that uses a group of rational points on elliptic curves over finite fields [69, 80]. The security of the elliptic curve cryptography is based on a discrete logarithm problem over elliptic curves. This problem is harder than the discrete logarithm over the finite filed and the integer factorization, since there is no efficient algorithm to solve this problem. Then the size of keys of the elliptic curve cryptography is smaller than that of the RSA and ElGamal cryptography, and the functions of the elliptic curve cryptography can be efficiently computed. There are some solving algorithms for the general elliptic curve cryptography such as ρ-method.

Now we usually apply the RSA cryptography and elliptic curve cryptography on public key infrastructure for encryption and signature in the network services.

1.2 Pairing-based Cryptography

In former public key cryptosystems, a user public key is generated from a user private key, and then the user public key becomes a random bit-string. Therefore the user cannot know whether the public key used in encryption is correct. Pairing-based cryptosystems are widely shown by ID-based encryption scheme having been proposed. Moreover, by using computable bilinear pairing over elliptic curves, various novel cryptosystems have been achieved such as a short signature [30], keyword searchable encryption [27], broad cast encryption [29], proxy re-encryption [77], and functional encryption [75, 84].

By the short signature, the length of the signature is shorter than that in the previous signature scheme. The keyword searchable encryption searches a keyword in the ciphertext without decryption. In the proxy re-encryption, a proxy can convert the ciphertext encrypted one public key into the different ciphertext which can be

decrypted with another private key. The broadcast encryption make a ciphertext for some appointed users of a group. Each user have own private key, respectively, however, the users decrypt the same ciphertext with own private key. The functional encryption has the ability of flexible decryption. For example, a user encrypts a message with attributes, and the others can decrypt it if own private key constructed by AND, OR, etc. satisfies the attributes. It can use many applications such as access control, mail services, and contents distribution.

In addition, the cryptosystems with various convenient properties have been proposed until now. Thus, for construction of secure and useful cryptosystem, pairing-based cryptography by using bilinearity of pairings is very effective cryptosystem.

ID-based encryption scheme:

ID-based encryption is one of the most well-known pairing-based cryptosystems. In this scheme, when Bob sends a message to Alice securely, he can encrypt the message with her identity such as e-mail address as Alice's public key. He can easily check whether the public key is correct without a certificate authority, since the identity is the known string. In the ID-based cryptosystems, a public key generator that is trusted third party like the certificate authority is required in order to make a master key, system parameters, and user public and private keys.

In 1984, Shamir showed a general idea of identity based cryptosystems [98], that can make the public key as arbitrary known information (e.g., e-mail address). The practical encryption scheme of the identity based cryptosystems was proposed by Sakai, Ogishi, Kasahara by using bilinear pairing over elliptic curves [93, 92]. Boneh and Franklin also suggested the identity based cryptosystem and its concrete implementation [28]. After that, there are many ID-based encryption schemes with additional functions or high security.

Here, we present the Boneh-Franklin ID-based encryption scheme as an example of ID-based encryption. The scheme is described by the four algorithms: Setup, Extract, Encrypt and Decrypt. Let $\mathbb{G}_1$ be an additive group, and $\mathbb{G}_2$ be a multiplicative group of order q. The basic idea of the ID-based encryption scheme is presented as follows:

Setup. Given a security parameter $k \in \mathbb{Z}$. The setup phase for the ID-based encryption executes the following steps.

(1) Run a randomized algorithm on input k to generate a prime q, two

groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of order q generated by random generators $P \in \mathbb{G}_1$ and $g \in \mathbb{G}_2$, and an admissible bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_2$.

(2) Pick a random $s \in \mathbb{Z}_q^*$ and set $P_{pub} \leftarrow sP$.

(3) Choose cryptographic hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1^*$ and $H_2 : \mathbb{G}_2 \to \{0,1\}^n$ for some positive integer n.

(4) Set message space $\mathcal{M} = \{0,1\}^*$ and ciphertext space $\mathcal{C} = \mathbb{G}_1^* \times \{0,1\}^n$. Then publish $\mathsf{param} = \{q, \mathbb{G}_1, \mathbb{G}_2, e, n, P, P_{pub}, H_1, H_2\}$ as system parameters, and conceal $s \in \mathbb{Z}_q^*$ as a master key which has a public key generator.

Extract. For a given user identity $\mathsf{ID} \in \{0,1\}^*$, compute $Q_{\mathsf{ID}} \leftarrow H_1(\mathsf{ID}) \in \mathbb{G}_1^*$, and set the private key $d_{\mathsf{ID}} \leftarrow sQ_{\mathsf{ID}}$ where s is the master key.

Encrypt. To encrypt a message $M \in \mathcal{M}$ using the user identity ID, compute $Q_{\mathsf{ID}} \leftarrow H_1(\mathsf{ID}) \in \mathbb{G}_1^*$. Then choose $r \in \mathbb{Z}_q^*$ randomly, and set the ciphertext to be

$$C = (rP, M \oplus H_2(g_{\mathsf{ID}}^r)),$$

where $g_{\mathsf{ID}} = e(Q_{\mathsf{ID}}, P_{pub}) \in \mathbb{G}_2^*$ and the n-bit bitwise XOR instruction $\oplus$.

Decrypt: Let $C = (U, V) \in \mathcal{C}$ be a ciphertext encrypted using the user identity ID. To decrypt the ciphertext C using the private key $d_{\mathsf{ID}} \in \mathbb{G}_1^*$ corresponds to the user identity ID, compute

$$V \oplus H_2(e(d_{\mathsf{ID}}, U)) \ (= M).$$

1.3 Motivation and Contribution

Pairing-based cryptosystems can be constructed many novel and useful cryptosystems with bilinearity such as identity based encryption. Then various convenient network services with cryptography can be provided securely. The efficient computation of pairing such as Tate and Weil pairings are required for the efficient construction of the pairing-based cryptosystems. However the computational cost of the pairing is slower than that of the previous public key cryptosystems.

Miller proposed the first efficient algorithm in order to compute the Tate pairing, and it can compute the pairing value in polynomial time [82, 81], Then some algorithms which calculates the pairing value more faster such as [9, 10, 42] were shown. Duursma and Lee proposed an efficient algorithm for computing the Tate

pairing on hyperelliptic curves of the form $y^2 = x^p - x \pm 1$ over $\mathbb{F}_{p^m}$, where a prime p such that $p \equiv 3 \bmod 4$ and $\gcd(m, 2p) = 1$ [36]. Barreto et al. indicated an η_T pairing [7], which is a different version of the Duursma-Lee algorithm, and it is about twice faster than the Duursma-Lee algorithm. In addition, various algorithms to compute pairing such as Ate pairing [57], Ate$_i$ pairing [115], R-ate pairing [74], Optimal pairing [110] have been proposed until now. Currently, the η_T pairing over $\mathbb{F}_{3^m}$ is one of the fastest pairings.

The previous public key cryptography uses prime fields $\mathbb{F}_p$ or binary fields $\mathbb{F}_{2^m}$ for a prime integer p and a positive integer m. Thus many efficient algorithms to compute arithmetic in $\mathbb{F}_p$, $\mathbb{F}_{2^m}$ have been proposed until now. On the other hand, the η_T paring computation is based on arithmetic in $\mathbb{F}_{3^m}$, and arithmetic in $\mathbb{F}_{3^m}$ had not been used before proposed the η_T paring. Therefore it is required to construct efficient algorithms in $\mathbb{F}_{3^m}$.

In this thesis, we propose some efficient algorithms for constructing pairing-based cryptosystems using the η_T pairing in characteristic three. We firstly construct the optimal addition and subtraction in $\mathbb{F}_3$. The addition and subtraction in $\mathbb{F}_3$ are the functions used as the base of all computations for the η_T pairing. For implementation of the addition and subtraction in $\mathbb{F}_3$, the instruction sequences with the minimum number of logical instructions need to be searched. An element in $\mathbb{F}_3$ is assigned to two bits, and the addition and subtraction are considered as a map $(\mathbb{F}_2)^2 \times (\mathbb{F}_2)^2 \to (\mathbb{F}_2)^2$. The instruction sequence consists of logical instructions such as AND, OR and XOR. Then we search the minimum instruction sequences by an exhaustive search. As the result of the search, we found many implementation of the addition and subtraction sequences with six logical instructions, and no implementation of them with less than six logical instructions for any two-bit encodings and binary logical instructions. In other words, we proved that the minimum construction of the addition and subtraction in $\mathbb{F}_3$ is six logical instructions.

Secondly, we suggest an efficient MapToPoint algorithm on supersingular elliptic curves in $\mathbb{F}_3$. MapToPoint is a hash function onto a point on an elliptic curve from a bit-string. It is used for constructing pairing-based cryptosystems as one of the components. For example, in encryption of the Boneh-Franklin ID-based encryption, the elliptic curve point is computed from a user ID. In the conventional MapToPoint algorithm on supersingular elliptic curves in $\mathbb{F}_3$, y-coordinate is computed from x-coordinate using a square root computation in $\mathbb{F}_{3^m}$. We propose *1/3-trace* over $\mathbb{F}_{3^m}$, which solves a solution x of $x^3 - x = t$ without multiplication

in $\mathbb{F}_{3^m}$. Then the proposed algorithm computes y-coordinate from x-coordinate using *1/3-trace*, and the number of $\mathbb{F}_{3^m}$-multiplications in the proposed algorithm is reduced from $O(\log m)$ to $O(1)$.

In addition, we describe the implementation detail of η_T pairing, point multiplication on supersingular elliptic curves over $\mathbb{F}_{3^m}$, and exponentiation in $\mathbb{F}_{3^{6m}}$ for the efficient construction of the paring-based cryptosystems. Then we implement them in C and Java, which are the most well-known programming languages and generally used for constructing systems. Moreover we measure the running time of these functions on an Intel Core i7-950 processor.

1.4 Organization

This thesis organizes as follows: Chapter 2 provides a preliminary of a finite field, an elliptic curve, the Weil and Tate pairings, and Miller's algorithm. In Chapter 3, we review the papers dealing with the efficient computation of pairings on supersingular elliptic curves in characteristic three, and then Chapter 4 details efficient implementation of the η_T pairing over $\mathbb{F}_{3^m}$. Chapter 5 explores the minimum number of logical instruction for optimizing the addition and subtraction in $\mathbb{F}_3$. Chapter 6 shows the efficient construction of a hash function from a bit-string to a point on supersingular elliptic curves over $\mathbb{F}_{3^m}$. Chapter 7 shows the timing results of the functions used by constructing the pairing-based cryptosystems in C and Java. Finally, the thesis is concluded in Chapter 8.

Chapter 2

Mathematical Background

2.1 Introduction

Cryptography, which consists of using some properties based on algebra, requires some mathematical background for constructing secure system. This chapter describes an overview of the mathematical techniques in order to build pairing-based cryptography.

We firstly explain the properties of finite fields, elliptic curves that are important components in cryptography. Moreover the notion of a bilinear pairing that has the property of bilinearity such as the Weil and Tate pairings are examined. Then we indicate Miller's algorithm that is the efficient algorithms to compute the result of the Tate and Weil pairings in polynomial time.

This chapter explains the basic facts of the mathematics. For the more detailed description such as theorems and proofs, refer to the books [25, 31, 32, 55, 76, 104] and so on.

2.2 Finite Fields

Definition 2.1. *A field K is a commutative ring such that every non-zero element is invertible. The field is equipped with two operations, addition and multiplication. Subtraction and division in the field are defined in term of addition and multiplication, respectively.*

If the number of elements contained in a field K is a finite number, then the field K is said to be a finite field. The additive identity element in the field is 0 and the multiplicative one is 1.

Let L and K be fields. An element $a \in L$ is algebraic over K if there is a non-zero polynomial $f(x)$ with all coefficients in K, such that $f(a) = 0$. Then a is called a root of the polynomial $f(x)$. A field L is an algebraic extension of K if every element in L is algebraic over K.

The characteristic of a field K is the least positive integer p if p-time addition of the multiplicative identity $\underbrace{1 + 1 + \cdots + 1}_{p \text{ times}}$ is equal to the additive identity 0; otherwise the characteristic is 0. If the characteristic of the field K is a prime p, then K is an extension of the prime field $\mathbb{F}_p = \{0, 1, ..., p-1\}$. For any positive integer k and prime field $\mathbb{F}_p$, there exists the extension field $\mathbb{F}_q$ with $q = p^k$ in characteristic p, then the positive integer k is said to an extension degree of $\mathbb{F}_q$. Arithmetic of addition and multiplication in a prime field $\mathbb{F}_p$ is given by $(a + b) \bmod p$ and $(a \cdot b) \bmod p$ for given $a, b \in \mathbb{F}_p$, respectively.

Definition 2.2. *Let K be a field. Then a field $\overline{K}$ is said to be an algebraic closure of K, if $\overline{K}$ is algebraic over K, and every polynomial of degree at least 1 with coefficients in $\overline{K}$ has a root in $\overline{K}$.*

For a given finite field $\mathbb{F}_q$, the algebraic closure of $\mathbb{F}_q$ is defined by

$$\overline{\mathbb{F}_q} = \bigcup_{i \geq 1,\ i \in \mathbb{N}} \mathbb{F}_{q^i}.$$

The set of non-zero elements of $\mathbb{F}_q$, denoted $\mathbb{F}_q^* = \mathbb{F}_q \setminus \{0\}$, is a cyclic group with respect to multiplication. The set of all polynomials with coefficients in $\mathbb{F}_q$ forms a commutative ring, denoted $\mathbb{F}_q[x]$. A polynomial $f(x)$ in $\mathbb{F}_q[x]$ is said to be irreducible over $\mathbb{F}_q[x]$ if f has a positive degree and $f = gh$ with $g, h \in \mathbb{F}_q[x]$ implies that either g or h is a constant value.

Let $\mathbb{F}_q$ be a finite field and $f(x)$ be a polynomial of degree n over $\mathbb{F}_q$. Then $\mathbb{F}_q[x]/(f(x))$ denotes the set of all polynomials of degree less than n, and is a field if and only if $f(x)$ is irreducible.

2.2.1 Bases

Let $\mathbb{F}_q$ and $\mathbb{F}_{q^k}$ be finite fields such that $\mathbb{F}_{q^k}$ is a finite extension of $\mathbb{F}_q$. If every element $c \in \mathbb{F}_{q^k}$ has a unique representation of the form

$$c = \sum_{i=0}^{k-1} c_i \alpha_i \ (c_0, ..., c_{k-1} \in \mathbb{F}_q),$$

with the elements $\alpha_0, ..., \alpha_{k-1} \in \mathbb{F}_{q^k}$, the set $(\alpha_0, ..., \alpha_{k-1}) \in (\mathbb{F}_{q^k})^k$ is said to be a basis of the finite field $\mathbb{F}_{q^k}$. Then the element in $\mathbb{F}_{q^k}$ is denoted by the vector representation $(c_0, c_1, ..., c_{k-1})$. For implementing arithmetic in finite fields, there are two basis representations; a polynomial basis and a normal basis.

Polynomial basis. The most familiar basis for implementation is a polynomial basis. It can easily and efficiently implement the field operations with polynomial calculation. Let $f(x)$ be an irreducible polynomial of degree k in $\mathbb{F}_q[x]$. If $\alpha \in \mathbb{F}_{q^k}$ is a root of $f(x)$, then a polynomial basis of $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ is a set of the form

$$(1, \alpha, \alpha^2, ..., \alpha^{k-1}).$$

Normal basis. A normal basis is a less commonly used basis rather than the polynomial basis. Let α be an element in $\mathbb{F}_{q^k}$. If $\alpha, \alpha^q, ..., \alpha^{q^{k-1}}$ are linearly independent, α is called normal over $\mathbb{F}_q$, and then a set of the form

$$(\alpha, \alpha^q, \alpha^{q^2}, ..., \alpha^{q^{k-1}})$$

is denoted by a normal basis of $\mathbb{F}_{q^k}$. There exists the normal bases of $\mathbb{F}_{q^k}$ over $\mathbb{F}_q$ for any $\mathbb{F}_q$ and positive integer k. By using a normal basis, the exponentiation c^q of the element $c = \sum_{i=0}^{k-1} c_i \alpha^{q^i} = (c_0, c_1, ..., c_{k-1}) \in \mathbb{F}_{q^k}$ can be easily computed by the shift operation of the vector representation

$$(c_{k-1}, c_0, c_1, ..., c_{k-2}).$$

2.2.2 Legendre Symbol

Let p be an odd prime number. An integer a such that $a \not\equiv 0 \pmod{p}$ is a quadratic residue modulo p if there exists x satisfying

$$x^2 \equiv a \bmod p,$$

otherwise a is a quadratic non-residue modulo p. The number of elements which are quadratic residue is $(p-1)/2$ in $\mathbb{F}_p$.

Definition 2.3. *The Legendre symbol is a function, for an integer a and prime integer p, defined by*

$$\left(\frac{a}{p}\right) = \begin{cases} -1 & \textbf{if} \quad a \textit{ is a quadratic non-residue}, \\ 0 & \textbf{if} \quad a \equiv 0 \pmod{p}, \\ +1 & \textbf{if} \quad a \textit{ is a quadratic residue}. \end{cases}$$

The Legendre symbol satisfies the following properties, for input integers a, b and an odd prime p,

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}, \left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$$

If p and q are prime integer, then one has the quadratic reciprocity law

$$\left(\frac{q}{p}\right)\left(\frac{p}{q}\right) = (-1)^{(p-1)(q-1)/4}.$$

The Jacobi symbol is a generalization of the Legendre symbol. For an integer a and a positive odd integer n such that $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$, where primes p_i and positive integers α_i, the Jacobi symbol is given by the product of the Legendre symbol:

$$\left(\frac{a}{n}\right) = \left(\frac{a}{p_1}\right)^{\alpha_1}\left(\frac{a}{p_2}\right)^{\alpha_2}\cdots\left(\frac{a}{p_k}\right)^{\alpha_k}.$$

If $n = 1$, then the Jacobi symbol is defined by

$$\left(\frac{a}{1}\right) = 1.$$

The Legendre symbol for an extension field $\mathbb{F}_{p^k}$ in odd characteristic is defined similarly to the case of the prime field. Let $f(x)$ be an irreducible polynomial of degree k in $\mathbb{F}_p[x]$ such that $\mathbb{F}_p[x]/(f(x))$ is isomorphic to $\mathbb{F}_{p^k}$. For a polynomial $t(x) \in \mathbb{F}_p[x]$, the Legendre symbol in $\mathbb{F}_p[x]$ is defined by

$$\left(\frac{t(x)}{f(x)}\right) = \begin{cases} -1 & \textbf{if} \quad t(x) \text{ is not a square modulo } f(x) \\ 0 & \textbf{if} \quad t(x) \mid f(x) \\ +1 & \textbf{if} \quad t(x) \text{ is a non-zero square modulo } f(x) \end{cases}$$

The Legendre symbol of $\mathbb{F}_{p^k}$ can be extended to the Jacobi symbol when $f(x)$ is not an irreducible polynomial. The algorithm to compute the Legendre symbol in $\mathbb{F}_p[x]$ is shown in Algorithm 2.1.

2.2.3 Trace and Norm

Let $\mathbb{F}_{q^k}$ and $\mathbb{F}_q$ be finite fields. The trace and norm of the endomorphism over $\mathbb{F}_q$ are defined as follows.

Definition 2.4. *For $\alpha \in \mathbb{F}_{q^k}$, the trace $\mathrm{Tr}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(\alpha)$ of α over $\mathbb{F}_q$ is defined by*

$$\mathrm{Tr}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(\alpha) = \alpha + \alpha^q + \alpha^{q^2} + \cdots + \alpha^{q^{k-1}}.$$

The norm $\mathrm{N}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(\alpha)$ of α over $\mathbb{F}_q$ is defined by

$$\mathrm{N}_{\mathbb{F}_{q^k}/\mathbb{F}_q}(\alpha) = \alpha \cdot \alpha^q \cdot \alpha^{q^2} \cdot \cdots \cdot \alpha^{q^{k-1}}.$$

Algorithm 2.1 Legendre symbol in $\mathbb{F}_p[x]$

INPUT: an irreducible polynomial $f(x)$ and a polynomial $t(x)$ in $\mathbb{F}_p[x]$

OUTPUT: $c = \left(\frac{t(x)}{f(x)}\right)$

1: $c \leftarrow 1$
2: **for** $\deg f \neq 0$ **do**
3: **if** $t(x) = 0$ **then return** 0
4: $a \leftarrow$ the leading coefficient of $t(x)$
5: $t(x) \leftarrow t(x)/a$
6: **if** $\deg f \equiv 1 \pmod 2$ **then** $c \leftarrow c \cdot \left(\frac{a}{p}\right)$
7: **if** $p^{\deg f} \equiv 3 \pmod 4$ **and** $\deg f \deg t \equiv 1 \pmod 2$ **then** $c \leftarrow -c$
8: $r(x) \leftarrow t(x)$
9: $t(x) \leftarrow f(x) \bmod r(x)$
10: $f(x) \leftarrow r(x)$
11: **end for**
12: **return** c

2.3 Elliptic Curves

2.3.1 Weierstrass equation

Let K be a field and $\overline{K}$ be its algebraic closure. A projective plane $\mathbb{P}^2$ over K is the set of the form

$$\mathbb{P}^2 = \{(X : Y : Z) \mid X, Y, Z \in \overline{K}\} \backslash \{0 : 0 : 0\},$$

with respect to an equivalence relation $\sim$. For given $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ in $\mathbb{P}^2$, the equivalence relation $\sim$ of $\mathbb{P}^2$ is so defined that

$$(X_1 : Y_1 : Z_1) \sim (X_2 : Y_2 : Z_2)$$

if and only if there exists $\lambda \in \overline{K}^*$ such that $X_1 = \lambda X_2$, $Y_1 = \lambda Y_2$, $Z_1 = \lambda Z_2$. An equivalence class of the form

$$\{(\lambda X : \lambda Y : \lambda Z) \mid \lambda \in \overline{K}^*\}$$

is represented by $[X : Y : Z]$. Then the set of K-rational points in $\mathbb{P}^2$ is the set

$$\mathbb{P}^2(K) = \{[X : Y : Z] \in \mathbb{P}^2 \mid X, Y, Z \in K\}.$$

Let $\mathbb{P}^2(K)$ be the set of all projective points in $\mathbb{P}^2$ over K. A Weierstrass equation is a homogeneous equation of the form

$$Y^2Z + a_1XYZ + a_3YZ^2 = X^3 + a_2X^2Z + a_4XZ^2 + a_6Z^3,$$

where a_1, a_2, a_3, a_4, $a_6 \in \overline{K}$. Here $\mathcal{O} = [0 : 1 : 0]$ is the base point and said to be the point at infinity of the curve.

The Weierstrass equation is said to be a smooth or non-singular if at least one of three partial derivatives $\partial F/\partial X$, $\partial F/\partial Y$, $\partial F/\partial Z$ is non-zero at P for all points $[X : Y : Z] \in \mathbb{P}^2(K)$ satisfying the equation

$$F(X, Y, Z) = Y^2Z + a_1XYZ + a_3YZ^2 - X^3 - a_2X^2Z - a_4XZ^2 - a_6Z^3 = 0.$$

If all partial derivatives is zero at some point P in $\in \mathbb{P}^2(K)$, the point P and the Weierstrass equation is called a singular point and singular curve, respectively.

In order to use the equation conveniently, we generally write the Weierstrass equation for an elliptic curve over K using non-homogeneous coordinate

$$E/K : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

Here each point of the curve using Affine coordinate is given by $x = X/Z$ and $y = Y/Z$ for $[X : Y : Z] \in \mathbb{P}^2(K)$.

Quantities Δ_E and j_E of E/K is defined as follows:

$$\begin{cases} d_2 &= a_1^2 + 4a_2, \\ d_4 &= 2a_4 + a_1a_3, \\ d_6 &= a_3^2 + 4a_6, \\ d_8 &= a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2, \\ c_4 &= d_2^2 - 24d_4, \\ \Delta_E &= -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6, \\ j_E &= c_4^3/\Delta_E. \end{cases}$$

The quantity Δ_E is called a discriminant of the Weierstrass equation, and j_E is also called a j-invariant of E/K if $\Delta_E \neq 0$. The curve E/K is the non-singular elliptic curve if and only if the discriminant is not equal to 0. For two elliptic curves E and E' over a field K, if these curves are isomorphic, then $j_E = j_{E'}$.

Let K be any extension field of a field K_0. The set of K-rational points on E is defined by

$$E(K) = \{(x, y) \in K^2 \mid y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\mathcal{O}\}.$$

A point in $E(K)$ is written $P = (x, y)$. The n-exponent group of $E(K)$ is the following set:

$$nE(K) = \{nP \mid P \in E(K)\},$$

and the group of n-torsion point that is the subgroup of $E(K)$ is defined by

$$E(K)[n] = \{P \in E(K) \mid nP = \mathcal{O}\}.$$

2.3.2 Group Law and Group Order

Let E be an elliptic curve given by a Weierstrass equation, and K be a field. The K-rational points on E become an Abelian group with respect to the addition described below. For the set $E(K)$ and the point at infinity $\mathcal{O}$, an addition low of the elliptic curve is defined as follows:

1. For P in $E(K)$, then $P + (-P) = \mathcal{O}$. The point $-P$ is the negative point of P, and it is $(x, -y - a_1x - a_2)$ for a given $P = (x, y)$.

2. Let P, Q be points on $E(K)$ such that $Q \neq \pm P$ and $P, Q \neq \mathcal{O}$. The line ℓ is through P, Q, then there exists a third point T of intersection of ℓ with E. Then $R = P + Q$ over $E(K)$ is the point $-T$.

3. Let P be a point on $E(K)$ such that $P \neq -P$ and $P \neq \mathcal{O}$, and let ℓ be the tangent line to E at P. The line ℓ intersects the curve E at a second point T. Then $R = 2P$ over $E(K)$ is the point $-T$.

4. $P + \mathcal{O} = P$ for all points P in $E(K)$.

5. $P + Q = Q + P$ for all points P, Q in $E(K)$.

6. Let $P, Q, R \in E(K)$. Then $(P + Q) + R = P + (Q + R)$

Let p be a prime and $q = p^m$, and let $\mathbb{F}_q$ be a finite field with p^m elements. Then the number of points on $E(\mathbb{F}_q)$, denoted $\#E(\mathbb{F}_q)$, is the order of the elliptic curve. The order of a point P is defined by the least positive integer r such that $rP = \mathcal{O}$.

Let E be an elliptic curve over a finite field $\mathbb{F}_q$. For the order of the elliptic curve, we obtain

$$\#E(\mathbb{F}_q) = q + 1 - t \text{ and } |t| \leq 2\sqrt{q}$$

from Hasse's theorem. The integer t is called the trace of the Frobenius, and the interval $[q + 1 - 2\sqrt{q}, q + 1 + 2\sqrt{q}]$ is called the Hasse interval.

Let p be a prime integer and $\mathbb{F}_q$ be a finite field of characteristic p. An elliptic curve E defined over $\mathbb{F}_q$ is supersingular, if it satisfies one of the following equivalent conditions:

1. p divides the trace of the Frobenius t.

2. E has no point of order p over $\overline{\mathbb{F}_q}$.

3. The endomorphism ring of E over $\overline{\mathbb{F}_q}$ is non-commutative.

Otherwise, it is an ordinary (non-supersingular) elliptic curve.

2.3.3 Simplified Weierstrass Equation

In order to construct group operations over elliptic curves efficiently, we apply admissible change of variables to the Weierstrass equation for simplifying the Weierstrass equation.

The two elliptic curve E and E' over K given by the equations:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$
$$E' : y^2 + a'_1xy + a'_3y = x^3 + a'_2x^2 + a'_4x + a'_6$$

are said to be isomorphic over K, if there are $u, r, s, t \in K$, and $u \neq 0$ such that the curve E' is obtained from E by the following admissible change of variables

$$(x, y) \to (u^2x + r, u^3y + u^2sx + t).$$

We may use the Weierstrass equation for any field K including that of characteristic 2 or 3. Generally, we deal with the simplified elliptic curves that are isomorphic curves of the Weierstrass equation for efficiently calculating. There are the following three transformations according to the characteristic of the field.

Char $= 2$. If the characteristic of the field K is equal to 2, then the admissible change of variables is

$$(x, y) \to \left(a_1^2x + \frac{a_3}{a_1}, a_1^3y + \frac{a_1^2a_4 + a_3^2}{a_1^3}\right).$$

The Weierstrass equation is transformed to the following curve by the above admissible change of variables:

$$y^2 + xy = x^3 + ax^2 + b,$$

where $a, b \in K$. The discriminant and j-invariant of this elliptic curve are $\Delta_E = b$ and $j_E = 1/b$. If $a_1 = 0$, then the curve transforms by the admissible change of variables

$$(x, y) \to (x + a_2, y)$$

to the curve

$$y^2 + cy = x^3 + ax + b,$$

where $a, b, c \in K$. The discriminant and j-invariant are $\Delta_E = c^4$ and $j_E = 0$, and thus this curve is a supersingular elliptic curve.

Char $= 3$. In an ordinary (non-singular) curve of the field in characteristic 3, if $a_1^2 \neq -a_2$, then the admissible change is

$$(x, y) \to \left(x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_4}{d_2} + a_3 \right)$$

where $d_2 = a_1^2 + a_2$, $d_4 = a_4 - a_1 a_3$, and the Weierstrass equation is transformed into the curve

$$y^2 = x^3 + ax^2 + b,$$

where $a, b \in K$. The discriminant and j-invariant are $\Delta_E = -a^3 b$ and $j_E = -a^3/b$. If $a_1^2 = -a_2$, then the admissible change is

$$(x, y) \to (x,\ y + a_1 x + a_3)\,.$$

For $a, b \in K$, the Weierstrass equation then transforms to the curve

$$y^2 = x^3 + ax + b.$$

The discriminant and j-invariant are $\Delta_E = -a^3$ and $j_E = 0$. This curve is also said to be a supersingular elliptic curve in characteristic 3. This form is the same equation of non-supersingular curves in characteristic $p \neq 2, 3$.

Char $\neq$ 2, 3. If the characteristic of K is not equal to 2 or 3, then the admissible change of variables

$$(x, y) \to \left(\frac{x - 3a_1^2 - 12a_2}{36},\ \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24} \right)$$

transforms E to the curve

$$y^2 = x^3 + ax + b,$$

where $a, b \in K$. The discriminant and j-invariant of this curve are

$$\Delta_E = -16(4a^3 + 27b^2),\ j_E = -1728\frac{(4a)^3}{\Delta_E},$$

respectively. The elliptic curve is a supersingular elliptic curve if and only if the trace t of the Frobenius satisfies $t \equiv 0 \pmod{p}$; otherwise it is an ordinary elliptic curve.

Table 2.1 lists the curves of simplified Weierstrass equations applying the admissible change of variables for each characteristic.

Table 2.1: List of Simplified Weierstrass Equations

Char	Curve	Determinant Δ_E	j-invariant j_E
$\neq 2, 3$	$y^2 = x^3 + ax + b$	$-16(4a^3 + 27b^2)$	$-1728(4a)^3/\Delta_E$
$= 2$	$y^2 + xy = x^3 + ax^2 + b$	b	$1/b$
$= 2$	$y^2 + cy = x^3 + ax + b$	c^4	0
$= 3$	$y^2 = x^3 + ax^2 + b$	$-a^3b$	$-a^3/b$
$= 3$	$y^2 = x^3 + ax + b$	$-a^3$	0

2.3.4 Explicit Addition Formulas on E/K

For each simplified Weierstrass equation, there are the explicit addition formulas of the group law. Now, we indicate the explicit formulas of a negative point, point addition and point doubling for each characteristic and each ordinary and supersingular elliptic curve, respectively.

Let E/K be an elliptic curve over K and $E(K)$ be a set of all points on E/K. The identity $\mathcal{O}$ satisfies $P + \mathcal{O} = \mathcal{O} + P = P$ for all $P \in E(K)$, and the point Q where $P + Q = \mathcal{O}$ is said to the negative point of $P \in E(K)$, denoted by $-P$.

For the elliptic curve $y^2 = x^3 + ax + b$, the formulas of the negative point, the point addition and point doubling are as follows:

Negative point. If $P = (x, y) \in E(K)$, then $-P$ is $(x, -y)$.

Point addition. Let $P = (x_1, y_1), Q = (x_2, y_2)$ be points in $E(K)$ such that $P \neq \pm Q$. Then the point addition $R = (x_3, y_3) = P + Q$ is computed as follows:

$$\begin{cases} \lambda &= \dfrac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= \lambda^2 - (x_1 + x_2) \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

Point doubling. Let $P = (x_1, y_1)$ be a point in $\in E(K)$ where $P \neq -P$. Then the point doubling $R = (x_3, y_3) = 2P$ is computed as follows:

$$\begin{cases} \lambda &= \dfrac{3x_1^2 + a}{2y_1} \\ x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

These formulas can be applied to ordinary and supersingular elliptic curves in characteristic $\neq 2, 3$ and supersingular elliptic curves in characteristic 3. If the

characteristic of the field K is 3, then the computational cost of the point doubling can be reduced, since $3x_1^2$ modulo 3 is equal to 0.

For the ordinary elliptic curve $y^2 + xy = x^3 + ax^2 + b$ in characteristic 2, the formulas of the negative point, the point addition and point doubling are as follows:

Negative point. If $P = (x, y) \in E(K)$, then $-P$ is $(x, x + y)$.

Point addition. Let $P = (x_1, y_1), Q = (x_2, y_2)$ be points in $E(K)$ where $P \neq \pm Q$. Then the point addition $R = (x_3, y_3) = P + Q$ is calculated by:

$$\begin{cases} \lambda &= \dfrac{y_1 + y_2}{x_1 + x_2} \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 \\ y_3 &= \lambda(x_1 + x_3) + x_3 + y_1 \end{cases}$$

Point doubling. Let $P = (x_1, y_1)$ be a point in $\in E(K)$ where $P \neq -P$. Then the point doubling $R = (x_3, y_3) = 2P$ is computed by:

$$\begin{cases} \lambda &= \dfrac{x_1 + y_1}{x_1} \\ x_3 &= \lambda^2 + \lambda + a = x_1^2 + \dfrac{b}{x_1^2} \\ y_3 &= x_1^2 + \lambda x_3 + x_3 \end{cases}$$

When the elliptic curve is the supersingular $y^2 + cy = x^3 + ax + b$, their formulas are as follows:

Negative point. If $P = (x, y) \in E(K)$, then $-P$ is $(x, y + c)$.

Point addition. For given points $P = (x_1, y_1), Q = (x_2, y_2)$ in $E(K)$ where $P \neq \pm Q$, the point addition $R = (x_3, y_3) = P + Q$ is calculated as:

$$\begin{cases} \lambda &= \dfrac{y_1 + y_2}{x_1 + x_2} \\ x_3 &= \lambda^2 + x_1 + x_2 \\ y_3 &= \lambda(x_1 + x_3) + y_1 + c \end{cases}$$

Point doubling. Let $P = (x_1, y_1)$ be a point in $\in E(K)$ where $P \neq -P$, then the point doubling $R = (x_3, y_3) = 2P$ is calculated by:

$$\begin{cases} \lambda &= \dfrac{x_1^2 + a}{c} \\ x_3 &= \lambda^2 \\ y_3 &= \lambda(x_1 + x_3) + y_1 + c \end{cases}$$

For the ordinary elliptic curve $y^2 = x^3+ax^2+b$ in characteristic 3, the negative point, the point addition and the point doubling are computed by the following formulas:

Negative point. If $P = (x, y) \in E(K)$, then $-P$ is $(x, -y)$.

Point addition. Let $P = (x_1, y_1), Q = (x_2, y_2)$ be points in $E(K)$ such that $P \neq \pm Q$. Then the point addition $R = (x_3, y_3) = P + Q$ is computed by:

$$\begin{cases} \lambda &= \dfrac{y_2 - y_1}{x_2 - x_1} \\ x_3 &= \lambda^2 - (x_1 + x_2 + a) \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

Point doubling. Let $P = (x_1, y_1)$ be a point in $\in E(K)$ such that $P \neq -P$. Then the point doubling $R = (x_3, y_3) = 2P$ is computed by:

$$\begin{cases} \lambda &= \dfrac{ax_1}{y_1} \\ x_3 &= \lambda^2 - 2x_1 \\ y_3 &= \lambda(x_1 - x_3) - y_1 \end{cases}$$

2.3.5 Point Multiplication

Point multiplication is one of the most important operation for constructing elliptic curve cryptography and pairing-based cryptography. Let P be a point on an elliptic curve $E(K)$ and d be an integer. The point multiplication is defined by

$$dP = \begin{cases} \sum_{i=1}^{d} P & \textbf{if} \quad n > 0 \\ \mathcal{O} & \textbf{if} \quad n = 0 \\ -\sum_{i=1}^{|d|} P & \textbf{if} \quad n < 0 \end{cases}$$

The computational cost of the point multiplication is usually $d - 1$ point additions. There exists some efficient algorithms such as binary method for point multiplication. The binary method for point multiplication consists of a sequence of point additions and point doublings and it is executed using Algorithm 2.2 with an integer d represented by a signed n-digit binary string $(-1)^s \sum_{i=0}^{n-1} d_i 2^i$. The computational cost of Algorithm 2.2 is approximately $(\lfloor \log_2 d \rfloor + 1)/2$ point additions and $\lfloor \log_2 d \rfloor$ point doubling for a random choosing d.

Algorithm 2.2 Binary method for point multiplication

INPUT: a point $P \in E(K)$, a signed binary string $d = (-1)^s \sum_{i=0}^{n-1} d_i 2^i$ $(d_i \in \{0,1\}$ for $i \in [0, n-1]$, $s \in \{0,1\})$

OUTPUT: $Q = dP$

1: $Q \leftarrow \mathcal{O}$
2: **for** $i \leftarrow n-1$ **downto** 0 **do**
3: $\quad Q \leftarrow 2Q$
4: $\quad$ **if** $d_i = 1$ **then** $Q \leftarrow Q + P$
5: **end for**
6: **return** $(-1)^s Q$

2.3.6 Frobenius Map

Let $\mathbb{F}_q$ be a field, and $\overline{\mathbb{F}_q}$ be its algebraic closure. Then the Frobenius map for $\mathbb{F}_q$ is given by:

$$\begin{array}{rccc} \phi_q & : & \overline{\mathbb{F}_q} & \rightarrow & \overline{\mathbb{F}_q}, \\ & & x & \mapsto & x^q. \end{array}$$

Let E be an elliptic curve defined over $\mathbb{F}_q$. The Frobenius map ϕ_q of a point (x, y) in $E(\overline{\mathbb{F}_q})$ acts:

$$\phi_q(x, y) \mapsto (x^q, y^q),\ \phi_q(\mathcal{O}) \mapsto \mathcal{O}.$$

2.4 Pairings

In cryptography, pairings are functions which map a pair of points on elliptic curve to an element in a multiplicative group of a finite field. Pairing can be applied to various novel cryptographic systems such as ID-based cryptosystems using bilinearity. In this section, we describe the definition and properties of pairings, and the construction of pairings for Miller's algorithm.

Definition 2.5. *Let n be a positive integer, and let $\mathbb{G}_1$ and $\mathbb{G}_2$ be additive groups of order n with identity 0, and then $\mathbb{G}_3$ be a multiplicative group of order n with identity 1. A pairing is a map*

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3.$$

If $\mathbb{G}_1 = \mathbb{G}_2$, then it is said to be a symmetric pairing; otherwise it is called an asymmetric pairing.

The pairings we consider for cryptography will satisfy the following properties:

Bilinearity: For all $P, P' \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, we have

$$e(P + P', Q) = e(P, Q)e(P', Q).$$

Similarly, for all $P \in \mathbb{G}_1$ and $Q, Q' \in \mathbb{G}_2$, we have

$$e(P, Q + Q') = e(P, Q)e(P, Q').$$

Then for any integers $a, b \in \mathbb{Z}$, we obtain

$$e(aP, bQ) = e(bP, aQ) = e(P, Q)^{ab}.$$

Non-degeneracy: For all $P \in \mathbb{G}_1$ with $P \neq 0$, there is some $Q \in \mathbb{G}_2$ such that $e(P, Q) \neq 1$. Similarly, for all $Q \in \mathbb{G}_2$ with $Q \neq 0$, there is some $P \in \mathbb{G}_1$ such that $e(P, Q) \neq 1$.

For given elements $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$, we give the following consequences about bilinearity from the above properties.

$$\begin{cases} e(P, 0) = e(0, Q) = 1, \\ e(-P, Q) = e(P, -Q) = e(P, -Q)^{-1}. \end{cases}$$

There are two commonly examples of pairings, the Weil and Tate pairings defined on elliptic curves over finite fields, and they are generally used to construct the pairing-based cryptosystems.

2.4.1 Divisors

Let K be a field and E be an elliptic curve over K, and let $E(\overline{K})$ be the set of all points on the curve defined over $\overline{K}$. Then a divisor and divisor group are defined by the following:

Definition 2.6. *The divisor group of $E(\overline{K})$ is the free Abelian group generated by the points of the curve E, and denoted by $\mathrm{Div}_{\overline{K}}(E)$. An element D in the divisor group is said to a divisor, and is given by a formal sum of finite points:*

$$D = \sum_{P_i \in E(\overline{K})} n_i(P_i),$$

where $n_i \in \mathbb{Z}$ and $n_i = 0$ for all but finitely many $P_i \in E(\overline{K})$.

The divisor group has the natural group structure under the addition:

$$D + D' = \sum n_i(P_i) + \sum m_i(P_i) = \sum (n_i + m_i)(P_i),$$

for the divisors D and D' in $\mathrm{Div}_{\overline{K}}(E)$. The degree of the divisor D in $\mathrm{Div}_{\overline{K}}(E)$ is defined by

$$\deg D = \sum_{P_i \in E(\overline{K})} n_i$$

A divisor is called a zero divisor if all n_i equal to zero, and written by 0. If $\deg D = 0$, then the divisor is said to be a degree zero divisor. The divisor group of degree zero divisors is represented by

$$\mathrm{Div}^0(E) = \{D \in \mathrm{Div}(E) \mid \deg D = 0\},$$

and it forms a subgroup of $\mathrm{Div}_{\overline{K}}(E)$. The sum of the divisor D is given by the addition of all points

$$\mathrm{sum}(D) = \sum_{P_i \in E(\overline{K})} n_i P_i \in E(\overline{K}).$$

Let f be a non-zero function on E. The multiplicity of f at a point P is given by $\mathrm{ord}_P(f)$. The divisor of the non-zero function f, written (f), is represented by the following divisor:

$$(f) = \sum_{P_i \in E(\overline{K})} \mathrm{ord}_{P_i}(f)(P_i).$$

If $\mathrm{ord}_P(f) > 0$, then f has a zero at P, and if $\mathrm{ord}_P(f) < 0$, then f has a pole at P. If f has no zeros or poles at P that means $(f) = 0$, then f is a constant. For given functions f, g, we obtain $(fg) = (f) + (g)$ and $(f/g) = (f) - (g)$. Two divisors D and D' are equivalent, denoted as $D \sim D'$, if $D' = D + (f)$ for some function f. A divisor is called a principal divisor if the divisor is equal to (f) for some function f.

Let E be an elliptic curve over a field K. Let $D = \sum_{P_i \in E(\overline{K})} n_i(P_i)$ be a degree zero divisor on E. Then there exists functions f on E satisfied with $D = (f)$ if and only if $\mathrm{sum}(D) = \mathcal{O}$. The divisor is the principal divisor, and its degree is $\deg((f)) = 0$.

The support of the divisor D is given by the set of all point P_i such that $n_i \neq 0$. Let f be a function and $D = \sum_{P_i \in E(\overline{K})} n_i(P_i)$. Assume that the support of D is disjoint to the support of (f). Then we define

$$f(D) = \prod_{P_i \in E(\overline{K})} f(P_i)^{n_i}.$$

2.4.2 Tate Pairing

Let E be an elliptic curve over a field K_0 of characteristic p, and n be a positive integer such that $\gcd(p, n) = 1$. The set of n-th roots of unity is defined by $\mu_n = \{u \in \overline{K_0^*} \mid u^n = 1\}$, and $K = K_0(\mu_n)$ is defined by the extension of K_0.

Let $P \in E(K)[n]$ and $Q \in E(K)/nE(K)$ that has the order n. Since $nP = \mathcal{O}$, there exists a function f such that $(f) = n(P) - n(\mathcal{O})$. We construct the degree zero divisor $D = (Q + S) - (S)$ by choosing a suitable point $S \in E(K)$ such that the support of D is disjoint to the support of (f). Since f and D are defined over K, $f(D)$ is an element in K^*.

The Tate pairing of two points P and Q is defined as a map

$$\begin{aligned} \langle\ ,\ \rangle_n : E(K)[n] \times E(K)/nE(K) &\to K^*/(K^*)^n, \\ \langle P, Q\rangle_n &\mapsto f(D). \end{aligned}$$

Since the result of the Tate pairing is in $K^*/(K^*)^n$, The Tate pairing is not a well defined element in K^*.

Suppose that $K_0 = \mathbb{F}_q$ is a finite filed with q elements. Let E be an elliptic curve over $\mathbb{F}_q$, and let n be a positive integer such that $\gcd(n, q) = 1$ and $n \mid \#E(\mathbb{F}_q)$. Then $K = K_0(\mu_n)$ is some extension field $\mathbb{F}_{q^k}$. The quantity k is said to an embedding degree (or a MOV degree), and it is simply chosen as the least positive integer such that $n \mid (q^k - 1)$. We suppose that n is a large prime integer as below.

In the conditions $K_0 = \mathbb{F}_q$ and $K = \mathbb{F}_{q^k}$, the result of the Tate pairing is an element in equivalence class $\mathbb{F}_{q^k}^*/(\mathbb{F}_{q^k}^*)^n$. For practical usage, we need to compute the result as a unique representation. The natural way is to compute the power $(q^k - 1)/n$ of the result. Hence we denote the reduced Tate pairing as a map into $\mu_n \subset \mathbb{F}_{q^k}^*$,

$$e(P, Q) = \langle P, Q\rangle_n^{(q^k-1)/n},$$

where P is a n-torsion point of $E(\mathbb{F}_q)$ from $E(\mathbb{F}_{q^k})$, and Q is in $E(\mathbb{F}_{q^k})/nE(\mathbb{F}_{q^k})$.

2.4.3 Weil Pairing

Let E be an elliptic curve defined over a field K_0 and let n be an integer which is co-prime to the characteristic of K_0. The set $E[n]$ is isomorphic to $\mathbb{Z}_n \oplus \mathbb{Z}_n$. We define $K = K_0(E[n])$ as the field extension of K_0 generated by the coordinate of all points in $E(\overline{K})$ of which point order is divisible by n.

The Weil pairing is defined as a map

$$e_n : E[n] \times E[n] \to \mu_n \subseteq K^*.$$

Let P and Q be points in $E[n]$ and let D and D' be degree zero divisors such that the supports of D and D' are disjoint, and $D \sim (P)-(\mathcal{O})$ and $D' \sim (Q)-(\mathcal{O})$. Then there exist functions f and g such that $(f) = nD$ and $(g) = nD'$. The Weil pairing computed by

$$e_n(P, Q) = f(D')/g(D)$$

The Weil pairing satisfies the following additional properties with the basic properties, i.e. bilinearity and non-degeneracy.

Alternating: Let P, Q be points in $E[n]$. Then $e_n(P, P) = 1$ and so $e_n(P, Q) = e_n(Q, P)^{-1}$.

Compatibility If $P \in E[nm]$, where m is a positive integer, and $Q \in E[n]$, then $e_{nm}(P, Q) = e_n(mP, Q)$.

Let E be an elliptic curve over a finite field $\mathbb{F}_q$ and let r be a prime such that $r \mid \#E(\mathbb{F}_q)$. Suppose that $r \nmid (q-1)$ and $\gcd(r, q) = 1$. Then $E[r] \subset E(\mathbb{F}_{q^k})$ if and only if $r \mid q^k - 1$ for some embedded degree k.

2.4.4 Miller's Algorithm

Miller proposed an explicit polynomial-time algorithm to compute the Weil pairing, and this approach can apply to compute the Tate pairing. The Miller's algorithm is to construct a function f satisfies $(f) = r(P) - r(\mathcal{O})$ by a double-and-addition method.

Let P be a r-torsion point of $E(K)$. We then write the divisor of a function f_i is denoted by

$$(f_i) = i(P) - (iP) - (i-1)(\mathcal{O})$$

for $1 \leq i \leq r$. Then the divisor of f_r is

$$r(P) - (rP) - (r-1)(\mathcal{O}) = r(P) - r(\mathcal{O}).$$

The function f_i satisfies the following conditions.

1. $f_1 = 1$.
2. $f_{i+j} = f_i f_j \cdot \ell/v$ for the lines ℓ and v used in the addition of $iP + jP = (i+j)P$.

For the proof of the second condition, let $P_1 = iP$ and $P_2 = jP$. The line ℓ is through the points P_1 and P_2 (if $P_1 = P_2$, then ℓ is the tangent line at P_1), and the vertical line v passes through $P_3 = P_1 + P_2 = (i+j)P$. Then the divisors of each line on E are given by

$$\begin{aligned}(\ell) &= (P_1) + (P_2) + (-P_3) - 3(\mathcal{O}),\\ (v) &= (-P_3) + (P_3) - 2(\mathcal{O}).\end{aligned}$$

Therefore,

$$\begin{aligned}(f_i f_j \cdot \ell/v) &= (f_i) + (f_j) + (\ell) - (v)\\ &= i(P) - (iP) - (i-1)(\mathcal{O}) + j(P) - (jP) - (i-1)(\mathcal{O})\\ &\quad +(P_1) + (P_2) + (-P_3) - 3(\mathcal{O}) - (-P_3) - (P_3) + 2(\mathcal{O})\\ &= (i+j)(P) - (iP) - (jP) - (i+j-2)(\mathcal{O})\\ &\quad +(iP) + (jP) - ((i+j)P) - (\mathcal{O})\\ &= (i+j)(P) - ((i+j)P) - (i+j-1)(\mathcal{O})\\ &= (f_{i+j})\end{aligned}$$

When $2c = i + j$ is even, $f_{2c} = f_c^2 \cdot \ell/v$ such that ℓ is the tangent line at cP, and v is the vertical line through $2cP$. When $2c + 1 = i + j$ is odd, then $f_{2c+1} = f_{2c} \cdot f_1 \cdot \ell/v = f_{2c} \cdot \ell/v$ such that ℓ is the line which passes $2cP$ and P, and v is the vertical line through $(2c + 1)P$.

The Miller's algorithm is performed as Algorithm 2.3.

2.4.5 Distorsion Map

Let r be a prime, k be an embedded degree which is a least positive integer such that $r \mid (q^k - 1)$. Let P be a point of order r in $E(\mathbb{F}_q)$. There is an endomorphism ψ such that $\psi(P)$ lie in disjoint cyclic groups of order r in $E(\mathbb{F}_{q^k})$. It is called a distortion map on E.

The reduced Tate pairing is also given by $e(P, \psi(Q))$, where P, Q in $E(\mathbb{F}_q)[r]$ using the distortion map. If E is an elliptic curve over $\mathbb{F}_q$ which has the distortion map, then E is a supersingular elliptic curve.

2.4.6 Hard Problems of Pairings for Security

The security of cryptographic schemes is based on some hard problem such as integer factorization and discrete logarithm problems. Here we briefly show the basic hard problems for pairing-based cryptosystems.

Algorithm 2.3 Miller's algorithm

INPUT: $P, Q \in E(K)$, where the order of P is $r = (r_{t-1}, ..., r_0)$ in binary

OUTPUT: $\langle P, Q \rangle_r$

1: Choose a suitable point $S \in E(K)$
2: $Q' \leftarrow Q + S$
3: $T \leftarrow P$
4: $f \leftarrow 1$
5: **for** $i \leftarrow t-1$ **downto** 0 **do**
6: Calculate lines ℓ and v for doubling $2T$
7: $T \leftarrow 2T$
8: $f \leftarrow f^2 \cdot \frac{\ell(Q')v(S)}{v(Q')\ell(S)}$
9: **if** $r_i = 1$ **then**
10: Calculate lines ℓ and v for addition $T + P$
11: $T \leftarrow T + P$
12: $f \leftarrow f \cdot \frac{\ell(Q')v(S)}{v(Q')\ell(S)}$
13: **end if**
14: **end for**
15: **return** f

The basic hard problem in bilinear maps is Bilinear Diffie-Hellman problem proposed by Boneh and Franklin [28]. There are two types of the problems, the computational and decisional problems. Let $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_2$ be a bilinear pairing on an additive group $\mathbb{G}_1$, and multiplicative group $\mathbb{G}_2$ of prime order p generated by $P \in \mathbb{G}_1$ and $g \in \mathbb{G}_2$. We define two bilinear Diffie-Hellman problems as follows:

Computational Bilinear Diffie-Hellman problem. In the computational bilinear Diffie-Hellman, an adversary actually computes a specific value from a given distribution. The problem is defined as follows: given a tuple $(P, aP, bPcP) \in \mathbb{G}_1$ for uniform random elements in $\mathbb{G}_1$, the adversary computes $e(P, P)^{abc}$.

Decisional Bilinear Diffie-Hellman problem. In the decisional version of the bilinear Diffie-Hellman problem, the adversary distinguishes two values whether they are same one or not. The adversary gets a distribution (P, aP, bP, cP, Z) where $a, b, c \in \mathbb{Z}_p^*$ and $Z \in \mathbb{G}_2$ are chosen by independent and uniform random. Then he determines whether $Z = e(P, P)^{abc}$ or Z is random element in $\mathbb{G}_2$.

When the adversary who can solve the computational BDH problem obviously computes the decisional one, but in converse setting this does not imply that the computational BDH is easy. By the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ for which the decisional BDH is easy but the computational BDH is hard, the problem is said to be a gap bilinear Diffie-Hellman problem.

There exists several problems except for these basic problems such as bilinear Diffie-Hellman inversion problem. Thus we consider the discrete logarithm problems for input and output groups and the bilinear maps, with respect to the construction of the secure pairing-based cryptosystems.

Chapter 3

η_T Pairing over $\mathbb{F}_{3^m}$

3.1 Introduction

Miller's algorithm can compute the value of the Weil and Tate pairing in polynomial time [82, 81], and it can be adopted by many implementations with elliptic curves and parameters. However, this algorithm requires much computing time compared with general public key cryptography such as elliptic curve cryptography, and hence the pairing-based cryptosystem using the general Tate pairing is difficult to be applied in the practical cryptosystem. For the efficient algorithm to compute the Tate pairing, there are many construction such as η_T pairing [7], Ate pairing and its variations [57, 74, 110, 115]. The η_T pairing over $\mathbb{F}_{3^m}$ is the most typical construction as a variant of the Tate pairing.

In this chapter, we outline a construction of the η_T pairing which is an efficient computation of a pairing on supersingular elliptic curves over $\mathbb{F}_{3^m}$. Firstly, we show the Duursma-Lee algorithm [36] and η_T pairing [7] which improve the efficiency from the computation of the Tate pairing. Moreover we present the modified algorithms of the η_T pairing, such as the η_T pairing without cube root [24] and an universal η_T pairing [99], for efficiency and simple implementation. We also show an efficient final exponentiation of the η_T pairing using torus [101].

3.2 Duursma-Lee Algorithm

Let r be an integer such that $r \mid \#E(\mathbb{F}_q)$ and $r \mid (q^k - 1)$, where an elliptic curve E, a finite field $\mathbb{F}_q$ and an embedded degree k. Suppose that the reduced Tate pairing is $f_{r,P}(Q)^{\frac{q^k-1}{r}}$, for given $P \in E(\mathbb{F}_q)[r]$ and $Q \in E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$. Let

N be an integer such that $N \mid (q^k - 1)$ and $r \mid N$. Then we obtain the following property:

$$f_{r,P}(Q)^{\frac{q^k-1}{r}} = f_{N,P}(Q)^{\frac{q^k-1}{N}},$$

with respect to the reduced Tate pairing.

Duursma and Lee dealt with the efficient implementation on the hyperelliptic curve $y^2 = x^p - x + d$ in characteristic p such that $p \equiv 3 \bmod 4$ for the efficient Tate pairing [36]. However, we now describe the Duursma-Lee algorithm on supersingular elliptic curves in characteristic three.

Let $\mathbb{F}_{3^m}$ be a finite field of characteristic three, and m be an extension degree of $\mathbb{F}_{3^m}$ such that $\gcd(m, 6) = 1$. We apply the following supersingular elliptic curve defined over $\mathbb{F}_{3^m}$:

$$E : y^2 = x^3 - x + b,$$

where $b = \pm 1$. In this setting, the embedded degree k which satisfies $r \mid (3^{km} - 1)$ is equal to six. The basis of $\mathbb{F}_{3^{6m}}$ over $\mathbb{F}_{3^m}$ is given by $\{1, \sigma, \rho, \rho\sigma, \rho^2, \rho^2\sigma\}$, where $\rho^3 - \rho - b = 0$ and $\sigma^2 + 1 = 0$. The representation of the element A in $\mathbb{F}_{3^{6m}}$ is written by $A = a_0 + a_1\sigma + a_2\rho + a_3\rho\sigma + a_4\rho^2 + a_5\rho^2\sigma$, for $a_i \in \mathbb{F}_{3^m}$. Let $P = (x_P, y_P)$ be a point in $E(\mathbb{F}_{3^m})$ such that $P \neq 3P$. The point tripling can be efficiently computed by

$$3P = (x_P^9 - b, -y_P^9).$$

The 3-power Frobenius ϕ_3 is

$$\phi_3(x_P, y_P) \mapsto (x_P^3, y_P^3).$$

Let r be a prime such that $r \mid \#E(\mathbb{F}_{3^m})$. Since r satisfies $r \mid (3^{6m} - 1)$ and $r \nmid (3^{3m} - 1)$, hence, $r \mid (3^{3m} + 1)$. Then we can compute the reduced Tate pairing using $N = 3^{3m} + 1$ instead of r by:

$$f_{N,P}(Q)^{\frac{3^{6m}-1}{N}} = f_{N,P}(Q)^{3^{3m}-1}.$$

The equation of the line h_P, which passes through P and $-3P$ in $E(\mathbb{F}_{3^m})$, is given by

$$h_P(x, y) = y_P^3 y - (x_P^3 - x + b)^2$$

Then the divisor of h_P is denoted by

$$(h_P) = 3(P) + (-3P) - 4(\mathcal{O}).$$

Let v_P be a vertical line through P and $-P$. The divisor of v is $(v_P) = (P) + (-P) - 2(\mathcal{O})$. Therefore

$$\begin{aligned}(h_P/v_{3P}) &= (h_P) - (v_{3P}) \\ &= 3(P) + (-3P) - 4(\mathcal{O}) - (3P) - (-3P) + 2(\mathcal{O}) \\ &= 3(P) - (3P) - 2(\mathcal{O})\end{aligned}$$

We denote a divisor (g_P) as (h_P/v_{3P}). Then for an arbitrary $z \in \mathbb{N}$, we obtain

$$\begin{aligned}(g_P^{3^{z-1}} g_{3P}^{3^{z-2}} \cdots g_{3^{z-1}P}) &= 3^{z-1}(g_P) + 3^{z-2}(g_{3P}) + \cdots + (g_{3^{z-1}P}) \\ &= 3^z(P) - (3^zP) - (3^z - 1)(\mathcal{O}) \\ &= (g_{3^zP}) \\ &= (f_{3^z+1,P})\end{aligned}$$

Here we define a distortion map ψ of $E(\mathbb{F}_{3^m})[r]$ to a point of order r in $E(\mathbb{F}_{3^{6m}})$. For a given point $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})[r]$, the distortion map is computed as

$$\psi(P) = (\rho - x_P, -y_P\sigma).$$

Thus for $P \in E(\mathbb{F}_{3^m})[r]$ and $Q \in E(\mathbb{F}_{3^m})/rE(\mathbb{F}_{3^m})$, we obtain the result of the Tate pairing using $N = 3^{3m} + 1$ as

$$f_{N,P}(\psi(Q)) = \prod_{i=1}^{3m} g_{3^{i-1}P}^{3^{3m-i}}(\psi(Q)) = \prod_{i=1}^{3m} \left(\frac{h_{3^{i-1}P}(\psi(Q))}{v_{3^iP}(\psi(Q))} \right)^{3^{3m-i}}.$$

In this setting, since the embedded degree k is even and x-coordinate of $\psi(Q)$ is in $\mathbb{F}_{3^{2m}}$ that is a subfield of $\mathbb{F}_{3^{6m}}$, we omit the computation of v for computing the reduced Tate pairing [9]. Moreover for decreasing the number of loops, we get

$$(h_{3^{i-1}P}(\psi(Q)))^{3^{3m-i}} = (h_{3^{i+m-1}P}(\psi(Q)))^{3^{2m-i}} = (h_{3^{i+2m-1}P}(\psi(Q)))^{3^{m-i}},$$

for $1 \leq i \leq m$. Therefore, the reduced Tate pairing without final exponentiation can be computed as

$$\begin{aligned}f_{N,P}(\psi(Q)) &= \prod_{i=1}^{m} h_{\phi_3^i(P)} \left(\phi_3^{m+1-i}(\psi(-Q)) \right), \\ &= \prod_{i=1}^{m} \left(-y_P^{3^i} y_Q^{3^{m+1-i}} \sigma - (x_P^{3^i} + x_Q^{3^{m+1-i}} + b - \rho)^2 \right).\end{aligned}$$

In addition, if $Q \in E(\mathbb{F}_{3^m})$, then $\phi_3^m(Q) = Q$. As a result, the reduced Tate pairing using Duursma-Lee algorithm is computed by

$$(f_{N,P}(\psi(Q)))^{3^{3m}-1} = \left(\prod_{i=1}^{m} \left(-y_P^{3^i} y_Q^{3^{1-i}} \sigma - (x_P^{3^i} + x_Q^{3^{1-i}} + b - \rho)^2 \right) \right)^{3^{3m}-1}.$$

Algorithm 3.1 Duursma-Lee Algorithm for Tate Pairing over $\mathbb{F}_{3^m}$ [36]

INPUT: $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})[r], Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})/rE(\mathbb{F}_{3^m})$

OUTPUT: $\langle P, \psi(Q) \rangle_N \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^N$

1: $f \leftarrow 1$
2: **for** $i \leftarrow 0$ **to** $m-1$ **do**
3: $\quad x_P \leftarrow x_P^3, y_P \leftarrow y_P^3$
4: $\quad u \leftarrow x_P + x_Q + b$
5: $\quad g \leftarrow -y_P y_Q \sigma - \mu^2 - \mu\rho - \rho^2$
6: $\quad f \leftarrow fg$
7: $\quad x_Q \leftarrow x_Q^{1/3}, y_Q \leftarrow y_Q^{1/3}$
8: **end for**
9: **return** f

Algorithm 3.1 is computed by the Tate pairing using Duursma-Lee algorithm. For $\mathbb{F}_{2^m}$, Kwon proposed the modified algorithm used by this approach [72].

3.3 Construction of η_T Pairing over $\mathbb{F}_{3^m}$

Barreto et al. proposed η_T pairing over $\mathbb{F}_{3^m}$ that a generalized algorithm of the Duursma-Lee algorithm [7]. It decreases the number of loops in half of the Duursma-Lee algorithm. Thus the η_T pairing over $\mathbb{F}_{3^m}$ becomes about twice faster than the Duursma-Lee algorithm algorithm.

Suppose that $b' = b$ if $m \equiv \pm 1 \pmod{12}$, and $b' = -b$ if $m \equiv \pm 5 \pmod{12}$. Then the order $\#E(\mathbb{F}_{3^m})$ of the curve is given by $\#E(\mathbb{F}_{3^m}) = 3^m + b'3^{(m+1)/2} + 1$. Let T be an integer such that

$$T = -b'3^{(m+1)/2} - 1.$$

Then the η_T pairing is defined as

$$\eta_T(P,Q) = \begin{cases} f_{T,P}(\psi(Q)) & \textbf{if} \quad T > 0, \\ f_{-T,-P}(\psi(Q)) & \textbf{if} \quad T < 0. \end{cases}$$

The η_T pairing can be represented by the following closed form

$$\begin{aligned} \eta_T(P,Q) &= \ell_{3^{(m+1)/2}P, b'P}(\psi(Q)) f_{3^{(m+1)/2}+1, P}(\psi(Q)), \\ &= \ell_{3^{(m+1)/2}P, b'P}(\psi(Q)) \prod_{i=0}^{(m-1)/2} h_{3^i P}(\psi(Q))^{3^{(m-1)/2-i}}. \end{aligned}$$

For $P' = 3^{(m-1)/2}P$ and $j = (m-1)/2 - i$, it can be modified into:

$$\ell_{3P',b'P}(\psi(Q)) \prod_{j=0}^{(m-1)/2} h_{3^{-j}P'}(\psi(Q))^{3^j}.$$

With respect to the line $\ell_{3P',b'P}(x,y)$, the slope of ℓ is $\lambda = y_P$ if $m \equiv 7, 11 \pmod{12}$ and $\lambda = -y_P$ if $m \equiv 7, 11 \pmod{12}$. Then the equation of ℓ is $y = \lambda(x - x_P) + b'y_P$.

Here we define $\xi(x,y) = (x-b, -y) \in E(\mathbb{F}_{3^m})$. Then P' is transformed into

$$P' = \xi^{(m-1)/2}\phi_3^{m-1}P = \xi^{(m-1)/2}\phi_3^{-1}P = \xi^{(m-1)/2}(x_P^{1/3}, x_P^{1/3}).$$

In addition, in the case of $m \equiv 1 \pmod{12}$, we can calculate $h_{3^{-j}P'}(\psi(Q))^{3^j}$ by:

$$h_{3^{-j}P'}(\psi(Q))^{3^j} = \sigma y_P^{3^{-j}} y_Q^{3^j} - u^2 - u\rho - \rho^2,$$

where $u = x_P^{3^{-j}} + x_Q^{3^j} + b$.

The final exponentiation of the η_T pairing is

$$f^{(3^{6m}-1)/\#E(\mathbb{F}_{3^m})} = f^{(3^{3m}-1)(3^m+1)(3^m+1-b'3^{(m+1)/2})}$$

and it is in $\mathbb{F}_{3^{6m}}^*$. The relation between the reduced Tate pairing and η_T pairing is

$$(\eta_T(P,Q)^W)^{3T^2} = e(P,\psi(Q))^{-b'3^{(n+3)/2}}$$

for $W = (3^{6m}-1)/\#E(\mathbb{F}_{3^m})$.

The η_T pairing algorithm has some branches by chosen m. The η_T pairing performed as Algorithm 3.2 when $m \equiv \pm 1 \bmod 12$.

3.4 η_T Pairing over $\mathbb{F}_{3^m}$ without Cube Root

There exists an efficient algorithm for cube root computation in $\mathbb{F}_{3^m}$. However, it requires to choose suitable an irreducible polynomial and to precompute the result of $x^{1/3}$, $x^{2/3}$, with respect to the construction of the cube root. Thus it is to be desired that the cube root does not use. Beuchat et al. proposed the modified η_T pairing without the cube root computation [24].

By bilinearity of pairing, we obtain the following relation:

$$\left(\eta_T(P,Q)^W\right)^{3^m} = \left(\eta_T(3^{(m-1)/2}P, Q)^{3^{(m+1)/2}}\right)^W.$$

Then by calculating $3^{(m-1)/2}P$, we can remove the cube root computation. The η_T pairing without the cube root in $\mathbb{F}_{3^m}$ performed as Algorithm 3.3 when $m \equiv \pm 1 \bmod 12$.

Algorithm 3.2 η_T Pairing over $\mathbb{F}_{3^m}$ when $m = \pm 1 \bmod 12$

INPUT: $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})[r]$, $Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})/rE(\mathbb{F}_{3^m})$

OUTPUT: $\eta_T(P, Q) \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$

1: $P_0 \leftarrow -P_0$
2: **if** $T < 0$ **then** $P \leftarrow -P$
3: $\ell \leftarrow$ the line between $3^{(m+1)/2}P$ and P_0
4: $f \leftarrow \ell(\psi(Q))$
5: **for** $i \leftarrow 0$ **to** $(m-1)/2$ **do**
6: $\quad u \leftarrow x_P + x_Q + b$
7: $\quad g \leftarrow y_P y_Q \sigma - u^2 - u\rho - \rho^2$
8: $\quad f \leftarrow fg$
9: $\quad x_P \leftarrow x_P^{1/3}$, $y_P \leftarrow y_P^{1/3}$
10: $\quad x_Q \leftarrow x_Q^3$, $y_Q \leftarrow y_Q^3$
11: **end for**
12: **return** f

Algorithm 3.3 η_T Pairing over $\mathbb{F}_{3^m}$ w/o Cube Root when $m = \pm 1 \bmod 12$ [24]

INPUT: $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})[r]$, $Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})/rE(\mathbb{F}_{3^m})$

OUTPUT: $\eta_T(P, Q)^{3^{(m+1)/2}} \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$

1: $y_P \leftarrow -by_P$, $d \leftarrow b$
2: $f \leftarrow -y_P(x_P + x_Q + b) + y_Q\sigma + y_P\rho$
3: **for** $i \leftarrow 0$ **to** $(m-1)/2$ **do**
4: $\quad u \leftarrow x_P + x_Q + d$
5: $\quad g \leftarrow y_P y_Q \sigma - u^2 - u\rho - \rho^2$
6: $\quad f \leftarrow (fg)^3$
7: $\quad y_P \leftarrow -y_P$
8: $\quad x_Q \leftarrow x_Q^9$, $y_Q \leftarrow y_Q^9$
9: $\quad d \leftarrow (d - b) \bmod 3$
10: **end for**
11: **return** f

3.5 Universal η_T Pairing over $\mathbb{F}_{3^m}$

Since the η_T pairing has some branches for chosen m and b, we cannot explicitly write the algorithm of the η_T pairing. Shirase et al. proposed the universal η_T pairing, written $\widetilde{\eta_T}(P, Q)$, that is modified based on the η_T pairing without cube root [99]. Hence this algorithm has no branch, then we can implement it simply.

Algorithm 3.4 $\widetilde{\eta_T}$ Pairing over $\mathbb{F}_{3^m}$ [99]

INPUT: $P = (x_P, y_P) \in E(\mathbb{F}_{3^m})[r]$, $Q = (x_Q, y_Q) \in E(\mathbb{F}_{3^m})/rE(\mathbb{F}_{3^m})$

OUTPUT: $\widetilde{\eta_T}(P, Q) \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$

1: $f \leftarrow -y_P(x_P + x_Q + b) + y_Q\sigma + y_P\rho$
2: $d \leftarrow b$
3: **for** $i \leftarrow 0$ **to** $(m-1)/2$ **do**
4: $\quad u \leftarrow x_P + x_Q + d$
5: $\quad g \leftarrow y_P y_Q \sigma - u^2 - u\rho - \rho^2$
6: $\quad f \leftarrow (fg)^3$
7: $\quad y_P \leftarrow -y_P$
8: $\quad x_Q \leftarrow x_Q^9$, $y_Q \leftarrow y_Q^9$
9: $\quad d \leftarrow (d - b) \bmod 3$
10: **end for**
11: **return** f

Let Z be an integer $-b'3^{(n+3)/2}$. We relate the universal η_T pairing with the reduced Tate pairing by:

$$\widetilde{\eta_T}(P, Q)^W = e(P, \psi(Q))^U,$$

where $U = 3^{(m-1)/2}VZT^{-2} \bmod \#E(\mathbb{F}_{3^m})$, and V is given by the following table:

	$b = 1$	$b = -1$
$m \equiv 1 \pmod{12}$	-1	1
$m \equiv 5 \pmod{12}$	$3^{(m+1)/2} - 2$	$3^{(m+1)/2} + 2$
$m \equiv 7 \pmod{12}$	-1	1
$m \equiv 11 \pmod{12}$	$-3^{(m+1)/2} - 2$	$-3^{(m+1)/2} + 2$

The algorithm of the universal η_T pairing is shown in Algorithm 3.4.

3.6 Final Exponentiation of η_T Pairing over $\mathbb{F}_{3^m}$

In regard to the result $f \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$ of the Miller loop using the η_T pairing over $\mathbb{F}_{3^m}$, we require to compute a final exponentiation f^W with

$$W = (3^{3m} - 1)(3^m + 1)(3^m + 1 - b'3^{(m+1)/2}) = (3^{6m} - 1)/\#E(\mathbb{F}_{3^m}).$$

This exponentiation can be efficiently computed by using the properties of torus $T_2(\mathbb{F}_{3^{3m}})$ proposed by Shirase et al. [101].

We recall that the basis of $\mathbb{F}_{3^{6m}}$ over $\mathbb{F}_{3^{3m}}$ is denoted with $\{1, \sigma\}$ satisfies $\sigma^2 = -1$, and the element $A \in \mathbb{F}_{3^{6m}}$ is represented as $A = A_0 + A_1\sigma$. Granger et al. [53] introduced the torus $T_2(\mathbb{F}_{3^{3m}})$ for compressing the element of $\mathbb{F}_{3^{6m}}$. The torus $T_2(\mathbb{F}_{3^{3m}})$ is given by

$$T_2(\mathbb{F}_{3^{3m}}) = \{A_0 + A_1\sigma \in \mathbb{F}^*_{3^{6m}} \mid A_0^2 + A_1^2 = 1\}.$$

Then the element $A \in T_2(\mathbb{F}_{3^{3m}})$ can be compressed to the half by storing either the part of the element A_0 or A_1.

Here we use the torus $T_2(\mathbb{F}_{3^{3m}})$ for efficient final exponentiation. The torus $T_2(\mathbb{F}_{3^{3m}})$ has the following properties:

1. $A_0 - A_1\sigma = (A_0 + A_1\sigma)^{-1}$ for $A \in T_2(\mathbb{F}_{3^{3m}})$.
2. $(A_0 + A_1\sigma)^{3^{3m}-1} \in T_2(\mathbb{F}_{3^{3m}})$ for $A \in \mathbb{F}^*_{3^{6m}}$.

Let $f = f_0 + f_1\sigma$ be a result of Miller loop under the η_T pairing over $\mathbb{F}_{3^m}$. Firstly, the part of the final exponentiation $f^{(3^{3m}-1)}$ is computed as

$$g = f^{(3^{3m}-1)} = \frac{(f_0^2 - f_1^2) + f_0 f_1 \sigma}{f_0^2 + f_1^2}.$$

Then g is an element in $T_2(\mathbb{F}_{3^{3m}})$. The remains of the final exponentiation computes by

$$\begin{aligned} h &= g^{3^m+1}, \\ i &= h^{3^m+1} \cdot (h^{3^{(m+1)/2}})^{-b'}, \end{aligned}$$

and return $i \in \mathbb{F}^*_{3^{6m}}$. In $T_2(\mathbb{F}_{3^{3m}})$, A^{3^m+1} can compute few multiplication in $\mathbb{F}_{3^m}$, and A^{-1} can only compute the negation of A_1 for a given $A \in T_2(\mathbb{F}_{3^{3m}})$.

Refer to Algorithm 4.19, 4.20 for the explicit algorithm of the final exponentiation of the η_T pairing.

Chapter 4

The Detail of Implementation of η_T Pairing over $\mathbb{F}_{3^m}$

4.1 Introduction

An operation of cryptosystems requires to be computed as fast as possible, since encryption and decryption generally act in the background of main services. Finite fields $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ have been mostly utilized to construct cryptosystems, then many efficient algorithms to compute arithmetic in $\mathbb{F}_{2^m}$ and $\mathbb{F}_p$ have been proposed. However $\mathbb{F}_{3^m}$ was scarcely used until pairing-based cryptosystems was proposed. Now various pairings such as Tate pairing, Weil pairing [82, 81], η_T pairing [7], Ate pairing [57], Ate$_i$ pairing [115], R-ate pairing [74], Optimal pairing [110] have been proposed, and these are constructed by arithmetic in the finite fields and on the elliptic curves. η_T pairing is constructed on supersingular elliptic curves in characteristic two and three, and Ate, Ate$_i$, R-ate, Optimal pairings are used with ordinary elliptic curves such as Barreto-Naehrig curves [11]. The η_T pairing in characteristic three is one of the fastest pairing algorithms, and it computes based on the arithmetic in $\mathbb{F}_{3^m}$.

In this chapter, we introduce the efficient algorithms to implement the pairing based cryptosystems which used the η_T pairing in characteristic three. In order to construct these cryptosystems, we use arithmetic in the finite field $\mathbb{F}_{3^m}$ and its extension fields $\mathbb{F}_{3^{3m}}$, $\mathbb{F}_{3^{6m}}$, arithmetic on the supersingular elliptic curves $y^2 = x^3 - x \pm 1$, and the η_T pairing computation and its variation for efficiently computing.

4.2 Arithmetic in $\mathbb{F}_{3^m}$

4.2.1 Element Representation in $\mathbb{F}_{3^m}$

Let m be an extension degree in $\mathbb{N}$, $\mathbb{F}_3[x]$ be a polynomial ring over $\mathbb{F}_3$, and $f(x)$ be an irreducible polynomial of degree m over $\mathbb{F}_3$. Then a finite field $\mathbb{F}_{3^m}$ is represented as

$$\mathbb{F}_{3^m} \cong \mathbb{F}_3[x]/(f(x)).$$

For choosing the suitable irreducible polynomial, we assume that there is the following irreducible trinomial over $\mathbb{F}_3$:

$$f(x) = x^m + x^k + 2,$$

where m and $k \in \mathbb{N}$ and $0 < k < m$.

Let $\mathbb{F}_3$ be a ternary field consists of elements $\{0, 1, 2\}$. For given bits a_h and a_l in $\mathbb{F}_2$, an element a in $\mathbb{F}_3$ is represented by two bits:

$$a = (a_h, a_l) \in \mathbb{F}_3.$$

By using a polynomial representation, an element A in $\mathbb{F}_{3^m}$ is denoted as follows:

$$A = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_0$$

where every coefficient a_i of A is the element in $\mathbb{F}_3$ for $0 \leq i \leq m-1$. Then, the representation of the coefficient is $a_i = ((a_i)_h, (a_i)_l)$ by two-bit representation.

Here, by the bit-slice implementation [87], the element A in $\mathbb{F}_{3^m}$ is represented by using two bit-strings (A_H, A_L):

$$\begin{cases} A_H &= \big((a_{m-1})_h, (a_{m-2})_h, ..., (a_0)_h\big), \\ A_L &= \big((a_{m-1})_l, (a_{m-2})_l, ..., (a_0)_l\big). \end{cases}$$

The length of each bit-string is m bits, then the required memory for storing one element in $\mathbb{F}_{3^m}$ is $2m$ bits.

In the implementation, every element $A \in \mathbb{F}_{3^m}$ represented as the bit-strings (A_H, A_L) is stored by two array with the array size $N = \lceil m/W \rceil$, where W is the word size of the target processor, and mostly $W = 32, 64$. An array representation is the word representation as shown below:

$$\begin{cases} A &= \displaystyle\sum_{i=0}^{N-1} A[i], \\ A[i] &= a_{iW}x^{iW} + a_{iW+1}x^{iW+1} + \cdots + a_{iW+W-1}x^{iW+W-1}, \end{cases}$$

Algorithm 4.1 Addition in $\mathbb{F}_{3^m}$

INPUT: $A = (A_H, A_L), B = (B_H, B_L) \in \mathbb{F}_{3^m}$

OUTPUT: $C = (C_H, C_L) = A + B \in \mathbb{F}_{3^m}$

1: $t \leftarrow A_H \wedge B_L$

2: $C_H \leftarrow t \,\&\, (A_L \wedge B_H)$

3: $C_L \leftarrow (t \wedge B_H) \mid (A_L \wedge B_L)$

Algorithm 4.2 Subtraction in $\mathbb{F}_{3^m}$

INPUT: $A = (A_H, A_L), B = (B_H, B_L) \in \mathbb{F}_{3^m}$

OUTPUT: $C = (C_H, C_L) = A - B \in \mathbb{F}_{3^m}$

1: $t \leftarrow A_L \wedge B_L$

2: $C_H \leftarrow (t \wedge B_H) \,\&\, (A_H \wedge B_L)$

3: $C_L \leftarrow t \mid (A_H \wedge B_H)$

where the coefficient $a_j = 0$ for $j \geq m$. Each $A[i]$ of A is divided into two words which are the sets of W coefficients consist of *hi* and *lo* bits, then $2NW$ bits are required to store the element represented by the array representation.

For the implementation, we use the following assignment from $(\mathbb{F}_2)^2$ to $\mathbb{F}_3$,

$$\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}.$$

In this assignment, we can construct the addition and subtraction in $\mathbb{F}_3$ with six logical instructions, and they are the minimum constructions of them (see Chap. 5). On the other hand, the negative element cannot be only computed by bit replacement in this assignment. The negative element of $a \in \mathbb{F}_3$ is computed by $a_h \leftarrow a_h \wedge a_l$, and hence the additional cost of N logical instructions is required to compute the negative element in $\mathbb{F}_{3^m}$.

4.2.2 Addition and Subtraction $\mathbb{F}_{3^m}$

Let A and B be elements in $\mathbb{F}_{3^m}$. Then the addition $C = A + B$ in $\mathbb{F}_{3^m}$ is calculated by adding the coefficients with the same degree of each polynomial $C = \sum_{i=0}^{m-1}(a_i + b_i)x^i$. Algorithm 4.1 shows the algorithm to compute the addition.

The subtraction $C = A - B$ in $\mathbb{F}_{3^m}$ can easily compute as $C = A + (-B)$, however the additional cost is required for the negative element computation in the assignment. Therefore, we optimize the subtraction algorithm in $\mathbb{F}_{3^m}$. The subtraction in $\mathbb{F}_{3^m}$ is performed by using Algorithm 4.2.

Algorithm 4.3 Shift-Addition Method for Multiplication in $\mathbb{F}_{3^m}$

INPUT: $A, B \in \mathbb{F}_{3^m}$

OUTPUT: $C' = A \cdot B \in \mathbb{F}_3[x]$

1: $C' \leftarrow 0$
2: **for** $i \leftarrow 0$ **to** $m-1$ **do**
3: $\quad C' \leftarrow C' + b_i A x^i$
4: **end for**

Algorithm 4.4 Right-to-Left Comb Method for Multiplication in $\mathbb{F}_{3^m}$

INPUT: $A, B \in \mathbb{F}_{3^m}$

OUTPUT: $C' = A \cdot B \in \mathbb{F}_3[x]$

1: $C' \leftarrow 0$
2: **for** $j \leftarrow 0$ **to** $W-1$ **do**
3: $\quad$ **for** $i \leftarrow 0$ **to** $N-1$ **do**
4: $\quad\quad C' \leftarrow C' + b_{iW+j} A x^{iW}$
5: $\quad$ **end for**
6: $\quad$ **if** $j \neq W-1$ **then**
7: $\quad\quad A \leftarrow Ax$
8: $\quad$ **end if**
9: **end for**

4.2.3 Multiplication in $\mathbb{F}_{3^m}$

For the elements A, B in $\mathbb{F}_{3^m}$ and the irreducible polynomial $f(x)$, the multiplication $C = A \cdot B \bmod f(x)$ are computed by two steps, a polynomial multiplication and a reduction. The polynomial multiplication computes the polynomial C' of degree $2(m-1)$ in $\mathbb{F}_3[x]$, and it is constructed by using the addition and subtraction in $\mathbb{F}_{3^m}$ and the shift operation Ax^i that computed by $\sum_{j=0}^{m-1} a_j x^{j+i}$ from x^i and $A = \sum_{j=0}^{m-1} a_j x^j$.

A simple algorithm to compute the polynomial multiplication is shift-addition method, performed as $C' \leftarrow C' + b_i A x^i$ for $0 \leq i < m$. Moreover comb method is the algorithm which reduces the shift compared with the shift-addition method. There are two variants; right-to-left and left-to-right comb methods. By the shift Tx^i, its word size shift operation Tx^{jW+i} can be computed virtually free, so the comb method is faster than the shift-addition method. These polynomial multiplication methods are executed by Algorithms 4.3, 4.4 and 4.5.

Algorithm 4.5 Left-to-Right Comb Method for Multiplication in $\mathbb{F}_{3^m}$

INPUT: $A, B \in \mathbb{F}_{3^m}$

OUTPUT: $C' = A \cdot B \in \mathbb{F}_3[x]$

1: $C' \leftarrow 0$
2: **for** $j \leftarrow W-1$ **downto** 0 **do**
3: **for** $i \leftarrow 0$ **to** $N-1$ **do**
4: $C' \leftarrow C' + b_{iW+j} A x^{iW}$
5: **end for**
6: **if** $j \neq 0$ **then**
7: $C' \leftarrow C'x$
8: **end if**
9: **end for**

Window method. In the Window method, we precompute $U \cdot A$ for all polynomials U with degree less then w in $\mathbb{F}_3[x]$, and store the results using some memory. During the evaluation of B in the polynomial multiplication, we scan w coefficients $(b_i, ..., b_{i+w-1})$ at once, instead of b_i, and compute the addition using the on-line precomputed table. Therefore the number of $\mathbb{F}_{3^m}$-additions can be reduced by the window method. We deploy it in conjunction with the shift-addition and left-to-right comb methods. Right-to-left comb method cannot use it, since A is changed in the algorithm.

For the window width w, the precomputed table of $A \in \mathbb{F}_{3^m}$ is written by:

$$\begin{cases} \widetilde{A_w} & = & \{A_{\langle 0 \rangle}, A_{\langle 1 \rangle}, ..., A_{\langle 3^w-1 \rangle}\} \\ A_{\langle u \rangle} & = & A \cdot (u_{w-1}x^{w-1} + u_{w-2}x^{w-2} + \cdots + u_0) \in \mathbb{F}_3[x] \\ & & (u = \sum_{i=0}^{w-1} u_i 3^i,\ u_i \in \mathbb{F}_3,\ 0 \leq u < 3^w) \end{cases}$$

The computational cost of the precomputation is $(3^w - w)/2$ additions, $(3^w - 1)/2$ negative elements, and $w - 1$ shifts in $\mathbb{F}_{3^m}$ by the window width w. The number of the $\mathbb{F}_{3^m}$-additions in the polynomial multiplication with the window method is about $1/w$ times smaller than that without the window method. The required memory to store the precomputed table is about $3^w \cdot 2NW$ bits. Hence we apply the suitable window width for computing the multiplication. The precomputation algorithm and the left-to-right comb method with the window method show in Algorithms 4.6 and 4.7, respectively.

Algorithm 4.6 Precomputation for Window Method

INPUT: $A \in \mathbb{F}_{3^m}$, Window width w

OUTPUT: Precomputed table $\widetilde{A_w}$

1: $A_{\langle 0 \rangle} \leftarrow 0$, $A_{\langle 1 \rangle} \leftarrow A$, $A_{\langle 2 \rangle} \leftarrow -A$
2: **for** $j \leftarrow 1$ **to** w **do**
3: $\quad A_{\langle 3^j \rangle} \leftarrow Ax^j$, $A_{\langle 2 \cdot 3^j \rangle} \leftarrow -Ax^j$
4: $\quad$ **for** $i \leftarrow 1$ **to** $3^j - 1$ **do**
5: $\quad\quad A_{\langle 3^j + i \rangle} \leftarrow A_{\langle 3^j \rangle} + A_{\langle i \rangle}$
6: $\quad\quad A_{\langle 2 \cdot 3^j + 3^{j-1}(2i_{j-1} \bmod 3) + \cdots + (2i_0 \bmod 3) \rangle} \leftarrow -A_{\langle 3^j + i \rangle}$
$\quad\quad$ ($i = (i_{j-1}, ..., i_0)$ in ternary)
7: $\quad$ **end for**
8: **end for**

Algorithm 4.7 LR-Comb Multiplication with Window Method in $\mathbb{F}_{3^m}$

INPUT: $A, B \in \mathbb{F}_{3^m}$, Window width w

OUTPUT: $C' = A \cdot B \in \mathbb{F}_3[x]$

1: Precompute $\widetilde{A_w}$
2: $C' \leftarrow 0$
3: **for** $j \leftarrow \lceil W/w \rceil - 1$ **downto** 0 **do**
4: $\quad$ **for** $i \leftarrow 0$ **to** $N - 1$ **do**
5: $\quad\quad b' \leftarrow b_{iW+jw} + \cdots + 3^{w-1} \cdot b_{iW+jw+w-1}$
$\quad\quad$ ($b_k \in B[i]$ for $iW + jw \leq k \leq iW + jw + w - 1$. $b_k = 0$ if $k \geq (i+1)W$)
6: $\quad\quad C' \leftarrow C' + A_{\langle b' \rangle} x^{iW+jw}$
7: $\quad$ **end for**
8: $\quad$ **if** $j \neq 0$ **then** $C' \leftarrow C'x^w$
9: **end for**

4.2.4 Reduction

The reduction in $\mathbb{F}_{3^m}$ is the modular computation by the irreducible polynomial $f(x)$, that calculates the polynomial $C' = \sum_{i=0}^{t} c'_i x^i$ of degree $t \geq m$ in $\mathbb{F}_3[x]$ to the element of the polynomial representation in $\mathbb{F}_{3^m}$. The straightforward reduction algorithm with the irreducible trinomial $f(x) = x^m + x^k + 2$ is Algorithm 4.8.

If $f(x)$ is an irreducible trinomial and $m - k \geq W$, the reduction can be efficiently computed by operating W coefficients at once. The input C' by the array representation, the word of C' is $C'[j] = \sum_{i=jW}^{jW+(W-1)} c'_i x^i$. Then the coefficient of a maximal degree $x^{jW+(W-1)}$ of $C'[j]$ is computed to the coefficients of

Algorithm 4.8 Reduction in $\mathbb{F}_{3^m}$

INPUT: $C' = \sum_{i=0}^{t} c'_i x^i \in \mathbb{F}_3[x]$ of degree $t \geq m$, $f(x) = x^m + x^k + 2$
OUTPUT: $C = C' \bmod f(x) \in \mathbb{F}_{3^m}$

1: **for** $i \leftarrow t$ **downto** m **do**
2: $\quad c'_{i-m+k} \leftarrow c'_{i-m+k} - c'_i$
3: $\quad c'_{i-m} \leftarrow c'_{i-m} + c'_i$
4: $\quad c'_i \leftarrow 0$
5: **end for**
6: **return** C'

$x^{jW+(W-1)-m+k}$ and $x^{jW+(W-1)-m}$. If $m - k \geq W$, then $x^{jW+(W-1)-m+k}$ is smaller than a minimum degree x^{jW} of $C'[j]$. Thus W coefficients in $C'[j]$ can be computed at the same time.

4.2.5 Cubing in $\mathbb{F}_{3^m}$

For a given $A \in \mathbb{F}_{3^m}$, the cubing in $\mathbb{F}_{3^m}$ can be computed by:

$$A^3 = (a_{m-1}x^{3(m-1)} + \cdots + a_1 x^3 + a_0) \bmod f(x).$$

We compute it by the bit-expansion and reduction. The bit-expansion is efficiently computed by using t–$3t$ lookup table which outputs $\sum_{i=0}^{t-1} a_i x^{3i}$ by the input $\sum_{i=0}^{t-1} a_i x^i$, where $a_i \in \mathbb{F}_3$. If $W = 32, 64$, each word of the array, which consist of *hi* and *lo* bit-strings, can be computed by only 2 or 4 table lookup with the 11–33 lookup table, respectively. The size of the 11–33 lookup table is approximately $2^{11} \cdot 31$ bits.

4.2.6 Inversion in $\mathbb{F}_{3^m}$

The inversion in $\mathbb{F}_{3^m}$ computes A^{-1} satisfied with $A \cdot A^{-1} = 1$ for $A \in \mathbb{F}_{3^m}$. In order to compute the inversion in $\mathbb{F}_{3^m}$, there are some algorithms, especially the extended Euclidean algorithm and binary GCD algorithm. Here, we use the ternary GCD algorithm over $\mathbb{F}_3$, that is a modified algorithm of the binary GCD algorithm. Let $\deg()$ be a function to compute a degree of a polynomial over $\mathbb{F}_3$, then the inversion using the ternary GCD is performed by Algorithm 4.9.

Algorithm 4.9 Ternary GCD for Inversion in $\mathbb{F}_{3^m}$

INPUT: $A \in \mathbb{F}_{3^m}$, an irreducible polynomial $f(x) = x^m + x^k + 2$

OUTPUT: $A^{-1} \in \mathbb{F}_{3^m}$

1: $U \leftarrow A$, $V \leftarrow f(x)$, $U' \leftarrow 1$, $V' \leftarrow 0$
2: **while** $deg(U) \neq 0$ **do**
3: **while** $v_0 = 0$ **do**
4: $V' \leftarrow V' + v'_0 f(x)$
5: $V \leftarrow V/x$, $V' \leftarrow V'/x$
6: **end while**
7: **while** $u_0 = 0$ **do**
8: $U' \leftarrow U' + u'_0 f(x)$
9: $U \leftarrow U/x$, $U' \leftarrow U'/x$
10: **end while**
11: **if** $\deg(V) \geq \deg(U)$ **then**
12: $V \leftarrow V - (u_0 v_0)U$, $V' \leftarrow V' - (u_0 v_0)U'$
13: **else**
14: $U \leftarrow U - (u_0 v_0)V$, $U' \leftarrow U' - (u_0 v_0)V'$
15: **end if**
16: **end while**
17: **return** $u_0 U'$

4.2.7 Square Root in $\mathbb{F}_{3^m}$

Assuming that the element $A \in \mathbb{F}_{3^m}$ is a quadratic residue and m is odd. Then the square root of A is computed by

$$\sqrt{A} = A^{(3^m+1)/4}.$$

The exponent $(3^m + 1)/4$ of A can be modified into $6 \cdot \sum_{i=0}^{k-1} 3^{2i} + 1$, where $k = (m-1)/2$. Then $A^{(3^m+1)/4}$ is computed as

$$A^{(3^m+1)/4} = A \cdot (A^{\sum_{i=0}^{k-1} 3^{2i}})^6.$$

In order to compute $A^{\sum_{i=0}^{k-1} 3^{2i}}$ efficiently, we execute recursively the following calculation by using an analogy of the Itoh-Tsujii inversion [60]:

$$A^{\sum_{i=0}^{k-1} u^i} = \begin{cases} (A^{\sum_{i=0}^{\lfloor k/2 \rfloor - 1} u^i}) \cdot (A^{\sum_{i=0}^{\lfloor k/2 \rfloor - 1} u^i})^{u^{\lfloor k/2 \rfloor}} & (\textbf{if } k \text{ is even}), \\ A \cdot ((A^{\sum_{i=0}^{\lfloor k/2 \rfloor - 1} u^i}) \cdot (A^{\sum_{i=0}^{\lfloor k/2 \rfloor - 1} u^i})^{u^{\lfloor k/2 \rfloor}})^u & (\textbf{if } k \text{ is odd}). \end{cases}$$

The square root in $\mathbb{F}_{3^m}$ is computed by Algorithm 4.10.

Algorithm 4.10 Square root in $\mathbb{F}_{3^m}$ [9]

INPUT: $A \in \mathbb{F}_{3^m}$ such that A is a quadratic residue,
$k = (m-1)/2 = (k_{n-1}, ..., k_0)$ in binary

OUTPUT: $A^{1/2} \in \mathbb{F}_{3^m}$

1: $S \leftarrow A$
2: $k' \leftarrow 1 \in \mathbb{N}$
3: **for** $i \leftarrow n-2$ **downto** 0 **do**
4: $\quad T \leftarrow S$
5: $\quad$ **for** $j \leftarrow 0$ **to** $k'-1$ **do**
6: $\quad\quad T \leftarrow T^{3^2}$
7: $\quad$ **end for**
8: $\quad k' \leftarrow 2k' + k_i$
9: $\quad S \leftarrow S \cdot T$
10: $\quad$ **if** $k_i = 1$ **then**
11: $\quad\quad S \leftarrow A \cdot S^{3^2}$
12: $\quad$ **end if**
13: **end for**
14: $S \leftarrow A \cdot S^6$
15: **return** S

4.2.8 Cube Root in $\mathbb{F}_{3^m}$

With respect to the computation of the cube root for an element A in $\mathbb{F}_{3^m}$, one can exponentiate the element to 3^{m-1},

$$\sqrt[3]{A} = A^{3^{m-1}}.$$

However this computation requires $m-1$ cubings and some additions in $\mathbb{F}_{3^m}$. Although the cubing can be efficiently computed in $\mathbb{F}_{3^m}$, the cube root of this algorithm is not efficient. The more efficient algorithm for computing the cube root is proposed by Barreto [6]. Let $m = 3u + r$ where $u = \lfloor 3/m \rfloor$ and $r = m \bmod 3$. Suppose that a_i for $i \geq m$ is equal to zero. We can transform $A \in \mathbb{F}_{3^m}$ into

$$A = \sum_{i=0}^{u} a_{3i} x^{3i} + x \cdot \sum_{i=0}^{u} a_{3i+1} x^{3i} + x^2 \cdot \sum_{i=0}^{u} a_{3i+2} x^{3i}.$$

Then we obtain the cube root of A by:

$$\sqrt[3]{A} = A^{3^{m-1}} = \sum_{i=0}^{u} a_{3i} x^i + x^{1/3} \sum_{i=0}^{u} a_{3i+1} x^i + x^{2/3} \sum_{i=0}^{u} a_{3i+2} x^i.$$

When $A'_0 = \sum_{i=0}^{u} a_{3i}x^i$, $A'_1 = \sum_{i=0}^{u} a_{3i+1}x^i$, and $A'_2 = \sum_{i=0}^{u} a_{3i+2}x^i$, the element in $\mathbb{F}_{3^m}$ can be written in

$$\sqrt[3]{A} = A'_0 + A'_1 x^{1/3} + A'_2 x^{2/3}.$$

The values $x^{1/3}$ and $x^{2/3}$ are fixed by the degree m and irreducible polynomial $f(x)$. Therefore these are precomputed and stored in the off-line memory. The values A'_0, A'_1, A'_2 are made by using the logical instructions. The computational cost of this computation is virtually for free. Hence the computational cost of the cube root in $\mathbb{F}_{3^m}$ is about 2 multiplications in $\mathbb{F}_{3^m}$. Moreover by choosing the suitable irreducible polynomial, $x^{1/3}$ and $x^{2/3}$ become sparse polynomials over $\mathbb{F}_3[x]$. Then the cube root can be computed by only some additions in $\mathbb{F}_{3^m}$.

4.3 Arithmetic in $\mathbb{F}_{3^{3m}}$

In our implementation, an extension field $\mathbb{F}_{3^{6m}}$ is constructed by a tower of field extensions of $\mathbb{F}_{3^m}$ via $\mathbb{F}_{3^{3m}}$, and then an extension field $\mathbb{F}_{3^{3m}}$ is constructed by a field extension of $\mathbb{F}_{3^m}$. In this section, we describe the detail of the element representation and the arithmetic in $\mathbb{F}_{3^{3m}}$.

Let $g(\rho) = (\rho^3 - \rho - b_3)$ be an irreducible polynomial over $\mathbb{F}_{3^m}$, where $b_3 = \pm 1$. The field extension $\mathbb{F}_{3^{3m}}$ of $\mathbb{F}_{3^m}$ is denoted by

$$\mathbb{F}_{3^{3m}} \cong \mathbb{F}_{3^m}[\rho]/(g(\rho)).$$

Note that b_3 is suitably chosen by the constant value of the used supersingular elliptic curve for constructing the η_T pairing. Here, the element $A_\rho \in \mathbb{F}_{3^{3m}}$ is represented as:

$$A_\rho = A_2\rho^2 + A_1\rho + A_0 \ (A_0, A_1, A_2 \in \mathbb{F}_{3^m})$$

All operations in $\mathbb{F}_{3^{3m}}$ are constructed by the arithmetic in $\mathbb{F}_{3^m}$. We list the arithmetic in $\mathbb{F}_{3^{3m}}$ below.

Addition and subtraction. Let $A_\rho, B_\rho \in \mathbb{F}_{3^{3m}}$. The addition and subtraction in $\mathbb{F}_{3^{3m}}$ are calculated by

$$\begin{aligned} A_\rho + B_\rho &= (A_2 + B_2)\rho^2 + (A_1 + B_1)\rho + (A_0 + B_0), \\ A_\rho - B_\rho &= (A_2 - B_2)\rho^2 + (A_1 - B_1)\rho + (A_0 - B_0). \end{aligned}$$

Multiplication. For a given $A_\rho, B_\rho \in \mathbb{F}_{3^{3m}}$, we firstly compute $t_{00} = A_0B_0$, $t_{11} = A_1B_1$, $t_{22} = A_2B_2$, $t_{01} = (A_0+A_1)(B_0+B_1)$, $t_{02} = (A_0+A_2)(B_0+B_2)$, $t_{12} = (A_1 + A_2)(B_1 + B_2)$ as auxiliary values. Then the multiplication in $\mathbb{F}_{3^{3m}}$ is computed by

$$\begin{aligned} A_\rho B_\rho &= (t_{02} + t_{11} - t_{00})\rho^2 \\ &\quad +(t_{01} + t_{12} + t_{11} - t_{00} + (b_3 - 1)t_{22})\rho \\ &\quad +(t_{00} + b_3(t_{12} - t_{11} - t_{22})). \end{aligned}$$

Cuging. The cubing in $\mathbb{F}_{3^{3m}}$ is calculated for a given element $A \in \mathbb{F}_{3^{3m}}$ by

$$A_\rho^3 = A_2^3\rho^2 + (A_1^3 - b_3A_2^3)\rho + (A_0^3 + b_3A_1^3 + A_2^3).$$

Inversion. The explicit algorithm to compute the inversion in $\mathbb{F}_{3^{3m}}$ is shown by Kerins et al. [68]. For a given $A_\rho \in \mathbb{F}_{3^{3m}}$, we compute $t_{00} = A_0^2$, $t_{11} = A_1^2$, $t_{22} = A_2^2$, $t_{01} = A_0A_1$, $t_{02} = A_0A_2$. The inversion $B_\rho = B_0+B_1\rho+B_2\rho^2 = A_\rho^{-1}$ is calculated by

$$\begin{cases} \lambda &= (A_0 - A_2)t_{00} + (-A_0 + b_3A_1)t_{11} + (A_0 - b_2A_1 + A_2)t_{22}, \\ B_0 &= \lambda^{-1}(t_{00} - t_{11} + t_{22} - A_2(A_0 + b_3A_1)), \\ B_1 &= \lambda^{-1}(-t_{01} + b_3t_{22}), \\ B_2 &= \lambda^{-1}(t_{11} - t_{02} - t_{22}). \end{cases}$$

4.4 Arithmetic in $\mathbb{F}_{3^{6m}}$

The extension field $\mathbb{F}_{3^{6m}}$ is constructed by the field extension of $\mathbb{F}_{3^{3m}}$. In this section, we explain the element representation and arithmetic in $\mathbb{F}_{3^{6m}}$.

Let $h(\sigma) = \sigma^2 + 1$ be an irreducible polynomial over $\mathbb{F}_{3^{3m}}$, then the extension field $\mathbb{F}_{3^{6m}}$ is denoted by

$$\mathbb{F}_{3^{6m}} \cong \mathbb{F}_{3^{3m}}[\sigma]/(h(\sigma)).$$

Moreover the element $A_\sigma \in \mathbb{F}_{3^{6m}}$ is represented as follows:

$$\begin{cases} A_\sigma &= A_{\rho,1}\sigma + A_{\rho,0} \\ &= A_5\rho^2\sigma + A_4\rho^2 + A_3\rho\sigma + A_2\rho + A_1\sigma + A_0 \\ &= (A_0, A_1, A_2, A_3, A_4, A_5) \end{cases}$$

where $A_{\rho,i} \in \mathbb{F}_{3^{3m}}$ for $i = 0, 1$ and $A_i \in \mathbb{F}_{3^m}$ for $0 \leq i < 6$. Now we explain the detail of the arithmetic in $\mathbb{F}_{3^{6m}}$ below:

Addition and subtraction. Let $A_\sigma, B_\sigma \in \mathbb{F}_{3^{6m}}$, then the addition and subtraction in $\mathbb{F}_{3^{6m}}$ are computed by

$$\begin{aligned} A_\sigma + B_\sigma &= (A_{\rho,1} + B_{\rho,1})\sigma + (A_{\rho,0} + B_{\rho,0}), \\ A_\sigma - B_\sigma &= (A_{\rho,1} - B_{\rho,1})\sigma + (A_{\rho,0} - B_{\rho,0}). \end{aligned}$$

Multiplication. Let $A_\sigma, B_\sigma \in \mathbb{F}_{3^{6m}}$. We compute the auxiliary values $t_{00} = A_{\rho,0}B_{\rho,0}$, $t_{11} = A_{\rho,1}B_{\rho,1}$, $t_{01} = (A_{\rho,0} + A_{\rho,1})(B_{\rho,0} + B_{\rho,1})$, and then the multiplication is computed by

$$A_\sigma B_\sigma = (t_{01} - t_{00} - t_{11})\sigma + (t_{00} - t_{11}).$$

Cubing. The cubing is computed as follows for a given $A_\sigma \in \mathbb{F}_{3^{6m}}$:

$$A_\sigma^3 = -A_{\rho,1}^3\sigma + A_{\rho,0}^3.$$

Inversion. For a given $A_\sigma \in \mathbb{F}_{3^{6m}}$, the inversion $B_\sigma = B_{\rho,1}\sigma + B_{\rho,0} = A_\sigma^{-1}$ is computed by

$$\begin{cases} \lambda &= A_{\rho,0}^2 + A_{\rho,1}^2, \\ B_{\rho,0} &= \lambda^{-1}A_{\rho,0}, \\ B_{\rho,1} &= -\lambda^{-1}A_{\rho,1}. \end{cases}$$

Exponentiation. With respect to the exponentiation in $\mathbb{F}_{3^{6m}}$, the simple algorithm is a triple-and-multiply method, that computes A_σ^d with an input element $A \in \mathbb{F}_{3^{6m}}$ and a t-digit ternary exponent $d \in \mathbb{N}$, written $d = \sum_{i=0}^{t-1} d_i 3^i$ ($d_{t-1} \neq 0$, $d_i \in \mathbb{F}_3$ for $0 \leq i < t$). Moreover, for the exponentiation, there are some efficient algorithms by converting the exponent and precomputing like the window method of the multiplication in $\mathbb{F}_{3^m}$. Here, we use a sliding window method for the efficient exponentiation in $\mathbb{F}_{3^{6m}}$. The digit set of the sliding window method with the window width w is

$$D_{SW}^w = \{0, 1, 2, ..., (3^w - 1)\} \backslash \{3, 6, ..., (3^w - 3)\}.$$

We precompute the elements A^i for all $i \in D_{SW}^w$. We then convert the exponent d into the sliding window form, and compute the exponentiation in $\mathbb{F}_{3^{6m}}$ using the precomputed table and the exponent of the sliding window form.

The conversion of the positive integer into the sliding window form and the triple-and-multiply exponentiation with the sliding window method in $\mathbb{F}_{3^{6m}}$ show in Algorithms 4.11 and 4.12.

Algorithm 4.11 Conversion of Positive Integer into Sliding Window Form

INPUT: A positive integer d, a window width w

OUTPUT: A sliding window form of d: $d' = (..., d'_1, d'_0)$

1: $i \leftarrow 0$
2: **while** $d > 0$ **do**
3: **if** $d \bmod 3 = 0$ **then**
4: $d'_i \leftarrow 0$
5: **else**
6: $d'_i \leftarrow d \bmod 3^w$
7: **end if**
8: $d \leftarrow (d - d'_i)/3$, $i \leftarrow i + 1$
9: **end while**
10: **return** $d' = (..., d'_1, d'_0)$

4.5 Arithmetic on Supersingular Elliptic Curves in Characteristic Three

In order to construct the η_T pairing over $\mathbb{F}_{3^m}$, we use supersingular elliptic curves in characteristic three. The elliptic curves we use are:

$$E: \ y^2 = x^3 - x + b_3 \ (b_3 = \pm 1),$$

and the set of the rational points of the elliptic curves defined by $\mathbb{F}_{3^m}$ in Affine coordinate is as follows:

$$E(\mathbb{F}_{3^m}) = \{(x, y) \in (\mathbb{F}_{3^m})^2 \mid y^2 = x^3 - x + b_3, \ b_3 = \pm 1\} \cup \{\mathcal{O}\}$$

The point on the curve is written by $P = (x, y) \in E(\mathbb{F}_{3^m})$ $(x, y \in \mathbb{F}_{3^m})$ in Affine coordinate.

We assume the extension degree m of the finite field $\mathbb{F}_{3^m}$ is coprime to six. For fix the order of the elliptic curve, we set b' by $m \equiv \pm 1, \pm 5 \pmod{12}$,

$$b' = \begin{cases} b_3 & \textbf{if} \quad m \equiv \pm 1 \pmod{12}, \\ -b_3 & \textbf{if} \quad m \equiv \pm 5 \pmod{12}. \end{cases}$$

Then the order $\#E(\mathbb{F}_{3^m})$ of the elliptic curve is given by

$$\#E(\mathbb{F}_{3^m}) = 3^m + b' 3^{(m+1)/2} + 1.$$

With respect to the secure η_T pairing, we choose the suitable parameters m and b_3 which $\#E(\mathbb{F}_{3^m})$ has a large prime integer r. As follows, we list the explicit algorithms to compute the arithmetic over $E(\mathbb{F}_{3^m})$.

Algorithm 4.12 Triple-and-Multiply Exponentiation with Sliding Window Method in $\mathbb{F}_{3^{6m}}$

INPUT: $A \in \mathbb{F}_{3^{6m}}$, a ternary integer d

OUTPUT: A^d

1: Compute sliding window form $d' = (d'_{t'-1}, ..., d'_0)$ using Algorithm 4.11
2: Compute $U_i \leftarrow A^i$ for all $i \in D^w_{SW}$
3: $T \leftarrow 1$
4: **for** $i \leftarrow t' - 1$ **downto** 0 **do**
5: $\quad T \leftarrow T^3$
6: $\quad$ **if** $d'_i \neq 0$ **then**
7: $\quad\quad T \leftarrow T \cdot U_{d'_i}$
8: $\quad$ **end if**
9: **end for**
10: **return** T

MapToPoint. The MapToPoint is one of the hash function to compute the point from the x or y-coordinate. The efficient MapToPoint algorithm shows in Algorithm 6.3 of Sect. 6.3.

Point addition. Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ be points in $E(\mathbb{F}_{3^m})$ such that $P \neq \pm Q$. Then the point addition $R = (x_3, y_3) = P + Q$ is computed by

$$\begin{cases} \lambda &= \dfrac{y_2 - y_1}{x_2 - x_1}, \\ x_3 &= \lambda^2 - (x_1 + x_2), \\ y_3 &= (y_1 + y_2) - \lambda^3. \end{cases}$$

Point doubling. For a given $P = (x_1, y_1) \in E(\mathbb{F}_{3^m})$, where $P \neq -P$, the point doubling $R = (x_3, y_3) = 2P$ is computed as follows:

$$\begin{cases} \lambda &= y_1^{-1}, \\ x_3 &= \lambda^2 + x_1, \\ y_3 &= -(\lambda^3 + y_1). \end{cases}$$

Point tripling. Let $P = (x_1, y_1)$ be a point in $E(\mathbb{F}_{3^m})$, then the point tripling $R = (x_3, y_3) = 3P$ is computed as follows:

$$\begin{cases} x_3 &= x_1^9 - b_3, \\ y_3 &= -y_1^9. \end{cases}$$

4.5.1 Projective Coordinate

The formulas for addition and doubling of points in Affine coordinate require the inversion in $\mathbb{F}_{3^m}$. The computation of the inversion is much slower than the other operations such as the multiplication in $\mathbb{F}_{3^m}$. Therefore, we adopt a few alternative coordinates instead of the Affine coordinate.

In a natural conversion, the points in the Affine coordinate are transformed into the point $(X : Y : Z)$ in projective space. In the Projective coordinate, the projective equation of the elliptic curve is

$$E_P : Y^2Z = X^3 - XZ^2 + b_3Z^3.$$

The point $P = (X : Y : Z)$ on E_P corresponds to $(X/Z, Y/Z)$ when $Z \neq 0$. Then we compute the point $(x : y : 1)$ in the Projective coordinate from the point (x, y) in the Affine coordinate. The point at infinity of this curve is $(0 : 1 : 0)$, and the negative point $-P$ is $(X : -Y : Z)$.

The explicit formulas for point addition, doubling and tripling in the Projective coordinate are shown as follows:

Point addition. Let $P = (X_1 : Y_1 : Z_1), Q = (X_2 : Y_2 : Z_2) \in E_P(\mathbb{F}_{3^m})$. Then the point addition $R = (X_3 : Y_3 : Y_3) = P + Q$ is computed as:

$$\begin{cases} \lambda_1 &= X_2Z_1 - X_1Z_2, \\ \lambda_2 &= Y_2Z_1 - Y_1Z_2, \\ X_3 &= \lambda_1\lambda_2^2Z_1Z_2 - (X_1Z_2 + X_2Z_1)\lambda_1^3, \\ Y_3 &= (Y_1Z_2 + Y_2Z_1)\lambda_1^3 - \lambda_2^3Z_1Z_2, \\ Z_3 &= Z_1Z_2\lambda_1^3. \end{cases}$$

Point doubling. For $P = (X_1 : Y_1 : Z_1)$ on $E_P(\mathbb{F}_{3^m})$, the point doubling $R = (X_3 : Y_3 : Y_3) = 2P$ is computed as follows:

$$\begin{cases} X_3 &= X_1Y_1^3 + Y_1Z_1^3, \\ Y_3 &= -(Y_1^4 + Z_1^4), \\ Z_3 &= Y_1^3Z_1. \end{cases}$$

Point tripling. Let $P = (X_1 : Y_1 : Z_1)$ on $E_P(\mathbb{F}_{3^m})$, then the point tripling $R = (X_3 : Y_3 : Y_3) = 3P$ is computed as follows:

$$\begin{cases} X_3 &= X_1^9 - b_3Z_1^9, \\ Y_3 &= -Y_1^9, \\ Z_3 &= Z_1^9. \end{cases}$$

4.5.2 Jacobian Coordinate

We similarly construct the Jacobian coordinate that is a variant of the coordinate system used by the projective space. In the Jacobian coordinate, the elliptic curve is given by:

$$E_J : Y^2 = X^3 - XZ^4 + b_3 Z^6$$

The point $P = (X : Y : Z)$ on E_J corresponds to $(X/Z^2, Y/Z^3)$ on E when $Z \neq 0$. The point at infinity in the Jacobian coordinate is $(1 : 1 : 0)$, and the negative point is given by the same point in the Projective coordinate.

Point addition, doubling and tripling in the Jacobian coordinate is performed by the following formulas:

Point addition. Let $P = (X_1 : Y_1 : Z_1), Q = (X_2 : Y_2 : Z_2) \in E_J(\mathbb{F}_{3^m})$. Then the point addition $R = (X_3 : Y_3 : Y_3) = P + Q$ in the Jacobian coordinate is computed as:

$$\begin{cases} \lambda_1 &= X_2 Z_1^2 - X_1 Z_2^2, \\ \lambda_2 &= Y_2 Z_1^3 - Y_1 Z_2^3, \\ X_3 &= \lambda_2^2 - \lambda_1^2 (X_1 Z_2^2 + X_2 Z_1^2), \\ Y_3 &= \lambda_1^3 (Y_1 Z_2^3 + Y_2 Z_1^3) - \lambda_2^3, \\ Z_3 &= Z_1 Z_2 \lambda_1. \end{cases}$$

Point doubling. For $P = (X_1 : Y_1 : Z_1)$ on $E_J(\mathbb{F}_{3^m})$, the point doubling $R = (X_3 : Y_3 : Y_3) = 2P$ is computed as follows:

$$\begin{cases} X_3 &= X_1 Y_1^2 + Z_1^8, \\ Y_3 &= -(Y_1^4 + Z_1^{12}), \\ Z_3 &= Z_1 Y_1. \end{cases}$$

Point tripling. Let $P = (X_1 : Y_1 : Z_1)$ on $E_J(\mathbb{F}_{3^m})$, then the point tripling $R = (X_3 : Y_3 : Y_3) = 3P$ is computed by:

$$\begin{cases} X_3 &= X_1^9 - b_3 (Z_1^9)^2, \\ Y_3 &= -Y_1^9, \\ Z_3 &= Z_1^9. \end{cases}$$

4.5.3 Comparison of Computational Cost in Affine, Projective and Jacobian Coordinates

In this section, we summarize the computational cost of the point arithmetic in the Affine, Projective and Jacobian coordinates. Table 4.1 lists the computational

Table 4.1: Computational Cost of Arithmetic on Supersingular Elliptic Curve over $\mathbb{F}_{3^m}$ in Affine, Projective and Jacobian Coordinates

	Point Addition	Point Doubling	Point Tripling
Affine	$2\mathbf{M} + \mathbf{C} + \mathbf{I}$	$\mathbf{M} + \mathbf{C} + \mathbf{I}$	$4\mathbf{C}$
Projective	$12\mathbf{M} + 2\mathbf{C}$	$5\mathbf{M} + 2\mathbf{C}$	$6\mathbf{C}$
Jacobian	$12\mathbf{M} + 4\mathbf{C}$	$7\mathbf{M} + 3\mathbf{C}$	$\mathbf{M} + 6\mathbf{C}$

* $\mathbf{M}$, $\mathbf{C}$, and $\mathbf{I}$ correspond to the computational cost of multiplication, cubing and inversion in $\mathbb{F}_{3^m}$.

Algorithm 4.13 Triple-and-Addition Method for Point Multiplication

INPUT: $P \in E(\mathbb{F}_{3^m})$, $d = \sum_{i=0}^{t-1} d_i 3^i$ ($d_{t-1} \neq 0$, $d_i \in \mathbb{F}_3$ for $0 \leq i < t$)
OUTPUT: $dP \in E(\mathbb{F}_{3^m})$

1: $P_1 \leftarrow P$, $P_2 \leftarrow 2P$
2: $Q \leftarrow \mathcal{O}$
3: **for** $i \leftarrow t-1$ **downto** 0 **do**
4: $\quad Q \leftarrow 3Q$
5: $\quad$ **if** $d_i \neq 0$ **then**
6: $\quad\quad Q \leftarrow Q + P_{d_i}$
7: $\quad$ **end if**
8: **end for**
9: **return** Q

cost in each coordinate. For the supersingular elliptic curve in characteristic three, the Projective coordinate can compute the point arithmetic most efficiently.

On the other hand, the point transformation from the Projective coordinate to the Affine coordinate requires 2 multiplications and 1 inversion in $\mathbb{F}_{3^m}$, and that from the Jacobian coordinate to the Affine coordinate requires 3 multiplications, 1 inversion, and 1 cubing in $\mathbb{F}_{3^m}$. The conversion from Affine to Jacobian or Projective coordinate requires no computational cost.

4.5.4 Point Multiplication

Let P be a point in $E(\mathbb{F}_{3^m})$, and d be a positive integer. The point multiplication dP over $E(\mathbb{F}_{3^m})$ is efficiently computed by using a triple-and-addition method. The scalar d represents a t-digit ternary integer $d = \sum_{i=0}^{t-1} d_i 3^i$ ($d_{t-1} \neq 0$, $d_i \in \mathbb{F}_3$ for $0 \leq i < t$). The point multiplication using the triple-and-addition is performed by Algorithm 4.13.

Algorithm 4.14 Conversion of Positive Integer into rw-NAF Form for $E(\mathbb{F}_{3^m})$

INPUT: An positive integer d, a window width w

OUTPUT: The rw-NAF form of d: $d' = (..., d'_1, d'_0)$

1: $i \leftarrow 0$
2: **while** $d > 0$ **do**
3: **if** $d \bmod 3 = 0$ **then**
4: $d'_i \leftarrow 0$
5: **else**
6: $d'_i \leftarrow d \overline{\bmod}\, 3^w$
7: **end if**
8: $d \leftarrow (d - d'_i)/3$, $i \leftarrow i + 1$
9: **end while**
10: **return** $d' = (..., d'_1, d'_0)$

Radix-r width-w NAF method. There exists more efficient algorithms by converting the scalar and precomputing some points for calculating the point multiplication of $E(\mathbb{F}_{3^m})$. NAF method is the signed expression of the scalar to efficiently compute the point multiplication. A radix-r width-w NAF method, written rw-NAF, is the variant of the NAF method and one of the fastest algorithms to compute the point multiplication [107].

The rw-NAF method is the generalized version of the NAF for window width w and radix r. The rw-NAF representation of d has the smallest Hamming weight among all signed representation for d with digit set $D^{r,w}_{rwn}$. In the rw-NAF, an input positive integer $d = (d_{t-1}, ..., d_0)$ in radix-r is converted into rw-NAF form $d' = (d'_{t'-1}, ..., d'_0)$, and each d'_i of rw-NAF form of the scalar d is in

$$D^{r,w}_{rwn} = \{0, \pm 1, \pm 2, ..., \pm \lfloor (r^w - 1)/2 \rfloor\} \backslash \{\pm 1r, \pm 2r, ..., \pm \left\lfloor (r^{w-1} - 1)/2 \right\rfloor r\}.$$

Then we precompute the points P_i for all $i \in D^{r,w}_{rwn}$. In the point multiplication, the negative point does not need to be precomputed, since its computation can compute virtually for free.

The notation $\overline{\bmod}$ is denoted by the signed modulo operation that computes $(t \bmod r^w) - r^w$ if $(t \bmod r^w) \geq r^w/2$; otherwise $t \bmod r^w$, where a positive integer t, radix r, and window width w. Algorithms 4.14 and 4.15 display the conversion of the positive integer into rw-NAF from and the point multiplication using rw-NAF method for $E(\mathbb{F}_{3^m})$.

Algorithm 4.15 rw-NAF Method for Point Multiplication in $E(\mathbb{F}_{3^m})$

INPUT: a point $P \in E(\mathbb{F}_{3^m})$, an positive integer d, window width w

OUTPUT: $dP \in E(\mathbb{F}_{3^m})$

1: Compute rw-NAF form $d' = (d'_{t'-1}, ..., d'_0)$ using Algorithm 4.14
2: Compute $P_i \leftarrow iP$ for all $i \in D^{3,w}_{rwn}$
3: $Q \leftarrow \mathcal{O}$
4: **for** $i \leftarrow t' - 1$ **downto** 0 **do**
5: $\quad Q \leftarrow 3Q$
6: $\quad$ **if** $d'_i > 0$ **then**
7: $\quad\quad Q \leftarrow Q + P_{d'_i}$
8: $\quad$ **else if** $d'_i < 0$ **then**
9: $\quad\quad Q \leftarrow Q - P_{-d'_i}$
10: $\quad$ **end if**
11: **end for**
12: **return** Q

4.6 η_T Paring over $\mathbb{F}_{3^m}$

This section resumes the η_T pairing and its variant algorithms as the improved algorithms of the Tate pairing computation on supersingular elliptic curves in characteristic three.

We choose the suitable extension degree m of which $E(\mathbb{F}_{3^m})$ has a large prime r such that $r \mid \#E(\mathbb{F}_{3^m})$, and then r satisfies $r \mid (3^{6m} - 1)$. Let $E(\mathbb{F}_{3^m})[r]$ be the set of the rational point of $E(\mathbb{F}_{3^m})$ with order r. The distortion map of the elliptic curve from $Q = (x, y) \in E(\mathbb{F}_{3^m})[r]$ to $E(\mathbb{F}_{3^{6m}})/rE(\mathbb{F}_{3^{6m}})$ is

$$\psi(Q) = (-x + \rho, y\sigma).$$

Then for $P, Q \in E(\mathbb{F}_{3^m})[r]$, the η_T pairing over $\mathbb{F}_{3^m}$ is the map

$$\begin{aligned} \eta_T : E(\mathbb{F}_{3^m})[r] \times E(\mathbb{F}_{3^m})[r] &\rightarrow \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})} \\ (P, Q) &\mapsto \eta_T(P, Q). \end{aligned}$$

The security of the η_T pairing is decided based on the difficulty of the discrete logarithm problem over the elliptic curve $E(\mathbb{F}_{3^m})[r]$ and finite field $\mathbb{F}_{3^{6m}}$. Then we suitably choose the parameters, the degree m and curve E, of which $\#E(\mathbb{F}_{3^m})$ has the cyclic group with the large prime order r.

The η_T pairing consists of two parts, Miller loop and final exponentiation. In the following, we describe the detail algorithm of them.

Miller loop. For computing the η_T pairing over $\mathbb{F}_{3^m}$, we use the $\widetilde{\eta_T}$ pairing without cube root, that is removed by the branches of the original η_T pairing algorithm with the degree m. The result of the $\widetilde{\eta_T}$ pairing is not the same as that of the original η_T pairing. However the result can be simply related with the original η_T pairing using the cubing in $\mathbb{F}_{3^{6m}}$, since the cubing in $\mathbb{F}_{3^{6m}}$ can be efficiently computed. Thus the security of the $\widetilde{\eta_T}$ pairing is the same of the η_T pairing.

Moreover some efficient methods, loop unrolling and efficient $\mathbb{F}_{3^{6m}}$ multiplication, are combined with the $\widetilde{\eta_T}$ pairing . Algorithm 4.16 shows the explicit algorithm of the $\widetilde{\eta_T}$ pairing with loop unrolling when $(m-1)/2$ is even, and the multiplication in Steps 14 and 15 of Algorithm 4.16 are performed as Algorithms 4.17 and 4.18.

Final exponentiation. For the η_T pairing computation, we need to compute the output value of the Miller loop f_σ from $\mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$ to $\mathbb{F}^*_{3^{6m}}$. In the final exponentiation, we calculate

$$f_\sigma^{(3^{3m}-1)(3^m+1)(3^m-b'3^{(m+1)/2}+1)}$$

where f_σ is the result of the Miller loop.

Firstly, $f_\sigma^{(3^{3m}-1)}$ is computed as

$$f_\sigma^{(3^{3m}-1)} = \frac{(f_{\rho,0}^2 - f_{\rho,1}^2) + f_{\rho,0}f_{\rho,1}\sigma}{f_{\rho,0}^2 + f_{\rho,1}^2}.$$

$f_\sigma^{(3^{3m}-1)}$ is the element in $T_2(\mathbb{F}_{3^{3m}})$, then the remain of the exponentiation is computed by using the properties of $T_2(\mathbb{F}_{3^{3m}})$.

Algorithms 4.19, 4.20 show the efficient final exponentiation of the $\widetilde{\eta_T}$ pairing using the properties of the torus $T_2(\mathbb{F}_{3^{3m}})$.

Algorithm 4.16 $\widetilde{\eta_T}$ Pairing over $\mathbb{F}_{3^m}$ with Unroll Loops when $(m-1)/2$ is Even

INPUT: $P=(x_p,y_p), Q=(x_q,y_q) \in E(\mathbb{F}_{3^m})[r]$

OUTPUT: $\widetilde{\eta_T}(P,Q) \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$

1: $t \leftarrow x_p + x_q + b$
2: $f_\sigma \leftarrow -y_q\sigma\rho^2 + y_q(y_p^2 - t)\sigma\rho - y_q t(y_p^2 + u)\sigma - y_p\rho + y_p(t^3 - y_q^2 - b_3)$
3: $d \leftarrow 0$
4: **for** $i \leftarrow 1$ **to** $(m-1)/4$ **do**
5: $\quad f_\sigma \leftarrow f_\sigma^9$
6: $\quad x_q \leftarrow x_q^9$, $y_q \leftarrow y_q^9$
7: $\quad d \leftarrow d - b_3 \pmod 3$
8: $\quad t \leftarrow x_p + x_q + d$
9: $\quad g_\sigma \leftarrow (y_p y_q)^3\sigma - (t'^2 + b_3 + 1)^3 + (b_3 - t)^3\rho - \rho^2$
10: $\quad x_q \leftarrow x_q^9$, $y_q \leftarrow y_q^9$
11: $\quad d \leftarrow d - b_3 \pmod 3$
12: $\quad t' \leftarrow x_p + x_q + d$
13: $\quad h_\sigma \leftarrow y_p y_q \sigma - t'^2 - t'\rho - \rho^2$
14: $\quad g_\sigma \leftarrow g_\sigma h_\sigma$ using Algorithm 4.17
15: $\quad f_\sigma \leftarrow f_\sigma g_\sigma$ using Algorithm 4.18
16: **end for**
17: **return** f_σ^3

Algorithm 4.17 Computation of $(u_0+u_1\sigma+u_2\rho-\rho^2)(v_0+v_1\sigma+v_2\rho-\rho^2)$ [17]

INPUT: $u_\sigma=(u_0,u_1,u_2,0,-1,0)$, $v_\sigma=(v_0,v_1,v_2,0,-1,0) \in \mathbb{F}_{3^{6m}}$

OUTPUT: $w_\sigma = u_\sigma \cdot v_\sigma \in \mathbb{F}_{3^{6m}}$

1: $m_0 \leftarrow u_0v_0$, $m_1 \leftarrow u_1v_1$, $m_2 \leftarrow u_2v_2$
2: $m_3 \leftarrow (u_0+u_1)(v_0+v_1)$, $m_4 \leftarrow (u_0+u_2)(v_0+v_2)$, $m_5 \leftarrow (u_1+u_2)(v_1+v_2)$
3: $w_0 \leftarrow m_0 - m_1 - b_3(u_2+v_2)$
4: $w_1 \leftarrow m_3 - m_0 - m_1$
5: $w_2 \leftarrow m_4 - m_0 - m_2 - (u_2+v_2) + b_3$
6: $w_3 \leftarrow m_5 - m_1 - m_2$
7: $w_4 \leftarrow 1 + m_2 - (u_0+v_0)$
8: $w_5 \leftarrow -(u_1+v_1)$
9: **return** $(w_0,w_1,w_2,w_3,w_4,w_5)$

Algorithm 4.18 Multiplication in $\mathbb{F}_{3^{6m}}$ [17]

INPUT: $u_\sigma = (u_0, u_1, u_2, u_3, u_4, u_5), v_\sigma = (v_0, v_1, v_2, v_3, v_4, v_5) \in \mathbb{F}_{3^{6m}}$

OUTPUT: $w_\sigma = u_\sigma \cdot v_\sigma \in \mathbb{F}_{3^{6m}}$

1: $a_0 \leftarrow u_0 + u_2 + u_4$, $a_1 \leftarrow u_1 + u_3 + u_5$, $a_2 \leftarrow a_0 + a_1$
$a_3 \leftarrow v_0 + v_2 + v_4$, $a_4 \leftarrow v_1 + v_3 + v_5$, $a_5 \leftarrow a_3 + a_4$
2: $m_0 \leftarrow a_0 a_3$, $m_1 \leftarrow a_2 a_5$, $m_2 \leftarrow a_1 a_4$
3: $a_6 \leftarrow u_0 - u_3 - u_4$, $a_7 \leftarrow u_1 + u_2 - u_5$, $a_8 \leftarrow a_6 + a_7$
$a_9 \leftarrow v_0 - v_3 - v_4$, $a_{10} \leftarrow v_1 + v_2 - v_5$, $a_{11} \leftarrow a_9 + a_{10}$
4: $m_3 \leftarrow a_6 a_9$, $m_4 \leftarrow a_8 a_{11}$, $m_5 \leftarrow a_7 a_{10}$
5: $a_{12} \leftarrow u_0 - u_2 + u_4$, $a_{13} \leftarrow u_1 - u_3 + u_5$, $a_{14} \leftarrow a_{12} + a_{13}$
$a_{15} \leftarrow v_0 - v_2 + v_4$, $a_{16} \leftarrow v_1 - v_3 + v_5$, $a_{17} \leftarrow a_{15} + a_{16}$
6: $m_6 \leftarrow a_{12} a_{15}$, $m_7 \leftarrow a_{14} a_{17}$, $m_8 \leftarrow a_{13} a_{16}$
7: $a_{18} \leftarrow u_0 + u_3 - u_4$, $a_{19} \leftarrow u_1 - u_2 - u_5$, $a_{20} \leftarrow a_{18} + a_{19}$
$a_{21} \leftarrow v_0 + v_3 - v_4$, $a_{22} \leftarrow v_1 - v_2 - v_5$, $a_{23} \leftarrow a_{21} + a_{22}$
8: $m_9 \leftarrow a_{18} a_{21}$, $m_{10} \leftarrow a_{20} a_{23}$, $m_{11} \leftarrow a_{19} a_{22}$
9: $a_{24} \leftarrow u_4 + u_5$, $a_{25} \leftarrow v_4 + v_5$
10: $m_{12} \leftarrow u_4 v_4$, $m_{13} \leftarrow a_{24} a_{25}$, $m_{14} \leftarrow u_5 v_5$
11: **if** $b_3 = 1$ **then**
12: $t_0 \leftarrow m_0 + m_4 + m_{12}$, $t_1 \leftarrow m_2 + m_{10} + m_{14}$, $t_2 \leftarrow m_6 + m_{12}$, $t_3 \leftarrow -m_8 - m_{14}$,
$t_4 \leftarrow m_7 + m_{13}$, $t_5 \leftarrow t_3 + m_2$, $t_6 \leftarrow t_2 - m_0$,
$t_7 \leftarrow t_3 - m_2 + m_5 + m_{11}$, $t_8 \leftarrow t_2 + m_0 - m_3 - m_9$
13: $w_0 \leftarrow -t_0 + t_1 - m_3 + m_{11}$
14: $w_1 \leftarrow t_0 + t_1 - m_1 + m_5 + m_9 - m_{13}$
15: $w_2 \leftarrow t_5 + t_6$
16: $w_3 \leftarrow t_5 - t_6 + t_4 - m_1$
17: $w_4 \leftarrow t_7 + t_8$
18: $w_5 \leftarrow t_7 - t_8 + t_4 + m_1 - m_4 - m_{10}$
19: **else**
20: $t_0 \leftarrow m_4 + m_8 + m_{14}$, $t_1 \leftarrow m_6 + m_{12}$, $t_2 \leftarrow t_1 + m_{10}$, $t_3 \leftarrow m_2 + m_{14}$,
$t_4 \leftarrow t_3 - m_8$, $t_5 \leftarrow -m_0 + m_6 - m_{12}$, $t_6 \leftarrow -t_3 + m_5 - m_8 + m_{11}$,
$t_7 \leftarrow t_1 + m_0 - m_3 - m_9$, $t_8 \leftarrow m_1 + m_{13}$
21: $w_0 \leftarrow t_0 - t_2 + m_5 - m_9$
22: $w_1 \leftarrow t_0 + t_2 + m_3 - m_7 + m_{11} - m_{13}$
23: $w_2 \leftarrow t_4 + t_5$
24: $w_3 \leftarrow t_4 - t_5 - t_8 + m_7$
25: $w_4 \leftarrow t_6 + t_7$
26: $w_5 \leftarrow t_6 - t_7 + t_8 - m_4 + m_7 - m_{10}$
27: **end if**
28: **return** $(w_0, w_1, w_2, w_3, w_4, w_5)$

Algorithm 4.19 Final Exponentiation of η_T Pairing [101]

INPUT: $f_\sigma = (f_0, f_1, f_2, f_3, f_4, f_5) \in \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^{\#E(\mathbb{F}_{3^m})}$

OUTPUT: $f_\sigma^{(3^{3m}-1)(3^m+1)(3^m-b'3^{(m+1)/2}+1)}$

1: $f_\sigma \leftarrow f_\sigma^{3^{3m}-1}$
2: $f_\sigma \leftarrow g_\sigma \leftarrow \Lambda(f_\sigma) = f_\sigma^{3^m+1}$
3: $f_\sigma \leftarrow \Lambda(f_\sigma) = f_\sigma^{3^m+1}$
4: $g_\sigma \leftarrow g_\sigma^{3^{(m-1)/2}}$
5: **if** $b' = 1$ **then**
6: **return** $f_\sigma \cdot (g_0,\ -g_1,\ g_2,\ -g_3,\ g_4,\ -g_5)$
7: **else**
8: **return** $f_\sigma \cdot (g_0,\ g_1,\ g_2,\ g_3,\ g_4,\ g_5)$
9: **end if**

Algorithm 4.20 Computation of $\Lambda(f_\sigma)$ [101]

INPUT: $f_\sigma = (f_0, f_1, f_2, f_3, f_4, f_5) \in T_2(\mathbb{F}_{3^{3m}})$

OUTPUT: $\Lambda(f_\sigma) = f_\sigma^{3^m+1} \in T_2(\mathbb{F}_{3^{3m}})$

1: $z_0 \leftarrow f_0 f_4$, $z_1 \leftarrow f_1 f_5$, $z_2 \leftarrow f_2 f_4$, $z_3 \leftarrow f_3 f_5$
2: $z_4 \leftarrow (f_0 + f_1)(f_4 - f_5)$, $z_5 \leftarrow f_1 f_2$, $v_6 \leftarrow f_0 f_3$
3: $z_7 \leftarrow (f_0 + f_1)(f_2 + f_3)$, $z_8 \leftarrow (f_2 + f_3)(f_4 - f_5)$
4: **if** $m \equiv 1 \pmod 6$ **then**
5: $c_0 \leftarrow 1 + z_0 + z_1 - bz_2 - bz_3$
6: $c_1 \leftarrow z_1 + z_4 + bz_5 - z_0 - bz_6$
7: $c_2 \leftarrow z_7 - z_2 - z_3 - z_5 - z_6$
8: $c_3 \leftarrow bz_0 + z_3 + z_8 - z_2 - bz_1 - bz_4$
9: $c_4 \leftarrow b(z_2 + z_3 + z_7 - z_5 - z_6)$
10: $c_5 \leftarrow b(z_3 + z_8 - z_2)$
11: **else if** $m \equiv 5 \pmod 6$ **then**
12: $c_0 \leftarrow 1 + z_0 + z_1 + bz_2 + bz_3$
13: $c_1 \leftarrow z_1 + z_4 - bv_5 - z_0 + bz_6$
14: $c_2 \leftarrow z_5 + z_6 - z_7$
15: $c_3 \leftarrow -bz_0 + z_3 + z_8 - z_2 + bz_1 + bz_4$
16: $c_4 \leftarrow b(z_2 + z_3 + z_7 - z_5 - z_6)$
17: $c_5 \leftarrow b(-z_3 - z_8 + z_2)$
18: **end if**
19: **return** $c_\sigma = (c_0, c_1, c_2, c_3, c_4, c_5)$

Chapter 5

Construction of Addition and Subtraction in $\mathbb{F}_3$ using Minimum Number of Logical Instructions

This chapter contains joint work with Kazumaro Aoki and Tsuyoshi Takagi, which has been published in IPSJ Journal (in Japanese) [64], and was presented at the third international conference on Pairing-based Cryptography [63].

5.1 Introduction

For constructing the η_T pairing and the operations over the supersingular elliptic curves in characteristic three, we need to implement the arithmetic in $\mathbb{F}_{3^m}$. The arithmetic in $\mathbb{F}_{3^m}$ is constructed based on $\mathbb{F}_3$-addition and subtraction, which are easily calculable by $a \pm b \bmod 3$ for given $\mathbb{F}_3$ elements a and b. However they cannot be directly computed by virtually any typical CPU, such as one based on x86-architecture [59]. Moreover, if we implement $\mathbb{F}_3$-addition and subtraction straightforwardly, then the computational cost of $\mathbb{F}_{3^m}$-addition and subtraction is m additions and subtractions in $\mathbb{F}_3$, respectively. For an efficient implementation in $\mathbb{F}_{3^m}$, each element in $\mathbb{F}_3$ is represented by two bits, and the $\mathbb{F}_3$-addition and subtraction is constructed by using logical instructions, that are the binary instructions of a 2-bit input and an 1-bit output such as AND, OR, XOR. Galbraith et al. [42] demonstrated the $\mathbb{F}_3$-addition using 12 logical instructions, consisting of AND, OR, XOR, and NOT. Then, Harrison et al. [56] improved this to seven logical instructions, consisting of OR and XOR.

This chapter describes our search for implementations of the $\mathbb{F}_3$-addition and subtraction that use seven or fewer logical instructions. Every $\mathbb{F}_3$-element is assigned to two bits in $(\mathbb{F}_2)^2$, and the $\mathbb{F}_3$-addition and subtraction are considered as a map $(\mathbb{F}_2)^2 \times (\mathbb{F}_2)^2 \to (\mathbb{F}_2)^2$. An exhaustive search is performed for finding the sequences of logical instructions that can compute the $\mathbb{F}_3$-addition and subtraction for the map. Indeed, although we found many implementations of the $\mathbb{F}_3$-addition and subtraction with only six logical instructions, the representation of $\mathbb{F}_3$-elements is not the natural assignment $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$, or the special logical instructions, which are not equipped in typical CPU such as ANDN are used. Moreover, we found no instruction sequence that can compute the $\mathbb{F}_3$-addition and subtraction with five logical instructions. In other words, we have proven that the minimum number of logical instructions required for the $\mathbb{F}_3$-addition and subtraction are six for any assignment of elements in $\mathbb{F}_3$ used by two-bit representation.

To demonstrate the cryptographic implications of the new $\mathbb{F}_3$-addition and subtraction implementations using six logical instructions, we implement arithmetic in $\mathbb{F}_{3^m}$ and the η_T pairing over $\mathbb{F}_{3^m}$ on an AMD Opteron processor model 275 (2.2 GHz). For comparison, we choose the extension degree $m = 509$, since the η_T pairing over $\mathbb{F}_{3^{509}}$ has 128-bit security [71]. With the assignment $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$, the addition in $\mathbb{F}_{3^{509}}$ was about 12% ($\simeq 1/7$) faster than that with the natural assignment. Similarly, the multiplication in $\mathbb{F}_{3^{509}}$, computed by the left-to-right comb method with the window width $w = 4$, was about 8% faster than that with the natural assignment. As a result, the running time of the η_T pairing over $\mathbb{F}_{3^{509}}$ for the new $\mathbb{F}_3$-addition with six logical instructions was 16.3 milliseconds, which is about 7% faster than the running time for the previous $\mathbb{F}_3$-addition with seven logical instructions.

Firstly, we describe the addition and subtraction in $\mathbb{F}_3$ and discuss previous results on this topic. Secondly, we give the details of our search for instruction sequences to compute the $\mathbb{F}_3$-addition and subtraction. We then show the search results of the addition and subtraction in $\mathbb{F}_3$, and list some examples. Finally, we describe our implementations and running time results for arithmetic in $\mathbb{F}_{3^m}$ and the η_T pairing, and the chapter is conclude.

5.2 Addition and Subtraction in $\mathbb{F}_3$

Let $\mathbb{F}_3$ be a finite field with three elements $\{0, 1, 2\}$. We can represent $e \in \mathbb{F}_3$ by using two bits:

$$e = (e_h, e_l),$$

where $e_h, e_l \in \{0, 1\}$. The assignment of $\mathbb{F}_3$ to $(\mathbb{F}_2)^2$ is the following:

$$\{(0,0) \mapsto 0,\ (0,1) \mapsto 1,\ (1,0) \mapsto 2\}.$$

We call this the *natural* assignment.

For given a and b in $\mathbb{F}_3$, we can compute addition $c \leftarrow a + b$ in $\mathbb{F}_3$ as $c = a + b \bmod 3$. In the *natural* assignment, a negative element, $-e$, of $e = (e_h, e_l)$ is obtained as

$$\begin{cases} (-e)_h & \leftarrow & e_l, \\ (-e)_l & \leftarrow & e_h. \end{cases}$$

Let a and b be elements in $\mathbb{F}_3$. The subtraction $a - b$ can be performed as $a + (-b)$, with the same cost as the $\mathbb{F}_3$-addition.

We construct the $\mathbb{F}_3$-addition by using the logical instructions below.

$$\begin{cases} \mid & : & \text{bitwise OR operation} \\ \& & : & \text{bitwise AND operation} \\ \wedge & : & \text{bitwise XOR operation} \\ \overline{x} & : & \text{bitwise complement (NOT)} \end{cases}$$

Implementation of an efficient $\mathbb{F}_3$-addition and subtraction using these logical instructions is a non-trivial problem. For efficient implementation, it is preferable to keep the number of logical instructions as small as possible. Galbraith et al. [42] presented the following $\mathbb{F}_3$-addition using 12 logical instructions.

$$\begin{cases} c_h & \leftarrow & ((a_h \wedge b_h)\ \&\ \overline{(a_l \mid b_l)}) \mid (a_l\ \&\ b_l), \\ c_l & \leftarrow & ((a_l \wedge b_l)\ \&\ \overline{(a_h \mid b_h)}) \mid (a_h\ \&\ b_h). \end{cases}$$

Harrison et al. [56] improved the implementation to seven logical instructions by using an auxiliary variable t, as indicated below.

$$\begin{cases} t & \leftarrow & (a_h \mid b_l) \wedge (b_h \mid a_l), \\ c_h & \leftarrow & t \wedge (a_l \mid b_l), \\ c_l & \leftarrow & t \wedge (a_h \mid b_h). \end{cases} \tag{5.1}$$

There are many other implementations of the arithmetic in $\mathbb{F}_{3^m}$; the addition in $\mathbb{F}_3$ of all, however, is constructed using exactly the same logical instruction sequence given in Eq. (5.1). It has not yet been proven whether seven is the minimum number of logical instructions for the $\mathbb{F}_3$-addition and subtraction.

5.3 Search Algorithm for $\mathbb{F}_3$-Addition

This section describes the procedure of our search for the minimum number of logical instructions in order to compute the $\mathbb{F}_3$-addition.

5.3.1 Choice of Assignment in $\mathbb{F}_3$

Firstly, we decide the assignment which is the bit representation of $\mathbb{F}_3$. The assignment R of $\mathbb{F}_3$ to $(\mathbb{F}_2)^2$ is denoted by

$$R = \{(e_h, e_l) \mapsto e \mid e_h, e_l \in \mathbb{F}_2, e \in \mathbb{F}_3\}.$$

The addition $c \leftarrow a + b$ in $\mathbb{F}_3$ is a map $\mathbb{F}_3 \leftarrow \mathbb{F}_3 \times \mathbb{F}_3$. To specify each assignment of $\mathbb{F}_3$ in the map, we denote by R_a, R_b, R_c the assignment for the left part of the input, the right part of the input, and the output, respectively, of the addition $c \leftarrow a + b$ in $\mathbb{F}_3$. The set R_a, R_b, R_c, is called the assignment set, determines the implementation of the $\mathbb{F}_3$-addition. The *natural* assignment set in the previous $\mathbb{F}_3$-addition uses the set $R_a = R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$. Note that there are many possible assignment sets, such as this:

$$\begin{cases} R_a &= \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}, \\ R_b &= \{(1,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2, (0,0) \mapsto 2\}, \\ R_c &= \{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}. \end{cases} \tag{5.2}$$

Here, we consider the following cases to choose the assignments. By using them, we can take all patterns of assignments for constructing the $\mathbb{F}_3$-addition.

Redundant representation. The number of elements in $\mathbb{F}_3$ and $(\mathbb{F}_2)^2$ is three and four, respectively. Then one element e_0 in $\mathbb{F}_3$ has two different representations $e_0 = (e_{0h}, e_{0l}) = (e'_{0h}, e'_{0l})$ in $(\mathbb{F}_2)^2$, but the other two elements e_1 and e_2 in $\mathbb{F}_3$ are uniquely assigned as $e_1 = (e_{1h}, e_{1l})$ and $e_2 = (e_{2h}, e_{2l})$ in $(\mathbb{F}_2)^2$, where $(e_{0h}, e_{0l}), (e'_{0h}, e'_{0l}), (e_{1h}, e_{1l})$, and (e_{2h}, e_{2l}) are pairwise different in $(\mathbb{F}_2)^2$.

Independent assignments. All assignments R_a, R_b, R_c for the addition $c \leftarrow a+b$ in $\mathbb{F}_3$ are not necessarily the same. Thus we choose these independently in the search.

The cardinality of the elements in R_i is denoted by $\#R_i$, and we can define the inclusion relation $R_i \subseteq R_j$, where $i, j \in \{a, b, c\}$. Then, all patterns of the

assignment set R_a, R_b, R_c can be categorized according to the cardinality and inclusion relation, as follows. For search the sequence to compute $\mathbb{F}_3$-addition, we use the following categorization of the assignments R_a, R_b, R_c.

- $\#R_a = \#R_b = \#R_c = 3$
  - 1–i *Natural* assignment set $R_a = R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$
  - 1–ii Assignment set of common $R_a = R_b = R_c$ (It includes the *natural* assignment set.)
  - 1–iii Assignment set of independent R_a, R_b, R_c
- $\#R_a = \#R_b = 3,\ \#R_c = 4$
  - 2–i $R_a = R_b \subseteq R_c$
  - 2–ii Assignment set of independent R_a, R_b, R_c
- $\#R_a = \#R_c = 3,\ \#R_b = 4$
  - 3–i $R_a = R_c \subseteq R_b$
  - 3–ii Assignment set of independent R_a, R_b, R_c
- $\#R_a = 3,\ \#R_b = \#R_c = 4$
  - 4–i $R_a \subseteq R_b = R_c$
  - 4–ii Assignment set of independent R_a, R_b, R_c
- $\#R_a = \#R_b = 4,\ \#R_c = 3$
  - 5–i $R_a = R_b \supseteq R_c$
  - 5–ii Assignment set of independent R_a, R_b, R_c
- $\#R_a = \#R_b = \#R_c = 4$
  - 6–i Assignment set of common $R_a = R_b = R_c$ using the redundant representation for same element in $\mathbb{F}_3$
  - 6–ii Assignment set of independent R_a, R_b, R_c

Table 5.1: Conversion of Instruction Sequences Containing NOT

	t_0 & y		y ANDN x
	t_0 | y		y ORN x
	$t_0 \wedge y$		x XORN y
	t_0 ANDN y		x NOR y
$\overline{x} \to t_0$,	y ANDN $t_0 \to t_1$	$\Rightarrow$	x & y $\to t_1$
	t_0 ORN y		x NAND y
	y ORN t_0		x | y
	t_0 XORN y		$x \wedge y$
	t_0 NAND y		x ORN y
	t_0 NOR y		x ANDN y

5.3.2 Logical Instruction Set

Many CPUs, such as one based on x86-architecture, are equipped with AND, OR, and XOR as logical instructions. In some architectures, however, other logical instructions are also available. Given x, y in $\mathbb{F}_2$, ANDN (x ANDN $y = x$ & $\overline{y}$) can be used in MMX and SSE implementations [59]. The SPARC and Alpha architectures provide ANDN, ORN (x ORN $y = x$ | $\overline{y}$) and XORN (x XORN $y = x \wedge \overline{y}$) [33, 105]. Combining the AND and OR operations with the NOT operation, we further obtain NAND ($\overline{x \ \& \ y}$) and NOR ($\overline{x \ | \ y}$).

Our search algorithm deals with the following two logical instruction sets:

LI-Set 3 = {AND, OR, XOR},
LI-Set 8 = {AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR}.

LI-Set 8 contains all binary operations except for trivial operations such as $x*y = x$ for the binary operation $*$. Note that LI-Set 8 includes non-commutative instructions such as ANDN (x ANDN $y \neq y$ ANDN x). Hence both computations are required to compute in the search. In LI-Set 3, we choose some logical instructions that can be used in many CPUs from LI-Set 8.

NOT instruction. In our search, we do not apply NOT instruction, since the instruction sequence using NOT instruction can be replaced with another instruction sequence that is constructed without NOT instruction. For example, $\overline{x}$ ANDN y is the same result as x NOR y for given $x, y \in \mathbb{F}_2$. Table 5.1 displays all conversions of the instructions sequences containing NOT and the arbitrary instructions.

Table 5.2: Calculation of $x \in \{0, 1\}$ and Constant Values Zero or One

x	&	0	$\rightarrow$	0	x & 1	$\rightarrow$	x
x	$\mid$	0	$\rightarrow$	x	$x \mid 1$	$\rightarrow$	1
x	$\wedge$	0	$\rightarrow$	x	$x \wedge 1$	$\rightarrow$	$\overline{x}$

Constant values zero and one. The calculating result of $x \in \{0, 1\}$ and constant values zero or one lists in Table 5.2. From Table 5.2, the useful result is only $\overline{x}$, and $\overline{x}$ can be replaced with another instruction sequence. Then we do not apply the constant values zero and one in the search.

5.3.3 Search Procedure

In this section, we describe the search method for finding the addition $c \leftarrow a+b$ in $\mathbb{F}_3$. We prepare bit-strings a_H, a_L, b_H, b_L, c_H, c_L consisting of all calculated patterns for the $\mathbb{F}_3$-addition using the assignment set R_a, R_b, R_c.

Recall that in the redundant representation, one element in $\mathbb{F}_3$ has two different representations in $(\mathbb{F}_2)^2$. When both R_a and R_b do not use the redundant representation, the bit length of each bit-string a_H, a_L, b_H, b_L, c_H, c_L is 9. If either R_a or R_b uses the redundant representation, then the bit length of each bit-string is 12. Similarly, if both R_a and R_b use the redundant representation, then the bit length of each bit-string is 16. Furthermore, we need to consider the case in which R_c uses the redundant representation. In this case, one element in $\mathbb{F}_3$ is assigned two non-unique representations in $(\mathbb{F}_2)^2$. Therefore, each bit of the redundant part in (c_H, c_L) must be checked twice, individually.

Table 5.3 gives the truth table for the bit-strings created by the assignment set R_a, R_b, R_c of Eq. (5.2) of Sect. 5.3.1 as an example. In this case, since R_a does not use the redundant representation and R_b uses the redundant representation, the bit length of each bit-string is 12. Then since R_c does not use the redundant representation, all bit-strings have unique value, respectively.

We search instruction sequences to compute the $\mathbb{F}_3$-addition by depth-first search. The search algorithm proceeds as follows.

1. Choose the assignment set and construct the bit-strings a_H, a_L, b_H, b_L, c_H, c_L using it.

2. Initialize the search set S, which consists of the bit-strings, as $S = \{a_H, a_L, b_H, b_L\}$.

Table 5.3: Truth Table for Assignment Set R_a, R_b, R_c in Equation 5.2

a_H	0	0	0	0	0	0	0	0	1	1	1	1
a_L	0	0	0	0	1	1	1	1	0	0	0	0
b_H	1	0	1	0	1	0	1	0	1	0	1	0
b_L	0	1	1	0	0	1	1	0	0	1	1	0
c_H	1	0	1	1	0	1	1	1	1	1	0	0
c_L	1	1	0	0	1	0	1	1	0	1	1	1

3. Choose arbitrary two bit-strings x and y in S.
4. Choose an arbitrary logical instruction $*$ which is the bitwise logical operation in LI-Set, and compute $z \leftarrow x * y$.
5. Add the resultant bit-string z to S.
6. Iterate Steps 3–5 for a limited number of logical instructions.
7. Check whether both c_H and c_L are included in S.

The search algorithm is not efficient if all instruction sequences are computed, since most instruction sequences are not correct as the $\mathbb{F}_3$-addition. Let N_{max} be the limited number of logical instructions. Then we stop the search deeper than the present iteration when it satisfies the following conditions.

- R_a, R_b satisfied with $\#R_a = \#R_b$, and R'_a, R'_b such that $R'_a = R_b$ and $R'_b = R_a$ are already chosen.
- The computed result z is already contained in S.
- The computed result z is the bit-string $(00...00)$ or $(11...11)$.
- Neither c_H nor c_L is contained in S, when the number of iteration at Step 6 is $N_{max} - 1$.

If the computed result z is already contained in S, the instruction sequence can be constructed by using less instructions. Then in order to compute the $\mathbb{F}_3$-addition, both c_H and c_L need to be included in S. Therefore, either c_H or c_L must be included in S, when the number of iteration at Step 6 is $N_{max} - 1$. Hence we can search exhaustively even if we use these conditions.

Table 5.4: Number of Logical Instructions for Simple Search

N_{max}	1	2	3	4	5	6	7
LI-Set 3	$2^{5.2}$	$2^{11.1}$	$2^{17.6}$	$2^{24.6}$	$2^{31.9}$	$2^{39.7}$	$2^{47.8}$
LI-Set 8	$2^{6.9}$	$2^{14.6}$	$2^{22.8}$	$2^{31.5}$	$2^{40.6}$	$2^{50.1}$	$2^{59.9}$

5.3.4 Search Cost

This section discusses the computational cost of our search algorithm. The number of assignment patterns for $\#R_i = 3$ is 24 ($= 4 \times 3 \times 2$), since three elements in $\mathbb{F}_3$ are assigned to a different element in $(\mathbb{F}_2)^2$, respectively. Similarly, the number of assignment patterns for $\#R_i = 4$ is 72 ($= 24 \times 3$), since one element in $\mathbb{F}_3$ is assigned to two representations. Let r be the number of assignments of $\#R_i = 4$ in $\#R_a$, $\#R_b$, and $\#R_c$, where $0 \leq r \leq 3$. Then, the number of assignment set patterns is $24^{3-r} \times 72^r$. If $\#R_a = \#R_b$, then we can reduce 1/2 patterns. For example, with $\#R_a = \#R_b = \#R_c = 3$, the number of assignment set patterns is 6,912 ($= 24^3/2$).

Here, we estimate the search cost with the input assignment set and LI-Set. Since ANDN and ORN in LI-Set 8 are non-commutative instructions, we need to compute 10 instructions when we use LI-Set 8. Therefore the number of logical instructions used by the search is

$$\sum_{i=1}^{N} \prod_{j=1}^{i} (L(j+3)(j+2)),\ L = \begin{cases} 3 & \textbf{if} \quad \text{LI-Set 3}, \\ 10 & \textbf{if} \quad \text{LI-Set 8}. \end{cases}$$

Table 5.4 shows the number of logical instruction of the search without the condition for efficient search.

Most of the instruction sequences in the search do not contain c_H and c_L. Thus, we need to reduce the computational cost by using the conditions described in Sect. 5.3.3. Table 5.5 lists the efficiency to use the conditions by the experiment. For using the efficient search, the cost reduced about 2^{14} times rather than the cost of the simple search.

5.4 Search Algorithm for $\mathbb{F}_3$-Subtraction

Here, we consider the search algorithm for the minimum construction of the $\mathbb{F}_3$-subtraction. The differences of the search between the $\mathbb{F}_3$-addition and subtraction are the input assignments R_a, R_b, the output value c and its bit-strings. For the output value, we only change the output bit-strings c_H and c_L.

Table 5.5: Number of Logical Instructions and Running Time of Search with Conditions in Sect. 5.3.3 ($\#R_a = \#R_b = \#R_c = 4$)

LI-Set	N_{max}	# of instructions	Running time
3	5	$2^{19.9}$	0.03 sec
3	6	$2^{25.9}$	1.57 sec
8	5	$2^{27.2}$	4.67 sec
8	6	$2^{35.2}$	1,222 sec

In the $\mathbb{F}_3$-addition search, the assignments R_a and R_b are commutative. However, R_a and R_b are not commutative assignments in the $\mathbb{F}_3$-subtraction. Thus we consider additional patterns of the assignment set. For the exhaustive search of the instruction sequences for computing the $\mathbb{F}_3$-subtraction, we use the following additional categorization of the assignments.

- $\#R_a = 4,\ \#R_b = \#R_c = 3$

  3'–i $R_b = R_c \subseteq R_a$

  3'–ii Assignment set of independent R_a, R_b, R_c

- $\#R_a = 4,\ \#R_b = 3,\ \#R_c = 4$

  4'–i $R_b \subseteq R_a = R_c$

  4'–ii Assignment set of independent R_a, R_b, R_c

The other procedure for searching the logical instruction sequences to compute the $\mathbb{F}_3$-subtraction uses the same setting of the search of the $\mathbb{F}_3$-addition.

5.5 Search Results and Some Examples

In our experiment, we searched the instruction sequences that can be constructed by using less than or equal to seven logical instructions, because Harrison et al. already showed the $\mathbb{F}_3$-addition with seven logical instructions. Table 5.6 lists the specific results of our search for the $\mathbb{F}_3$-addition, leading to the following general results.

1. There is no implementation of the addition $c \leftarrow a + b$ in $\mathbb{F}_3$ that uses less than six logical instructions from the set $\{$AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR$\}$ for any assignment set R_a, R_b, R_c.

2. There are many implementations of the addition $c \leftarrow a + b$ in $\mathbb{F}_3$ that use six logical instructions from the set $\{$AND, OR, XOR$\}$ in the assignment set R_a, R_b, R_c with $\#R_a = \#R_b = \#R_c = 3$.

3. In the natural assignment set and the logical instruction set $\{$AND, OR, XOR$\}$, seven instructions is the minimum number for constructing the $\mathbb{F}_3$-addition.

Hence, we have proved that the minimum number of logical instructions for computing the $\mathbb{F}_3$-addition is six.

The search results for the $\mathbb{F}_3$-subtraction show in Table 5.7. In this search, we experimented additional search where the assignment set is 4' and 3'. However, the results are almost same of the $\mathbb{F}_3$-addition and there is no implementation of the subtraction in $\mathbb{F}_3$ with less than six logical instructions.

There is Osvik's search [86] as a thing similar to our search. He indicated the efficient instruction sequences to compute the four-bit input and output S-box. His search optimizes for the implementation on x86 processors. For example, his search use AND, OR, XOR, NOT, MOV instructions, and the instruction sequences he indicated can be constructed by using five registers, In our search, we apply eight logical instructions which contain all binary operations except for trivial operations, and we do not consider NOT and MOV instructions and the number of usable registers.

In the following, we show some examples of computing the $\mathbb{F}_3$-addition and subtraction $c \leftarrow a \pm b$ with six logical instructions. Let $a = (a_h, a_l)$, $b = (b_h, b_l)$, and $c = (c_h, c_l)$ be elements in $\mathbb{F}_3$.

Example 1. In the case of the assignment $R_a = R_b = R_c = \{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$, the $\mathbb{F}_3$-addition can be computed by

$$\begin{cases} t_0 & \leftarrow & a_l \wedge b_l, \\ t_1 & \leftarrow & a_h \wedge b_h, \\ c_h & \leftarrow & t_0 \mid (t_1 \wedge a_l), \\ c_l & \leftarrow & t_1 \mid (t_0 \wedge a_h). \end{cases} \tag{5.3}$$

In this assignment, the negative element, $-a$, of $a \in \mathbb{F}_3$ can be computed by the conversion of exchanging a_h and a_l.

Table 5.6: Search Results for Computing the $\mathbb{F}_3$-Addition

$\#R_a$	$\#R_b$	$\#R_c$	Assignment set	LI-Set	# of instructions	Existence
3	3	3	1-iii	8	Less than 6	No
3	3	3	1-i	3	Less than 7	No
3	3	3	1-ii	3	6	Yes
3	3	4	2-ii	8	Less than 6	No
3	3	4	2-i	3	6	Yes
3	4	3	3-ii	8	Less than 6	No
3	4	3	3-ii	3	Less than 7	No
3	4	3	3-i	3	7	Yes
3	4	4	4-ii	8	Less than 6	No
3	4	4	4-i	3	6	Yes
4	4	3	5-ii	8	Less than 6	No
4	4	3	5-ii	3	Less than 7	No
4	4	3	5-i	3	7	Yes
4	4	4	6-ii	8	Less than 6	No
4	4	4	6-i	3	6	Yes

LI-Set LI-Set 3 = {AND, OR, XOR},
LI-Set 8 = {AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR}.

Assignment set The assignment set was R_a, R_b, and R_c as indicated in Sect. 5.3.1.

Table 5.7: Search Results for Computing the $\mathbb{F}_3$-Subtraction

$\#R_a$	$\#R_b$	$\#R_c$	Assignment set	LI-Set	# of instructions	Existence
3	3	3	1-iii	8	Less than 6	No
3	3	3	1-i	3	Less than 7	No
3	3	3	1-ii	3	6	Yes
3	3	4	2-ii	8	Less than 6	No
3	3	4	2-i	3	6	Yes
3	4	3	3-ii	8	Less than 6	No
3	4	3	3-ii	3	Less than 7	No
3	4	3	3-i	3	7	Yes
3	4	4	4-ii	8	Less than 6	No
3	4	4	4-i	3	6	Yes
4	3	3	3'-ii	8	Less than 6	No
4	3	3	3'-ii	3	Less than 7	No
4	3	3	3'-i	3	7	Yes
4	3	4	4'-ii	8	Less than 6	No
4	3	4	4'-i	3	6	Yes
4	4	3	5-ii	8	Less than 6	No
4	4	3	5-ii	3	Less than 7	No
4	4	3	5-i	3	7	Yes
4	4	4	6-ii	8	Less than 6	No
4	4	4	6-i	3	6	Yes

LI-Set LI-Set 3 = {AND, OR, XOR},
LI-Set 8 = {AND, OR, XOR, ANDN, ORN, XORN, NAND, NOR}.

Assignment set The assignment set was R_a, R_b, and R_c as indicated in Sects. 5.3.1 and 5.4.

Example 2. In the case of the assignment $R_a = R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$, the $\mathbb{F}_3$-addition can be computed as

$$\begin{cases} t_0 & \leftarrow & a_h \wedge b_l, \\ c_h & \leftarrow & (a_l \wedge b_h) \,\&\, t_0, \\ c_l & \leftarrow & (a_l \wedge b_l) \mid (t_0 \wedge b_h). \end{cases} \tag{5.4}$$

In this assignment, the negative element can be computed simply by $a_h \leftarrow a_h \wedge a_l$. In this case, although additional cost is required to compute the negative element, the subtraction $a - b$ can be computed using six logical instructions:

$$\begin{cases} t_0 & \leftarrow & a_l \wedge b_l, \\ c_h & \leftarrow & (a_h \wedge b_l) \,\&\, (t_0 \wedge b_h), \\ c_l & \leftarrow & (a_h \wedge b_h) \mid t_0. \end{cases} \tag{5.5}$$

Example 3. We also indicate the addition sequence using the redundant representations. In the case of the assignment $R_a = \{(0,0) \mapsto 0, (1,0) \mapsto 1, (1,1) \mapsto 2\}$ and $R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 0, (1,0) \mapsto 1, (1,1) \mapsto 2\}$, the $\mathbb{F}_3$-addition can be computed by

$$\begin{cases} t_0 & \leftarrow & b_h \,\&\, (b_l \wedge a_h \wedge a_l), \\ c_l & \leftarrow & a_l \wedge t_0, \\ c_h & \leftarrow & (a_h \wedge b_h) \mid t_0. \end{cases}$$

Example 4. We have found that the $\mathbb{F}_3$-addition can be computed using six logical instructions from the set {AND, OR, XOR, ANDN} even for the natural assignment set $R_a = R_b = R_c = \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$:

$$\begin{cases} c_h & \leftarrow & ((a_h \wedge b_h) \wedge a_l) \,\&\, \overline{(a_l \wedge b_l)}, \\ c_l & \leftarrow & ((a_l \wedge b_l) \wedge a_h) \,\&\, \overline{(a_h \wedge b_h)}. \end{cases}$$

This instruction sequence can efficiently compute the $\mathbb{F}_3$-addition by using four registers, requiring no additional register. Therefore, it is suitable for an implementation, such as SSE, in which the computed result overwrites one of the operands. It can be performed using registers r_i $(i = 0, 1, 2, 3)$ as indicated below.

1. $r_0 \leftarrow a_h,\ r_1 \leftarrow a_l,\ r_2 \leftarrow b_h,\ r_3 \leftarrow b_l$
2. $r_2 \leftarrow r_0 \wedge r_2,\ r_3 \leftarrow r_1 \wedge r_3$
3. $r_1 \leftarrow r_2 \wedge r_1,\ r_0 \leftarrow r_3 \wedge r_0$
4. $r_3 \leftarrow r_1$ ANDN $r_3,\ r_2 \leftarrow r_0$ ANDN r_2
5. $c_h \leftarrow r_3, c_l \leftarrow r_2$

5.6 Application to η_T Pairing over $\mathbb{F}_{3^m}$

In this section, we explain how to implement operations in $\mathbb{F}_{3^m}$ and the η_T pairing by using the new addition implementation proposed in the previous section. We give the operations' running times on an AMD Opteron processor model 275 (2.2 GHz), using GCC 4.1.2 with the `-O3` option under Linux/x86_64. Then, we use the `timeval` structure and the `gettimeofday` function to measure time.

5.6.1 Bit Representation of $\mathbb{F}_{3^m}$

Let $\mathbb{F}_{3^m}$ be given by $\mathbb{F}_3[x]/(f(x))$ where $\mathbb{F}_3[x]$ is the set of all polynomials over $\mathbb{F}_3$ and $f(x)$ is an irreducible polynomial over $\mathbb{F}_3$. An element A in $\mathbb{F}_{3^m}$ can be represented as the polynomial $\sum_{i=0}^{m-1} a_i x^i$, where each coefficient a_i is an element in $\mathbb{F}_3$. Recall that each coefficient is converted to two bits, such as $a_i = ((a_i)_h, (a_i)_l) \in (\mathbb{F}_2)^2$. By using bit-sliced representation [87], an element A in $\mathbb{F}_{3^m}$ can be represented as two bit-strings (A_H, A_L) of $\mathbb{F}_3$-elements as indicated below:

$$\begin{cases} A_H &= ((a_{m-1})_h, (a_{m-2})_h, \cdots, (a_0)_h)\,, \\ A_L &= ((a_{m-1})_l, (a_{m-2})_l, \cdots, (a_0)_l)\,. \end{cases} \tag{5.6}$$

The bit-strings are stored in two arrays of size $N = \lceil m/W \rceil$, where $W = 64$ is the word size of the 64-bit processor.

Here, we use the following three assignments to represent the coefficients a_i: The first assignment is the natural assignment shown by Harrison et al., which can compute the $\mathbb{F}_3$-addition with seven logical instructions. The second and third ones are assignments in which the $\mathbb{F}_3$-addition can be computed using six logical instructions, as indicated in Sect. 5.5. They are written as follows:

$$\begin{aligned} \textit{Natural}\text{ assignment} &: \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}, \\ \textit{Type 1}\text{ assignment} &: \{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}, \\ \textit{Type 2}\text{ assignment} &: \{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}. \end{aligned}$$

5.6.2 Addition and Subtraction in $\mathbb{F}_{3^m}$

When the elements $A, B, C \in \mathbb{F}_{3^m}$ are represented according to Eq. (5.6), the addition $C \leftarrow A + B$ in $\mathbb{F}_{3^m}$ can be computed by the $\mathbb{F}_3$-addition $c_i \leftarrow a_i + b_i$ for each coefficient of elements A and B. In the *Natural* and *Types 1* and *2* assignments, the $\mathbb{F}_3$-addition can be implemented according to Eqs. (5.1), (5.3) and (5.4), respectively. A logical instruction can compute W coefficients at one

Table 5.8: Running Time Comparison for Addition and Subtraction in $\mathbb{F}_{3^{509}}$ (μsec)

	Natural	*Type 1*	*Type 2*
Addition	0.0197	0.0166	0.0175
Subtraction	0.0199	0.0166	0.0176

time. Therefore, the $\mathbb{F}_{3^m}$-addition is implemented using $7N$ logical instructions in the *Natural* assignment, but in the *Types 1* and *2* assignments, it is implemented using $6N$ logical instructions.

Similarly, the subtraction $C \leftarrow A - B$ in $\mathbb{F}_{3^m}$ can be constructed using logical instructions. In the *Natural* and *Type 1* assignments, the subtraction can be performed as $A + (-B)$, and its cost is the same as that of the $\mathbb{F}_{3^m}$-addition. In the *Type 2* assignment, the subtraction can be implemented using $6N$ logical instructions according to Eq. (5.5).

To compare the running times, we used the finite field $\mathbb{F}_{3^{509}}$, since it achieves the 128-bit security level [71]. Table 5.8 lists the running time results for addition and subtraction in $\mathbb{F}_{3^{509}}$. Each listed running time is the average of 10^9 executions. For fair comparison, we used the same program code for implementing the logical instructions, and we did not use extensions such as SSE.

Theoretically, the running times of addition and subtraction in $\mathbb{F}_{3^m}$ for the *Types 1* and *2* assignments should be about $6/7$ of those for the *Natural* assignment, since the number of logical instructions for computing the $\mathbb{F}_3$-addition is reduced from seven to six. In our experiment, both addition and subtraction in $\mathbb{F}_{3^m}$ using $6N$ logical instructions (*Types 1* and *2*) were about $12 \sim 16\%$ ($\simeq 1/7$) faster than using $7N$ logical instructions (*Natural*). These operations are very simple and require hardly any computational cost except for computing $6N$ or $7N$ logical instructions. Therefore, the running times of these operations became faster, as predicted from the theoretical estimation, by reducing the number of logical instructions.

5.6.3 Multiplication in $\mathbb{F}_{3^m}$

The multiplication $A \cdot B$ in $\mathbb{F}_{3^m}$ consists of the polynomial multiplication $C' \leftarrow A \cdot B$ in $\mathbb{F}_3[x]$ and the reduction $C \leftarrow C' \bmod f(x)$. If $f(x)$ is an irreducible trinomial $f(x) = x^m + x^k + 2$, then the computational cost of the reduction is negligibly small compared to that of the polynomial multiplication.

The polynomial multiplication is constructed by using the addition in $\mathbb{F}_{3^m}$ and

Table 5.9: Running Time Comparison for Multiplication in $\mathbb{F}_{3^{509}}$ (μsec)

	Natural	*Type 1*	*Type 2*
Shift-and-addition ($w = 4$)	7.09	6.91	6.62
Left-to-right Comb ($w = 4$)	4.98	4.74	4.59

the shift Ax^i computed by $\sum_{j=0}^{m-1} a_j x^{j+i}$ from A and x^i . In our experiment, we implemented the shift-and-addition method and the left-to-right comb method with window width $w = 4$. Table 5.9 lists the running time results for multiplication in $\mathbb{F}_{3^{509}}$. We used the trinomial $f(x) = x^{509} + x^{358} + 2$ for the polynomial basis of $\mathbb{F}_{3^{509}}$. Each listed running time is the average of 10^6 executions.

The running time with the *Type 2* assignment was about a few percent faster than that with the *Type 1* assignment. For the *Type 1* assignment, the shift requires some additional cost. When the shift Ax^i is computed, the bits less than x^i are padded by zeros. Since $0 \in \mathbb{F}_3$ is assigned as $(1, 1)$ in this assignment, it is necessary to change these bits from zero to one. On the other hand, the negative element is computed by $A_h \leftarrow A_h \wedge A_l$ for the *Type 2* assignment, so that it requires N logical instructions.

The running time of the comb method with $w = 4$ for the *Type 2* assignment is about 8% faster than that for the *Natural* assignment. On the other hand, the shift-and-addition method requires more shifts than does the comb method, and thus, the running time of the shift-and-addition method with $w = 4$ for the *Type 2* assignment was only about 7% faster than that for the *Natural* assignment. As a result, we found that the comb method with $w = 4$ for the *Type 2* assignment was the fastest multiplication implementation under these conditions.

5.6.4 η_T Pairing over $\mathbb{F}_{3^m}$ and its Efficient Implementation

For comparison the efficiency of the η_T pairing, we implemented it with the following algorithms: the η_T pairing algorithm without cube root [99]; the loop unrolling technique [19]; an efficient multiplication in $\mathbb{F}_{3^{6m}}$ [49]; reuse of the precomputed table for the window multiplication in $\mathbb{F}_{3^m}$ [108]; and final exponentiation using the torus $T_2(\mathbb{F}_{3^{3m}})$ [101]. Then cubing in $\mathbb{F}_{3^m}$ was performed with an 11-bit lookup table that computes $\sum_{i=t}^{t+10} a_i x^{3i}$ for a positive integer or zero t, and the reduction by the irreducible polynomial. The size of the lookup table is 32×2^{11} bits = 8 KByte, and it can be store in the cache of the processor. Inversion in $\mathbb{F}_{3^m}$ was implemented by the extended Euclidean algorithm for a ternary polynomial.

Table 5.10: Running Time Comparison for η_T Pairing over $\mathbb{F}_{3^{509}}$ (μsec)

	Natural	*Type 1*	*Type 2*
η_T Pairing	17,525	17,238	16,295

Table 5.11: Running Time for Arithmetic in $\mathbb{F}_{3^m}$ and the η_T Pairing with the *Type 2* Assignment (μsec)

Degree m	97	193	353	509
Addition	0.006	0.010	0.014	0.018
Subtraction	0.006	0.010	0.014	0.018
Multiplication [1]	0.77	1.82	3.31	4.59
Cubing	0.073	0.122	0.182	0.282
Inversion	6.9	19.2	49.4	94.3
η_T Pairing [2]	615	2,611	8,299	16,295

[1] Comb method with window width $w = 4$.

[2] Loop unrolling [19], fast multiplication in $\mathbb{F}_{3^{6m}}$ [49], without cube root [99], final exponentiation using torus T_2 [101], reuse of precomputed table [108].

Table 5.10 lists the running time results for the η_T pairing over $\mathbb{F}_{3^{509}}$. Each listed running time is the average of 10^5 executions. We used the comb method with $w = 4$ to compute the multiplication in $\mathbb{F}_{3^m}$. As a result, we obtained a running time of about 16.3 milliseconds for computing the η_T pairing over $\mathbb{F}_{3^{509}}$.

The efficiency improvement by using the *Type 2* assignment instead of the *Natural* assignment had almost the same ratio as that for the multiplication using the comb method with $w = 4$, since the dominant computational cost of the η_T pairing is the multiplication in $\mathbb{F}_{3^m}$. The η_T pairing for the *Type 1* assignment, however, was only 2% faster than that for the *Natural* assignment, because the shift requires additional cost in the cubing and inversion for the *Type 1* assignment.

Finally, we present the running times for arithmetic in $\mathbb{F}_{3^m}$ and the η_T pairing using the *Type 2* assignment with several degrees. We used irreducible trinomials $f(x) = x^m + x^k + 2$ with a given $(m, k) = \{(97, 12), (193, 12), (353, 142), (509, 358)\}$. Page et al. [88] estimated that the discrete logarithm problems in $\mathbb{F}_{3^{6m}}$ with $m = 97, 193, 353$ have the security level of factoring 845, 1080, 1976 bits, respectively. Table 5.11 lists the running time results for computing arithmetic in $\mathbb{F}_{3^m}$ and the η_T pairing over $\mathbb{F}_{3^m}$.

5.7 Conclusion

In this chapter, we have described our search for logical instruction sequences implementing the $\mathbb{F}_3$-addition and subtraction in software. We considered a redundant representation of $\mathbb{F}_3$ in $(\mathbb{F}_2)^2$ and an independent assignment of each $\mathbb{F}_3$ via the addition map $\mathbb{F}_3 \times \mathbb{F}_3 \to \mathbb{F}_3$. We also deployed several logical instructions provided in many CPUs, namely $\{$AND, OR, XOR$\}$, as well as $\{$ANDN, ORN, XORN, NAND, NOR$\}$, available in the SPARC and Alpha architectures, MMX, SSE, and other implementations.

As a result, we found many implementations of the $\mathbb{F}_3$-addition and subtraction with only six logical instructions. Their representations in $(\mathbb{F}_2)^2$, however, do not use the natural assignment $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$, or their logical instructions include not only $\{$AND, OR, XOR$\}$ but also other ones. Moreover, we proved that the minimum number of logical instructions for constructing the $\mathbb{F}_3$-addition and subtraction is six in any representation of $\mathbb{F}_3$ in $(\mathbb{F}_2)^2$ by using any binary operation. Finally, we implemented the arithmetic in $\mathbb{F}_{3^m}$ and the η_T pairing by applying the new $\mathbb{F}_3$-addition and subtraction with six logical instructions. The resulting running time of the η_T pairing over $\mathbb{F}_{3^{509}}$ was approximately 7% faster than that using the $\mathbb{F}_3$-addition with seven logical instructions, on an AMD Opteron processor model 275 (2.2 GHz). Therefore, for an efficient implementation of the η_T pairing, using the minimum number of logical instructions for computing the $\mathbb{F}_3$-addition and subtraction is effective in software implementation.

Chapter 6

MapToPoint over Supersingular Elliptic Curves in $\mathbb{F}_{3^m}$

This chapter contains joint work with Tetsutaro Kobayashi, Gen Takahashi, and Tsuyoshi Takagi, which has been published in IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences [65].

6.1 Introduction

In pairing-based cryptosystems, many useful and convenient protocols such as ID-based cryptosystems [28] have been proposed using bilinearity. Various operations, such as pairing computation, exponentiation in finite fields, and point multiplication on elliptic curves, are required to construct the pairing-based cryptosystems.

MapToPoint is a hashing algorithm onto a point $P = (x, y)$ on an elliptic curve, and it is one of the required functions for constructing pairing-based cryptosystems. It has two types algorithm: one is that computes the y-coordinate from an x-coordinate or vice versa in Affine coordinate. Another is that computes using point multiplication of the point on an elliptic curve from an input integer. Here, we apply the first algorithm that directly computes the point from x or y-coordinate. For example of MapToPoint, in every encryption of the ID-based cryptosystem proposed by Boneh and Franklin [28], the point on the elliptic curve is computed from a user's identification (e.g. a user's e-mail address), and the MapToPoint is used to compute the point from the y-coordinate. The MapToPoint acts on-line after the coordinate is received.

For the η_T pairing in characteristic two or three, the following supersingular elliptic curves over the finite field with characteristic two or three are used:

$$\begin{aligned} E_2 &: y^2 + y = x^3 + x + b_2 \quad (b_2 = 0, 1), \\ E_3 &: y^2 = x^3 - x + b_3 \qquad (b_3 = \pm 1). \end{aligned}$$

In $p = 2, 3$, the sets of all rational points on E_p over $\mathbb{F}_{p^m}$ are denoted by $E(\mathbb{F}_{p^m})$, and the point is represented by $P = (x, y) \in E(\mathbb{F}_{p^m})$ $(x, y \in \mathbb{F}_{p^m})$.

In the conventional MapToPoint algorithm on E_2, the y-coordinate is computed from the x-coordinate using *half-trace* over $\mathbb{F}_{2^m}$ [40]. The *half-trace* over $\mathbb{F}_{2^m}$ was proposed by Fong et al. to achieve efficient point multiplication algorithm on elliptic curves over $\mathbb{F}_{2^m}$ using point-halving. Then the *half-trace* over $\mathbb{F}_{2^m}$ can be computed by a solution of equation $y^2 + y = c$, where $c \in \mathbb{F}_{2^m}$. The computational cost of this algorithm is $O(1)$ multiplications, $O(m)$ squarings, and $O(m)$ additions in $\mathbb{F}_{2^m}$.

For computing the MapToPoint on E_3, there exists two algorithms: the first computes the y-coordinate from the x-coordinate using a square root computation in $\mathbb{F}_{3^m}$ [9], which is slower than the other operations in $\mathbb{F}_{3^m}$. The computational cost of this algorithm is $O(\log m)$ multiplications, $O(m)$ cubings, and $O(1)$ additions in $\mathbb{F}_{3^m}$. Cubing in $\mathbb{F}_{3^m}$ and squaring in $\mathbb{F}_{2^m}$ are respectively more efficient than multiplication in each field $\mathbb{F}_{3^m}$ and $\mathbb{F}_{2^m}$. Hence, we reduce the number of multiplications for an efficient MapToPoint on E_3. The second computes the x-coordinate from the y-coordinate using an $(m-1) \times (m-1)$ matrix over $\mathbb{F}_3$ [8]. Its computational cost is $O(1)$ multiplications and $O(m)$ additions in $\mathbb{F}_{3^m}$. However, it requires the off-line memory to store the matrix which consist of about m $\mathbb{F}_{3^m}$-elements.

We construct an efficient MapToPoint algorithm on E_3 using *1/3-trace* over $\mathbb{F}_{3^m}$. The *1/3-trace* $\sum_{i=0}^{\lfloor m/3 \rfloor} c^{3^{3i}}$ can be computed without multiplication or off-line memory, and the x-coordinate is computed from the y-coordinate since a solution x of the equation $x^3 - x = c$ can be computed using the *1/3-trace* for a given $c \in \mathbb{F}_{3^m}$. The computational cost of the proposed MapToPoint algorithm is $O(1)$ multiplications, $O(m)$ cubings, and $O(m)$ additions in $\mathbb{F}_{3^m}$, and it has almost the same computational cost as the MapToPoint algorithm on E_2. The proposed algorithm then requires less than m $\mathbb{F}_3$-elements to be stored in the off-line memory to efficiently compute *trace* over $\mathbb{F}_3$.

Moreover, we implement the proposed and conventional MapToPoint algorithms over $E(\mathbb{F}_{3^m})$ on an AMD Opteron processor model 275 (2.2 GHz). We choose $m = 97, 193, 353, 509$ as the extension degree. As a result, the running

time of the proposed MapToPoint over $E(\mathbb{F}_{3^{509}})$, that has 128-bit security [71], is about 116 microseconds, which is approximately 35% faster than that of the conventional MapToPoint using the square root computation.

Firstly, we explain the conventional algorithms for computing MapToPoint on E_2 and E_3. Then the proposed MapToPoint algorithm on E_3 is described, and we compares the computational cost of the proposed and conventional algorithms, Finally we show the running time results of their algorithms in our implementation, and the chapter is conclude.

6.2 Conventional MapToPoint Algorithms

Let a finite field $\mathbb{F}_{p^m}$ be given by $\mathbb{F}_p[x]/(f(x))$, where $f(x)$ is an irreducible polynomial in $\mathbb{F}_p[x]$ for given $p = 2, 3$, and $m \in \mathbb{N}$. Each element $c \in \mathbb{F}_{p^m}$ is represented as $c = \sum_{i=0}^{m-1} c_i x^i$, where $c_i \in \mathbb{F}_p$, by polynomial representation. Recall that, for $c \in \mathbb{F}_{p^m}$, *trace* $\mathrm{Tr}_{\mathbb{F}_{p^m}/\mathbb{F}_p}(c)$ over $\mathbb{F}_{p^m}$ is defined by $\mathrm{Tr}_{\mathbb{F}_{p^m}/\mathbb{F}_p}(c) = \sum_{i=0}^{m-1} c^{p^i}$, simply written by Tr_p after this.

Theorem 6.1. *Let $c, d \in \mathbb{F}_{p^m}$, $\alpha \in \mathbb{F}_p$ with $p = 2, 3$. The trace satisfies the following properties:*

(i) $\mathrm{Tr}_p(c) = \mathrm{Tr}_p(c^p) = \mathrm{Tr}_p(c)^p \in \mathbb{F}_p$;

(ii) $\mathrm{Tr}_p(c + d) = \mathrm{Tr}_p(c) + \mathrm{Tr}_p(d)$;

(iii) $\mathrm{Tr}_p(\alpha) = m\alpha$.

The supersingular elliptic curves in characteristics 2 and 3 correspond to $E_2 : y^2 + y = x^3 + x + b_2$ $(b_2 = 0, 1)$ and $E_3 : y^2 = x^3 - x + b_3$ $(b_3 = \pm 1)$. Then, the sets of all rational points on E_p over $\mathbb{F}_{p^m}$ are denoted by $E(\mathbb{F}_{p^m})$, and point P on the curve is denoted as $P = (x, y) \in E(\mathbb{F}_{p^m})$ $(x, y \in \mathbb{F}_{p^m})$.

6.2.1 MapToPoint over $E(\mathbb{F}_{2^m})$ using *half-trace*

In MapToPoint over $E(\mathbb{F}_{2^m})$, the y-coordinate $y \in \mathbb{F}_{2^m}$ is computed from the x-coordinate $x \in \mathbb{F}_{2^m}$ using *trace* and *half-trace* over $E(\mathbb{F}_{2^m})$ [40]. Assuming that m is odd, *half-trace* $\mathrm{HfTr}(c)$ over $\mathbb{F}_{2^m}$ is defined as:

$$\mathrm{HfTr}(c) = \sum_{i=0}^{\lfloor m/2 \rfloor} c^{2^{2i}} = c + c^{2^2} + c^{2^4} + \cdots + c^{2^{m-1}},$$

Algorithm 6.1 MapToPoint over $E(\mathbb{F}_{2^m})$ [40]

INPUT: $x \in \mathbb{F}_{2^m}$ s.t. $\mathrm{Tr}_2(x^3 + x + b_2) = 0$

OUTPUT: $P = (x, y) \in E(\mathbb{F}_{2^m})$

1: $s \leftarrow t \leftarrow x^3 + x + b_2$
2: **for** $i \leftarrow 1$ **to** $\lfloor m/2 \rfloor$ **do**
3: $\quad t \leftarrow t^{2^2}$
4: $\quad s \leftarrow s + t$
5: **end for**
6: **return** (x, s)

where c is the element in $\mathbb{F}_{2^m}$. Then *half-trace* over $\mathbb{F}_{2^m}$ satisfies the following properties.

Theorem 6.2. *Let $c, d \in \mathbb{F}_{2^m}$. Then we obtain:*

(i) $\mathrm{HfTr}(c + d) = \mathrm{HfTr}(c) + \mathrm{HfTr}(d)$.

(ii) $\mathrm{HfTr}(c)$ *is a solution of equation* $x^2 + x = c + \mathrm{Tr}_2(c)$.

(iii) $\mathrm{HfTr}(c) = \mathrm{HfTr}(c^2) + c + \mathrm{Tr}_2(c)$.

In the MapToPoint algorithm over $E(\mathbb{F}_{2^m})$ using *half-trace*, point $P = (x, y)$ in $E(\mathbb{F}_{2^m})$ is computed as follows: first, we choose random element x in $\mathbb{F}_{2^m}$, and compute $t \leftarrow x^3 + x + b_2$. If $\mathrm{Tr}_2(t) = 0$, we then compute $y \leftarrow \mathrm{HfTr}(t)$; otherwise, we compute the above operation again. The probability, which is satisfied with $\mathrm{Tr}_2(c) = 0$, is 1/2 for random element $c \in \mathbb{F}_{2^m}$. Algorithm 6.1 shows the detail of the MapToPoint algorithm over $E(\mathbb{F}_{2^m})$. The computational cost of Algorithm 6.1 is 1 multiplication, m squarings, and $\lfloor m/2 \rfloor + 1 + b_2$ additions in $\mathbb{F}_{2^m}$, and it requires less than m $\mathbb{F}_2$-elements to be stored in the off-line memory to efficiently compute *trace* over $\mathbb{F}_{2^m}$.

Note that there are more efficient algorithms for computing $\mathrm{HfTr}(c)$ by precomputing *half-trace* $\mathrm{HfTr}(x^i)$ off-line in [40], where $c \in \mathbb{F}_{2^m}$. They can be computed by less than m additions in $\mathbb{F}_{2^m}$ to compute $\mathrm{HfTr}(c)$. However, they require additional off-line memory to store the precomputed results of each *half-trace* $\mathrm{HfTr}(x^i)$.

6.2.2 MapToPoint over $E(\mathbb{F}_{3^m})$ using Square Root

The simple MapToPoint algorithm over $E(\mathbb{F}_{3^m})$ is constructed using a square root computation in $\mathbb{F}_{3^m}$ [9]. In this algorithm, point $P = (x, y) \in E(\mathbb{F}_{3^m})$ is

computed as follows: first, we choose random element x in $\mathbb{F}_{3^m}$, and compute $t \leftarrow x^3 - x + b_3$. If t is a quadratic residue, we then compute $y \leftarrow t^{1/2}$ using the square root computation in $\mathbb{F}_{3^m}$; otherwise, we compute the above operation again.

The square root computation in $\mathbb{F}_{3^m}$ is slower than the other operations in $\mathbb{F}_{3^m}$, such as the multiplication. An efficient algorithm for computing the square root in $\mathbb{F}_{3^m}$ has been shown by Barreto et al. [9]. Assuming that element $c \in \mathbb{F}_{3^m}$ is a quadratic residue and m is odd, solution y of equation $y^2 = c$ is computed by using $y = c^{(3^m+1)/4}$. The exponent $(3^m + 1)/4$ of c can then be modified into $6 \cdot \sum_{i=0}^{k-1} 3^{2i} + 1$, where $k = (m-1)/2$. Therefore, the solution y of equation $y^2 = c$ is computed as $y = c \cdot (c^{\sum_{i=0}^{k-1} 3^{2i}})^6$ and it computes using an analogy of the Itoh-Tsujii inversion [60]. Refer to Sect. 4.2.7 for the detail. In order to check efficiently whether c is the quadratic residue, we use the Legendre symbol of a polynomial over $\mathbb{F}_3$ (see Algorithm 2.1).

We denote, by the binary representation $(k_{\ell-1}, ..., k_0)_2$ ($k_{\ell-1} = 1$, $k_i \in \mathbb{F}_2$ for $i = 0, ..., \ell - 2$) and by $hw(k)$, the Hamming weight of the binary representation of k. The MapToPoint over $E(\mathbb{F}_{3^m})$ using the square root computation in $\mathbb{F}_{3^m}$ is executed using Algorithm 6.2. The computational cost of Algorithm 6.2 is $\lfloor \log k \rfloor + hw(k) + 1$ multiplications, $m - 1$ cubings, and 2 additions in $\mathbb{F}_{3^m}$.

6.2.3 MapToPoint over $E(\mathbb{F}_{3^m})$ using Matrix

Barreto and Kim proposed the MapToPoint algorithm over $E(\mathbb{F}_{3^m})$ using an $(m-1) \times (m-1)$ matrix over $\mathbb{F}_3$ [8], which can compute a solution x of the equation $y^2 - b_3 = x^3 - x$. The x-coordinate of $P \in E(\mathbb{F}_{3^m})$ in this algorithm is computed as follows: first, we choose random element y in $\mathbb{F}_{3^m}$, and compute $t \leftarrow y^2 - b_3$. If $\mathrm{Tr}_3(t) = 0$, we then compute the solution x of $t = x^3 - x$ using the matrix; otherwise, we compute the above operation again. The computational cost of this algorithm is 1 multiplication and m additions in $\mathbb{F}_{3^m}$, and it is required to store about m $\mathbb{F}_{3^m}$-elements in the off-line memory. In addition, less than m $\mathbb{F}_3$-elements are stored in the off-line memory to efficiently check whether $\mathrm{Tr}_3(t) = 0$.

6.3 Proposed MapToPoint Algorithm

This section describes an efficient MapToPoint algorithm over $E(\mathbb{F}_{3^m})$ using *trace* and *1/3-trace*. The x-coordinate is computed from the y-coordinate in the proposed algorithm.

Here, in order to compute a solution x of the equation $x^3 - x = y^2 - b_3$,

Algorithm 6.2 MapToPoint over $E(\mathbb{F}_{3^m})$ using Square Root [9]

INPUT: $x \in \mathbb{F}_{3^m}$ s.t. $(x^3 - x + b_3)$ is a quadratic residue,
$k = (m-1)/2 = (k_{\ell-1}, \cdots, k_0)_2$

OUTPUT: $P = (x, y) \in E(\mathbb{F}_{3^m})$

1: $s \leftarrow t \leftarrow x^3 - x + b_3$
2: $k' \leftarrow 1 \in \mathbb{N}$
3: **for** $i \leftarrow \ell - 2$ **down to** 0 **do**
4: $\quad u \leftarrow t$
5: $\quad$ **for** $j \leftarrow 0$ **to** $k' - 1$ **do**
6: $\quad\quad u \leftarrow u^{3^2}$
7: $\quad$ **end for**
8: $\quad k' \leftarrow 2k' + k_i \in \mathbb{N}$
9: $\quad t \leftarrow t \cdot u$
10: $\quad$ **if** $k_i = 1$ **then**
11: $\quad\quad t \leftarrow s \cdot t^{3^2}$
12: $\quad$ **end if**
13: **end for**
14: $s \leftarrow s \cdot t^6$
15: **return** (x, s)

we define *1/3-trace* over $\mathbb{F}_{3^m}$. Assuming that $m \bmod 3 \neq 0$ and $r = m \bmod 3$, *1/3-trace* $\mathrm{CrTr}(c)$ over $\mathbb{F}_{3^m}$ is defined as:

$$\mathrm{CrTr}(c) = \sum_{i=0}^{\lfloor m/3 \rfloor} c^{3^{3i}} = c + c^{3^3} + c^{3^6} + \cdots + c^{3^{m-r}},$$

where $c \in \mathbb{F}_{3^m}$. Then, the *1/3-trace* over $\mathbb{F}_{3^m}$ satisfies the following properties similar to the *half-trace* over $\mathbb{F}_{2^m}$.

Property: Let $c, d \in \mathbb{F}_{3^m}$ and $r = m \bmod 3$. Then we obtain the following properties for the *1/3-trace* over $\mathbb{F}_{3^m}$:

(i) $\mathrm{CrTr}(c + d) = \mathrm{CrTr}(c) + \mathrm{CrTr}(d)$,

(ii) $\mathrm{CrTr}(c) + \mathrm{CrTr}(c^3) + \mathrm{CrTr}(c^9) = \sum_{i=0}^{2-r} c^{3^i} + \mathrm{Tr}_3(c)$.

By using the *1/3-trace* over $E(\mathbb{F}_{3^m})$, the solution x of the equation $t = x^3 - x$ can be computed by using $\mathrm{CrTr}(t)^3 - \mathrm{CrTr}(t)$ when $m \equiv 2 \pmod 3$ and $t -$

Algorithm 6.3 Proposed MapToPoint over $E(\mathbb{F}_{3^m})$

INPUT: $y \in \mathbb{F}_{3^m}$ s.t. $\mathrm{Tr}_3(y^2 - b_3) = 0$

OUTPUT: $P = (x, y) \in E(\mathbb{F}_{3^m})$

1: $s \leftarrow t \leftarrow u \leftarrow y^2 - b_3$
2: **for** $i \leftarrow 1$ **to** $\lfloor m/3 \rfloor$ **do**
3: $\quad t \leftarrow t^{3^3}$
4: $\quad s \leftarrow s + t$
5: **end for**
6: $s \leftarrow s^3 - s$
7: **if** $m \bmod 3 = 1$ **then**
8: $\quad s \leftarrow u - s$
9: **end if**
10: **return** (s, y)

$\mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)$ when $m \equiv 1 \pmod 3$, where $t = y^2 - b_3$ with $\mathrm{Tr}_3(t) = 0$. Note that the algorithm in the case of $m \equiv 0 \pmod 3$ cannot be constructed by using *trace*, or *1/3-trace* over $\mathbb{F}_{3^m}$.

Let y be an element in $\mathbb{F}_{3^m}$ with $\mathrm{Tr}_3(y^2 - b_3) = 0$. The proposed MapToPoint algorithm over $E(\mathbb{F}_{3^m})$ is executed by Algorithm 6.3. In Algorithm 6.3, Steps 2–5 are computed by the *1/3-trace* $\mathrm{CrTr}(s)$ over $\mathbb{F}_{3^m}$, and Step 6 is computed by $\mathrm{CrTr}(s)^3 - \mathrm{CrTr}(s)$ for $s = y^2 - b_3$. Then, Steps 7–9 are computed by $s - (\mathrm{CrTr}(s)^3 - \mathrm{CrTr}(s))$ in the case of $m \bmod 3 = 1$. The computational cost of Algorithm 6.3 is 1 multiplication, $m + 1 - r$ cubings and $\lfloor m/3 \rfloor + 4 - r$ additions in $\mathbb{F}_{3^m}$. To compute the *trace* over $\mathbb{F}_{3^m}$ efficiently, this algorithm is required to store less than m $\mathbb{F}_3$-elements, which is satisfied with $\mathrm{Tr}_3(x^i) \neq 0$, in the off-line memory.

We next establish the correctness of Algorithm 6.3.

Case of $m \equiv 2 \pmod 3$**.** When $m \equiv 2 \pmod 3$, point $P = (x, y) \in E(\mathbb{F}_{3^m})$ is computed as follows: first, we compute $t \leftarrow y^2 - b_3$ for $y \in \mathbb{F}_{3^m}$. If $\mathrm{Tr}_3(t) = 0$, we obtain $\mathrm{CrTr}(t) + \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)^9 = t$ from Property (ii). Then, we obtain

$$\mathrm{CrTr}(t) + \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)^9 = (\mathrm{CrTr}(t)^3 - \mathrm{CrTr}(t))^3 - (\mathrm{CrTr}(t)^3 - \mathrm{CrTr}(t)).$$

Therefore, if $\mathrm{Tr}_3(t) = 0$, then $\mathrm{CrTr}(t)^3 - \mathrm{CrTr}(t)$ is the solution of equation $x^3 - x = t$, and we get the x-coordinate of point P.

Case of $m \equiv 1 \pmod 3$**.** In the case of $m \equiv 1 \pmod 3$, point $P = (x, y) \in E(\mathbb{F}_{3^m})$ is computed as follows: first, we compute $t \leftarrow y^2 - b_3$ for $y \in \mathbb{F}_{3^m}$. From Property (ii), we have

$$-(\mathrm{CrTr}(t) + \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)^9) = -(t + t^3 + \mathrm{Tr}_3(t)),$$

and we get

$$\begin{aligned}(t^3 - t) &- (\mathrm{CrTr}(t) + \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)^9) \\ &= (t^3 - t) - (t + t^3 + \mathrm{Tr}_3(t)) \\ &= t - \mathrm{Tr}_3(t).\end{aligned}$$

Then, we obtain

$$\begin{aligned}&(t^3 - t) - (\mathrm{CrTr}(t) + \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)^9) \\ &\quad = (t - \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t))^3 - (t - \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)).\end{aligned}$$

Therefore, if $\mathrm{Tr}_3(t) = 0$, then $t - \mathrm{CrTr}(t)^3 + \mathrm{CrTr}(t)$ is the solution of the equation $x^3 - x = t$, and we obtain the x-coordinate.

For all $c \in \mathbb{F}_{3^m}$, we have $\mathrm{Tr}_3(c) = 0$ if and only if equation $x^3 - x = c$ has solution $\beta \in \mathbb{F}_{3^m}$. Let c be a random element in $\mathbb{F}_{3^m}$. From the properties of the *trace*, $\mathrm{Tr}_3(c) = \sum_{i=0}^{m-1} c_i \mathrm{Tr}_3(x^i)$. Here, $\mathrm{Tr}_3(x^i)$ is included in $\mathbb{F}_3$, and each coefficient c_i is random in $\mathbb{F}_3$ for all $0 \leq i < m$. Therefore, if $\mathrm{Tr}_3(x^i) = 0$, then, $c_i \mathrm{Tr}_3(x^i) = 0$; otherwise, $c_i \mathrm{Tr}_3(x^i)$ takes a random value in $\{0, 1, 2\}$ for all $0 \leq i < m$. Hence, the probability with which $\mathrm{Tr}_3(c)$ takes each value in $\mathbb{F}_3$ is 1/3, when the element c is chosen uniformly at random from $\mathbb{F}_{3^m}$.

The cubing and addition in $\mathbb{F}_{3^m}$ are represented as a linear map. Therefore, in this algorithm, the x-coordinate of the point in $E(\mathbb{F}_{3^m})$ can be efficiently computed using the matrix, which computes $(\mathrm{CrTr}(t)^3 - \mathrm{CrTr}(t))$ for a given $t \in \mathbb{F}_{3^m}$. By using the matrix, the x-coordinate can be computed by approximately 1 multiplication and m additions in $\mathbb{F}_{3^m}$ from the y-coordinate $y \in \mathbb{F}_{3^m}$, which is satisfied with $\mathrm{Tr}_3(y^2 - b_3) = 0$. Then, m $\mathbb{F}_{3^m}$-elements are stored in the off-line memory for storing the matrix. Since the MapToPoint using the matrix constructed by *1/3-trace* uses the $m \times m$ matrix, the additional computational cost and additional off-line memory in this method is 1 $\mathbb{F}_{3^m}$-addition and about 2 $\mathbb{F}_{3^m}$-elements compared with the conventional MapToPoint algorithm using the matrix indicated in Sect. 6.2.3. However, the additional computational cost and additional memory cost are about 1% by $m = 97$, and they become small when m increases. Then the computational cost and the memory cost of both algorithms using the matrix are the almost same.

Table 6.1: Computational Cost of Proposed and Conventional MapToPoint Algorithms over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$

Field	Algorithm	Computational Cost [1]	Memory [2]
$\mathbb{F}_{2^m}$	*half-trace* (Alg.6.1)	$O(\mathbf{M_2} + m\mathbf{S_2} + m\mathbf{A_2})$	1
$\mathbb{F}_{3^m}$	Square Root (Alg.6.2)	$O((\log m)\mathbf{M_3} + m\mathbf{C_3} + \mathbf{A_3})$	0
	Matrix (Sect. 6.2.3)	$O(\mathbf{M_3} + m\mathbf{A_3})$	about m
	Proposed Alg. (Alg.6.3)	$O(\mathbf{M_3} + m\mathbf{C_3} + m\mathbf{A_3})$	1

[1] $\mathbf{M_p}$, $\mathbf{S_p}$, $\mathbf{C_p}$, $\mathbf{A_p}$ correspond to the computational cost of multiplication, squaring, cubing and addition in $\mathbb{F}_{p^m}$.

[2] Memory is the required off-line memory to store the number of $\mathbb{F}_{p^m}$-elements.

6.4 Comparison of Proposed and Conventional MapToPoint Algorithms

6.4.1 Theoretical Estimate of Cost for MapToPoint

We compare the computational costs and memory costs of computing MapToPoint using the proposed and conventional algorithms in this section. Table 6.1 lists the computational cost of MapToPoint algorithms over $E(\mathbb{F}_{p^m})$.

The computational cost in the proposed algorithm is 1 multiplication, $m+1-r$ cubings and $\lfloor m/3 \rfloor + 4 - r$ additions. By comparing of the proposed algorithm and the conventional algorithm using the square root computation over $E(\mathbb{F}_{3^m})$, the number of multiplications in $\mathbb{F}_{3^m}$ is reduced from $\lfloor \log k \rfloor + hw(k) + 1$ to 1, and the number of multiplications of the proposed algorithm becomes the same as that computed with the conventional algorithm over $\mathbb{F}_{2^m}$. Moreover, the number of cubings in $\mathbb{F}_{3^m}$ is almost the same as that with the conventional algorithm using the square root computation over $\mathbb{F}_{3^m}$.

In the proposed algorithm, by precomputing $\mathrm{Tr}_3(x^i)$ off-line for all $0 \leq i < m$, we can efficiently check whether $\mathrm{Tr}_3(y^2 - b_3) = 0$ for $y \in \mathbb{F}_{3^m}$. Then, the required off-line memory for storing the precomputed results is less than m $\mathbb{F}_3$-elements in the proposed algorithm and the conventional algorithm using the matrix.

6.4.2 Implementation Results for MapToPoint over $E(\mathbb{F}_{3^m})$

We implement arithmetic in $\mathbb{F}_{3^m}$ and the proposed and conventional MapToPoint algorithms over $E(\mathbb{F}_{3^m})$. We apply $m = 97, 193, 353, 509$ for the implementation. Then we use irreducible trinomials $f(x) = x^m + x^k + 2$ with a given

Table 6.2: Running Time Results of Arithmetic in $\mathbb{F}_{3^m}$ and MapToPoint Algorithms over $E(\mathbb{F}_{3^m})$ (μsec)

Degree m		97	193	353	509
Addition		0.006	0.010	0.014	0.018
Multiplication		0.806	1.765	3.244	4.624
Cubing		0.056	0.107	0.167	0.211
MapToPoint	Square Root (Alg.6.2)	11.75	36.71	95.63	178.30
	Matrix (Sect. 6.2.3)	1.90	4.56	9.15	16.03
	Proposed Alg. (Alg.6.3)	6.40	22.88	64.39	115.52

$(m, k) = \{(97, 12), (193, 12), (353, 142), (509, 358)\}$. To efficiently compute addition, multiplication, and cubing in $\mathbb{F}_{3^m}$, we use the following algorithms: addition is constructed by the logical instruction sequence with 6 logical instructions [63]. Multiplication is constructed by using LR-comb algorithm with window width $w = 4$. Cubing is performed with an 11-bit look-up table that computes $\sum_{i=t}^{t+10} a_i x^{3i}$ for part of an element $a \in \mathbb{F}_{3^m}$ and $t \geq 0$. Then, we implement using C language, and give the operations' running times on an AMD Opteron processor model 275 (2.2 GHz), using GCC 4.1.2 with the `-O3` option under Linux/x86_64. The `timeval` structure and the `gettimeofday` function are used to measure running time.

Table 6.2 lists the running time results for computing arithmetic in $\mathbb{F}_{3^m}$ and the proposed and conventional MapToPoint algorithms over $E(\mathbb{F}_{3^m})$. The running time of arithmetic in $\mathbb{F}_{3^m}$ is the average of 10^9 executions, and that of the MapToPoint algorithms over $E(\mathbb{F}_{3^m})$ is the average of 10^7 executions.

In our implementation, the cubing in $\mathbb{F}_{3^{509}}$ is at least 20 times as fast as the multiplication in $\mathbb{F}_{3^{509}}$. In the other fields, the cubing is at least 10 times as fast as the multiplication. The ratio of the timing of the cubing to that of the multiplication increases when we choose the bigger m. The conventional MapToPoint algorithm using the square root is required by $\lfloor \log k \rfloor + hw(k) + 1$ multiplications. Then, the number of the cubing in the proposed MapToPoint algorithm and the conventional MapToPoint algorithm using the square root is almost the same. As a result, the proposed MapToPoint with each degree m is $33 \sim 46\%$ faster than the conventional MapToPoint using the square root, respectively, and especially the proposed MapToPoint over $E(\mathbb{F}_{3^{509}})$ is about 35% faster than the conventional MapToPoint over $E(\mathbb{F}_{3^{509}})$ using the square root. Since the number of the multiplication is $O(\log m)$, the efficiency of the new MapToPoint becomes small asymptotically.

Table 6.3: Running Time Results to Check whether $\mathrm{Tr}_3(c) = 0$ and c is Quadratic Residue in $\mathbb{F}_{3^m}$ (μsec)

Degree m	97	193	353	509
Check whether $\mathrm{Tr}_3(c) = 0$	0.015	0.016	0.016	0.028
Check whether c is quadratic residue	9.713	20.209	46.365	73.092

However, we use $m = 509$ at most as the degree m in fact, and the new MapToPoint is effective to compute the point on $E(\mathbb{F}_{3^m})$.

6.4.3 Implementation Results for Checking Conditions of Input Coordinate

We measure the running time to check whether $\mathrm{Tr}_3(c) = 0$ and c is the quadratic residue for a given $c \in \mathbb{F}_{3^m}$. To compute $\mathrm{Tr}_3(c) = 0$ efficiently, we precompute $\mathrm{Tr}_3(x^i)$ off-line and store the results $\mathrm{Tr}_3(x^i)$, satisfied that $\mathrm{Tr}_3(x^i) \neq 0$. In $\mathbb{F}_{3^{509}}$ with $f(x) = x^{509} + x^{358} + 2$, the number of $\mathrm{Tr}_3(x^i)$ which satisfies the condition $\mathrm{Tr}_3(x^i) \neq 0$ is only 4 elements x^0, x^{151}, x^{302}, x^{453}, and the required off-line memory is 4 $\mathbb{F}_3$-elements. By the other fields, the number of $\mathrm{Tr}_3(x^i)$ which satisfies $\mathrm{Tr}_3(x^i) \neq 0$ is 2. Then, we use the Legendre symbol to check whether c is the quadratic residue. Table 6.3 lists the running time results to check whether $\mathrm{Tr}_3(c) = 0$ and c is the quadratic residue. The running time of each operation is the average of 10^9 executions.

The probability with which c satisfies $\mathrm{Tr}_3(c) = 0$ is 1/3, and that with which c is the quadratic residue is 1/2, where c is a random element in $\mathbb{F}_{3^m}$. However, by precomputing $\mathrm{Tr}_3(x^i)$ off-line, $\mathrm{Tr}_3(c)$ can be computed efficiently. In $\mathbb{F}_{3^{509}}$ with $f(x) = x^{509} + x^{358} + 2$, $\mathrm{Tr}_3(c)$ can be computed by

$$c_0\mathrm{Tr}_3(x^0) + c_{151}\mathrm{Tr}_3(x^{151}) + c_{302}\mathrm{Tr}_3(x^{302}) + c_{453}\mathrm{Tr}_3(x^{453}),$$

and its computational cost is only 4 multiplications and 3 additions in $\mathbb{F}_3$. $\mathrm{Tr}_3(c)$ with the other m can be computed by several multiplication and addition in $\mathbb{F}_3$. Then, the Legendre symbol computation requires many modulo operations over $\mathbb{F}_3[x]$. In addition, the timing to compute the quadratic residue become slower when m increases. Thus, the running time to check whether $\mathrm{Tr}_3(c) = 0$ is faster than that to check whether c is the quadratic residue.

6.5 Conclusion

We proposed an efficient MapToPoint algorithm on E_3 in characteristic three by using *1/3-trace* over $\mathbb{F}_{3^m}$. The *1/3-trace* over $\mathbb{F}_{3^m}$ we proposed can compute a solution of equation $x^3 - x = c$ for $c \in \mathbb{F}_{3^m}$, and by using *1/3-trace*, the proposed algorithm can compute the x-coordinate of the point on $E(\mathbb{F}_{3^m})$ from the y-coordinate. The computational cost of the new algorithm is $O(1)$ multiplications, $O(m)$ cubings and $O(m)$ additions in $\mathbb{F}_{3^m}$, and we reduced the number of multiplications in $\mathbb{F}_{3^m}$ from $O(\log m)$ to $O(1)$. The proposed algorithm requires the off-line memory to store less than m elements in $\mathbb{F}_3$ to efficiently compute *trace* over $\mathbb{F}_{3^m}$.

Moreover, we compared the running time of the proposed and conventional MapToPoint algorithms over $E(\mathbb{F}_{3^m})$ with $m = 97, 193, 353, 509$ by the software implementation. As a result, the running time of the proposed MapToPoint algorithm over $E(\mathbb{F}_{3^{509}})$ was about 116 microseconds on an AMD Opteron processor model 275 (2.2 GHz), which is approximately 35% faster than that of the conventional MapToPoint algorithm using the square root computation.

Chapter 7

Experiment and Timing Results

7.1 Introduction

In order to use practical pairing-based cryptosystems, it is necessary to implement and measure running time of functions that are applied in the cryptosystems. Many timing results of various pairing, Tate pairing [82, 81], η_T pairing [7], Ate pairing [57], Optimal pairing [110] with Barreto-Naehrig curves [11], etc., are shown until now, and their pairings are implemented on many devices such as PC [1, 3, 4, 22, 23, 50, 54, 109], a mobile phone [61, 66, 114], a sensor node [100, 106], a smart card [97], and FPGA [16, 18, 20, 39, 24, 68, 103] and ASIC [21]. These results mainly showed the timing of the pairing computations with some parameters.

We described the detailed algorithms of arithmetic in $\mathbb{F}_{3^m}$, $\mathbb{F}_{3^{3m}}$, $\mathbb{F}_{3^{6m}}$, arithmetic on supersingular elliptic curves $E(\mathbb{F}_{3^m})$, and η_T pairing computation over $\mathbb{F}_{3^m}$, whose functions are used for making pairing-based cryptosystems, in the previous chapters. In this chapter, we show the parameters and experimental environment for implementing the functions and measuring the running time of the functions in C and Java on PC. C and Java languages are generally used as a program language frequently for constructing systems.

We showed the timing results by using some devices in our former references [63, 64, 65, 66, 67, 99]. In this chapter, the timing results are newly experimented on a state-of-the-art device, which is equipped with an Intel Core i7-950 processor. We re-implement all functions by using some implementation techniques from scratch. In addition, we additionally implement fast point multiplication over $E(\mathbb{F}_{3^m})$ and exponentiation in $\mathbb{F}_{3^{6m}}$.

As the result of our implementation and experiment, the running time of the η_T

pairing over $\mathbb{F}_{3^{509}}$, the point multiplication over $E(\mathbb{F}_{3^{509}})$ and the exponentiation in $\mathbb{F}_{3^{6\cdot 509}}$, that have the 128-bit security of AES, are 9.8, 5.1, 8.6 milliseconds in C, and 33.2, 15.5, 29.6 milliseconds in Java on an Intel Core i7-950 3.06-GHz processor.

7.2 Parameters and Experimental Environment

We describe the parameters and experimental environment for measuring the running time of the η_T pairing over $\mathbb{F}_{3^m}$ in this section.

In the experiment, we use the extension degrees $m = 97, 193, 353, 509$ and the following irreducible polynomials, respectively.

$$f(x) = \begin{cases} x^{97} + x^{12} + 2 \\ x^{193} + x^{12} + 2 \\ x^{353} + x^{142} + 2 \\ x^{509} + x^{358} + 2 \end{cases}$$

The constant value b_3 of the supersingular elliptic curve $y^2 = x^3 - x + b_3$ in characteristic three is chosen by $b_3 = 1$ when $m = 97, 509$ and $b_3 = -1$ when $m = 193, 353$. Then, small coprime of $\#E(\mathbb{F}_{3^m})$ is 7 for $m = 97, 509$, and $\#E(\mathbb{F}_{3^m})$ is a prime for $m = 193, 353$. The security of each m based on the discrete logarithm problem over $\mathbb{F}_{3^{6m}}$ has AES 64, 80, 112, 128-bit security, respectively [5, 71, 88].

In C, we measure the running time using GCC 4.4.4 compiler with the `-O3` and `-fomit-frame-pointer` options, and the `timeval` structure and the `gettimeofday` function. The program is implemented without assembly and SSE implementation. In Java, we also use a HotSpot 64-bit Server VM (build 21, 1-b02, mixed mode) as a Java virtual machine, and `System.nanoTime()` function to measure the running time.

For measuring the running time, we use the following computer:

CPU : Intel Core i7-950 processor (3.06 GHz)
L1 cache: 64 KByte (32 KByte Instruction/Data) per core
L2 cache: 256 KByte per core
L3 cache: 8 MByte
RAM : 24 GByte
OS : Linux version 2.6.32-71.el6.x86_64

Table 7.1: Running Time for Arithmetic in $\mathbb{F}_{3^m}$ in C (μsec)

Degree m	97	193	353	509
Addition	0.003	0.006	0.008	0.011
Subtraction	0.003	0.006	0.009	0.011
Multiplication	0.427	1.057	1.876	2.747
Cubing	0.033	0.060	0.088	0.115
Inversion	3.501	10.152	23.006	39.277

7.3 Timing Results in C

C is a general-purpose programming language, and it is generally used for implementation of program and system. By using C compiler, a very high-speed program is generated from a source code, since we can optimize the implementation such as memory management, and the C compiler outputs the optimized native code for a target platform. Thus it is one of the fastest programming languages. In the efficient implementation of cryptography, C and C++, that is the extension of C, are mainly used now.

Since the arithmetic in $\mathbb{F}_{3^m}$ is the base arithmetic for the η_T pairing over $\mathbb{F}_{3^m}$ with respect to pairing-based cryptosystems, this arithmetic requires to be calculated as efficient as possible. Table 7.1 presents the result of the running time for the arithmetic in $\mathbb{F}_{3^m}$ implemented in C. The window width of the multiplication in $\mathbb{F}_{3^m}$ we used is $w = 4$. Each listed function computes more than 10^8 time executions, and the timing result measures the average of the executions. As the result, the multiplication is more than 10 times slower than the cubing, and the inversion is about 10 times slower than the multiplication in $\mathbb{F}_{3^m}$. Then when the degree m increases about twice, the running time of each operation becomes approximately 2 times slower.

For the arithmetic over $E(\mathbb{F}_{3^m})$, we provide three types of coordinate; Affine, Projective, and Jacobian coordinates. We show the result of the arithmetic using three coordinates in Table 7.2, and compare the running time between each coordinate. The running time of their functions computes the average of more than 10^7 executions. The point addition and doubling in the Affine coordinate are computed by few multiplication and cubing. However the timing is slower compared with that in the other coordinates, since it needs 1 inversion in $\mathbb{F}_{3^m}$ and the inversion is 10 times or more slower than the other functions in $\mathbb{F}_{3^m}$. The point tripling in the Jacobian coordinate is much slow, since it requires the multiplication in $\mathbb{F}_{3^m}$. The

Table 7.2: Running Time for Arithmetic over $E(\mathbb{F}_{3^m})$ in C (μsec)

Degree m		97	193	353	509
Affine	Point Addition	4.43	12.51	27.37	45.26
	Point Doubling	4.00	11.83	25.34	42.20
	Point Tripling	0.14	0.25	0.37	0.48
Projective	Point Addition	5.31	12.77	22.84	33.41
	Point Doubling	2.26	5.36	9.62	14.03
	Point Tripling	0.20	0.38	0.56	0.72
Jacobian	Point Addition	5.39	12.88	23.05	33.63
	Point Doubling	3.17	7.50	13.50	19.67
	Point Tripling	0.63	1.41	2.42	3.48

Table 7.3: Running Time for η_T Pairing over $\mathbb{F}_{3^m}$, Point Multiplication over $E(\mathbb{F}_{3^m})$, and Exponentiation in $\mathbb{F}_{3^{6m}}$ in C (μsec)

Degree m	97	193	353	509
η_T Pairing over $\mathbb{F}_{3^m}$	342	1,495	4,717	9,798
Point Multiplication over $E(\mathbb{F}_{3^m})$	280	970	2,635	5,088
Exponentiation in $\mathbb{F}_{3^{6m}}$	390	1,490	4,333	8,648

fastest coordinate of the arithmetic over $E(\mathbb{F}_{3^m})$ is the Projective coordinate.

Finally, we indicate the result of the running time for the η_T pairing over $\mathbb{F}_{3^m}$, the point multiplication over $E(\mathbb{F}_{3^m})$, and the exponentiation in $\mathbb{F}_{3^{6m}}$. For implementation of the point multiplication and the exponentiation, we use the width $w = 4$ for the rw-NAF method in the Projective coordinate and $w = 3$ for the sliding window method, respectively. Table 7.3 displays the timing result, that is the average of more than 10^5 time executions. In the pairing-based cryptosystems, the dominant cost is the multiplication and the cubing in $\mathbb{F}_{3^m}$. Then the running time can be estimated by the efficiency of the multiplication and cubing. The fastest function of these is the point multiplication, and it is approximately twice faster than the other in $\mathbb{F}_{3^{509}}$. Then the pairing and the exponentiation are almost the same running time in all m.

Table 7.4: Running Time for Arithmetic in $\mathbb{F}_{3^m}$ in Java (μsec)

Degree m	97	193	353	509
Addition	0.035	0.039	0.048	0.049
Subtraction	0.037	0.039	0.048	0.049
Multiplication	2.077	3.881	6.254	9.329
Cubing	0.137	0.202	0.268	0.331
Inversion	7.411	21.560	47.854	84.234

7.4 Timing Results in Java

Java is one of the general programming languages and the object-oriented languages, and is usually adopted in order to construct network system. The Java program acts on Java virtual machine via bytecode complied by Java, then it can run on JVM of any platform with same bytecode regardless of computer architecture. A garbage collector of JVM engages itself in memory management, therefore, we can implement without considering an object management, and the bug resulting from usage of a memory decreases. The efficiency of Java is almost slower the program implemented in C, since there exists some overheads such as a garbage collection and conversion of bytecode on-line.

Table 7.4 shows the result of the running time for the arithmetic in $\mathbb{F}_{3^m}$ implemented in Java. The window width of the multiplication is $w = 3$. In Java, since making and accessing objects are slow rather than that in C, then we set the window width $w = 3$. We measure the running time of arithmetic in $\mathbb{F}_{3^m}$ by the average of 10^7 executions. In Java implementation, the multiplication is more than 10 times slower than the cubing, and in $\mathbb{F}_{3^{509}}$, its differential becomes about 30 times. On the other hand, the ratio of multiplication to the inversion is few rather than that in C. In $\mathbb{F}_{3^{97}}$, the inversion is only 3.6 times slower than the multiplication.

We similarly measure the timing of the arithmetic over $E(\mathbb{F}_{3^m})$ by the Affine, Projective, and Jacobian coordinates. We indicate the table of the timing results using three coordinates in Table 7.5. The running time of their functions computes the average of more than 10^7 executions. Roughly speaking, in C, the Projective coordinate is the fastest coordinate. However, in Java, the timing of the point addition in the Affine coordinate is the fastest one. This is because the difference of the timing ratio of the inversion to the multiplication is small. In the Projective coordinate, the point addition additionally computes 10 multiplications instead of computing the inversion. In Java, the inversion relatively acts efficiently, then the

Table 7.5: Running Time for Arithmetic over $E(\mathbb{F}_{3^m})$ in Java (μsec)

Degree m		97	193	353	509
Affine	Point Addition	12.04	30.05	61.14	104.18
	Point Doubling	9.81	25.96	54.69	94.48
	Point Tripling	0.66	0.93	1.20	1.51
Projective	Point Addition	25.40	47.05	76.01	113.16
	Point Doubling	10.80	19.94	32.08	47.65
	Point Tripling	0.94	1.32	1.76	2.14
Jacobian	Point Addition	25.67	47.47	76.60	113.95
	Point Doubling	15.10	27.80	44.92	66.63
	Point Tripling	3.01	5.19	8.02	11.49

Table 7.6: Running Time for η_T Pairing over $\mathbb{F}_{3^m}$, Point Multiplication over $E(\mathbb{F}_{3^m})$, and Exponentiation in $\mathbb{F}_{3^{6m}}$ in Java (μsec)

Degree m	97	193	353	509
η_T Pairing over $\mathbb{F}_{3^m}$	1,716	5,681	15,934	33,231
Point Multiplication over $E(\mathbb{F}_{3^m})$	683	2,324	6,947	15,495
Exponentiation in $\mathbb{F}_{3^{6m}}$	1,967	5,716	14,727	29,556

Affine coordinate is fast.

We finally show the results of the main components of the pairing-based cryptography, the η_T pairing over $\mathbb{F}_{3^m}$, the point multiplication over $E(\mathbb{F}_{3^m})$, and the exponentiation in $\mathbb{F}_{3^{6m}}$. We then use the width $w = 4$ for the rw-NAF method and $w = 3$ for the sliding window method, respectively. In Java, the point multiplication acts under the Affine coordinate, since its coordinate is the fastest point addition over $E(\mathbb{F}_{3^m})$. Table 7.6 lists the running time of these functions, that is the average of more than 10^5 time executions.

As the result of our experiment, the timing of all functions less than 40 milliseconds. The point multiplication is also fastest in them, and it is twice faster.

These are using the same algorithm except coordinate of point and multiplication in $\mathbb{F}_{3^m}$.

Table 7.7: Comparison of Running Time between C and Java in $\mathbb{F}_{3^{509}}$ (μsec)

	C	Java
η_T Pairing over $\mathbb{F}_{3^{509}}$	9,798	33,231
Point Multiplication over $E(\mathbb{F}_{3^{509}})$	5,088	15,495
Exponentiation in $\mathbb{F}_{3^{6\cdot 509}}$	8,648	29,556

7.5 C vs. Java

In this section, we mention the comparison of the timing results between the programs implemented in C and Java. Java is easily implemented because we rely Java virtual machine such as a memory management. However the efficiency of the program in Java falls.

Table 7.7 shows the running time comparison of the main components for the construction of the pairing-based cryptosystems; the η_T pairing over $\mathbb{F}_{3^m}$, the point multiplication over $E(\mathbb{F}_{3^m})$, and the exponentiation in $\mathbb{F}_{3^{6m}}$ in $m = 509$. These are implemented by using same algorithms for fair comparison, except the coordinate of $E(\mathbb{F}_{3^m})$ and the window width of the multiplication in $\mathbb{F}_{3^m}$. From the result, the functions in Java is about 3 times slower than that in C. The Java program acts with some overheads, the running time is usually slower. However, the Java virtual machine tries some optimizations on-line, and therefore the running timing is enough fast.

By using the other devices such as an mobile phone, the behavior of JVM is restricted, for example, many garbage collections act in the program for saving a memory. In our former result of the mobile phones, the η_T pairing over $\mathbb{F}_{3^{97}}$ is about 200 milliseconds [67]. Therefore, we need to optimize the implementation for each device due to the performance of the device and JVM.

Chapter 8

Conclusion

8.1 Review

This thesis was shown that the efficient $\mathbb{F}_3$-addition, $\mathbb{F}_3$-subtraction and MapToPoint algorithms for an efficient implementation of the η_T pairing in characteristic three. We described the detail of how to implement efficiently the pairing-based cryptosystems using η_T pairing over $\mathbb{F}_{3^m}$. We also implemented the component functions and measured the running time of them in C and Java.

Firstly, we constructed the logical instruction sequences of the addition and subtraction in $\mathbb{F}_3$. The η_T pairing computation over $\mathbb{F}_{3^m}$ is based on these operations, then they require to be computed as efficient as possible. In order to represent an element in $\mathbb{F}_3$, we used a two-bit encoding. Then the $\mathbb{F}_3$-addition and subtraction are considered as a map $(\mathbb{F}_2)^2 \times (\mathbb{F}_2)^2 \rightarrow (\mathbb{F}_2)^2$, and computed by using logical instructions. The previous construction had used seven logical instructions, consisting of OR and XOR, with the natural assignment $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$. However, it has not yet been proven whether seven is a minimum number. For our exhaustive search of the instruction sequences, we found many implementations of the $\mathbb{F}_3$-addition and subtraction using only six logical instructions with different assignments such as $\{(1,1) \mapsto 0, (0,1) \mapsto 1, (1,0) \mapsto 2\}$ and $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$. We then proved that there is no implementation of the $\mathbb{F}_3$-addition and subtraction using less than six logical instructions with any assignment of $\mathbb{F}_3$-element by two bits. Moreover, we applied the new $\mathbb{F}_3$-additions to an efficient software implementation of the η_T pairing. The running time of the η_T pairing over $\mathbb{F}_{3^{509}}$ using the new $\mathbb{F}_3$-addition with the encoding $\{(0,0) \mapsto 0, (0,1) \mapsto 1, (1,1) \mapsto 2\}$ is 16.3 milliseconds on an AMD Opteron 2.2-GHz processor. This is approximately 7% faster than the implementation us-

ing the previous $\mathbb{F}_3$-addition with seven logical instructions.

Secondly, we proposed a faster MapToPoint algorithm on supersingular elliptic curves in characteristic three, which is a hashing algorithm onto an elliptic curve point from x or y-coordinate. The conventional algorithms are executed by using a square root computation in $\mathbb{F}_{3^m}$ or matrix over $\mathbb{F}_3$, however they require $O(\log m)$ multiplications as the computational cost or about m $\mathbb{F}_{3^m}$-elements as the off-line memory, respectively. In the proposed algorithm, we compute the point from the input y-coordinate by using *1/3-trace* over $\mathbb{F}_{3^m}$. The *1/3-trace* over $\mathbb{F}_{3^m}$ can compute a solution x of $x^3 - x = c$ using no multiplication in $\mathbb{F}_{3^m}$. The proposed algorithm is computed by $O(1)$ multiplications and $O(m)$ cubing in $\mathbb{F}_{3^m}$, and the number of the multiplication reduced from $O(\log m)$ to $O(1)$ compared with the MapToPoint using the square root computation. Moreover it only requires less than m $\mathbb{F}_3$-elements to be stored in the off-line memory to efficiently compute *trace* over $\mathbb{F}_{3^m}$. In our software implementation of $\mathbb{F}_{3^{509}}$, the proposed MapToPoint algorithm was approximately 35% faster than the conventional algorithm using the square root computation on an AMD Opteron 2.2-GHz processor.

Furthermore, we implemented the η_T pairing and the arithmetic of the supersingular elliptic curve and the finite field, which are used in order to construct the pairing-based cryptosystems. The implementation was used by C and Java languages for a practical usage, and performed on an Intel Core i7-950 3.06-GHz processor using the GCC compiler for C and the HotSpot server 64-bit VM for Java, respectively. The timing result of the η_T pairing over $\mathbb{F}_{3^m}$, the point multiplication over $E(\mathbb{F}_{3^m})$ and the exponentiation over $\mathbb{F}_{3^{6m}}$ is 9.8, 5.1, 8.6 milliseconds in C, and 33.2, 15.5, 29.6 milliseconds in Java.

8.2 Open Problem and Future Work

There are some open problems for construction of pairing-based cryptosystems using η_T pairing and finite fields $\mathbb{F}_{p^m}$.

The η_T pairing can be applied to supersingular hyperelliptic curves of genus $g = (p-1)/2$ of the form $y^2 = x^p - x + d$ in small characteristic p. In this construction, an embedded degree k is $2p$, then extension degree of field $\mathbb{F}_{p^m}$ can be reduced for achieving the security level. There exists some improvement for implementing the hyperelliptic pairings [73]. However pairings on hyperelliptic curves are still much slow rather than that on elliptic curves.

Moreover, for some pairings on hyperelliptic curves, we need to consider arith-

metic of a finite field $\mathbb{F}_p$ in small characteristic $p = 5, 7$, that is used for construction of an extension field $\mathbb{F}_{p^m}$. Each element in $\mathbb{F}_p$ is represented by at least three bits, then the addition and subtraction sequences become more complicated than that in $\mathbb{F}_2$ and $\mathbb{F}_3$. Hence we require to construct the minimum instruction sequences to efficiently compute the field operations. Since the search space is extensive, we cannot execute the exhaustive search in this condition. We will construct a suitable search algorithm, for example, we cut a node by some conditions, for reducing the search space.

Bibliography

[1] O. Ahmadi, D. Hankerson, and A. Menezes. Software Implementation of Arithmetic in $\mathbb{F}_{3^m}$. In C. Carlet and B. Sunar (eds.) *Arithmetic of Finite Fields – WAIFI 2007*, volume 4547 of *Lecture Notes in Computer Science*, pages 85–102. Springer, 2007.

[2] O. Ahmadi and F. Rodríguez-Henríquez. Low Complexity Cubing and Cube Root Computation over $\mathbb{F}_{3^m}$ in Polynomial Basis. *IEEE Transactions on Computers*, 59(10):1297–1308, 2010.

[3] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López. Faster Explicit Formulas for Computing Pairings over Ordinary Curves. In K. G. Paterson (ed.) *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 48–68. Springer, 2011.

[4] D. F. Aranha, J. López, and D. Hankerson. High-Speed Parallel Software Implementation of the η_T Pairing. In J. Pieprzyk (ed.) *Topics in Cryptology – CT-RSA 2010*, volume 5985 of *Lecture Notes in Computer Science*, pages 89–105. Springer, 2010.

[5] E. Barker, W. Barker, W. Burr, W. Polk, and M. Smid. Recommendation for Key Management – Part 1: General (Revised). NIST Special Publication 800-57, 2007. Available from `http://csrc.nist.gov/publications/nistpubs/800-57/sp800-57-Part1-revised2_Mar08-2007.pdf`.

[6] P. S. L. M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004.

[7] P. S. L. M. Barreto, S. D. Galbraith, C. ÓhÉigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Designs, Codes and Cryptography*, 42(3):239–271, 2007.

[8] P. S. L. M. Barreto and H. Y. Kim. Fast hashing onto elliptic curves over fields of characteristic 3. Cryptology ePrint Archive, Report 2001/098, 2001.

[9] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient Algorithms for Pairing-Based Cryptosystems. In M. Yung (ed.) *Advances in Cryptology – CRYPTO 2002*, volume 2442 of *Lecture Notes in Computer Science*, pages 354–369. Springer, 2002.

[10] P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient Implementation of Pairing-Based Cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.

[11] P. S. L. M. Barreto and M. Naehrig. Pairing-Friendly Elliptic Curves of Prime Order. In B. Preneel and S. E. Tavares (eds.) *Selected Areas in Cryptography – SAC 2005*, volume 3897 of *Lecture Notes in Computer Science*, pages 319–331. Springer, 2006.

[12] P. S. L. M. Barreto and J. F. Voloch. Efficient Computation of Roots in Finite Fields. *Designs, Codes and Cryptography*, 39(2):275–280, 2006.

[13] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards Curves. In S. Vaudenay (ed.) *Progress in Cryptology – AFRICACRYPT 2008*, volume 5023 of *Lecture Notes in Computer Science*, pages 389–405. Springer, 2008.

[14] D. J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. In K. Kurosawa (ed.) *Advances in Cryptology – ASIACRYPT 2007*, volume 4833 of *Lecture Notes in Computer Science*, pages 29–50. Springer, 2007.

[15] G. Bertoni, J. Guajardo, S. S. Kumar, G. Orlando, C. Paar, and T. J. Wollinger. Efficient $GF(p^m)$ Arithmetic Architectures for Cryptographic Applications. In M. Joye (ed.) *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 158–175. Springer, 2003.

[16] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, and F. Rodríguez-Henríquez. A Comparison between Hardware Accelerators for the Modified Tate Pairing over $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$. In S. D. Galbraith and K. G. Paterson (eds.) *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 297–315. Springer, 2008.

[17] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and Arithmetic Operators for Computing the η_T Pairing in Characteristic Three. Cryptology ePrint Archive, Report 2007/417, 2007.

[18] J.-L. Beuchat, N. Brisebarre, J. Detrey, E. Okamoto, M. Shirase, and T. Takagi. Algorithms and Arithmetic Operators for Computing the η_T Pairing in Characteristic Three. *IEEE Transactions on Computers*, 57(11):1454–1468, 2008.

[19] J.-L. Beuchat, N. Brisebarre, M. Shirase, T. Takagi, and E. Okamoto. A Coprocessor for the Final Exponentiation of the η_T Pairing in Characteristic Three. In C. Carlet and B. Sunar (eds.) *Arithmetic of Finite Fields – WAIFI 2007*, volume 4547 of *Lecture Notes in Computer Science*, pages 25–39. Springer, 2007.

[20] J.-L. Beuchat, J. Detrey, N. Estibals, E. Okamoto, and F. Rodríguez-Henríquez. Fast Architectures for the η_T Pairing over Small-Characteristic Supersingular Elliptic Curves. *IEEE Transactions on Computers*, 60(2):266–281, 2011.

[21] J.-L. Beuchat, H. Doi, K. Fujita, A. Inomata, P. Ith, A. Kanaoka, M. Katouno, M. Mambo, E. Okamoto, T. Okamoto, T. Shiga, M. Shirase, R. Soga, T. Takagi, A. Vithanage, and H. Yamamoto. FPGA and ASIC implementations of the η_T pairing in characteristic three. *Computers & Electrical Engineering*, 36(1):73–87, 2010.

[22] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya. High-Speed Software Implementation of the Optimal Ate Pairing over Barreto-Naehrig Curves. In M. Joye, A. Miyaji, and A. Otsuka (eds.) *Pairing-Based Cryptography – Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 21–39. Springer, 2010.

[23] J.-L. Beuchat, E. López-Trejo, L. Martínez-Ramos, S. Mitsunari, and F. Rodríguez-Henríquez. Multi-core Implementation of the Tate Pairing over Supersingular Elliptic Curves. In J. A. Garay, A. Miyaji, and A. Otsuka (eds.) *Cryptology and Network Security – CANS 2009*, volume 5888 of *Lecture Notes in Computer Science*, pages 413–432. Springer, 2009.

[24] J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto. An Algorithm for the η_T Pairing Calculation in Characteristic Three and its Hardware Imple-

mentation. In *IEEE Symposium on Computer Arithmetic – ARITH18*, pages 97–104. IEEE Computer Society, 2007.

[25] I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in Elliptic Curve Cryptography*. London Mathematical Society Lecture Note Series 317. Cambridge University Press, 2005.

[26] D. Boneh and X. Boyen. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In C. Cachin and J. Camenisch (eds.) *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 223–238. Springer, 2004.

[27] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search. In C. Cachin and J. Camenisch (eds.) *Advances in Cryptology – EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[28] D. Boneh and M. K. Franklin. Identity-Based Encryption from the Weil Pairing. *SIAM Journal on Computing*, 32(3):586–615, 2003.

[29] D. Boneh, C. Gentry, and B. Waters. Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys. In V. Shoup (ed.) *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[30] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[31] S. Chatterjee and P. Sarkar. *Identity-Based Encryption*. Springer, 2011.

[32] H. Cohen and G. Frey. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, 2006.

[33] Compaq Computer Corporation. *Alpha Architecture Reference Manual: Fourth Edition*, 2002.

[34] M. P. L. Das and P. Sarkar. Pairing Computation on Twisted Edwards Form Elliptic Curves. In S. D. Galbraith and K. G. Paterson (eds.) *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 192–210. Springer, 2008.

[35] W. Diffie and M. E. Hellman. New Directions in Cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[36] I. M. Duursma and H.-S. Lee. Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$. In C.-S. Laih (ed.) *Advances in Cryptology – ASIACRYPT 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2003.

[37] H. M. Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American Mathematical Society*, 44(3):393–422, 2007.

[38] T. ElGamal. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. *IEEE Transactions on Information Theory*, 31(4):469–472, 1985.

[39] N. Estibals. Compact Hardware for Computing the Tate Pairing over 128-Bit-Security Supersingular Curves. In M. Joye, A. Miyaji, and A. Otsuka (eds.) *Pairing-Based Cryptography – Pairing 2010*, volume 6487 of *Lecture Notes in Computer Science*, pages 397–416. Springer, 2010.

[40] K. Fong, D. Hankerson, J. López, and A. Menezes. Field Inversion and Point Halving Revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, 2004.

[41] S. D. Galbraith. Supersingular Curves in Cryptography. In C. Boyd (ed.) *Advances in Cryptology – ASIACRYPT 2001*, volume 2248 of *Lecture Notes in Computer Science*, pages 495–513. Springer, 2001.

[42] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the Tate Pairing. In C. Fieker and D. R. Kohel (eds.) *Algorithmic Number Theory – ANTS-V*, volume 2369 of *Lecture Notes in Computer Science*, pages 324–337. Springer, 2002.

[43] S. D. Galbraith and X. Lin. Computing pairings using x-coordinates only. *Designs, Codes and Cryptography*, 50(3):305–324, 2009.

[44] S. D. Galbraith, K. G. Paterson, and N. P. Smart. Pairings for cryptographers. *Discrete Applied Mathematics*, 156(16):3113–3121, 2008.

[45] R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian (ed.)

Advances in Cryptology – CRYPTO 2001, volume 2139 of *Lecture Notes in Computer Science*, pages 190–200. Springer, 2001.

[46] C. Gentry. Practical Identity-Based Encryption Without Random Oracles. In S. Vaudenay (ed.) *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 445–464. Springer, 2006.

[47] C. Gentry and A. Silverberg. Hierarchical ID-Based Cryptography. In Y. Zheng (ed.) *Advances in Cryptology – ASIACRYPT 2002*, volume 2501 of *Lecture Notes in Computer Science*, pages 548–566. Springer, 2002.

[48] D. M. Gordon. A Survey of Fast Exponentiation Methods. *Journal of Algorithms*, 27(1):129–146, 1998.

[49] E. Gorla, C. Puttmann, and J. Shokrollahi. Explicit Formulas for Efficient Multiplication in $\mathbb{F}_{3^{6m}}$. In C. M. Adams, A. Miri, and M. J. Wiener (eds.) *Selected Areas in Cryptography – SAC 2007*, volume 4876 of *Lecture Notes in Computer Science*, pages 173–183. Springer, 2007.

[50] P. Grabher, J. Großschädl, and D. Page. On Software Parallel Implementation of Cryptographic Pairings. In R. M. Avanzi, L. Keliher, and F. Sica (eds.) *Selected Areas in Cryptography – SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2009.

[51] R. Granger, D. Page, and N. P. Smart. High Security Pairing-Based Cryptography Revisited. In F. Hess, S. Pauli, and M. E. Pohst (eds.) *Algorithmic Number Theory – ANTS-VII*, volume 4076 of *Lecture Notes in Computer Science*, pages 480–494. Springer, 2006.

[52] R. Granger, D. Page, and M. Stam. Hardware and Software Normal Basis Arithmetic for Pairing-Based Cryptography in Characteristic Three. *IEEE Transactions on Computers*, 54(7):852–860, 2005.

[53] R. Granger, D. Page, and M. Stam. On Small Characteristic Algebraic Tori In Pairing-Based Cryptography. *LMS Journal of Computation and Mathematics*, 9:64–85, 2006.

[54] D. Hankerson, A. Menezes, and M. Scott. Software implementation of pairings. Centre for Applied Cryptographic Research (CACR) Technical Reports, CACR 2008-08, 2008. Available

from `http://www.cacr.math.uwaterloo.ca/techreports/2008/cacr2008-08.pdf`.

[55] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer Professional Computing. Springer-Verlag, 2004.

[56] K. Harrison, D. Page, and N. P. Smart. Software implementation of finite fields of characteristic three, for use in pairing-based cryptosystems. *LMS Journal of Computation and Mathematics*, 5:181–193, 2002.

[57] F. Hess, N. P. Smart, and F. Vercauteren. The Eta Pairing Revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[58] J. Horwitz and B. Lynn. Toward Hierarchical Identity-Based Encryption. In L. R. Knudsen (ed.) *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 466–481. Springer, 2002.

[59] Intel Corporation. *Intel 64 and IA-32 Architectures Software Developer's Manual*, 2011.

[60] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computation*, 78(3):171–177, 1988.

[61] T. Iyama, S. Kiyomoto, K. Fukushima, T. Tanaka, and T. Takagi. Efficient Implementation of Pairing on BREW Mobile Phones. In I. Echizen, N. Kunihiro, and R. Sasaki (eds.) *Advances in Information and Computer Security – IWSEC 2010*, volume 6434 of *Lecture Notes in Computer Science*, pages 326–336. Springer, 2010.

[62] A. Joux. A One Round Protocol for Tripartite Diffie-Hellman. In W. Bosma (ed.) *Algorithmic Number Theory – ANTS-IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 385–394. Springer, 2000.

[63] Y. Kawahara, K. Aoki, and T. Takagi. Faster Implementation of η_T Pairing over GF(3^m) Using Minimum Number of Logical Instructions for GF(3)-Addition. In S. D. Galbraith and K. G. Paterson (eds.) *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2008.

[64] Y. Kawahara, K. Aoki, and T. Takagi. Faster Implementation of η_T Pairing Using Minimum Number of Logical Instructions for GF(3)-addition. *IPSJ Journal*, 50(11):2717–2726, 2009. (in Japanese).

[65] Y. Kawahara, T. Kobayashi, G. Takahashi, and T. Takagi. Faster MapTo-Point on Supersingular Elliptic Curves in Characteristic 3. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E94-A(1):150–155, 2011.

[66] Y. Kawahara, T. Takagi, and E. Okamoto. Efficient Implementation of Tate Pairing on a Mobile Phone Using Java. In Y. Wang, Y. ming Cheung, and H. Liu (eds.) *Computational Intelligence and Security – CIS 2006*, volume 4456 of *Lecture Notes in Computer Science*, pages 396–405. Springer, 2007.

[67] Y. Kawahara, T. Takagi, and E. Okamoto. Efficient Implementation of Tate Pairing on Mobile Phones Using Java. *IPSJ Journal*, 49(1):427–435, 2008. (in Japanese).

[68] T. Kerins, W. P. Marnane, E. M. Popovici, and P. S. L. M. Barreto. Efficient Hardware for the Tate Pairing Calculation in Characteristic Three. In J. R. Rao and B. Sunar (eds.) *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 412–426. Springer, 2005.

[69] N. Koblitz. Elliptic Curve Cryptosystems. *Mathematics of Computation*, 48(177):203–209, 1987.

[70] N. Koblitz. An Elliptic Curve Implementation of the Finite Field Digital Signature Algorithm. In H. Krawczyk (ed.) *Advances in Cryptology – CRYPTO '98*, volume 1462 of *Lecture Notes in Computer Science*, pages 327–337. Springer, 1998.

[71] N. Koblitz and A. Menezes. Pairing-Based Cryptography at High Security Levels. In N. P. Smart (ed.) *Cryptography and Coding – C&C 2005*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.

[72] S. Kwon. Efficient Tate Pairing Computation for Elliptic Curves over Binary Fields. In C. Boyd and J. M. González Nieto (eds.) *Information Security and Privacy – ACISP 2005*, volume 3574 of *Lecture Notes in Computer Science*, pages 134–145. Springer, 2005.

[73] E. Lee, H.-S. Lee, and Y. Lee. Eta Pairing Computation on General Divisors over Hyperelliptic Curves $y^2 = x^7 - x \pm 1$. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto (eds.) *Pairing-Based Cryptography – Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 349–366. Springer, 2007.

[74] E. Lee, H.-S. Lee, and C.-M. Park. Efficient and Generalized Pairing Computation on Abelian Varieties. *IEEE Transactions on Information Theory*, 55(4):1793–1803, 2009.

[75] A. B. Lewko, T. Okamoto, A. Sahai, K. Takashima, and B. Waters. Fully Secure Functional Encryption: Attribute-Based Encryption and (Hierarchical) Inner Product Encryption. In H. Gilbert (ed.) *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 62–91. Springer, 2010.

[76] R. Lidl and H. Niederreiter. *Finite Fields*. Encyclopedia of Mathematics and its Applications, Volume 20. Cambridge University Press, 2nd edition, 1997.

[77] T. Matsuo. Proxy Re-encryption Systems for Identity-Based Encryption. In T. Takagi, T. Okamoto, E. Okamoto, and T. Okamoto (eds.) *Pairing-Based Cryptography – Pairing 2007*, volume 4575 of *Lecture Notes in Computer Science*, pages 247–267. Springer, 2007.

[78] A. J. Menezes. *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic Publishers, 1993.

[79] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. Discrete Mathematics and Its Applications. CRC Press, 1997.

[80] V. S. Miller. Short Programs for Functions on Curves. Unpublished manuscript, 1986. Available from `http://crypto.stanford.edu/miller/miller.pdf`.

[81] V. S. Miller. Use of Elliptic Curves in Cryptography. In H. C. Williams (ed.) *Advances in Cryptology – CRYPTO '85*, volume 218 of *Lecture Notes in Computer Science*, pages 417–426. Springer, 1986.

[82] V. S. Miller. The Weil Pairing, and Its Efficient Calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

[83] T. Okamoto and K. Takashima. Hierarchical Predicate Encryption for Inner-Products. In M. Matsui (ed.) *Advances in Cryptology – ASIACRYPT 2009*, volume 5912 of *Lecture Notes in Computer Science*, pages 214–231. Springer, 2009.

[84] T. Okamoto and K. Takashima. Fully Secure Functional Encryption with General Relations from the Decisional Linear Assumption. In T. Rabin (ed.) *Advances in Cryptology – CRYPTO 2010*, volume 6223 of *Lecture Notes in Computer Science*, pages 191–208. Springer, 2010.

[85] K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi. Signed Binary Representations Revisited. In M. K. Franklin (ed.) *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 123–139. Springer, 2004.

[86] D. A. Osvik. Speeding up Serpent. In *AES Candidate Conference – AES3*, pages 317–329, 2000.

[87] D. Page and N. P. Smart. Hardware Implementation of Finite Fields of Characteristic Three. In B. S. Kaliski Jr., Çetin Kaya Koç, and C. Paar (eds.) *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 529–539. Springer, 2003.

[88] D. Page, N. P. Smart, and F. Vercauteren. A comparison of MNT curves and supersingular curves. *Applicable Algebra in Engineering, Communication and Computing*, 17(5):379–392, 2006.

[89] G. C. C. F. Pereira, M. A. Simplício Jr., M. Naehrig, and P. S. L. M. Barreto. A family of implementation-friendly BN elliptic curves. *Journal of Systems and Software*, 84(8):1319–1326, 2011.

[90] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[91] R. Ronan, C. ÓhÉigeartaigh, C. C. Murphy, T. Kerins, and P. S. L. M. Barreto. A Reconfigurable Processor for the Cryptographic η_T Pairing in Char-

acteristic 3. In *Information Technology: New Generations – ITNG 2007*, pages 11–16. IEEE Computer Society, 2007.

[92] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystem Based on Pairing over Elliptic Curve. In Proceedings of *Symposium on Cryptography and Information Security – SCIS 2001*, 7B-2, 2001. (in Japanese).

[93] R. Sakai, K. Ohgishi, and M. Kasahara. Cryptosystems Based on Pairings. In Proceedings of *Symposium on Cryptography and Information Security – SCIS 2000*, C20, 2000. (in Japanese).

[94] Y. Sasaki, S. Nishina, M. Shirase, and T. Takagi. An Efficient Residue Group Multiplication for the η_T Pairing over $\mathbb{F}_{3^m}$. In M. Jacobson, V. Rijmen, and R. Safavi-Naini (eds.) *Selected Areas in Cryptography – SAC 2009*, volume 5867 of *Lecture Notes in Computer Science*, pages 364–375. Springer, 2009.

[95] M. Scott. On the Efficient Implementation of Pairing-Based Protocols. Cryptology ePrint Archive, Report 2011/334, 2011.

[96] M. Scott and P. S. L. M. Barreto. Compressed Pairings. In M. K. Franklin (ed.) *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2004.

[97] M. Scott, N. Costigan, and W. Abdulwahab. Implementing Cryptographic Pairings on Smartcards. In L. Goubin and M. Matsui (eds.) *Cryptographic Hardware and Embedded Systems – CHES 2006*, volume 4249 of *Lecture Notes in Computer Science*, pages 134–147. Springer, 2006.

[98] A. Shamir. Identity-Based Cryptosystems and Signature Schemes. In G. Blakley and D. Chaum (eds.) *Advances in Cryptology – CRYPTO '84*, volume 196 of *Lecture Notes in Computer Science*, pages 47–53. Springer, 1985.

[99] M. Shirase, Y. Kawahara, T. Takagi, and E. Okamoto. Universal η_T Pairing Algorithm over Arbitrary Extension Degree. In S. Kim, M. Yung, and H.-W. Lee (eds.) *Information Security Applications – WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2008.

[100] M. Shirase, Y. Miyazaki, T. Takagi, D.-G. Han, and D. Choi. Efficient Implementation of Pairing-Based Cryptography on a Sensor Node. *IEICE Transactions on Information and Systems*, E92-D(5):909–917, 2009.

[101] M. Shirase, T. Takagi, and E. Okamoto. Some Efficient Algorithms for the Final Exponentiation of η_T Pairing. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E91-A(1):221–228, 2008.

[102] V. Shoup. Lower Bounds for Discrete Logarithms and Related Problems. In W. Fumy (ed.) *Advances in Cryptology – EUROCRYPT '97*, volume 1233 of *Lecture Notes in Computer Science*, pages 256–266. Springer, 1997.

[103] C. Shu, S. Kwon, and K. Gaj. FPGA Accelerated Tate Pairing Based Cryptosystems over Binary Fields. In *IEEE International Conference on Field Programmable Technology – FPT 2006*, pages 173–180, 2006.

[104] J. H. Silverman. *The Arithmetic of Elliptic Curves*. Graduate Texts in Mathematics, Volume 106. Springer, 2nd edition, 2009.

[105] SPARC International, Inc. *The SPARC Architecture Manual*, 2000.

[106] P. Szczechowiak and M. Collier. TinyIBE: Identity-Based Encryption for Heterogeneous Sensor Networks. In *Intelligent Sensors, Sensor Networks and Information Processing – ISSNIP 2009*, pages 349–354, 2009.

[107] T. Takagi, D. Reis Jr., S.-M. Yen, and B.-C. Wu. Radix-r Non-Adjacent Form and Its Application to Pairing-Based Cryptosystem. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E89-A(1):115–123, 2006.

[108] G. Takahashi, F. Hoshino, and T. Kobayashi. Efficient GF(3^m) Multiplication Algorithm for ηT Pairing. Cryptology ePrint Archive, Report 2007/463, 2007.

[109] S.-Y. Tan, S.-H. Heng, and B.-M. Goi. Java Implementation for Pairing-Based Cryptosystems. In D. Taniar, O. Gervasi, B. Murgante, E. Pardede, and B. O. Apduhan (eds.) *Computational Science and Its Applications – ICCSA 2010 Part IV*, volume 6019 of *Lecture Notes in Computer Science*, pages 188–198. Springer, 2010.

[110] F. Vercauteren. Optimal pairings. *IEEE Transactions on Information Theory*, 56(1):455–461, 2010.

[111] J. von zur Gathen. Irreducible trinomials over finite fields. *Mathematics of Computation*, 72(244):1987–2000, 2003.

[112] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography*. Discrete Mathematics and Its Applications. Chapman & Hall/CRC, 2nd edition, 2008.

[113] B. Waters. Efficient Identity-Based Encryption Without Random Oracles. In R. Cramer (ed.) *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2005.

[114] M. Yoshitomi, T. Takagi, S. Kiyomoto, and T. Tanaka. Efficient Implementation of the Pairing on Mobilephones Using BREW. *IEICE Transactions on Information and Systems*, E91-D(5):1330–1337, 2008.

[115] C.-A. Zhao, F. Zhang, and J. Huang. A note on the Ate pairing. *International Journal of Information Security*, 7(6):379–382, 2008.

History

Publications

Journals

1. Yuto Kawahara, Tsuyoshi Takagi, and Eiji Okamoto. Efficient Implementation of Tate Pairing on Mobile Phones Using Java. *IPSJ Journal*, 49(1):427–435, 2008. (in Japanese).

2. Yuto Kawahara, Kazumaro Aoki, and Tsuyoshi Takagi. Faster Implementation of η_T Pairing Using Minimum Number of Logical Instructions for GF(3)-addition. *IPSJ Journal*, 50(11):2717–2726, 2009. (in Japanese).

3. Yuto Kawahara, Tetsutaro Kobayashi, Gen Takahashi, and Tsuyoshi Takagi. Faster MapToPoint on Supersingular Elliptic Curves in Characteristic 3. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E94-A(1):150–155, 2011.

Refereed International Conference Papers

1. Yuto Kawahara, Tsuyoshi Takagi, and Eiji Okamoto. Efficient Implementation of Tate Pairing on a Mobile Phone Using Java. In Yuping Wang, Yiu ming Cheung, and Hailin Liu (eds.) *Computational Intelligence and Security – CIS 2006*, volume 4456 of *Lecture Notes in Computer Science*, pages 396–405. Springer, 2007.

2. Masaaki Shirase, Yuto Kawahara, Tsuyoshi Takagi, and Eiji Okamoto. Universal η_T Pairing Algorithm over Arbitrary Extension Degree. In Sehun Kim, Moti Yung, and Hyung-Woo Lee (eds.) *Information Security Applications – WISA 2007*, volume 4867 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2007.

3. Yuto Kawahara, Kazumaro Aoki, and Tsuyoshi Takagi. Faster Implementation of η_T Pairing over GF(3^m) Using Minimum Number of Logical Instructions for GF(3)-Addition. In Steven D. Galbraith and Kenneth G. Paterson (eds.) *Pairing-Based Cryptography – Pairing 2008*, volume 5209 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2008.

Unrefereed International/Domestic Workshops/Symposiums

1. Yuto Kawahara, Tsuyoshi Takagi, and Eiji Okamoto. Efficient implementation of Tate pairing on a mobile phone using Java. *Computer Security Symposium – CSS 2006*, pages 591–596, 2006. (in Japanese).

2. Yuto Kawahara, Masaaki Shirase, Tsuyoshi Takagi, and Eiji Okamoto. Efficient Software Implementation of η_T Pairing. *Symposium on Cryptography and Information Security – SCIS 2007*, 2E3-2, 2007. (in Japanese).

3. Kazumaro Aoki, Yuto Kawahara, and Tsuyoshi Takagi. The optimal addition operation on GF(3) using logical instructions. *Symposium on Cryptography and Information Security – SCIS 2008*, 3D2-3, 2008. (in Japanese).

4. Yuto Kawahara, Tetsutaro Kobayashi, Gen Takahashi, and Tsuyoshi Takagi. A Faster MapToPoint on Supersingular Elliptic Curves in Characteristic 3. *Symposium on Cryptography and Information Security – SCIS 2009*, 3C3-3, 2009. (in Japanese).

5. Yuto Kawahara, Tetsutaro Kobayashi, Gen Takahashi, and Tsuyoshi Takagi. Implementation of Pairing-based Cryptography on Java Card. *Computer Security Symposium – CSS 2010*, pages 237–242, 2010. (in Japanese).

6. Yuto Kawahara, Go Yamamoto, Tetsutaro Kobayashi, and Tsuyoshi Takagi. Implementation of Decryption with Self-corrector in Pairing-based Cryptosystems. *Symposium on Cryptography and Information Security – SCIS 2011*, 2F1-2, 2011. (in Japanese).

Poster

1. Yuto Kawahara, Go Yamamoto, Tetsutaro Kobayashi, and Tsuyoshi Takagi. Cryptographic Self-Corrector for Cloud Computing Security. *International Workshop on Security – IWSEC 2011*, 2011.

Awards

1. Computer Security Symposium 2006, Student Paper Award. October, 2006.
2. Future-University Award. March, 2007.
3. IPSJ Yamashita SIG Research Award. March, 2008.
4. IEICE Hokkaido Branch Director Award. March, 2009.

Internship

NTT Information Sharing Platform Laboratories. Constructing New Decrypt System in Pairing-based Cryptosystem using Cryptographic Self-Corrector and Its Implementation. July – September, 2010.

Other

Research Fellow of the Japan Society for the Promotion of Science. April, 2009 – March, 2012.