Elliptic Curve Arithmetic Using SIMD

Kazumaro Aoki[1]*, Fumitaka Hoshino[2], Tetsutaro Kobayashi[2], and Hiroaki Oguro[2]

[1] NTT Communications
[2] NTT Information Sharing Platform Laboratories, NTT Corporation
{maro,fhoshino,kotetsu,oguro}@isl.ntt.co.jp

Abstract. Focusing on servers that process many signatures or ciphertexts, this paper proposes two techniques for parallel computing with SIMD, which significantly enhances the speed of elliptic curve scalar multiplication. We also evaluate one of them based on a real implementation on a Pentium III, which incorporates the SIMD architecture. The results show that the proposed method is about 4.4 times faster than the conventional method.

1 Introduction

Elliptic curve cryptosystems have become an essential technology since they provide many advantages such as a short key length [1] and fast computation. The use of elliptic curves in cryptography was suggested independently by Miller [2] and Koblitz [3] in 1985. The elliptic curve cryptosystem is an elliptic curve analog of a discrete logarithm based scheme. Elliptic curves can be defined over any finite field $\mathbb{F}_q$. If q is prime then $\mathbb{F}_q$ is a 'prime field.' If q is of the form 2^m then $\mathbb{F}_q$ is a 'binary field.' Elliptic curves over prime and binary fields have become widely accepted and included in standards approved by accredited organizations such as ANSI, IEEE, and ISO.

Many efficient computational algorithms have been proposed for the elliptic curve cryptosystems [4,5]. Several kinds of smart cards have a co-processor which accelerate elliptic curve calculations. Moreover, the recent results shows that we can implement elliptic curve on a smart card without co-processor [6].

As many smart cards are widely distributed, elliptic curve implementations must be accelerated especially for server systems that process a large number of signatures or ciphertexts such as electronic voting, electronic cash, time stamping, and certificate authentication. Koyama and Tsuruoka [7] studied a parallel multiplication environment for elliptic curve arithmetic. Recently, Smart [8] proposed the parallel implementation for a special class of elliptic curves called "Hessian form," namely those with a point of order three.

This paper discusses parallel implementations of elliptic curves over both prime and binary fields. We describe the implementations over prime fields in Sect. 2 and those over binary fields in Sect. 3.

* This work was done while the author was in NTT Information Sharing Platform Laboratories.

G.I. Davida and Y. Frankel (Eds.): ISC 2001, LNCS 2200, pp. 235–247, 2001.

2 Parallel Elliptic Curve Arithmetic

2.1 SIMD

The CPU word size has increased year by year from 32 to 64 and now to 128 bits. Most of these CPUs have the Single Instruction Multiple Data (SIMD) architecture which means that multiple data sets can be processed in parallel. This architecture becomes particularly important as the clock frequency of a CPU approaches the limit. Now, many processors incorporate the SIMD architecture such as the Pentium III has MMX and SSE, SPARC has VIS, and the PowerPC has AltiVec.

Lipmaa [9] presented results on a secret key cryptosystem. Dixon and Lenstra [10] presented results on a factoring. The papers showed that the SIMD characteristic of parallel execution is useful in the quick encryption of a secret key cipher and a factoring. However, the characteristics of SIMD, already explained before, does not depend on secret key cipher properties, but also that it can perform fast public key cryptosystems. This paper utilizes those characteristic to optimize elliptic curve implementations.

It is efficient to use SIMD technology if every operation can be executed simultaneously. However, there is a problem with the data dependency chain. Operations that SIMD processes must be independent of each other.

Consider the following example. When we compute $A + B + C + D$, first, we compute $U \leftarrow A + B$. Next, we compute $V \leftarrow U + C$. Finally, we compute $R \leftarrow V + D$. Then, we have the summation $A+B+C+D$. We cannot apply SIMD to such an algorithm. Changing the order of the computation solves the problem in this case. First, we compute $S \leftarrow A + B$ and $T \leftarrow C + D$ simultaneously. Next, we compute $R \leftarrow S + T$. Then, we have the summation $A + B + C + D$ using the SIMD processor.

We propose semi-optimal methods using SIMD to compute elliptic curve addition and doubling over $\mathbb{F}_q$, where $\mathrm{char}\,\mathbb{F}_q > 3$.

2.2 Elliptic Curve Arithmetic over $\mathbb{F}_q$, $\mathrm{char}\,\mathbb{F}_q > 3$

A non-supersingular elliptic curve defined over the finite field, $\mathbb{F}_q$; $\mathrm{char}\,\mathbb{F}_q > 3$ is given by (1).

$$E : y^2 = x^3 + ax + b \qquad (a, b \in \mathbb{F}_q, 4a^3 + 27b^2 \neq 0) \tag{1}$$

We denote the group of $\mathbb{F}_q$-rational points (x, y) on E together with a 'point at infinity' $\mathcal{O}$ as $E(\mathbb{F}_q)$.

In Jacobian coordinates, we set $(x, y) = (X/Z^2, Y/Z^3)$, giving the equation

$$E_J : Y^2 = X^3 + aXZ^4 + bZ^6, \tag{2}$$

and represent rational points on E by (X, Y, Z).

If a scalar multiplication is implemented by the binary method [11, pp.614-615], we can assume that the addend of elliptic curve additions is normalized, i.e.,

$Z = 1$, since the addend is the input of the method. Even if a scalar multiplication is implemented by the Window method [11, pp.615-616] [1], we can also assume that the addend of the elliptic curve additions is normalized according to [4]. In this case, elliptic curve addition $(X_2, Y_2, Z_2) \leftarrow (X_0, Y_0, Z_0) + (X_1, Y_1, 1)$ and elliptic curve doubling $(X_2, Y_2, Z_2) \leftarrow 2(X_0, Y_0, Z_0)$ are defined in Figure 1.

Addition formulae	Doubling formulae
$X_2 \leftarrow -H^3 - 2U_1H^2 + r^2$ $Y_2 \leftarrow -S_1H^3 + r(U_1H^2 - X_2)$ $Z_2 \leftarrow Z_0H$ where $U_1 = X_1{Z_0}^2$ $S_1 = Y_1{Z_0}^3$ $H = X_0 - U_1$ $r = S_1 - Y_0$	$X_2 \leftarrow T$ $Y_2 \leftarrow -8{Y_0}^4 + M(S - T)$ $Z_2 \leftarrow 2Y_0Z_0$ where $S = 4X_0{Y_0}^2$ $M = 3{X_0}^2 + a{Z_0}^4$ $T = -2S + M^2$

Fig. 1. Elliptic curve addition and doubling formulae over $\mathbb{F}_q$, $\mathrm{char}\,\mathbb{F}_q > 3$.

2.3 Elliptic Curve Arithmetic with Parallel Multiplication

Currently, more than two calculations can be performed in parallel using SIMD instructions. However, the following problems occur.

- Calculations that depend on the results of one of the other calculations cannot be computed simultaneously.
- Calculations of two elements in the same register require data place-adjustment operations. The cost cannot be ignored.

Because of these characteristics, it is difficult to implement a field multiplication using SIMD. On the other hand, we need to perform several field multiplications to establish an elliptic curve addition or doubling. If we can perform two or more independent field multiplications of an elliptic curve addition or doubling in parallel, these SIMD problems can be resolved because each parallel calculator of the SIMD can perform independent field multiplications.

Koyama and Tsuruoka [7] studied a parallel multiplication environment for elliptic curve arithmetic. However, they only estimated the number of parallel processes and did not show how to compute an elliptic curve arithmetic using multiple field multipliers in a practical situation. We carefully observed field multiplication calculations in elliptic curve arithmetic and found that we can compute field multiplication calculations in parallel without many multiplier stalls for an elliptic curve, whose elements are represented as a projective coordinate.

We propose an algorithm of the elliptic curve addition and doubling of Jacobian coordinates (Figure 1) using two field multipliers is presented in Figures 2

[1] Sometimes referred as the k-ary method for an exponentiation.

and 3. Note that field additions and subtractions are partially omitted. The asterisk ($*$) in the figures represents squaring.

We propose a modified Jacobian coordinate system that is suitable for parallel operations. In modified Jacobian coordinates, we represent a rational point on E by (X, Y, Z, Z^2) instead of (X, Y, Z) of the original Jacobian coordinates. Though there were similar modified representation of Jacobian coordinates to decrease the number of field multiplication for elliptic curve arithmetic [4], our viewpoint is new, because we can compute elliptic curve addition and doubling with the same number of field multiplications but with fewer steps in the modified Jacobian coordinates than in the original Jacobian coordinates. We also show an algorithm of the elliptic curve addition and doubling in the modified Jacobian coordinates using three field multipliers in Figures 4 and 5. Note that field additions and subtractions are partially omitted.

We can compute the elliptic curve addition and doubling in Figure 1 with 11 and 9 steps, respectively, by using one field multiplier. On the other hand, [7] shows the theoretical bound of the parallelization. To establish the elliptic curve addition and doubling, we need at least 3 steps even if we use 29 parallel field multipliers. Recently, Smart [8] proposed the parallel implementation of a special class of elliptic curves called "Hessian form," namely those with a point of order three. He showed that the elliptic curve addition and doubling derived from Hessian form can be computed with 4 and 3 steps, respectively, by using three parallel field multipliers.

Step	Multiplier #1	Multiplier #2
1	Z_0^2	$Y_1 \cdot Z_0$
2	$U_1 \leftarrow X_1 \cdot Z_0^2$	$S_1 \leftarrow Y_1 Z_0 \cdot Z_0^2$
3*	H^2	r^2
4	$H \cdot H^2$	$U_1 \cdot H^2$
5	$S_1 \cdot H^3$	$Z_2 \leftarrow Z_0 \cdot H$
6	$r \cdot (U_1 H^2 - X_2)$	—

Fig. 2. Parallel elliptic curve addition for Jacobian coordinate

Step	Multiplier #1	Multiplier #2
1*	Y_0^2	Z_0^2
2*	X_0^2	$(Z_0^2)^2$
3*	$(Y_0^2)^2$	M^2
4	$S \leftarrow 4X_0 \cdot Y_0^2$	$Z_2 \leftarrow 2Y_0 \cdot Z_0$
5	$M \cdot (S - T)$	—

Fig. 3. Parallel elliptic curve doubling for Jacobian coordinate

Thus, we believe that the proposed method is semi-optimal because Figures 2 and 3 compute elliptic curve addition and doubling with 6 and 5 steps, respectively, by using two parallel field multipliers and Figures 4 and 5 compute them with 4 and 3 steps, respectively, by using three parallel field multipliers. Though it can use a general elliptic curve parameter, it is the same as efficient as parallel algorithm for special elliptic curves [8]. Their number of field multiplication is the same as the case of one field multiplier and the probability is only 10% or less that a multiplier sleeps.

Step	Multiplier #1	Multiplier #2	Multiplier #3
1	$U_1 \leftarrow X_1 \cdot Z_0^2$	$Z_0 \cdot Z_0^2$	—
2	H^2	$S_1 \leftarrow Y_1 \cdot Z_0^3$	$Z_2 \leftarrow Z_0 \cdot H$
3	$U_1 \cdot H^2$	$H \cdot H^2$	r^2
4	$S_1 \cdot H^3$	$r \cdot (U_1 H^2 - X_2)$	Z_2^2

Fig. 4. Parallel elliptic curve addition for modified Jacobian coordinate

Step	Multiplier #1	Multiplier #2	Multiplier #3
1*	Y_0^2	X_0^2	$(Z_0^2)^2$
2	M^2	$S \leftarrow 4X_0 \cdot Y_0^2$	$Z_2 \leftarrow 2Y_0 \cdot Z_0$
3	$(Y_0^2)^2$	$M \cdot (S - T)$	Z_2^2

Fig. 5. Parallel elliptic curve doubling for modified Jacobian coordinate

3 Bitslice Implementation

Bitslice implementation is a technique that regards the w-bit CPU as a w-parallel 1 bit CPU, where w is the word size of the CPU. This is also a kind of SIMD. This technique has been applied to secret key cryptosystems [12,13].

But there has been no application to public key cryptosystem because the usual public key algorithm requires conditional branches. This complicates the bitslice implementation. This paper first applies the bitslice technique to public key cryptosystem and apply it to elliptic curve cryptosystems defined over $\mathbb{F}_{2^m}$. When independent w elliptic curve operations are simultaneously performed, each of the word bit is allocated to each curve. We can avoid conditional branches by introducing a new technique, which realizes a conditional move, for the bitslice implementation.

The organization of this section is as follows. Subsection 3.1 describes the data structure of the bitslice implementation. Subsection 3.2 describes the conditional move techniques. Subsections 3.3, 3.5, and 3.6 describe the performance in terms of efficiency.

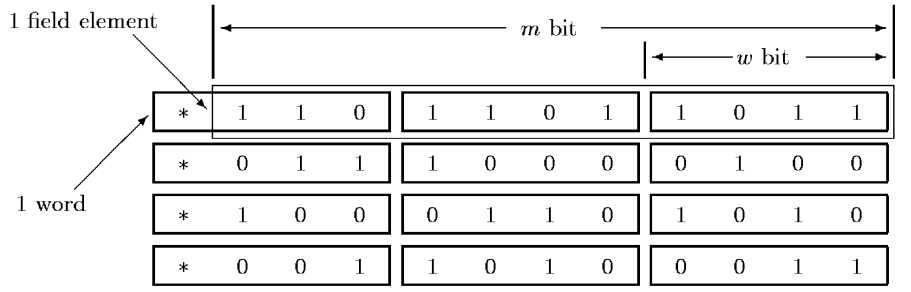

Fig. 6. Data representation of non-bitslice implementation, $(m = 11, w = 4)$

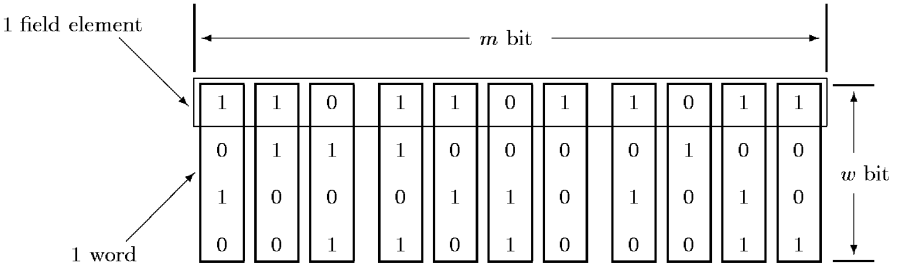

Fig. 7. Data representation of bitslice implementation, $(m = 11, w = 4)$

3.1 Data Structure

In this subsection, we describe how to represent $\mathbb{F}_{2^m}$ arithmetic by using the bitslice implementation.

We assume parallel processing in w, i.e., the word size of the processor, with independent elements over $\mathbb{F}_{2^m}$. Using a non-bitslice implementation, w elements in $\mathbb{F}_{2^m}$ are represented in Figure 6, where one row is divided into several words and one word is denoted by one rectangle. We represent them in Figure 7 when using the bitslice implementation. That is, we denote the i-th bit of the j-th element in $\mathbb{F}_{2^m}$ as $e_{(i,j)}$,

$$e_i = \sum_{j=0}^{w-1} e_{(i,j)} 2^j. \qquad (0 \leq i < m)$$

w elements in $\mathbb{F}_{2^m}$ can be represented by m word data $e_i (0 \leq i < m)$. The e_i is a bitslice representation.

The advantage of the bitslice implementation is that bitwise operation becomes easy, because each bit of data is contained in a different register. On the other side, bitslice implementation is inefficient if there are less data to be processed than w.

3.2 Conditional Move

It is difficult to realize a conditional branch when using the bitslice implementation, because one register has w independent data and each of w branches independently. For example, when we use a binary method to compute the elliptic curve scalar multiplication by k, we must establish a branch corresponding to each bit of k. In this subsection, the basic idea of the conditional move in the bitslice implementation is shown and the problem of a binary method without a conditional branch is solved.

Let the bold face characters denote the vector of w data, e.g., $\boldsymbol{A}$ denotes the vector of w rational points $(A(0), A(1), \ldots, A(w-1))$ and $\boldsymbol{k}$ denotes the vector of w independent scalars $(k(0), k(1), \ldots, k(w-1))$. We define the two-case conditional move $\texttt{cmov}_2(\boldsymbol{k}; \boldsymbol{A}, \boldsymbol{B})$ such that

$$\texttt{cmov}_2(\boldsymbol{k}; \boldsymbol{A}, \boldsymbol{B}) := \boldsymbol{T},$$
$$\text{where } T(i) = \begin{cases} A(i) & \text{if } k(i) = 1, \\ B(i) & \text{if } k(i) = 0, \end{cases}$$
$$\text{for } 0 \le i < w.$$

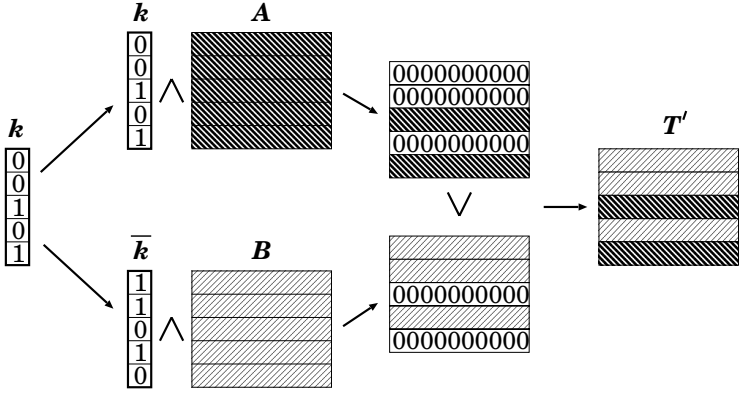

Fig. 8. Example of bitslice conditional move realization for bitslice implementation

The basic idea of the conditional move realization for bitslice implementation is shown in Figure 8. The figure shows that we can compute $\texttt{cmov}_2$ using the fact that

$$\begin{aligned} \boldsymbol{T}' &= (\boldsymbol{k} \wedge \boldsymbol{A}) \vee (\overline{\boldsymbol{k}} \wedge \boldsymbol{B}) \\ &= \texttt{cmov}_2(\boldsymbol{k}; \boldsymbol{A}, \boldsymbol{B}) \end{aligned}$$

where $\overline{\boldsymbol{k}}$ is a bitwise complementation of $\boldsymbol{k}$. $\boldsymbol{T}'$ is equivalent to output $\boldsymbol{T}$ of $\texttt{cmov}_2$ using conditional branches for each point.

The cost of $\texttt{cmov}_2$ is nearly equal to three field additions, which is negligible compared to an elliptic curve addition.

3.3 The Window Method for Bitslice Implementation

The conditional move enables us to use the window method that is faster than a binary method. Although we have more complicated conditional branches in the conventional window method, we can replace all of them using conditional moves for multiple elements.

We show the window method for bitslice implementation by using a four-case conditional move $\texttt{cmov}_4$ in Figure 9, where the window size equals 2. We can construct $\texttt{cmov}_4$ by using three $\texttt{cmov}_2$ as

$$\texttt{cmov}_4(\boldsymbol{k}_1, \boldsymbol{k}_0; \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{D}) := \texttt{cmov}_2(\boldsymbol{k}_1; \texttt{cmov}_2(\boldsymbol{k}_0; \boldsymbol{A}, \boldsymbol{B}), \texttt{cmov}_2(\boldsymbol{k}_0; \boldsymbol{C}, \boldsymbol{D})).$$

Since $\texttt{cmov}_s, (s > 2)$ can be constructed similarly for any s, we can choose any window size for the window method in the bitslice implementation. Note that $\boldsymbol{O}$ denotes $(\mathcal{O}, \mathcal{O}, \ldots, \mathcal{O})$ in Figure 9. [2]

Input : $\boldsymbol{P}$, $\boldsymbol{k}_j$, $\quad (0 \le j < t,\ t \text{ is even})$

Output : $\displaystyle\sum_{j=0}^{t-1} 2^j k_j(i) P(i), \quad (0 \le i < w)$

1. Compute $\boldsymbol{P}_2 \leftarrow 2\boldsymbol{P}$, $\boldsymbol{P}_3 \leftarrow 3\boldsymbol{P}$, $\boldsymbol{T} \leftarrow \boldsymbol{O}$.
2. For $j \leftarrow t-1$ down to 1 decremented by 2
   - 2.1 $\boldsymbol{T} \leftarrow 4\boldsymbol{T}$.
   - 2.2 $\boldsymbol{U} \leftarrow \texttt{cmov}_4(\boldsymbol{k}_j, \boldsymbol{k}_{j-1}; \boldsymbol{P}_3, \boldsymbol{P}_2, \boldsymbol{P}, \boldsymbol{O})$.
   - 2.3 $\boldsymbol{T} \leftarrow \boldsymbol{T} + \boldsymbol{U}$.
3. Return $\boldsymbol{T}$.

Fig. 9. Window method for bitslice implementation (window size = 2)

The numbers of computations for the non-bitslice implementation and bitslice implementation are shown in Table 1, where l is the size of window. A_b and A_n denote the cost of elliptic curve addition using the bitslice and non-bitslice implementations, respectively. Also D_b and D_n denote the cost of elliptic curve doubling using the bitslice and the non-bitslice implementation, respectively. We give the detailed evaluation of A_b and A_n in Appendix. We omit the detail about D_b and D_n becasuse we can compute scalar multiplication without elliptic curve doublings using the parameter K-163 [14].

We can assume a window size to be at least $l = 4$, in the conventional implementation of the window method for a secure curve. The difference between the implementation of a conditional branch and a conditional move

$$\frac{\left(\left(\frac{t}{l} + 2^l - 2\right) A_b + tD_b\right)}{\left(\left(\frac{(2^l-1)}{2^l}\frac{t}{l} + 2^l - 2\right) A_n + tD_n\right)} - 1$$

[2] We must use conditional branch for elliptic curve addition with $\mathcal{O}$. We can also use $\texttt{cmov}_2$ to solve the problem.

Table 1. Number of Computations for Bitslice and Non-bitslice Scalar Multiplications

	Bitslice (with conditional move)	Non-bitslice (with conditional branch)
Binary method	$w(tA_b + tD_b)$	$\frac{1}{2}tA_n + tD_n$
Window method	$w\left(\left(\frac{t}{l} + 2^l - 2\right)A_b + tD_b\right)$	$\left(\frac{(2^l-1)}{2^l}\frac{t}{l} + 2^l - 2\right)A_n + tD_n$

is only 6.7% at most, assuming usual situation as $0 \leq D_b \leq D_n$, $0 \leq A_b \leq A_n$ and $t > 0$.

3.4 Efficiency

In this section, we describe the performance in terms of efficiency, assuming the following.

- There are two typical representation of $\mathbb{F}_{2^m}$, namely in a normal basis and a polynomial basis. We select the polynomial basis because it is efficient if we use the Karatsuba algorithm [15, pp.294-302] to compute field multiplication.
- We can apply the Karatsuba algorithm recursively to both, the bitslice and the non-bitslice implementations until single precision multiplication. Because single precision is w-bit for the non-bitslice and 1-bit for the bitslice implementation, the depth of recursions of the Karatsuba algorithm differs between bitslice and non-bitslice implementations.
- We use Itoh-Tsujii inversion algorithm [16] for the bitslice implementation, and almost inverse algorithm [17] for the non-bitslice implementation.
- We assume that rational points are in $E(\mathbb{F}_{2^m})$, $m = 163$, and the CPU word size, w, is 64.
- We show the number of computations per scalar multiplication in Table 2, because the bitslice implementation independently computes the scalar multiplication w times at once. We assume $\boldsymbol{s} = \boldsymbol{x} = \boldsymbol{a} = 1$ cycle, where $\boldsymbol{a}, \boldsymbol{x}$ and $\boldsymbol{s}$ denote the costs of one word operations **AND**, **XOR** and **SHIFT**, respectively.
- We ignore the costs to convert the non-bitslice representation and bitslice representation because they are less compared to one field multiplication in $\mathbb{F}_{2^m}$.
- We use the NIST recommended elliptic curve parameter [14] over $\mathbb{F}_{2^m}$. Digits written in a courier font denote a hexadecimal number.

  [NIST-recommended elliptic curve K-163]
  $E : y^2 + xy = x^3 + ax^2 + b,\ \ a = 1, b = 1$
  The field polynomial: $p(t) = t^{163} + t^7 + t^6 + t^3 + 1$
  The base point order:
  $r =$ `4 00000000 00000000 00020108 a2e0cc0d 99f8a5ef`
  The base point $G = (G_x, G_y)$
  $G_x =$ `2 fe13c053 7bbc11ac aa07d793 de4e6d5e 5c94eee8`
  $G_y =$ `2 89070fb0 5d38ff58 321f2e80 0536d538 ccdaa3d9`

3.5 Comparison Based on an Ideal Model

In this subsection, we compare bitslice and non-bitslice implementations based on an ideal model. The numbers are based on the tables in Appendix.

Table 2. Number of Computations for One Elliptic Curve Addition ($w = 64, m = 163$)

Coordinate	Bitslice	Non-bitslice
Affine	4302	5245
Projective [5]	2690	16790

(cycles)

Table 2 shows that elliptic curve addition by the projective coordinate is faster when we use the bitslice implementation, and by the affine coordinate is faster when we use the non-bitslice implementation. In comparison with the bitslice implementation in the projective coordinate and the non-bitslice implementation in the affine coordinate, the bitslice one is about twice as fast as the non-bitslice one.

3.6 Comparison by Implementation

In this subsection, we compare the bitslice and the non-bitslice implementations by real implementation on a Pentium III PC. The specifications for the computer we used are

CPU : Pentium III 850MHz
Memory : 128MB

We show the results of the implementation in Table 3. It shows that the bitslice implementation is 4.4 times faster than non-bitslice.

Table 3. Time of Computations for an Elliptic Curve Scalar Multiplication

Coordinate	Bitslice (with conditional move)	Non-bitslice (with conditional branch)
Affine	552.9	1444
Projective [5]	326.8	1918

($m = 163$, μsec., Pentium III 850 MHz, Memory 128MB)

4 Conclusion

With the focus of processing many signatures or ciphertexts, we proposed two techniques, the "parallel elliptic curve algorithm" and use of "bitslice implementation," to accelerate elliptic curve arithmetic over prime and binary fields, respectively.

The parallel elliptic curve addition and doubling methods for prime field enable two or three parallel computations. By the technique, we can utilize SIMD architecture effectively. Their number of computation is the same as the case of one field multiplier and the probability is only 10% or less that a multiplier sleeps.

The bitslice implementation accelerates field arithmetic over a binary field. Though the usual public key algorithm requires conditional branches, we first applied the bitslice technique to public key cryptosystem, avoiding conditional branches by introducing a new technique, which realizes a "conditional move, for the bitslice implementation." The performance depends on the circumstances. Our implementation shows that the speed of the bitslice implementation is 4.4 times faster than the conventional non-bitslice implementation on a Pentium III.

References

1. Lenstra, A.K., Verheu, E.R.: Selecting cryptographic key sizes. In Imai, H., Zheng, Y., eds.: Public Key Cryptography — Third International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2000. Volume 1751 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (2000) 446–465
2. Miller, V.S.: Use of elliptic curves in cryptography. In Williams, H.C., ed.: Advances in Cryptology — CRYPTO'85. Volume 218 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1986) 417–426
3. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation **48** (1987) 203–209
4. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In Ohta, K., Pei, D., eds.: Advances in Cryptology — ASIACRYPT'98. Volume 1514 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1998) 51–65
5. Lopez, J., Dahab, R.: Improved algorithms for elliptic curve arithmetic in $gf(2^n)$. In Tavares, S., Meijer, H., eds.: Selected Areas in Cryptography — 5th Annual International Workshop, SAC'98. Volume 1556 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (1999) 210–212
6. Woodbury, A., Bailey, D., Paar, C.: Elliptic curve cryptography on smart cards without coprocessors. In: the Fourth Smart Card Research and Advanced Applications (CARDIS 2000) Conference. CADIS'2000, Bristol, UK (2000)
7. Koyama, K., Tsuruoka, Y.: Speeding up elliptic curve cryptosystems by using a signed binary windows method. In Brickell, E.F., ed.: Advances in Cryptology — CRYPTO'92. Volume 740 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1993) 345–357
8. Smart, N.P.: The hessian form of an elliptic curve. In: Preproceedings of Cryptographic Hardware and Embedded Systems. CHES2001 (2001) 121–128
9. Lipmaa, H.: Idea: A cipher for multimedia architectures? In Tavares, S., Meijer, H., eds.: Selected Areas in Cryptography — 5th Annual International Workshop, SAC'98. Volume 1556 of Lecture Notes in Computer Science., Berlin, Heidelberg, New York, Springer-Verlag (1999) 248–263
10. Dixon, B., Lenstra, A.K.: Factoring integers using simd sieves. In Helleseth, T., ed.: Advances in Cryptology — EUROCRYPT'93. Volume 765 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1993) 28–39

11. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography. CRC Press (1997)
12. Biham, E.: A fast new des implementation in sofware. In Biham, E., ed.: Fast Software Encryption — 4th International Workshop, FSE'97. Volume 1267 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1997) 260–272
13. Nakajima, J., Matsui, M.: Fast software implementations of misty1 on alpha processors. IEICE Transactions Fundamentals of Electronics, Communications and Computer Sciences (Japan) **E82-A** (1999) 107–116
14. NIST: Recommended elliptic curves for federal government use (1999) (available at `http://csrc.nist.gov/csrc/fedstandards.html`).
15. Knuth, D.E.: Seminumerical Algorithms. Third edn. Volume 2 of The Art of Computer Programming. Addison Wesley (1997)
16. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in $gf(2^m)$ using normal bases. In: Information and Computation. Volume 78. (1988) 171–177
17. Schroeppel, R., Orman, H., O'Malley, S., Sparscheck, O.: Fast key exchange with ellipic curve systems. In Coppersmith, D., ed.: Advances in Cryptology — CRYPTO'95. Volume 963 of Lecture Notes in Computer Science. Springer-Verlag, Berlin, Heidelberg, New York (1995) 43–56

Appendix

In this appendix we show in detail the number of computations for elliptic curve additions using the bitslice and non-bitslice implementations A_b and A_n in Tables 4 and 5.

M, S and I denote the cost of multiplication, squaring, and an inversion of elements in $\mathbb{F}_{2^m}$. We assume the base of log is 2.

Table 4. Number of Computations for Bitslice Implementation

Coordinate	Operation	Bitslice (per one element)
Affine		$2M + S + I$
	$\boldsymbol{s}$	0
	$\boldsymbol{x}$	$\frac{1}{3w}\{26 + 13 \cdot w_H(m-1)\}m^{\log 3} + \frac{1}{2w}tm$
	$\boldsymbol{a}$	$\frac{1}{9w}\{32 + 16 \cdot w_H(m-1)\}m^{\log 3}$
Projective [5]		$9M + 4S$
	$\boldsymbol{s}$	0
	$\boldsymbol{x}$	$\frac{1}{w}\{39m^{\log 3} + 2tm\}$
	$\boldsymbol{a}$	$\frac{16}{w}m^{\log 3}$

Table 5. Number of Computations for Non-bitslice Implementation

Coordinate	Operation	Non-bitslice
Affine		$2M + S + I$
	$\boldsymbol{s}$	$2w(\frac{m}{w})^{\log 3} + 2(2m + \log w)\left\lceil \frac{m}{w} \right\rceil$
	$\boldsymbol{x}$	$2(9w-2)(\frac{m}{w})^{\log 3} + 2(2m + \log w)\left\lceil \frac{m}{w} \right\rceil$
	$\boldsymbol{a}$	$2w(\frac{m}{w})^{\log 3} + 2\left\lceil \frac{m}{w} \right\rceil \log w$
Projective [5]		$9M + 4S$
	$\boldsymbol{s}$	$9w \cdot (\frac{m}{w})^{\log 3} + 8\left\lceil \frac{m}{w} \right\rceil \log w$
	$\boldsymbol{x}$	$9(9w-2)w \cdot (\frac{m}{w})^{\log 3} + 8\left\lceil \frac{m}{w} \right\rceil \log w$
	$\boldsymbol{a}$	$9w \cdot (\frac{m}{w})^{\log 3} + 8\left\lceil \frac{m}{w} \right\rceil \log w$