Efficient Implementation of the Pairing on Mobilephones using BREW

Motoi Yoshitomi†, Tsuyoshi Takagi†,
Shinsaku Kiyomoto‡, and Toshiaki Tanaka‡

†Future University - Hakodate, School of System Information Science
116-2, Kamedanakano-cho, Hakodate, 041-0806, Japan

‡KDDI R&D Laboratories Inc.
2-1-15, Ohara, Fujimino, Saitama, 356-8502, Japan

Abstract. Pairing based cryptosystems can accomplish novel security applications such as ID-based cryptosystems, which have not been constructed efficiently without the pairing. The processing speed of the pairing based cryptosystems is relatively slow compared with the other conventional public key cryptosystems. However, several efficient algorithms for computing the pairing have been proposed, namely Duursma-Lee algorithm and its variant η_T pairing. In this paper, we present an efficient implementation of the pairing over some mobilephones. Moreover, we compare the processing speed of the pairing with that of the other standard public key cryptosystems, i.e. RSA cryptosystem and elliptic curve cryptosystem. Indeed the processing speed of our implementation in ARM9 processors on BREW achieves under 100 milliseconds using the supersingular curve over $\mathbb{F}_{3^{97}}$. In addition, the pairing is more efficient than the other public key cryptosystems, and the pairing can be achieved enough also on BREW mobilephones. It has become efficient enough to implement security applications, such as short signature, ID-based cryptosystems or broadcast encryption, using the pairing on BREW mobilephones.

Keywords: Pairing Based Cryptosystem, Mobilephone, BREW, Efficient Implementation

1 Introduction

The pairing can realize novel cryptographic applications e.g. short signature [6], ID-based cryptosystem [5], broadcast encryption [2], which have not been achieved by conventional public key cryptosystems. Short signature is a digital signature suitable for applications on memory-constrained devices because the signature length in the pairing becomes about a half of that in elliptic curve cryptosystem. ID-based cryptosystems can replace the public key with an E-mail address or an IP address which can be easily memorized. However, the processing speed is slower than that of other public key cryptosystem. The timing in paper [3] shows that the processing speed of pairing based cryptosystem is about 5 times or more slower than that of RSA cryptosystem or elliptic curve cryptosystem.

Recently, Duursma and Lee proposed a very efficient algorithm for computing the pairing over supersingular curves [7]. Barreto et al. then presented η_T pairing which is about twice faster than Duursma-Lee algorithm [1]. The two algorithms can compute the pairing relatively efficient.

By the recent progress of devices technology, we are able to implement pairing based cryptosystems on ubiquitous devices that the processing speed is comparatively slow. Especially, it is important to implement and evaluate of the pairing based cryptosystem on mobilephones which are the most familiar as ubiquitous devices. For mobilephones, there are two kinds of platforms such that JAVA and BREW. An implementation of the pairing with mobilephones for JAVA was already reported [13].

In this paper, we report about an efficient implementation of the pairing by mobilephones in ARM9 processors on BREW. We implement both Duursma-Lee algorithm and η_T pairing over finite fields $\mathbb{F}_{3^m}$ for extension degrees $m = \{97,167,193,239,313\}$. We try to improve the efficiency of pairing by the following procedures. At first, functions of the finite field and the pairing are implemented from scratch in C language. Then, we investigate the functions which requires a lot of the processing time by using a profiling tool. It turns out that the multiplications in the finite field are about 80% of the whole computation of the pairing programs, and thus we focus on enhancing the speed of multiplication in $\mathbb{F}_{3^m}$.

Note that a substitution operation in $\mathbb{F}_{3^m}$ on ARM9 processors is relatively slow to our experiments. Comb method can perform the polynomial multiplication fast because there are few substitution operations than normal multiplication algorithm [11]. We propose the improved Comb method which reduces the number of the substitution operations by eliminating the first loop of the Comb method. The improved Comb method enhances the whole speed of pairing by about 20%. In addition, the unrollment of the loop in the programs further improves the speed due to the reducement of pipeline hazard.

Moreover, we compare the processing speed of the pairing with that of other standard public key cryptosystem, i.e. RSA cryptosystem and elliptic curve cryptosystem, in this paper. Indeed in our implementation in ARM9 processors on BREW, the pairing achieves the processing speed which is more effective than the other cryptosystem. Therefore the pairing on BREW mobilephones has become practical enough.

2 Algorithms for Implementing the Pairing

In this section, we explain some efficient algorithms for computing in finite fields and the pairing.

2.1 The Elements Representation in $\mathbb{F}_{3^m}$

Let $\mathbb{F}_3[x]$ be the polynomials with coefficients over field $\mathbb{F}_3 = \{0, 1, 2\}$. Finite field $\mathbb{F}_{3^m}$ is the set of all polynomials represented by $\mathbb{F}_{3^m} = \mathbb{F}_3[x]/f(x)$, where

$f(x)$ is an irreducible polynomial. An element $A(x)$ in $\mathbb{F}_{3^m}$ can be represented as follows:

$$A(x) = (a_{m-1}, a_{m-2}, \cdots, a_1, a_0),\ a_i \in \mathbb{F}_3,$$

where a_i is an element in $\mathbb{F}_3$ for $i = 0, 1, \cdots, m-1$. Note that $\mathbb{F}_3$ has the three values 0,1 and 2. Therefore, we represent an element in $\mathbb{F}_3$ as (hi,lo)-bit, where hi and lo are binary bits. Let W be the word size of a targeted computer, and let $A[i]$ be a sequence of (hi, lo)-bits with size of W, where i is a positive integer including zero. Let $t = \lceil m/W \rceil$. Then $A(x)$ can be represented by the right-to-left array $A[t-1], \cdots, A[1], A[0]$. In this case of $W = 32$ and $m = 97$, $A(x)$ in $\mathbb{F}_{3^{97}}$ is represented by the array such that $A[3] = (0, 0, \cdots, 0, a_{96})$, $A[2] = (a_{95}, a_{94}, \cdots, a_{65}, a_{64})$, $A[1] = (a_{63}, a_{62}, \cdots, a_{33}, a_{32})$, $A[0] = (a_{31}, a_{30} \cdots, a_1, a_0)$. Finally, let $A[i]_k$ be the k-th element of $A[i]$ (for example, $A[1]_0$ is a_{32}).

2.2 Arithmetic in Finite Field $\mathbb{F}_{3^m}$

Addition/Subtraction $A(x) \pm B(x)$ in $\mathbb{F}_{3^m}$ can be efficiently implemented by logical operations such as AND, OR and XOR [8].

Shift-and-Add method [11] is well known as a standard algorithm for computing the polynomial multiplication over finite fields. This method is an algorithm which shifts $A(x)$ from right to left, and performs addition $C(x) \leftarrow A(x) \pm B(x)$ based on each (hi, lo)-bit of $B(x)$, where $A(x), B(x)$ and $C(x)$ are elements in $\mathbb{F}_{3^m}$. Therefore Shift-and-Add method requires $\frac{2}{3}m^2$ additions $a + b$ for $a, b \in \mathbb{F}_3$ and $\frac{2}{3}m^2$ substitution operations $c \leftarrow a + b$ for the resulting addition. In this paper the leftarrow ($\leftarrow$) is called the substitution operation. Comb method [11] is another efficient algorithm for computing the polynomial multiplication, which shifts $A(x)$ from right to left only after performing t additions in $\mathbb{F}_{3^m}$ based on each (hi, lo)-bit of the array $B[t-1], \cdots, B[1], B[0]$. $A(x) \cdot x^{iW+k}$ is computed by shifting $A(x)$ for $k \in [0, W-1]$, $j \in [0, t-1]$ and the word size W. Therefore the number of the substitution operations in Comb method becomes $\lceil m/W \rceil$ times smaller than that of Shift-and-Add method, namely it is $1/t$. In our implementation in Section 3, we use the Comb method because the substitution operations on ARM9 processors are inefficient to our experiment.

In characteristic 3, cube is represented by $A(x)^3 = \sum_{i=0}^{m-1} a_i x^{3i}$. Cube can be computed analogously to square in characteristic 2 by using table look-up [8]. Therefore, this process is fast because it does not require the multiplication step actually. Inversion can be computed by using the extended Euclidean algorithm for the polynomials over $\mathbb{F}_3[x]$ [13]. This algorithm was improved extended Euclidean algorithm in characteristic 2.

Note that each operation in $\mathbb{F}_{3^m}$ has the following relationship of the cost:

$$\mathrm{A} < \mathrm{C} < \mathrm{M} < \mathrm{I},$$

where A, C, M and I are the cost for addition/subtraction, cube, multiplication and inversion, respectively. In our implementation of $\mathbb{F}_{3^{97}}$, the difference of the costs in each inequality sign ($<$) is about 10 times (see for Section 3.4).

2.3 Arithmetic in Extended Field $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$

We can represent $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$ by a tower of extensions to use the operations of $\mathbb{F}_{3^m}$. Let $\mathbb{F}_{3^{3m}} = \mathbb{F}_{3^m}[\rho]/(\rho^3-\rho-1)$ and $\mathbb{F}_{3^{6m}} = \mathbb{F}_{3^{3m}}[\sigma]/(\sigma^2+1)$. An element A in $\mathbb{F}_{3^{6m}}$ is represented as follows:

$$\begin{aligned} A &= \alpha_1\sigma + \alpha_0 && (1)\\ &= a_5\sigma\rho^2 + a_4\sigma\rho + a_3\sigma + a_2\rho^2 + a_1\rho + a_0 && (2)\\ &= (a_5, a_4, a_3, a_2, a_1, a_0) && (3) \end{aligned}$$

where α_i is an element in $\mathbb{F}_{3^{3m}}$, a_j is an element in $\mathbb{F}_{3^m}$ and $i = 0, 1, j = 0, 1, \cdots, 5$. Operations in $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$ are addition, subtraction, cube, multiplication and inversion, and the operations require operations in $\mathbb{F}_{3^m}$. The operations in $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$ are computed the same way in [9].

We show the costs of operations in $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$ in Table 1. The operations cost of addition/subtraction and cube in $\mathbb{F}_{3^{6m}}$ is just 2 times of that in $\mathbb{F}_{3^{3m}}$. Meanwhile the operation cost of multiplication and inversion in $\mathbb{F}_{3^{6m}}$ is more than 2 times of that in $\mathbb{F}_{3^{3m}}$.

Table 1. The costs of operations in $\mathbb{F}_{3^{3m}}$ and $\mathbb{F}_{3^{6m}}$

Operations	$\mathbb{F}_{3^{3m}}$	$\mathbb{F}_{3^{6m}}$
Addition/Subtraction	3A	6A
Multiplication	12A + 6M	51A + 18M
Cube	3A + 3C	6A + 6C
Inversion	6A + 15M + 1I	57A + 38M + 1I

2.4 Tate Pairing

There is Tate pairing as one of the kinds of the pairing. Tate pairing requires operations of elliptic curve over finite fields. In this paper, we use the following supersingular elliptic curve over $\mathbb{F}_{3^m}$,

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

where $\mathcal{O}$ is the point at infinity. The group order $\sharp E$ of $E(\mathbb{F}_{3^m})$ is $\sharp E = 3^m + 3^{(m+1)/2} + 1$. Let r be a prime number which satisfies with $r|\sharp E$ and $r|(3^{6m}-1)$. Tate pairing is defined as follows:

$$e\langle\cdot,\cdot\rangle : E(\mathbb{F}_{3^m})[r] \times E(\mathbb{F}_{3^{6m}})[r] \rightarrow \mathbb{F}^*_{3^{6m}}/(\mathbb{F}^*_{3^{6m}})^r$$

where $E(\mathbb{F}_{3^m})[r]$ is the subgroup of order r in $E(\mathbb{F}_{3^m})$. Point in $E(\mathbb{F}_{3^{6m}})$ is generated from point in $E(\mathbb{F}_{3^m})$ by using distortion map $\phi(x, y) = (-x + \rho, y\sigma)$.

Algorithm 1 Duursma-Lee algorithm [15]

```
INPUT: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})[r]$
OUTPUT: $e\langle P, Q\rangle \in \mathbb{F}_{3^{6m}}$
 1: initialization:
    $T \leftarrow 1$   (in $\mathbb{F}_{3^{6m}}$)
    $a \leftarrow x_p, b \leftarrow y_p, x \leftarrow {x_q}^3, y \leftarrow {y_q}^3$   (in $\mathbb{F}_{3^m}$)
    $d \leftarrow 1$   (in $\mathbb{F}_3$)
 2: for $i \leftarrow 0$ to $m-1$ do
 3:     $a \leftarrow a^9,\ b \leftarrow b^9$   (in $\mathbb{F}_{3^m}$)
 4:     $c \leftarrow a + x + d$   (in $\mathbb{F}_{3^m}$)
 5:     $R \leftarrow -by\sigma - \rho^2 - c\rho - c^2$   (in $\mathbb{F}_{3^{6m}}$)
 6:     $T \leftarrow T^3$   (in $\mathbb{F}_{3^{6m}}$)
 7:     $T \leftarrow TR$   (in $\mathbb{F}_{3^{6m}}$)
 8:     $y \leftarrow -y$   (in $\mathbb{F}_{3^m}$)
 9:     $d \leftarrow d - 1$   (in $\mathbb{F}_3$)
10: end for
11: final exponentiation:
    $T \leftarrow$ Algorithm_2 $(T)$
12: return $T$
```

Algorithm 2 Final exponentiation (Duursma-Lee algorithm) [14]

```
INPUT: $T = \tau_1\sigma + \tau_0 \in \mathbb{F}_{3^{6m}}$
OUTPUT: $T^{(3^{3m}-1)} \in \mathbb{F}_{3^{6m}}$
1: $U \leftarrow T^{-1}$   (in $\mathbb{F}_{3^{6m}}$)
2: $\tau_1 \leftarrow -\tau_1$   (in $\mathbb{F}_{3^{3m}}$)
3: $T \leftarrow UT$   (in $\mathbb{F}_{3^{6m}}$)
4: return $T$
```

The pairing $e\langle P,\ Q\rangle$ satisfies bilinearity $e\langle aP,\ Q\rangle = e\langle P,\ aQ\rangle = e\langle P,\ Q\rangle^a$, where $P,\ Q$ are points in $E(\mathbb{F}_{3^m})[r]$ and a is an integer.

Miller proposed the first polynomial time algorithm for computing Tate pairing [17]. Duursma and Lee proposed the efficient algorithm using supersingular elliptic curve over finite fields in characteristic 3 [7]. Kwon proposed an improved algorithm of Duursma-Lee algorithm which requires no cube root. We present Duursma-Lee algorithm without cube root in Algorithm 1.

Duursma-Lee algorithm has the step which is called the final exponentiation. The step is necessary to compute $T^{(3^{3m}-1)}$, where $T = \tau_1\sigma + \tau_0$ is the representation in equation (2). One final exponentiation usually requires $3m$ multiplications and 1 inversion in $\mathbb{F}_{3^{6m}}$. However it can be computed by 1 multiplication and 1 inversion due to $T^{(3^{3m}-1)} = (-\tau_1\sigma + t_0)(\tau_1\sigma + \tau_0)^{-1}$ [14].

2.5 η_T Pairing

η_T pairing can reduce the cost of Duursma-Lee algorithm to the half by using Frobenius map [1]. An improved algorithm without cube root was also

Algorithm 3 η_T pairing [4]

```
INPUT: $P = (x_p, y_p), Q = (x_q, y_q) \in E(\mathbb{F}_{3^m})[r]$,
       $S = (3^{3m} - 1)(3^m + 1)(3^m - 3^{(m+1)/2} + 1)$
OUTPUT: $(\eta_T \langle P, Q \rangle^S)^{3^{(m+1)/2}} \in \mathbb{F}_{3^{6m}}$
 1: initialization:
    $a \leftarrow x_p, b \leftarrow -y_p, x \leftarrow x_q, y \leftarrow y_q$   (in $\mathbb{F}_{3^m}$)
    $d \leftarrow 1$   (in $\mathbb{F}_3$)
    $c \leftarrow a + x + d$   (in $\mathbb{F}_{3^m}$)
    $T \leftarrow y\sigma + b\rho - bc$   (in $\mathbb{F}_{3^{6m}}$)
 2: for $i \leftarrow 0$ to $(m-1)/2$ do
 3:    $c \leftarrow a + x + d$   (in $\mathbb{F}_{3^m}$)
 4:    $R \leftarrow by\sigma - \rho^2 - c\rho - c^2$   (in $\mathbb{F}_{3^{6m}}$)
 5:    $T \leftarrow TR$   (in $\mathbb{F}_{3^{6m}}$)
 6:    $T \leftarrow T^3$   (in $\mathbb{F}_{3^{6m}}$)
 7:    $b \leftarrow -b$   (in $\mathbb{F}_{3^m}$)
 8:    $x \leftarrow x^9,\ y \leftarrow y^9$   (in $\mathbb{F}_{3^m}$)
 9:    $d \leftarrow d - 1$   (in $\mathbb{F}_3$)
10: end for
11: final exponentiation:
    $T \leftarrow$ Algorithm_4 $(T)$
12: return $T$
```

Algorithm 4 Final exponentiation (η_T pairing) [20]

```
INPUT: $K \in \mathbb{F}_{3^{6m}}$, $S = (3^{3m} - 1)(3^m + 1)(3^m - 3^{(m+1)/2} + 1)$
OUTPUT: $K^S \in \mathbb{F}_{3^{6m}}$
 1: $K \leftarrow K^{3^{3m}-1}$, $G \leftarrow \Lambda(K) = K^{3^m+1}$
 2: $K \leftarrow G$, $K \leftarrow \Lambda(K) = K^{3^m+1}$
 3: for $i \leftarrow 0$ to $(m-1)/2$ do
 4:    $G \leftarrow G^3$
 5: end for
 6: $g_2 \leftarrow -g_2$, $g_1 \leftarrow -g_1$, $g_0 \leftarrow -g_0$
 7: return $K \cdot G$
```

proposed in η_T pairing [4]. We show η_T pairing without cube root in Algorithm 3. η_T pairing need a final exponentiation step to compute T^S, where $S = (3^{3m} - 1)(3^m + 1)(3^m - 3^{(m+1)/2} + 1)$. Because S is a large value, it takes much time to compute the final exponentiation compared with Duursma-Lee algorithm. However, an efficient algorithm which uses the torus T_2 for $\mathbb{F}^*_{3^{6m}}$ was proposed, where T_2 is defined $T_2(\mathbb{F}_{3^{3m}}) = \{A_0 + A_1\sigma \in \mathbb{F}^*_{3^{6m}} : {A_0}^2 + {A_1}^2 = 1\}$ [20]. This algorithm for computing T^S is shown in Algorithm 4.

The output of Duursma-Lee algorithm and η_T pairing relates as follows:

$$(\eta_T \langle P, Q \rangle^S)^{3(3^{(m+1)/2}+1)^2} = e\langle P, Q \rangle^{-3^{(m+3)/2}},$$

and $e\langle P, Q \rangle$ can compute from $U = \eta_T \langle P, Q \rangle^S$ as follows [4]:

$$e\langle P, Q \rangle = \left(U^{3^{(m+1)/2}+2} \cdot \sqrt[3^m]{U^{(m-1)/2}} \right)^{-1}.$$

Table 2. Computation costs of Duursma-Lee algorithm and η_T pairing in $\mathbb{F}_{3^{97}}$

Duursma-Lee algorithm (Alg. 1)	4635A + 972C + 1511M + 1I	≈ 1664M
(final exponentiation (Alg. 2))	(108A + 56M + 1I)	(≈ 67M)
η_T pairing (Alg. 3)	2785A + 784C + 871M + 1I	≈ 987M
(final exponentiation (Alg. 4))	(496A + 294C + 86M + 1I)	(≈ 130M)

Here we estimate the computational costs of Duursma-Lee algorithm and η_T pairing. The extension degree of the underlying finite fields is usually chosen as $m = \{97, 167, 193, 239, 313\}$ [1, 3, 4, 9, 13, 14]. The computational cost with $m = 97$ is shown in Table 2. The third column is the estimated number of multiplications with A= 0.01M, C= 0.1M, I= 10M appeared in Section 3.4. Actually, the cost of η_T pairing is smaller than that of Duursma-Lee algorithm.

3 Implementation of the Pairing on BREW

In this section, we explain our efficient implementation of the pairing in mobilephones on BREW [1].

3.1 Experimental Environment and Analysis of the Program

In this paper, we try to implement the pairing on ARM9 processors which is currently often used for mobilephones. BREW supports an emulator of ARM processors on a PC whose programs are written in C language. A source file in C is complied using an ARM compiler on BREW, and then an executable file (`*.mod`) of BREW applications for ARM processors is generated.

Here we are interested in the timing of the executable files on ARM processors. The same source code can be also compiled using a standard C compiler, and we can examine the timing of the compiled codes on a PC. We deploy a PC (AMD Opteron Processor 246 (2.0 GHz), RAM : 1 GByte) with GCC version 3.4.2 using the flags "-O2 -fomit-frame-pointer", and mobilephones (150MHz ARM9 processor and 225MHz ARM9 processor) with an ARM complier using "-Otime" for optimizing the speed.

In order to implement the pairing, we implement the functions of the finite field in Section 2.2 and the pairing in Sections 2.4-2.5 from scratch in C language. The extension degrees and their irreducible trinomial are chosen as $m = \{97, 167, 193, 239, 313\}$ and $x^{97} + x^{12} + 2$, $x^{167} + x^{96} + 2$, $x^{193} + x^{12} + 2$, $x^{239} + x^{24} + 2$, $x^{313} + x^{126} + 2$, respectively. The functions in our implementation are named as follows: `FF_Add` (addition in $\mathbb{F}_{3^m}$), `FF_Multi` (Comb method in $\mathbb{F}_{3^m}$), `FF_Cube` (cube in $\mathbb{F}_{3^m}$) and so on. Then we examine the timing of the basic functions (`FF_Add`, `FF_Multi`, `FF_Cube`, etc) by GCC using a profiling tool. The

[1] BREW is a registered trademark of Qualcomm company and it is an application platform developed for mobilephones of cdmaOne and cdma2000.

timings of our initial implementation of Duursma-Lee algorithm and η_T pairing in $\mathbb{F}_{3^{97}}$ by profiling is shown in Table 3.

Table 3. Timings by profiling the functions in $\mathbb{F}_{3^{97}}$

`Duursma-Lee algorithm`

Time of Function	%	Hit Count	Function
927.176	84.1	199900	FF_Multi
83.649	7.6	97776	FF_Cube
22.580	2.0	408606	FF_Add

η_T `pairing`

Time of Function	%	Hit Count	Function
338.506	78.6	113200	FF_Multi
53.508	12.4	165976	FF_Cube
10.281	2.4	351406	FF_Add

In Table 3, the multiplication speeds of both Duursma-Lee algorithm and η_T pairing are about 80% in the whole program of the pairing. Accordingly, we try to optimize the speed of multiplication in $\mathbb{F}_{3^m}$.

3.2 Optimized Multiplication for BREW

In the following we propose *the improved Comb method*, which reduces the number of the substitution operations by unrolling the first loop of Comb method. Here we explain the improved Comb method with extension degree $m = 97$, but it can be applicable to other extension degrees $m = \{167, 193, 237, 313\}$.

We now focus on $B[3]$ which is one of the array representing $B(x) \in \mathbb{F}_{3^{97}}$. Note that $B[3]$ only contains the 96-th coefficient of $B(x)$ as $B[3]_0$, namely,

$$B[3] = (B[3]_{31}, B[3]_{30}, \cdots, B[3]_1, B[3]_0) = (0, 0, \cdots, 0, b_{96}).$$

When we compute $A(x) \cdot B(x)$ using Comb method in Section 2.2, we have to perform the addition of $A(x) \cdot B[3]$ and substitution operations for a temporary save of the addition even $B[3]_j = 0$ for $j = 31, 30, \cdots, 1$. There are many substitution operations and additions which do not affect the result of $A(x) \cdot B(x)$. Those operations can be eliminated by unrolling the loop of computing $B[3]$. In other words, the loop corresponding to $B[i]_j$ with $j > 0$ is not computed for $i = 3$, and we only fulfill the complete loop for $B[i]_0$ with $i = 0, 1, 2, 3$. As a result, the proposed scheme can save 31 substitution operations compared with Comb method, and we achieve about 20% faster multiplication. We show the improved Comb method in Algorithm 5.

Algorithm 5 Improved Comb method $m = 97$

```
INPUT: $A(x), B(x) \in \mathbb{F}_{3^m}$, $W = 32$ : word length
OUTPUT: $C(x) = A(x) \cdot B(x) \bmod f(x) \in \mathbb{F}_{3^m}$
 1: $C(x) \leftarrow 0$
 2: for $i \leftarrow 0$ to 3 do
 3:     $C(x) \leftarrow C(x) + B[i]_0 A(x) x^{iW}$
 4: end for
 5: for $j \leftarrow 1$ to $W$ do
 6:     for $i \leftarrow 0$ to 2 do
 7:         $C(x) \leftarrow C(x) + B[i]_j A(x) x^{iW+j}$
 8:     end for
 9: end for
10: for $i \leftarrow 2m - 2$ downto $m$ do
11:     $c_{i-85} \leftarrow c_{i-85} - c_i$
12:     $c_{i-97} \leftarrow c_{i-97} + c_i$
13:     $c_i \leftarrow 0$
14: end for
15: return $C(x)$
```

This algorithm has two steps, the polynomial multiplication step (line 1-9) and the reduction step (line 10-14), where the reduction step is the computation of $c(x) \bmod f(x)$. The main loop is line 5-9, and process of loop unrolling is line 2-4. The difference of the proposed scheme from Comb method is line 6. The number of iteration in the loop of line 6 becomes one time shorter, and the omitted process is moved to line 2.

3.3 Further Discussion on Speed-up

We carry out the following two methods for speed-up of the pairing.

In the one method, we perform effectively multiplication in $\mathbb{F}_{3^{6m}}$ in the pairing algorithms. The multiplication $T \cdot R$ in $\mathbb{F}_{3^{6m}}$ is computed in line 7 of Algorithm 1 and line 5 of Algorithm 3. Kerins et al. pointed out the element $R = r_5\sigma\rho^2 + r_4\sigma\rho + r_3\sigma + r_2\rho^2 + r_1\rho + r_0$ in $\mathbb{F}_{3^{6m}}$ satisfies $r_4 = r_5 = 0$ and $r_2 = 2$ [14]. The number of multiplication in $\mathbb{F}_{3^m}$ required for $T \cdot R$ is reduced because the multiplication with the constants (r_2, r_4 and r_5) is virtually for free. We can reduce the processing time of the whole pairing about 10% by developing the optimized multiplication for $T \cdot R$.

In the other method, we unroll the loop used in all the functions of $\mathbb{F}_{3^m}$. The functions in $\mathbb{F}_{3^m}$ processes 32 coefficients depending on word length at a time. For example, the addition of m (hi, lo)-bits is constructed $\lceil m/W \rceil$ times of loop. By unrolling this loop, the count of a pipeline hazard in the target processor can be reduced. Actually, the processing speed of the whole pairing can be improved about 30% by unrolling the loop.

Finally, we also implemented a window method in Algorithm 5. However the speed of the window method of width 2 was slower on the ARM9 processors

Table 4. The average time of the operations in $\mathbb{F}_{3^m}$ and the pairing algorithms on the 150MHz ARM9 processor (msec)

150MHz ARM9	$opt\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{167}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$	$\mathbb{F}_{3^{313}}$
Addition (A)	0.0006	0.0009	0.0012	0.0014	0.0016	0.0020
Subtraction (A)	0.0006	0.0009	0.0012	0.0014	0.0016	0.0019
Cube (C)	0.0070	0.0067	0.0156	0.0189	0.0229	0.0261
Multiplication (M)	0.0642	0.0852	0.2055	0.2200	0.3410	0.5308
Inversion (I)	0.6915	0.8360	1.8540	2.3765	3.4115	5.8725
Duursma-Lee algorithm	98.96	129.19	549.39	701.18	1303.07	2616.63
η_T pairing	56.50	76.68	337.25	401.27	738.23	1459.65

Table 5. The average time of the operations in $\mathbb{F}_{3^m}$ and the pairing algorithms on the 225MHz ARM9 processor (msec)

225MHz ARM9	$opt\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{167}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$	$\mathbb{F}_{3^{313}}$
Addition (A)	0.0004	0.0005	0.0007	0.0008	0.0009	0.0012
Subtraction (A)	0.0004	0.0005	0.0007	0.0008	0.0010	0.0012
Cube (C)	0.0050	0.0051	0.0107	0.0133	0.0155	0.0162
Multiplication (M)	0.0393	0.0530	0.1448	0.2313	0.2200	0.3420
Inversion (I)	0.4590	0.5825	1.6890	1.4480	2.3040	3.7280
Duursma-Lee algorithm	66.00	84.70	356.11	457.93	847.56	1702.64
η_T pairing	37.52	50.34	218.27	261.88	478.54	947.30

(Note that it was faster on the Opteron processor). Therefore we do not use a window method in this paper. The main reason is that the precomputation table in the window method can not be stored in the CPU cache of the mobilephones and thus we have to road it from outside the CPU cache.

3.4 Implementation Result

We show the average time of the operations in $\mathbb{F}_{3^m}$ and the pairing algorithms, Duursma-Lee algorithm and η_T pairing, on the ARM9 processors and on the Opteron processor in Table 4-6. We compute the average time for the pairing algorithm with random input at least 200 times on the ARM9 processors and at least 20,000 times on the Opteron processor. The optimized program at $m = 97$ is denoted by "$opt\mathbb{F}_{3^{97}}$". This program uses the improved Comb method with unrolling the loop for all the programs in $\mathbb{F}_{3^m}$, and other programs use Comb method without it.

In $opt\mathbb{F}_{3^{97}}$ and $\mathbb{F}_{3^{97}}$, it turns out that the addition/subtraction (A) was about 0.01 times of the multiplication (M), the cube (C) was about 0.1 times and the inversion (I) was about 10 times. This estimation have little differences with the ARM9 processors and the Opteron processor. Based on the values, we can estimate the ratios of final exponentiation of Duursma-Lee algorithm and η_T pairing

Table 6. The average time of the operations in $\mathbb{F}_{3^m}$ and the pairing algorithms on the 2.0 GHz Opteron processor (μ sec)

2.0 GHz Opteron	$opt\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{97}}$	$\mathbb{F}_{3^{167}}$	$\mathbb{F}_{3^{193}}$	$\mathbb{F}_{3^{239}}$	$\mathbb{F}_{3^{313}}$
Addition (A)	0.0118	0.0158	0.0219	0.0250	0.0280	0.0421
Subtraction (A)	0.0118	0.0160	0.0221	0.0250	0.0281	0.0421
Cube (C)	0.1631	0.1631	0.3062	0.3406	0.4151	0.4440
Multiplication (M)	1.5955	2.3438	5.1805	5.9468	8.6590	14.7052
Inversion (I)	16.6686	19.7985	44.3534	62.2334	86.1984	137.4680
Duursma-Lee algorithm	2,610	3,810	13,980	18,531	32,800	71,780
η_T pairing	1,480	2,240	8,540	10,473	18,400	39,720

Table 7. The time of RSA and ECC on the ARM9 processors (msec)

	RSA		ECC ($\mathbb{F}_{2^n}$)		ECC ($\mathbb{F}_p$)	
150MHz ARM9 processor	512 bits	80.90	`sect113r1`	587.50	`secp112r1`	557.00
	768 bits	209.97	`sect131r1`	827.00	`secp128r1`	663.90
	1,024 bits	447.93	`sect163r1`	1,360.40	`secp160r1`	887.60
	1,536 bits	1,404.21	`sect193r1`	2,018.60	`secp192r1`	1,105.80
	2,048 bits	3,083.93	`sect233r1`	3,238.70	`secp224r1`	1,391.70
225MHz ARM9 processor	512 bits	55.97	`sect113r1`	360.40	`secp112r1`	338.70
	768 bits	150.91	`sect131r1`	527.40	`secp128r1`	399.20
	1,024 bits	327.33	`sect163r1`	902.40	`secp160r1`	499.00
	1,536 bits	1,026.70	`sect193r1`	1,368.90	`secp192r1`	645.20
	2,048 bits	2,304.70	`sect233r1`	2,237.70	`secp224r1`	803.10

as 4.03% and 13.2%, respectively. The ratios required in $\mathbb{F}_{3^m}$ in the pairing computation become 2.78% addition/subtraction (A), 5.84% cube (C), 90.78% multiplication (M) and 0.60% inversion (I) in Duursma-Lee algorithm, and 2.82% addition/subtraction (A), 7.94% cube (C), 88.22% multiplication (M) and 1.01% inversion (I) in η_T pairing, respectively. In our programs except $opt\mathbb{F}_{3^{97}}$, when extension degree becomes about 2 times, the processing speed becomes about 5 times slower on both the ARM9 processors and the Opteron processor.

The size of executable file (`*.mod`) in $opt\mathbb{F}_{3^{97}}$ was 36,524 Bytes. The size of other programs $m = 97, 167, 193, 239, 313$ are 32,336 Bytes, 32,464 Bytes, 33,000 Bytes, 33,012 Bytes, 32,704 Bytes, respectively. Because the program of $opt\mathbb{F}_{3^{97}}$ increase the amount of the size of the loop unrollment, the size of $opt\mathbb{F}_{3^{97}}$ is the largest. As an average size of executable files currently, 300 Kbytes or less is standard. Then, the size of our executable files becomes the size of around 10% in BREW applications.

η_T pairing is more efficient than Duursma-Lee algorithm in ARM9 processors and the Opteron processor. The processing speed achieves 56.5 msec in the 150MHz ARM9 processor and 37.52 msec in the 225MHz ARM9 processor for computing η_T pairing on the supersingular curve over $\mathbb{F}_{3^{97}}$.

Remark 1 In paper [13], there is a report on the implementation of η_T pairing using Java mobilephones. They achieved about 500 msec for $m = 97$ on FOMA 901iS, where its CPU specification is not available from the manufacture. However, FOMA 901 is a commercial product of the same generation with the mobilephones which equips 150MHz and 225Mhz ARM9 processors. An executable file on Java mobilephones is in general slower than that in BREW mobilephones. In this case our implementation is about 10 times faster.

3.5 Comparison with Other Public Key Cryptosystems

In order to demonstrate the efficiency of our implementation of the pairing, we implement the standard public key cryptosystems in ARM9 processors on BREW, namely RSA cryptosystem (RSA) and elliptic curve cryptosystem (ECC).

The processing speed of RSA is measured with modular exponentiation $f^g \bmod h$, where f and g are integers as large as the RSA modulus h of $\{512, 1024, 1536, 2048\}$ bits. We use the Chinese remainder theorem [19], the Montgomery multiplication [16, Algorithm 14.36] and the sliding window method of width 3 [16, Algorithm 14.85]. Next, let E be an elliptic curve over finite field $\mathbb{F}_{2^n}$ or $\mathbb{F}_p$, where n is an integer and p is a prime number. We choose the elliptic curves appeared in SECG [21]. The processing speed of ECC is measured with scalar multiplication $dP \in E$, where d is a scalar as large as $\#E$ and P is a point of elliptic curve E: `sect113r1`, `sect131r1`, `sect163r1`, `sect193r1`, `sect233r1` for ECC over $\mathbb{F}_{2^n}$ and `secp112r1`, `secp128r1`, `secp160r1`, `secp192r1`, `secp224r1` for ECC over $\mathbb{F}_p$, respectively. The non-adjacent form (NAF) is used for computing the scalar multiplication [12].

The current key size of RSA is 1,024 bits. It is known that the key size of ECC with the same security level as 1,024 bits RSA is 160 bits. In our implementation, this is equivalent to ECC over $\mathbb{F}_{2^{163}}$ and $\mathbb{F}_p$ ($p = 160$ bits). On the other hand, Page et al. estimated an equivalent size of the security parameter between the pairing and RSA [18]. The security parameter of the η_T pairing over $\mathbb{F}_{3^m}$, which has 1,024 bits RSA security should be as large as $m = 193$.

The processing speed of RSA and that of ECC over $\mathbb{F}_{2^n}$ and $\mathbb{F}_p$ are shown Table 7. The speeds of 1,024 bits RSA on 150MHz and 225MHz ARM9 processors are 447.93 msec and 327.33 msec, respectively. The speeds of η_T pairing over $\mathbb{F}_{3^{193}}$ on 150MHz and 225MHz ARM9 processors are 401.27 msec and 261.88 msec in Tables 4 and 5, respectively. Thus, the speeds of η_T pairing is slightly faster than that of RSA. In addition, the speeds of ECC over $\mathbb{F}_{2^{163}}$ (and $\mathbb{F}_p$ ($p = 160$ bits)) on 150MHz and 225MHz ARM9 processors are 1,360.40, 887.60 msec (and 902.40, 499.00 msec), respectively. The speed of ECC over $\mathbb{F}_p$ is faster than that over $\mathbb{F}_{2^n}$. However the speed of ECC over $\mathbb{F}_p$ is about 2 times slower than that of η_T pairing with the same security level.

4 Conclusion

In this paper, we presented efficient implementation of Duursma-Lee algorithm and η_T pairing over $\mathbb{F}_{3^m}$ using BREW mobilephones. In our initial implementation in $\mathbb{F}_{3^{97}}$, the whole time required for the multiplication is about 80% in the computation of the pairing. We thus proposed *improved Comb method*, which is particularly effective multiplication for BREW mobilephones, namely with fewer substitution operations. Moreover, we improved the multiplication $T \cdot R$ in $\mathbb{F}_{3^{6m}}$ of the pairing algorithms and we performed loop unrolling in the finite field.

As a result, the processing speed of our optimized pairing implementation using BREW on 150MHz and 225MHz ARM9 processors achieved under 100 milliseconds. In addition, we represented that the processing speed of η_T pairing is relatively faster than RSA or ECC with the equivalent security. It has become efficient enough to implement security applications, such as short signature, ID-based cryptosystems or broadcast encryption, using the pairing on BREW mobilephones.

References

1. P. Barreto, S. Galbraith, C. O'hEigeartaigh and M. Scott, "Efficient Pairing Computation on Supersingular Abelian Varieties", Designs, Codes and Cryptography, Vol.42, No.3, pp.239-271, 2007.
2. D. Boneh, C. Gentry and B. Waters, "Collusion Resistant Broadcast Encryption with Short Ciphertexts and Private Keys", CRYPTO 2005, LNCS 3621, pp.258-275, 2005.
3. P. Barreto, H. Kim, B. Lynn and M. Scott, "Efficient Algorithms for Pairing-Based Cryptosystems", CRYPTO2002, LNCS 2442, pp.354-368, 2002.
4. J. Beuchat, M. Shirase, T. Takagi and E. Okamoto, "An Algorithm for the η_T Pairing Calculation in Characteristic Three and its Hardware Implementation", IEEE International Symposium on Computer Arithmetic, ARITH-18, pp.97-104, 2007.
5. D. Boneh and M. Franklin, "Identity Based Encryption from the Weil Pairing", SIAM Journal of Computing, Vol.32, No.3, pp.514-532, 2001.
6. D. Boneh, B. Lynn and H. Shacham, "Short Signatures from the Weil Pairing", ASIACRYPT 2001, LNCS 2248, pp.514-532, 2001.
7. I. Duursma and H. Lee, "Tate Pairing Implementation for Hyperelliptic Curves $y^2 = x^p - x + d$", ASIACRYPT 2003, LNCS 2894, pp.111-123, 2003.
8. S. Galbraith, K. Harrison and D. Soldera, "Implementing the Tate Pairing", ANTS V, LNCS 2369, pp.324-337, 2001.
9. R. Granger, D. Page and M. Stam, "On Small Characteristic Algebraic Tori in Pairing-Based Cryptography", LMS Journal of Computation and Mathematics, Vol.9, pp.64-85, 2006.
10. R. Granger, D. Page and M. Stam, "Hardware and Software Normal Basis Arithmetic for Pairing-Based Cryptography in Characteristic Three", IEEE Transactions on Computers, Vol.54, No.7, pp.852-860, 2005.
11. D. Hankerson, A. Menezes and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, 2004.
12. IEEE P1363, Standard Specifications for Public-Key Cryptography. http://grouper.ieee.org/groups/1363/
13. Y. Kawahara, T. Takagi and E. Okamoto, "Efficient Implementation of Tate Pairing on a Mobile Phone using Java", CIS 2006, LNAI 4456, pp.396-405, 2007.

14. T. Kerins, W. Marnane, E. Popovici and P. Barreto, "Efficient Hardware for the Tate Pairing Calculation in Characteristic Three", CHES 2005, LNCS 3659, pp.412-426, 2005.
15. S. Kwon, "Efficient Tate Pairing Computation for Supersingular Elliptic Curves over Binary Fields", Cryptology ePrint Archive, Report 2004/303, 2004.
16. A. Menezes, P. van Oorschot and S. Vanstone, Handbook of Applied Cryptography, CRC Press, 1966.
17. V. Miller, "Short Programs for Functions on Curves", Unpublished Manuscript, 1986.
18. D. Page, N. Smart, F. Vercauteren, "A comparison of MNT curves and supersingular curves", Applicable Algebra in Engineering, Communication and Computing, Vol.17, No.5, pp.379-392, 2006.
19. Public-Key Cryptography Standards, PKCS#1, Version 2.1, RSA Laboratories. http://www.rsasecurity.com/rsalabs/pkcs/
20. M. Shirase, T. Takagi and E. Okamoto, "Some Efficient Algorithms for the Final Exponentiation of η_T Pairing", ISPEC 2007, LNCS 4464, pp.254-268, 2007.
21. SECG, Standards for Efficient Cryptography Group. http://www.secg.org/

A Function $\Lambda(K)$ in Algorithm 4

In this appendix, we describe the function $\Lambda(K)$ used in Algorithm 4 of this paper.

Algorithm 6 Computation of $\Lambda(K)$ [20]

```
INPUT: $K = (k_5, k_4, k_3, k_2, k_1, k_0) \in \mathbb{F}_{3^{6m}}$
OUTPUT: $\Lambda(K) = K^{3^m+1} \in T_2(\mathbb{F}_{3^{3m}})$
 1: $v_0 \leftarrow k_0k_2$, $v_1 \leftarrow k_3k_5$, $v_2 \leftarrow k_1k_2$, $v_3 \leftarrow k_4k_5$
 2: $v_4 \leftarrow (k_0 + k_3)(k_2 - k_5)$, $v_5 \leftarrow k_3k_1$, $v_6 \leftarrow k_0k_4$
 3: $v_7 \leftarrow (k_0 + k_3)(k_1 + k_4)$, $v_8 \leftarrow (k_1 + k_4)(k_2 - k_5)$
 4: $c_0 \leftarrow 1 + v_0 + v_1 \mp v_2 \mp v_3$
 5: $c_1 \leftarrow v_7 - v_2 - v_3 - v_5 - v_6$   $(m \equiv 1 \bmod 12)$
    $c_1 \leftarrow v_5 + v_6 - v_7$               $(m \equiv -1 \bmod 12)$
 6: $c_2 \leftarrow v_2 + v_3 + v_7 - v_5 - v_6$
 7: $c_3 \leftarrow v_1 + v_4 \pm v_5 - v_0 \mp v_6$
 8: $c_4 \leftarrow v_3 + v_8 \pm v_0 - v_2 \mp v_1 \mp v_4$
 9: $c_5 \leftarrow \pm v_3 \pm v_8 \mp v_2$
10: return $C = (c_5, c_4, c_3, c_2, c_1, c_0)$
```