IEICE TRANS. FUNDAMENTALS, VOL. E82–A, NO. 1 JANUARY 1999

PAPER *Special Section on Cryptography and Information Security*

A Small and Fast Software Implementation of Elliptic Curve Cryptosystems over $GF(p)$ on a 16-Bit Microcomputer

Toshio HASEGAWA[†], ***Nonmember***, **Junko NAKAJIMA**[†], ***and*** **Mitsuru MATSUI**[†], ***Members***

SUMMARY Recently the study and implementation of elliptic curve cryptosystems (ECC) have developed rapidly and its achievements have become a center of attraction. ECC has the advantage of high-speed processing in software even on restricted environments such as smart cards. In this paper, we concentrate on complete software implementation of ECC over a prime field on a 16-bit microcomputer M16C (10 MHz). We propose a new type of prime characteristic of base field suitable for small and fast implementation, and also improve basic elliptic arithmetic formulas. We report a small and fast software implementation of a cryptographic library which supports 160-bit elliptic curve DSA (ECDSA) signature generation, verification and SHA-1 on the processor. This library also includes general integer arithmetic routines for applicability to other cryptographic algorithms. We successfully implemented the library in 4 Kbyte code/data size including SHA-1, and confirmed a speed of 150 msec for generating an ECDSA signature and 630 msec for verifying an ECDSA signature on M16C.

key words: *public key cryptography, elliptic curve cryptosystem, GF(p), 16-bit microcomputer, implementation*

1. Introduction

When we use a software public key cryptosystem on high speed hardware platforms such as modern PCs, its computation time or program size is now rarely a serious problem. However in restricted hardware environments with limited computational power and small ROM/RAM size such as smart cards, it is still a heavy load to process public key cryptographic algorithms such as digital signature. We need a public key cryptosystem with small code/data size and less computational complexity to enjoy benefits of digital signature in these circumstances.

Recently the study and implementation of elliptic curve cryptosystems (ECC) [1]–[3], [8]–[11] have developed rapidly and its achievements have become a center of attraction because of its many advantages in comparison to conventional public key systems. ECC has the highest strength-per-key-bit of any known public-key system. For example, ECC with a 160-bit modulus offers the same level of cryptographic security as RSA with 1024-bit moduli. The smaller key sizes result in smaller system parameters, bandwidth savings, faster implementations and lower power consumptions. These advantages lead to the possibility to achieve public key schemes more easily in software.

For a software implementation of ECC over a field of characteristic 2, a smart card system with an 8-bit microprocessor is known [4]. It contains an elliptic curve digital signature function, which generates a signature in 600 msec, but does not have a signature verification function, which generally requires much heavier computation than signature generation.

As for ECC over a prime field, many papers have been presented about high performance implementation on powerful software computational environments with high clock rates [11] or on a special public key coprocessor hardware. An implementation with an arithmetic coprocessor on a IC card is reported in [16]. It has an 160-bit ECDSA signature generation function only (308 msec at 5 MHz) without hash function. In this case, code/data size is about 7.7 Kbyte. But special hardware is generally expensive. On the other hand, as far as the authors know, a serious example of complete software implementation on processors with low clock rates has not been reported yet. Therefore we focus on a complete software implementation.

ECC over a prime field has two advantages: one is a potential use of an existent co-processor (for instance, a modular-multiplier for RSA), and the other is a possibility for efficiently using a multiplier/divider embedded in a target processor. Hence we tried to implement a ECC over a prime field enough small and fast only in software. As a target processor, we have chosen a 16-bit microcomputer M16C (10 MHz) [12], [13], which has been widely used for engineering applications such as mobile telecommunication systems.

One of our purposes is to design a small and fast cryptographic library which supports SHA-1 [15], ECDSA signature generation and verification [7], which is currently being standardized in the ANSI X9F1 and IEEE P1363 standards committees. We also regarded applicability to other cryptographic algorithms, not only ECC but also RSA, for example, as important. To achieve this, we designed two independent integer arithmetic modules, where one is a group of routines

Manuscript received March 27, 1998.
Manuscript revised July 27, 1998.

[†]The authors are with Information Technology R&D Center, Mitsubishi Electric Corporation, Kamakura-shi, 247–8501 Japan.

that execute modular arithmetic modulo a fixed prime characteristic p for high speed computation, and the other is a collection of general integer routines that accept any positive integers with arbitrary length for wider applicability.

In order to realize small and fast ECC library, a choice of system parameters, particularly a prime characteristic is important. In this paper we propose a new type of prime charactersitic of a base field suitable for small implementation.

For arithmetic algorithms of an elliptic curve, in order to realize 4 Kbyte code/data size, which we regard as a reasonably small size for most applications on M16C, we mainly adopted standard methods described in [6]. We also show an improved version of basic arithmetic formulas, that is, elliptic doubling and addition formulas on projective coordinates, which reduce the code size and the number of temporary variables.

As a result, we successfully implemented the library in 4 Kbyte code/data size including SHA-1, and confirmed a speed of 150 msec for generating a 160-bit ECDSA signature and 630 msec for verifying an ECDSA signature on M16C. There are many trade-offs between speed and code/data size. We can further reduce code/data size at the cost of processing speed, or it is also possible to get higher performance with more code/data size.

This paper is organized as follows. In Sect. 2, we give a brief explanation of arithmetic and DSA algorithm on an elliptic curve. In Sect. 3, we introduce a 16-bit engineering microcomputer M16C. In Sect. 4, we discuss the curve parameters and algorithms used in this paper for a small and fast implementation. In Sect. 5, we show our software architecture and programming style. Results and possible enhancements are given in detail in Sect. 6.

2. Arithmetic on Elliptic Curve

In this section, we briefly explain arithmetic and DSA algorithm on an elliptic curve over a field of prime characteristic.

2.1 Elliptic Curve

An elliptic curve over K is given as follows:

$$E : y^2 = x^3 + ax + b \quad (a, b \in K, 4a^3 + 27b^2 \neq 0), \quad (1)$$

where K is a finite field of characteristic $\neq 2, 3$. Then the set of K-rational points on E (with a special element O at infinity), denoted $E(K)$, is a finite abelian group, where $E(K) = \{(x, y) \in K^2 | y^2 = x^3 + ax + b\} \cup \{O\}$.

2.2 Addition Formula in the Affine Coordinates

We show the addition formulas in the affine coordinates. Let $P = (x_1, y_1)$, $Q = (x_2, y_2)$ and $P + Q = (x_3, y_3)$ be points on $E(K)$.

Curve addition formula in the affine coordinates ($P \neq \pm Q$):

$$x_3 = \lambda^2 - x_1 - x_2, \quad (2)$$

$$y_3 = \lambda(x_1 - x_3) - y_1, \quad (3)$$

$$\lambda = \frac{y_2 - y_1}{x_2 - x_1}. \quad (4)$$

Curve addition formula in the affine coordinates ($P = Q$):

$$x_3 = \lambda^2 - 2x_1, \quad (5)$$

$$y_3 = \lambda(x_1 - x_3) - y_1, \quad (6)$$

$$\lambda = \frac{3{x_1}^2 + a}{2y_1}. \quad (7)$$

2.3 Addition Formula in the Projective Coordinates

Because the addition formula in the affine coordinates requires a modular inversion which is expensive, we often use the projective coordinates to reduce the number of modular inversions. The conversion from affine to projective is as follows:

$$X = x, Y = y, Z = 1. \quad (8)$$

After this conversion, the elliptic addition and doubling can be done on this projective coordinates without any inversion. And then we convert from projective to affine as follows.

$$x = \frac{X}{Z^2}, y = \frac{Y}{Z^3}. \quad (9)$$

In this procedure, we need a modular inverse operation only once.

Next we give a brief explanation of the addition formula on projective coordinates. Let $P = (X_1, Y_1, Z_1), Q = (X_2, Y_2, Z_2)$ and $P+Q = (X_3, Y_3, Z_3)$.

Doubling formula in the projective coordinates ($P = Q$) is as follows.

$$2(X_1, Y_1, Z_1) = (X_2, Y_2, Z_2), \quad (10)$$

where

$$X_2 = \left(3{X_1}^2 + a{Z_1}^4\right)^2 - 8X_1{Y_1}^2, \quad (11)$$

$$Y_2 = (3{X_1}^2 + a{Z_1}^4)(4X_1{Y_1}^2 - X_2) - 8{Y_1}^4, \quad (12)$$

$$Z_2 = 2Y_1Z_1. \quad (13)$$

Addition formula in the projective coordinates ($P \neq \pm Q$) is as follows.

$$(X_0, Y_0, Z_0) + (X_1, Y_1, Z_1) = (X_2, Y_2, Z_2), \quad (14)$$

where

$$V = (X_0{Z_1}^2 + X_1{Z_0}^2)(X_0{Z_1}^2 - X_1{Z_0}^2)^2 - 2X_2, \quad (15)$$

$$X_2 = (Y_0{Z_1}^3 - Y_1{Z_0}^3)^2 - (X_0{Z_1}^2 + X_1{Z_0}^2)(X_0{Z_1}^2 - X_1{Z_0}^2)^2, \quad (16)$$

$$2Y_2 = V(Y_0{Z_1}^3 - Y_1{Z_0}^3) - (Y_0{Z_1}^3 + Y_1{Z_0}^3)(X_0{Z_1}^2 - X_1{Z_0}^2)^3, \quad (17)$$

$$Z_2 = Z_0 Z_1 (X_0{Z_1}^2 - X_1{Z_0}^2). \quad (18)$$

2.4 DSA Signature Generation and Verification on Elliptic Curve

We define P as the base point whose order is a large prime number n. e is a hashed message. The private key d and the public key Q have the relation that $Q = dP$. The DSA signature $sig = (r, s)$ generation scheme is as follows:

1. Generate a random number k.
2. Calculate elliptic scalar multiplication $R = kP$.
3. Calculate $r = (R)_x \bmod n$.
   If $r = 0$, then go to step 1.
4. Calculate $s = \frac{e+dr}{k} \bmod n$.
   If $s = 0$, then go to step 1.
5. Output $sig = (r, s)$.

The DSA verification scheme is as follows:

1. If r is not in $[1, n-1]$ or s is not in $[1, n-1]$, then output 'false.'
2. Calculate modular inverse $\frac{1}{s} \bmod n$
3. Calculate elliptic scalar multiplication $(\frac{e}{s} \bmod n\ )P$
4. Calculate elliptic scalar multiplication $(\frac{r}{s} \bmod n)\ Q$
5. Calculate elliptic addition $r' = ((\frac{e}{s} \bmod n)P + (\frac{r}{s} \bmod n)Q)_x$
6. If $r = r'$ then output 'true,' else output 'false.'

This elliptic curve DSA is currently being standardized within the ANSI X9F1 and IEEE P1363 standards committees.

3. 16-Bit Microcomputer M16C

We have implemented an elliptic curve cryptosystem on a 16-bit single-chip microcomputer M16C designed by Mitsubishi Electric Corporation. M16C series were developed for embedded applications and have been widely used in practice such as mobile telecommunications, industrial applications, audio-visual systems, etc. M16C family currently has the following characteristics:

- **Memory:**
  - ROM: romless, 32 KB, 64 KB, 96 KB
  - RAM: 2 KB, 3 KB, 4 KB, 10 KB
- **Shortest instruction execution time:**
  - 100 ns (at 10 MHz input clock rate)
- **Registers:**
  - Data register: 16-bit × 4 (two of them are usable as two 8-bit registers)
  - Address register: 16-bit × 2
  - Base register: 16-bit × 2
- **Instruction Sets:** 91 instructions
  - multiplication (hardwired) and division (microcode) operations
  - stack frame generation/release instructions
  - register-to-register, register-to-memory and memory-to-memory modes.
  - single-bit operations and four-bit operations.

M16C is a typical CISC processor; memory-to-memory operations can reduce code size significantly, and stack frame generation/release instructions make a simple and small subroutine calling sequence possible. M16C is designed to assign instructions of high frequency to short opcode, and has some complex instructions and various addressing modes. Moreover because it includes multiplier and DMAC, it carries out many instructions in the shortest execution time (about 70% of the instructions are executed in less than 3 cycles).

However, because of the unequal registers and complex instructions, optimizing size and performance of the code is not an easy task; for instance, a C language-like subroutine interface often results in a large code due to redundancy of parameter passing. A module interface design plays an important role for code optimization.

Therefore we wrote programs in assembly language carefully to extract maximum performance of M16C, analyzing the architecture in detail. To improve performance, we utilized special operations of M16C which can condense several instructions, and took priority over choosing short instructions. For the code/data size, we aimed at a total of 4 KB so that our program can be applicable to a future M16C family with a smaller ROM/RAM size.

4. Elliptic Curve Parameters and Algorithms

4.1 Elliptic Curve Parameters

Base Field

In this paper we propose a prime characteristic of the base field in the following form:

$$p = e2^a \pm 1. \tag{19}$$

where e is an integer whose size is within machine word size (16 in our case), and a is an integer whose value is a multiple of machine word size. Use of this form enables us to execute a galios field multiplication in a smaller code than when adopting $2^n - \epsilon$, where ϵ is a small integer, for instance. In our form, an efficient use of divide instruction results in a small code size.

By taking an assembly source example, we will briefly explain why our proposed form leads to smaller code implementation in modular multiplication arithmetic. Let us consider $a \times b \bmod p$. Here let a, b and p be n-bit numbers. Then we have the relation as follows:

$$a \times b = H \cdot 2^n + L, \tag{20}$$

where H is the upper n bits and L is the lower n bits.

In the case that $p = 2^n - \epsilon$, one can utilize the relation $a \times b = H \times 2^n + L \equiv \epsilon \times H + L \bmod p$. Therefore one needs the calculation $\epsilon \times H$. One can calculate it by performing the following sequence for each i (increasing order), where $H = \sum_{i=0} h_i \cdot 2^{16i}$; for instance,

```
mov  hi, R0          : set hi to R0 register
mul  #ε,R0           : calculate ε × R0 = R0R1
add  R3,R0           : add R0 and R3 (carry-in)
adcf R1              : add R1 with carry
add  R0,memoryi      : add R0 to result
adcf R1              : add R1 with carry
mov  R1,R3           : set R1 to R3(carry-out).
```

The code size of this sequence is 18 bytes on M16C.

In our case, for example, when $p = e \cdot 2^{n-16} - 1$, where e is 16 bit-data, the equation $H \cdot 2^{n-16} = \frac{H}{e} \cdot (p + 1)$ holds. Therefore we can get the equation $a \times b = H \cdot 2^n + L \equiv (Q \cdot 2^{16} + R \cdot 2^n + L) \bmod p$, where Q and R are the quotient and remainder of $\frac{H}{e}$ respectively. Then the essential instruction sequence is as follows.

```
mov  hi,R0           : set hi to R0 register
div  #e, R0R1        : divide R0R1 by e;
                       quotient in R0/ remainder in R1
movmovR0,memoryi     : move R0(quotient) to memoryi
```

This sequence requires 10 bytes on M16C. Note that by performing instructions above in sequence for each i (decreasing order), we can find Q and also get R at the same time.

For 160-bit ECC, the above sequence must be repeated ten times. Code size of one sequence in modular multiplication is shown in Table 1. Because we adopt loop unrolling technique, our modular multiplication arithmetic is 80 byte smaller than that of $p = 2^n - \epsilon$. As for processing time, although it seems a little slower (the detailed timing depends on the value e), we gives priority over small code size.

Table 1 Code size of one sequence in modular multiplication on M16C.

prime characteristic	Code Size
$p = e \cdot 2^{n-16} - 1$ (Our proposal)	10 bytes
$p = 2^n - \epsilon$ (Conventional)	18 bytes

Specifically in our implementation on M16C, we have adopted the following 160-bit prime number:

$$p = 65112 \times 2^{144} - 1. \tag{21}$$

Curve Parameters

We generated an elliptic curve over the prime field using complex multiplication, and for speeding up an elliptic doubling, we adjusted the constant term a of the curve equation to $p-3$. The base point (P_x, P_y) was selected randomly:

$$a = 1452046121366725933991673688168680114377396846588 = p - 3, \tag{22}$$

$$b = 6214682355133916515067362290845349684168005016 22, \tag{23}$$

$$n = 1452046121366725933991671292371452349213344743009, \tag{24}$$

$$d = -6259 \quad \text{(discriminant)}, \tag{25}$$

$$P_x = 9159058152596341855059567352513494265732120342 66, \tag{26}$$

$$P_y = 1431589911282020630356314725439635170402984187 78. \tag{27}$$

4.2 Elliptic Addition/Doubling

Table 2 shows our improved version of basic elliptic arithmetic formulas. The IEEE-P1363 document [6] describes a detailed implementation algorithm that realizes elliptic addition and doubling, where the addition formula requires four temporary variables (excluding three output variables). We however used the following improved addition algorithm which requires only two temporary variables even when $Z' \neq 1$. This algorithm can be implemented using galois field addition, subtraction, multiplication, and halving routines only. If we accept one more temporary variable, the six steps from step 09 to step 14 can be four. Note that the case $Z' \neq 1$ appears only once in the ECDSA signature verification procedure (see step 5 in the DSA verification scheme in 2.4).

Our elliptic doubling algorithm can be applied to the case $a = p-3$ and it reduces the program size. Another advantage of our addition/doubling implementation is that the third variable of the subtraction (the

Table 2 Elliptic addition/doubling formula.

Elliptic Addition Formula $(X,Y,Z) = (X,Y,Z) + (X',Y',Z')$	Elliptic Doubling Formula $(X,Y,Z) = 2(X,Y,Z)$

if ($Z' \neq 1$)
01: $T_1 = Z' * Z'$
02: $X = X * T_1$
03: $T_1 = Z' * T_1$
04: $Y = Y * T_1$
endif
 	
05: $T_1 = Z * Z$
06: $T_2 = X' * T_1$
07: $T_1 = Z * T_1$
08: $T_1 = Y' * T_1$
 	
09: $Y = Y - T_1$
10: $T_1 = 2T_1$
11: $T_1 = Y + T_1$
12: $X = X - T_2$
13: $T_2 = 2T_2$
14: $T_2 = X + T_2$
 	
if ($Z' \neq 1$)
15: $Z = Z * Z'$
endif
16: $Z = Z * X$
 	
17: $T_1 = T_1 * X$
18: $X = X * X$
19: $T_2 = T_2 * X$
20: $T_1 = T_1 * X$
 	
21: $X = Y * Y$
22: $X = X - T_2$
 	
23: $T_2 = T_2 - X$
24: $T_2 = T_2 - X$
25: $T_2 = T_2 * Y$
26: $Y = T_2 - T_1$
27: $Y = Y/2$
	01: $T_1 = Z * Z$
	02: $Z = Y * Z$
	03: $Z = 2Z$
	
	if ($a = p - 3$)
	04: $T_2 = X - T_1$
	05: $T_1 = X + T_1$
	06: $T_2 = T_1 * T_2$
	07: $T_1 = 2T_2$
	08: $T_1 = T_1 + T_2$
	else
	09: $T_1 = T_1 * T_1$
	10: $T_1 = a * T_1$
	11: $T_2 = X * X$
	12: $T_1 = T_1 + T_2$
	13: $T_2 = 2T_2$
	14: $T_1 = T_1 + T_2$
	endif
	
	15: $Y = 2Y$
	16: $Y = Y * Y$
	17: $T_2 = Y * Y$
	18: $T_2 = T_2/2$
	19: $Y = Y * X$
	
	20: $X = T_1 * T_1$
	21: $X = X - Y$
	22: $X = X - Y$
	
	23: $Y = Y - X$
	24: $Y = Y * T_1$
	25: $Y = Y - T_2$

Table 3 Number of multiplications in addition formula.

method	addition		doubling	
	$Z_1 \neq 1$	$Z_1 = 1$	$a \neq p-3$	$a = p-3$
IEEE P1363	16	11	10	8
Our proposal	16	11	10	8

Table 4 Number of temporary variables in addition formula.

method	addition		doubling	
	$Z_1 \neq 1$	$Z_1 = 1$	$a \neq p-3$	$a = p-3$
IEEE P1363	4	3	2	2
Our proposal	2	2	2	2

subtracting variable) is always different from the first variable (the output variable). This often enables a small implementation of the subtraction function.

We show the number of multiplications and temporary variables in addition formula in both our proposal and IEEE P1363 in Table 3 and Table 4. We can estimate the processing time by the number of modular multiplications. Table 3 shows that the processing time is almost the same as that in IEEE P1363. Also, the code size is almost the same. But our improvement of elliptic arithemtic results in 40 bytes smaller RAM size than IEEE P1363 because of the reduction of temporary variables.

4.3 Scalar Multiplication of a Random Point

In ECDSA, a scalar multiplication of a given random point is used in signature verification and is the most time-consuming part of the total signature computation. Although there are various works for speeding up this calculation, we here adopted a standard binary method for minimizing temporary memory, which occupies RAM area that is expensive in our environments M16C.

For example, in the case of addition-subtraction method, we can get shorter processing time, but need additional 20 byte RAM size. For another example, in the case of the window method (window size = 4), although it requires only about 1920 field multiplications in average, it needs at least additional 140 byte RAM area compared with the binary method.

Algorithm [Random Point]

- Input : a positive integer k and a random point P on an elliptic curve.
- Output: the elliptic curve point $Q = kP$.

1. Let $(0 \cdots 01k_{l-2}k_{l-3} \cdots k_1k_0)$ be the binary representation of k.
2. Set $Q \leftarrow P$.
3. For i from $l-2$ down to 0 do
   a. Set $Q \leftarrow 2Q$.
   b. If $k_i = 1$, then set $Q \leftarrow Q + P$.
4. Output Q.

4.4 Scalar Multiplication of a Fixed Point

In ECDSA, this procedure is used for a scalar multiplication of the base point, which is a fixed system parameter. We can hence reduce the calculation time by having a precomputed loop-up table in ROM area, which is less expensive than RAM area. Again to reduce the code and temporary buffer size, which is important on M16C with small RAM size, we adopted the following standard window method for the scalar multiplication.

Algorithm [Fixed Point]

Precomputed Table Generation (31 entries):
$table[n] = \sum_{j=0}^{4} n_j 2^{32j} P \quad (1 \leqq n \leqq 31)$, where n_j is the j-th bit of n.
Online Scalar Multiplication:

- Input : a positive integer k.
- Output: the elliptic curve point $Q = kP$.

1. Let $(k_{159}k_{158} \cdots k_1k_0)$ be the binary representation of k.
2. For $i = 31$ down to 0 do
   a. Set $n_i = \sum_{j=0}^{4} k_{32j+i} 2^j$.
   b. If $n_i \neq 0$, set $Q = table[n_i]$ and go to Step 3.
3. For $i = i - 1$ down to 0 do
   a. Set $n_i = \sum_{j=0}^{4} k_{32j+i} 2^j$.
   b. Set $Q \leftarrow 2Q$.
   c. If $n_i \neq 0$, set $Q \leftarrow Q + table[n_i]$.
4. Output Q.

The above algorithm requires a total of at most 31 elliptic additions and 31 elliptic doublings with a 1240-byte precomputed table. This leads to 589 field multiplications. On the other hand, the binary method on a random point requires 2160 field multiplications in average.

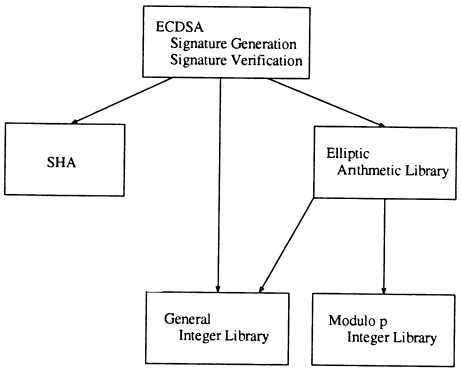

Fig. 1 Library architecture.

5. Software Architecture and Programming

To balance the signature generation/verification speed with the total program size, we designed the ECDSA software on M16C on the basis of the following hierarchical module architecture in Fig. 1. We also regarded independence of modules as important so that we could easily change elliptic curve parameters or even change cryptographic algorithms when necessary.

Modulo p Integer Library

This module consists of modular addition, subtraction, multiplication, and halving routines for the embedded modulus p, which is the characteristic of the base field. The first three routines have two interfaces; one has two arguments, one of which is an input/output parameter; and the other has three independent arguments, one of which is used for an output parameter only. The speed of these routines dominates the overall performance of the signature. Hence for maximizing the performance rather than reducing the code size, we adopted a 160-bit fixed I/O interface and introduced an extensive use of loop unrolling technique. We also designed this module so that no other modules could depend on elliptic parameters. For reducing the code size, we have not included an optimized modular square routine, which is generally 20–30% faster than a modular multiplication routine. Our estimation is that another 100–200 bytes are required for realizing this modular square routine.

General Integer Library

This module is intended to be a collection of general purpose routines for treating integers with arbitrary length. It covers move, addition, subtraction, multiplication, division and inversion (modulo an integer) routines, accepting any length as input. In ECDSA, this module is mainly used for deriving an inverse element both in modulus n, the order of the curve, and in modulus p, the order of the field, where the latter is necessary for transforming a projective coordinate into an affine coordinate.

This module is slightly size-redundant for implementing ECDSA only, but we decided to accept this structure to maintain an ability of extension to other cryptographic algorithms.

Elliptic Arithmetic Library

This library consists of elliptic addition/doubling, elliptic scalar multiplication and coordinate conversion routines. The addition and doubling routines carry out an addition and a doubling of elliptic curve points in projective coordinates, respectively. The algorithms we used are described in the preceeding section. The scalar multiplication consists of two routines: one for a scalar multiplication of the fixed base point and the other for a scalar multiplication of a randomly given point. Mainly for reducing the code size, we adopted standard algorithms for realizing these routines, as described in the preceeding section. The coordinate conversion routine changes projective coordinates into affine coordinates.

These routines require a 160-bit interface, but are independent of the value of the prime characteristic p or the base point (P_x, P_y). While the conversion routine requires both of the modulo p library and the general integer library because of necessity for an inversion operation in modulus p, the other routines use the modulo p library only.

It could be possible to design a special (modulo p) inversion routine which would slightly speed up the coordinate conversion procedure. But we did not adopt this strategy because the improvement of the total performance of signature generation/verification would be small, while the increase in the code size would not be negligible.

ECDSA Signature Generation/Verification

This top level module makes procedure calls for SHA-1, the elliptic scalar multiplication, the elliptic coordinate conversion and the general integer operations. The general integer library is needed for modulo n arithmetic operations, particularly an inversion modulo n.

6. Implementation Results

In this section, we summarize our results of implementation, including the code size of each module and performance of some major functions. We also discuss possible size/speed enhancements of this library at the end of this section.

Table 5 Code size of our library.

Module	Size
ECDSA Signature Generation and Verfication	220 bytes
Elliptic Arithmetic Library	690 bytes
Modulo p Integer Library	600 bytes
General Integer Library	780 bytes
SHA-1	430 bytes
DATA Table	1280 bytes
Total Size	4000 bytes

Table 6 Performance of our library.

Module	Elapsed Time
Elliptic Scalar Multiplication (Random Point)	480 msec
Elliptic Scalar Multiplication (Fixed Point)	130 msec
Inversion Modulo 160-bit Intege	8 msec
SHA-1 (time for processing one block)	2 msec

Table 7 Performance of 160-bit ECDSA on M16C.

Function	Elapsed Time
ECDSA signature generation	150 msec
ECDSA signature verification	630 msec

6.1 Code/Data Size

We have successfully designed the program in a total size of 4 Kbyte code/data, which was one of our goals in the implementation. The size of each module is shown in Table 5.

The fixed data table consists of the 1240 byte precomputed table, the 20 byte base prime p and the 20 byte curve order n. Therefore the real code size is within 3 Kbyte.

6.2 Processing Time

Table 6 shows the processing time of a scalar multiplication of a randomly given point and the fixed point (the base point), a modulo inversion of a given 160-bit data, and SHA-1.

This result clearly shows that although the inversion routine and the hash function take much less time than the elliptic scalar multiplication, their computational time can not be ignored.

Table 7 shows a total computational time of ECDSA signature generation and verification. In both signature generation and verification, we have included time for one-block processing of SHA-1 for hashing a message.

Table 8 Performance of 1024-bit RSA on M16C.

Function	Elapsed Time
RSA signature generation (Chinese remainder theorem)	10 sec
RSA signature verification ($e = 2^{16} + 1$)	400 msec

6.3 Estimation of 1024-Bit RSA

Since our program includes the general integer library, it is easy to implement RSA cryptosystem using the library. Table 8 shows our estimation of 1024-bit RSA signature generation/verification operations. Because the library is not speed-optimized, the signature generation is slow, but the verification is still faster than that of ECDSA.

6.4 Possible Enhancements

In this paper, to realize 4 KB code/data, we adopted standard computation algorithms and coding methods. In the following, we give a list for possible enhancements for further speeding up and/or reducing code size.

Since the modulo p integer library has been coded with a loop unrolling for fast computation, the code size is expected to be reduced by up to 200–300 bytes by introducing a usual loop rolling. It is also possible to reduce the precomputed table or even ignore scalar multiplication of the base point, but this will lead to a heavy penalty of time for the signature generation. Another possibility is to reduce the code size of the general integer library at the cost of applicability to other algorithms. We estimate that it is possible to reduce another 300–400 bytes with 50% loss of the overall performance.

On the other hand, adding an optimized modulo square routine is expected to speed up the signature generation/verification by 10 to 20%. Optimizing the speed of the general integer routine will make RSA faster. We estimate that RSA can be up to twice faster with another 500 bytes.

7. Conclusions

In this paper, we dealt with a small and fast software implementation of ECC over prime field on limited resources. We proposed a new type of prime characteristic of base field and improved basic elliptic arithmetic formulas suitable for small and fast implementation. As a result, we successfully implemented 4 KB software cryptographic library for 160-bit ECDSA signature generation, verification and SHA-1 on M16C (10 MHz) and confirmed a speed of 150 msec for generating an ECDSA signature and 630 msec for verifying an ECDSA signature. We conclude that it is possible to provide a small and fast cryptographic library of elliptic curve cryptosystems over a prime field on restricted resources. We can also apply this library to other cryptographic algorithms using the general integer library, such as RSA.

References

[1] N. Koblitz, "Elliptic curve cryptosystems," Mathematics of Computation, vol.48, pp.203–209, 1987.
[2] V.S. Miller, "Use of elliptic curves in cryptography," Advances in Cryptology-Proceedings of Crypto'85, Lecture Notes in Computer Science, vol.218, pp.417–426, Springer-Verlag, 1986.
[3] G. Agnew, R. Mullin, and S. Vanston, "An Implementation of elliptic curve cryptosystems over $F_{2^{155}}$," IEEE J. Sel. Areas Commun., vol.11, pp.804–813, 1993.
[4] Certicom SigGen Smart Card, http://205.150.149.57/ce2/embed.htm.
[5] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithm," IEEE Trans. Inf. Theory, vol.IT-31, pp.469–472, 1985.
[6] IEEE P1363 Working Draft Appendices, Feb. 6, 1997.
[7] IEEE P1363 Draft Version 1, Dec. 19, 1997.
[8] G. Harper, A. Menezes, and S. Vanstone, "Public-key cryptosystems with very small key lengths," Advances in Cryptology-Proceedings of Eurocrypt'92, Lecture Notes in Computer Science, vol.658, pp.163–173, Springer-Verlag, 1993.
[9] R. Schroeppel, H. Orman, S. O'Malley, and O. Spatscheck, "Fast key exchange with elliptic curve systems," Advances in Cryptology-Proceedings of Crypt'95, Lecture Notes in Computer Science, vol.963, pp.43–56, Springer-Verlag, 1995.
[10] E.D. Win, A. Bosselaers, and S. Vandenberghe, "A fast software implementation for arithmetic operations in $GF(2^n)$," Advances in Cryptology-Proceedings of Asiacrypt'95, Lecture Notes in Computer Science, vol.1163, pp.65–76, Springer-Verlag, 1996.
[11] A. Miyaji, T. Ono, and H. Cohen, "Efficient elliptic curve exponentiation," Information and Communications Security ICICS'97, Lecture Notes in Computer Science, vol.1334, pp.282–291, Springer-Verlag, 1997.
[12] "User Manual of M16C/60 Series," Mitsubishi Electric Corporation, 1996.
[13] "Software Manual of M16C/60 Series," Mitsubishi Electric Corporation, 1996.
[14] National Institute of Standards and Technology, NIST FIPS PUB 186, "Digital Signature Standard," U.S. Department of Commerce, 1994.
[15] National Institute of Standards and Technology, NIST FIPS PUB 180, "Secure Hash Standard," U.S. Department of Commerce, 1995.
[16] S. Miyazaki, S. Furuya and K. Takaragi, "Fast elliptic curve signature method on smart card," Symposium on Cryptography and Information Security, 1998.

Toshio Hasegawa was born in Tokyo, on August 19, 1970. He received the B.S. and M.S. degrees in physics from University of Tokyo, Tokyo, Japan, in 1993 and 1995 respectively. In 1995 he joined Mitsubishi Electric Corporation and engaged in cryptography and information security. His research interests include public key cryptosystems, computer arithmetic and computer architecture. He was awarded SCIS'98 paper prize. He is a member of IPSJ and JSAI.

Junko Nakajima received the B.S. degree in physics from Nara Woman's University, Nara, Japan in 1988. In 1988 she joined Mitsubishi Electric Corporation and engaged in designing microcomputer, cryptography and information security. Her research interests include fast software implementation and public key cryptosystems. She was awarded SCIS'98 paper prize.

Mitsuru Matsui received the B.S. and M.S. degrees in mathematics from Kyoto University, Kyoto, Japan, in 1985 and 1987 respectively. In 1987 he joined Mitsubishi Electric Corporation and engaged in coding theory, cryptography and information security. His research interests include block ciphers, public key cryptosystems and their implementation. He was awarded SCIS'93 and SCIS'94 paper prizes and also received the Best Paper Award of the Institute of Electrical and Communication Engineers of Japan in 1995. He is a member of IACR.