Fast Multi-scalar Multiplication Methods on Elliptic Curves with Precomputation Strategy Using Montgomery Trick

Katsuyuki Okeya[1] and Kouichi Sakurai[2]

[1] Hitachi, Ltd., Systems Development Laboratory,
292, Yoshida-cho, Totsuka-ku, Yokohama, 244-0817, Japan
ka-okeya@sdl.hitachi.co.jp
[2] Kyushu University,
Graduate School of Information Science and Electrical Engineering, 6-10-1, Hakozaki,
Higashi-ku, Fukuoka, 812-8581, Japan
sakurai@csce.kyushu-u.ac.jp

Abstract. Our development of efficient methods for the precomputation of multi-scalar multiplication for elliptic curve cryptosystems (ECCs) is presented. Multi-scalar multiplication is required in many forms of ECC, including schemes for the verification of ECDSA signatures. The simultaneous method is one known method for fast multi-scalar multiplication. The method has two stages: a precomputation stage and an evaluation stage. Points for use in the evaluation stage are computed in the precomputation stage. The actual multi-scalar multiplication is carried out on the basis of the precomputed points in the evaluation stage. In the evaluation stage of the simultaneous method, we are able to quickly compute the points of the multi-scalar multiple because few additions are required. On the other hand, if we use a large window width, we have to compute an enormous number of points in the precomputation stage. Hence, we have to compute an abundance of inversions, which carries a high computational cost. The result is that a large amount of time is required by the precomputation stage. This is the well-known draw-back of the simultaneous method. In our proposed method, we apply the Montgomery trick to reduce the number of inversions required with a width window w from $O(2^{2w})$ to $O(w)$. In addition, our proposed method computes uP and vQ for any u, v, then compute $uP + vQ$, where P, Q are elliptic points. This procedure enables us to remove points that will not be used later from the process of precomputation. Without our proposed method, an algorithm to compute precomputation table would have to be changed dependently on unused points. Compared with the method without Montgomery trick, our proposed method is 3.6 times faster than the conventional simultaneous method, i.e., than in the absence of the Montgomery trick. Moreover, the optimal window width for our proposed method is 3, whereas the corresponding width for conventional simultaneous methods is 2.

Keywords: *elliptic curve cryptography, multi-scalar multiplication, precomputation, Montgomery trick, simultaneous*

1 Introduction

During the world-wide deployment of the electronic-signature law, the infrastructure for digital signatures is spreading. To support the expected rapid growth in the scale of electronic commerce, efficient implementation of public key cryptosystems such as digital signatures is becoming more and more important. Several digital signature schemes have been developed, studied, and to some extent, applied; these include RSA [RSA78] and DSA [DSA] schemes. The ECDSA signature scheme [ANSI,IEEEp1363,SEC-1], which is based on elliptic curve cryptography [Kob87,Mil86], is particularly noteworthy, because it provides high levels of security with short keys. In this article, we will propose an elliptic multi-scalar multiplication method that includes an efficient form of precomputation. This method provides a faster way to carry out the multi-scalar multiplication that is required in such elliptic curve cryptosystems as the signature verification procedure of an ECDSA signature scheme.

1.1 Elliptic Curve Operations

Some elliptic curve cryptosystems such as signature generation of ECDSA signature scheme need operations of scalar multiplication. In many cases, scalar multiplication is dominant in the overall time taken in computation. Several methods for the fast computation of scalar multiplication have been proposed; these include methods base on the use of more efficient coordinate systems (such as projective coordinates [CC87] and Jacobian coordinates [CC87,CMO98]), on the use of precomputation tables (such as the window method [Knu81] and comb method [LL94]), on the subtraction of points (such as NAF [MO90]), and on the non-use of the y-coordinate (such as the Montgomery-form elliptic curve [Mon87]), and others.

On the other hand, verifying an ECDSA signature requires an operation in which the point $kP+lQ$ is computed from the elliptic points P, Q and the integers k, l. This operation is referred to as multi-scalar multiplication. While two rounds of scalar multiplication are conventionally used to obtain kP and lQ, certain methods operate by the direct computation of the multi-scalar multiple, i.e., by the simultaneous computation of $kP + lQ$. The simultaneous method [Elg85, HHM00,BHLM01] and interleaving exponentiation [Möl01] are two examples of methods that operate in this way.

Multi-scalar multiplication has other applications. A method of scalar multiplication in which endomorphisms [GLV01] are applied for faster computation has recently been proposed. This method requires multi-scalar multiplication. Thus, fast methods of multi-scalar multiplication are desirable for use in elliptic curve cryptosystems.

1.2 Our Contributions

We propose a simultaneous method of multi-scalar multiplication in which Montgomery trick [Coh93] is applied to obtain an efficient form of precomputation.

A simultaneous method for calculating multi-scalar multiple consists of two stages: a stage of precomputation and a stage of evaluation [Möl01]. Points for use in multi-scalar multiplication in the evaluation stage are computed in the precomputation stage. The additions calculated in the precomputation stage are in affine coordinates, since the points have to be in this form for the fast computation of elliptic addition in the evaluation stage. However, each addition of points in affine coordinates requires a finite field operation of inversion, and inversion carries a high computational cost. In particular, a given increase in the number of points to be precomputed leads to a much greater increase in the computational cost of precomputation.

The main contribution of our method is that Montgomery trick is used in the precomputation of additions of points in affine coordinates. Montgomery trick [Coh93] provides a way of inverting n elements of a finite field with a single inversion operation and $3(n-1)$ multiplications rather than with n inversion operations. A further advantage of Montgomery trick is that no more memory is consumed in computation than with the straightforward method. Montgomery trick thus reduces the number of inversion operations required in plural additions of points in affine coordinates, while taking up no more memory. This reduction allows us to compute $O(w)$ inversions instead of $O(2^{2w})$ inversions for a window width w. As a result, precomputation according to our proposed method is 3.6 times faster than with an otherwise equivalent method in which Montgomery trick is not applied.

Another known example of the use of Montgomery trick in fast computation is in precomputation for elliptic scalar multiplication by the window method [CMO98]. While uP is computed in precomputation for scalar multiplication, $uP+vQ$ is precomputed in multi-scalar multiplication, so the relation between the procedures of computation is complicated. This is particularly, in cases where some pairs (u,v) need not be computed and in the case of NAF (Non Adjacent Form) pairs $(\pm u, \pm v)$. In our proposed method, uP and vQ are computed first, and this is followed by the computation of $uP+vQ$. Following this procedure simplifies the relation between the procedures of computation.

This simplification adds a further improvement to that which we obtain by using Montgomery trick. As the window width is increased, increasingly large numbers of precomputed points go unused in the evaluation stage. We eliminate the computation of such points in the precomputation stage. The simplification allows us to discard such points without modifying the procedure of computation for the remainder of the points of precomputation. Without our proposed method, the algorithm for computing the precomputation table would have to be changed according to the unused points. Eliminating the computation of the unused points achieves further speedup and further reduces the consumption of memory. The respective effects are estimated as a 20% speedup and a reduction of about ten points in the precomputation stage for the simultaneous sliding window NAF method with a window width w of 3 and 160-bit scalars. Moreover, the optimum window width for our proposed method is 3, whereas the corresponding width for conventional simultaneous methods is 2.

As well as the simultaneous sliding window NAF method, our proposed method is adaptable to simultaneous methods in general, including the NAF method [Aki01], the interleaving exponentiation method [Möl01], and so on.

The remainder of this article is organized as follows: Section 2 explains multi-scalar multiplication. Section 3 is a review of conventional simultaneous methods of scalar multiplication. Section 4 outlines the efficient method of precomputation in which Montgomery trick is applied. Section 5 gives a comparison of the method and conventional methods in terms of computational cost. We confirm that the proposed method is faster than an otherwise equivalent method in which Montgomery trick is not applied.

2 Multi-scalar Multiplication

Let P and Q be elliptic points, and k and l be integers. In multi-scalar multiplication, an elliptic point $kP+lQ$ is computed from points P and Q and integers k and l. Multi-scalar multiplication is widely used in such elliptic curve cryptosystems as the procedure for signature verification in the ECDSA signature [ANSI, IEEEp1363,SEC-1], and EC-MQV [IEEEp1363,SEC-1] schemes. Furthermore, multi-scalar multiplication has other applications. Recently, a method of scalar multiplication [GLV01] in which endomorphisms are used for fast computation has been proposed. This method of scalar multiplication involves multi-scalar multiplication. In most cases where multi-scalar multiplication is applied, the process is dominant in determining the overall computational cost.

Methods for the computation of multi-scalar multiples can be classified into two types. In methods of one type, independent computation of the scalar multiples kP and lQ is followed by addition $(kP)+(lQ)$. In methods of the other type, the multi-scalar multiple $kP + lQ$ is computed in one stage, without separate computation of kP and lQ. An example of the former type is a method in which kP is computed by a comb method [LL94] and lQ is computed by a window method; these steps are followed by computation of $kP+lQ$. This approach has been applied in the signature verification process of the ECDSA, where we can assume that the point P is fixed; we can thus use a fixed-base comb method to compute kP and a window method to compute lQ, then compute the addition of the two. Examples of the latter type are examples of simultaneous methods. This article is mainly concerned with such examples.

3 Simultaneous Scalar Multiplication

A simultaneous method for scalar multiplication has two stages; one of precomputation and the other of evaluation. All of the elliptic points which will be required in the evaluation stage are computed and stored in a table during the precomputation stage. In the evaluation stage, the multi-scalar multiplied point is computed by using the table which was prepared in the precomputation stage.

Concrete algorithms for the precomputation stage of conventional simultaneous methods of scalar multiplication are often omitted; for example, this step will be given as "Compute points of precomputation". In the remainder of this section, we describe some actual algorithms for the precomputation stage and review simultaneous methods of scalar multiplication.

Hereafter, we assume that p is a prime and $\mathbf{F}_p$ is the prime field of characteristic p, and we limit our discussion to those elliptic curves which are defined over the finite field $\mathbf{F}_p$. The argument applies to all elliptic curves defined over finite fields of characteristic 2 and those over optimal extension fields (OEF) [BP98]. In that case, while our proposed method may be fast, it is unfortunately, not particularly fast. This is because there are fast methods for inversion in such fields [HHM00,BP99]. For further details on elliptic curve cryptography, see [BSS99,Eng99,Men93,Sil86].

We choose the simultaneous sliding window NAF method from among the simultaneous methods, and now explain this method. However, the method we propose below is adaptable to the other simultaneous methods. The NAF and sliding window tricks are applied in the simultaneous sliding window NAF method. The original simultaneous method is known as Shamir's trick [Elg85, HHM00,BHLM01]. We assume that $w \geq 2$ applies to the window width w in the simultaneous sliding window NAF method.

Precomputation Stage. In the precomputation stage of the simultaneous sliding window NAF method, we compute elliptic points $uP + vQ$ for all $u, v \in [-f(w), f(w)]$ such that $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2$, where $f(w)$ is the integer which is given by $f(w) = \frac{2^{w+2}-(-1)^w-3}{6}$. Here, the condition $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2$ is needed because the least significant bits of u and v which are equal to 0 are not used in the evaluation stage when we apply the sliding window technique.

The relation $-R = (x, -y)$ holds for any elliptic point $R = (x, y)$. Thus, once we have computed $uP + vQ$ and $-uP + vQ$, we are easily able to get the points $-uP - vQ$ and $uP - vQ$ without significantly adding to computational costs, by simply substituting $-y$ for the y-coordinate y.

The formulae for the addition of points in affine coordinates are as follows: $x_3 = (\frac{y_2-y_1}{x_2-x_1})^2 - x_1 - x_2$, $y_3 = (\frac{y_2-y_1}{x_2-x_1})(x_1 - x_3) - y_1$, where $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$ and $P_3 = P_1 + P_2$. Hence, the addition formulae require an inversion, $(x_2 - x_1)^{-1}$. Since the points uP and $-uP$ have the same x-coordinates, the inverse of the addition $uP + vQ$ is the same as that of $-uP + vQ$.

Algorithm 1 : The precomputation stage in the simultaneous sliding window NAF method
INPUT : Elliptic points P and Q and the window width w
OUTPUT : The precomputation table $\{uP + vQ | u, v \in [-f(w), f(w)]$ s.t. $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$

```
1. for $u = 2$ to $f(w)$ do
   1.1. $uP \leftarrow (u-1)P + P$.
2. for $u = 1$ to $f(w)$ do
   2.1. $-uP \leftarrow -(uP)$.
3. for $v = 2$ to $f(w)$ do
```

```
   3.1. $vQ \leftarrow (v-1)Q + Q$.
4. $PreComp = \{(u,v) \in [1, f(w)] \times [1, f(w)] | u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$.
5. for any $(u,v) \in PreComp$ do
   5.1. $uP + vQ \leftarrow (uP) + (vQ)$, $-uP + vQ \leftarrow (-uP) + (vQ)$.
6. for $v = 1$ to $f(w)$ do
   6.1. $-vQ \leftarrow -(vQ)$.
7. for any $(u,v) \in PreComp$ do
   7.1. $-uP - vQ \leftarrow -(uP + vQ)$, $uP - vQ \leftarrow -(-uP + vQ)$.
```

Evaluation Stage. In the evaluation stage, we use the table prepared in the stage to compute the multi-scalar multiple $kP + lQ$.

Algorithm 2 : The evaluation stage in the simultaneous sliding window NAF method

INPUT : t-bit integers k and l, elliptic points P and Q, the window width w, and a precomputation table $\{uP + vQ | u, v \in [-f(w), f(w)]$ s.t. $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$

OUTPUT : The multi-scalar multiple $kP + lQ$

```
1. Write $k = (k_{t-1}, k_{t-2}, \cdots, k_0)$ and $l = (l_{t-1}, l_{t-2}, \cdots, l_0)$, where each $k_i$ and $l_i$ is an NAF bit.
2. $R \leftarrow \mathcal{O}$, $i \leftarrow t-1$
3. while $i \geq 0$ do
   3.1. if $k_i = 0$, $l_i = 0$ then $R \leftarrow 2R$, $i \leftarrow i-1$
        else do
      3.1.1 $j \leftarrow \max\{i - w + 1, 0\}$.
      3.1.2 while $k_j = 0$, $l_j = 0$ do
         3.1.2.1 $j \leftarrow j+1$.
      3.1.3 $k' \leftarrow (k_i, k_{i-1}, \cdots, k_j)$, $l' \leftarrow (l_i, l_{i-1}, \cdots, l_j)$.
      3.1.4 $R \leftarrow 2^{i-j+1}R + (k'P + l'Q)$.
      3.1.5 $i \leftarrow j-1$.
4. Output $R$.
```

For fast computation, the precomputation stage uses the addition formulae of $A \leftarrow A + A$ and the doubling formulae of $A \leftarrow A$, and the evaluation stage uses the addition formulae of $J^m \leftarrow J + A$, the doubling formulae of $J \leftarrow J^m$ for doubling prior to addition, and the doubling formulae of $J^m \leftarrow J^m$ for doubling prior to doubling, where A, J and J^m indicate affine coordinates, Jacobian coordinates, and modified Jacobian coordinates [CMO98], respectively.

Other examples of simultaneous methods of scalar multiplication are the interleaving exponentiation method [Möl01] and the method of simultaneous scalar multiplication [Aki01] on a Montgomery-form elliptic curve [Mon87].

4 Proposed Precomputation Stage

We explain our proposed method in this section, that is, the method of computation for use in the precomputation stage in which Montgomery trick is applied to obtain efficiency. Our proposed method reduces the number of inversions required and thus provides a quick way of preparing the precomputation table.

Given a window width of w, precomputation for a simultaneous method requires the computation of $O(2^{2w})$. Thus, a large window width requires the computation of an enormous number of points. Moreover, the computation of points in affine coordinates[1] requires the computation of inversion, which carries a high computational cost. Therefore, we have to reduce the number of inversion operations to obtain faster computation in the precomputation stage.

For faster computation in our proposed method, Montgomery trick is applied in computing plural inverses with a single inversion operation rather than with plural inversion operations. Montgomery trick has been applied for fast computation in, for example, precomputation for elliptic scalar multiplication with the window method [CMO98]. While precomputation for scalar multiplication is required to provide $\{uP|u \in [0, 2^w - 1]\}$, precomputation for multi-scalar multiplication is required to provide $\{uP + vQ|u, v \in [0, 2^w - 1]\}$. In the case of scalar multiplication, u is the only variable we need to consider, so we compute $\{2P\}, \{3P, 4P\}, \{5P, 6P, 7P, 8P\}, \cdots$. However, we have two variables in the case of multi-scalar multiplication, namely u and v, so the relation between procedures of computation is complicated. This is particular so in cases where some pairs (u, v) need not be computed and in cases of NAF pairs $(\pm u, \pm v)$. This is because the flow of the algorithm used to compute the precomputation table may have to change according to the presence of such unused points. In our proposed method, $\{uP|u \in [0, 2^w - 1]\}$ and $\{vQ|v \in [0, 2^w - 1]\}$ are computed first, and these are followed by computation of $\{uP + vQ|(u, v) \in [1, 2^w - 1] \times [1, 2^w - 1]\}$. This procedure simplifies the relation between procedures of computation.

In the rest of this section, after reviewing Montgomery trick, we go on to describe our proposed method of precomputation in which we apply the trick.

4.1 Montgomery Trick

Given n elements $a_1, a_2, \cdots, a_n$ from a finite field $\mathbf{F}_p$, Montgomery trick[2] may be used to compute their inverses $b_1, b_2, \cdots, b_n$ in the following way [Coh93]:

Algorithm 3 : Montgomery trick
INPUT : $a_1, a_2, \cdots, a_n$
OUTPUT : Inverses $b_1, b_2, \cdots, b_n$ of $a_1, a_2, \cdots, a_n$

[1] If the precomputation table is not represented in affine coordinates, computation of the multi-scalar multiples in the evaluation stage will be slow.

[2] The algorithm of Montgomery trick which is given in [Coh93] is for integer factorization. We have modified algorithm 3 for use as the precomputation stage of simultaneous scalar multiplication by removing the part that carries out integer factorization.

```
1. $c_1 \leftarrow a_1$.
2. for $i = 2$ to $n$ do
   2.1. $c_i \leftarrow c_{i-1}a_i$.
3. $u \leftarrow c_n^{-1}$.
4. for $i = n$ down to 2 do
   4.1. $b_i \leftarrow c_{i-1}u$.
   4.2. $u \leftarrow ua_i$.
5. $b_1 \leftarrow u$.
```

The computational cost of Montgomery trick is $3(n-1)M+I$, where M and I respectively denote operations of multiplication and inversion in $\mathbf{F}_p$.

Lemma 1. *For n elements, we apply Montgomery trick m times, and compute n inverses. The computational cost of this is then $3(n-m)M+mI$.*

Proof. Assume that we separate n into $n_1+n_2+\cdots+n_m$. The computational cost is then

$$\begin{aligned}\sum_{j=1}^{m}(3(n_j-1)M+I) &= \left(3\sum_{j=1}^{m} n_jM\right) - 3mM + mI \\ &= 3(n-m)M+mI.\end{aligned}$$

□

Remark 1. Lemma 1 shows that dividing n elements into m groups and applying Montgomery trick to each group leads to a computational cost of $3(n-m)M+mI$, which is independent of the division into m groups. This implies that the computational cost of the precomputation stage is solely dependent on number of times Montgomery trick is applied in the precomputation stage.

4.2 Simultaneous Sliding Window NAF Method

We describe a fast algorithm, in which Montgomery trick is applied, for the precomputation stage of the simultaneous sliding window NAF method. For a point R, x_R and y_R respectively denote the x- and y-coordinates.

Algorithm 4 : Applying Montgomery trick in a precomputation stage for the simultaneous sliding window NAF method
INPUT : Elliptic points P and Q and window width w
OUTPUT : The precomputation table $\{uP+vQ \mid u,v \in [-f(w),f(w)]$ s.t. $u \neq 0 \pmod 2$ or $v \neq 0 \pmod 2\}$

```
1. for $i = 1$ to $w-1$ do
   1.1. Compute points $2^{i-1}P + jP, 2^{i-1}Q + jQ$ for any $j \in [1, 2^{i-1}]$:
      1.1.1 Use Montgomery trick to compute inverses of $(x_{jP} - x_{2^{i-1}P})$, $(x_{jQ} - x_{2^{i-1}Q})$, $(2y_{2^{i-1}P})$ and $(2y_{2^{i-1}Q})$ for any $j \in [1, 2^{i-1}-1]$.
```

```
    1.1.2 Compute points $(2^{i-1}+j)P, (2^{i-1}+j)Q$ using $(x_{jP}-x_{2^{i-1}P})^{-1}$,
          $(x_{jQ}-x_{2^{i-1}Q})^{-1}$, $(2y_{2^{i-1}P})^{-1}$ and $(2y_{2^{i-1}Q})^{-1}$.
2. Compute points $2^{w-1}P+jP, 2^{w-1}Q+jQ$ for any $j \in [1, f(w)-2^{w-1}]$:
  2.1. Use Montgomery trick to compute inverses of $(x_{jP}-x_{2^{w-1}P}), (x_{jQ}-x_{2^{w-1}Q})$ for any $j \in [1, f(w)-2^{w-1}]$.
  2.2. Compute points $(2^{w-1}+j)P, (2^{w-1}+j)Q$ using $(x_{jP}-x_{2^{w-1}P})^{-1}, (x_{jQ}-x_{2^{w-1}Q})^{-1}$.
3. for $u=1$ to $f(w)$ do
  3.1. $-uP \leftarrow -(uP)$.
4. $PreComp = \{(u,v) \in [1,f(w)] \times [1,f(w)] | u \neq 0 \pmod 2 \text{ or } v \neq 0 \pmod 2\}$.
5. Compute points $uP+vQ, -uP+vQ$ for any $(u,v) \in PreComp$:
  5.1. Use Montgomery trick to compute inverses of $(x_{vQ}-x_{uP})$ for any $(u,v) \in PreComp$.
  5.2. Compute points $uP+vQ$,$-uP+vQ$ using $(x_{vQ}-x_{uP})^{-1}$.
6. for $v=1$ to $f(w)$ do
  6.1. $-vQ \leftarrow -(vQ)$.
7. for any $(u,v) \in PreComp$ do
  7.1. $-uP-vQ \leftarrow -(uP+vQ)$, $uP-vQ \leftarrow -(-uP+vQ)$.
```

Remark 2. The use of a coordinate system other than affine coordinates leads to greatly increased computational costs. This is because the computational cost of the precomputation stage with affine coordinates is $5M+S$ per point due to the application of Montgomery trick, whereas the equivalent cost with Jacobian coordinates ($J \leftarrow J+A$) is $8M+3S$ per point; the former approach thus provides a faster way to compute elliptic addition, where S denotes an operation of squaring in $\mathbf{F}_p$.

4.3 Further Improvement: Reducing the Number of Points in Precomputation

We now propose a further improvement to the precomputation stage. As was mentioned above, while a larger window width requires the computation of larger numbers of points in precomputation, it also leads to fewer additions in the evaluation stage. Hence, a relatively large window width requires the precomputation of points which are not actually used in the evaluation stage. For example, in the case of the simultaneous method with a window width w of 3, while 63 points are precomputed, only 46 additions are computed, on average in the evaluation stage for 160-bit scalars. Thus, about 17 points are unnecessarily precomputed. Therefore, we thus use a trick to avoid the precomputation of points that will not actually be used. This provides us with a further increase in speed along with reduced memory consumption, since fewer points have to be stored. For the simultaneous sliding window NAF method with a window width w of 3 and 160-bit scalars, the effects are estimated as a 20% speedup and the elimination of about ten points in the precomputation stage.

Remark 3. This improvement is only achieved within the framework of our proposed method. For example, improvement in this way is not applicable to the following method. $P, Q \to 2P, P+Q, 2Q \to 3P, 4P, 2P+Q, 3P+Q, P+2Q, 2P+2Q, 3Q, P+3Q, 4Q \to \cdots$. If $P+Q$ is an unused point which is eliminated from the computation, then $3P+Q$ is not computable in the third phase, since $(P+Q)+2P$ is the only available way to compute $3P+Q$. Thus, unused points require modification of the algorithm.

Next, we show a concrete algorithm which determines precomputation points that are used in the evaluation stage.

Algorithm : An algorithm which determines precomputation points that are used in the evaluation stage of the simultaneous sliding window NAF method
INPUT : t-bit integers k, l, a window width w.
OUTPUT : Pairs of integers (k', l') that are used in the evaluation stage.

```
1. Write $k = (k_{t-1}, k_{t-2}, \cdots, k_0)$ and $l = (l_{t-1}, l_{t-2}, \cdots, l_0)$, where each $k_i$
   and $l_i$ is an NAF bit.
2. $i \leftarrow t-1$, $num \leftarrow 0$.
3. while $i \geq 0$ do
  3.1. if $k_i = 0$, $l_i = 0$ then $i \leftarrow i-1$
       else do
     3.1.1 $j \leftarrow \max\{i-w+1, 0\}$.
     3.1.2 while $k_j = 0$, $l_j = 0$ do
      3.1.2.1 $j \leftarrow j+1$.
     3.1.3 $k' \leftarrow (k_i, k_{i-1}, \cdots, k_j)$, $l' \leftarrow (l_i, l_{i-1}, \cdots, l_j)$.
     3.1.4 for $j = 1$ to $num$
      3.1.4.1 If $T[j] = (k', l')$ then go to Step 3.1.6.
     3.1.5 $num \leftarrow num+1$, $T[num] \leftarrow (k', l')$.
     3.1.6 $i \leftarrow j-1$.
4. Output $T$.
```

Algorithm 4, the proposed precomputation stage, computes points $u_jP + v_jQ$ for $T[j] = (u_j, v_j)$ for $j = 1, \cdots, num$ in Step 5.

5 Computational Cost and Comparison

5.1 Precomputation Stage

Here, we start by estimating the computational cost of Algorithm 1, which is the conventional method of precomputation. Assume that $w > 1$. Doubling is the operation at both Steps 1.1 and 3.1 if u or v is equal to 2. If not, both steps are additions. The two elliptic additions in Step 5.1 only require a single 1 inversion, because they have a common inverse. The number of iterations of Step 5 is $\#PreComp = f(w)^2 - \left(\lfloor \frac{f(w)}{2} \rfloor\right)^2$. That is, $(2\#PreComp + 2f(w) - 4)$ elliptic additions and 2 elliptic doublings are computed by Algorithm 1. Meanwhile, the number of inversions is reduced by $(\#PreComp)$. Thus, the computational cost is

$$\begin{aligned}&(2\#PreComp+2f(w)-4)\,(2M+S+I)+2(2M+2S+I)-(\#PreComp)\,I\\ =\ &(4\#PreComp+4f(w)-4)\,M+(2\#PreComp+2f(w))\,S\\ &+(\#PreComp+2f(w)-2)\,I.\end{aligned}$$

In the case of $w=1$, the computational cost is $4M+2S+I$.

Next, we estimate the computational cost of Algorithm 4, our proposed method. Assume that $w>2$. In our proposed method, $(2\#PreComp+2f(w)-2w)$ elliptic additions and $(2w-2)$ elliptic doublings are computed. Meanwhile, the number of inversions is reduced by $(\#PreComp)$ and Montgomery trick is applied $(w+1)$ times in computing $(\#PreComp+2f(w)-2)$ inverses. Using Lemma 1, we obtain the following results for computational cost.

$$\begin{aligned}&(2\#PreComp+2f(w)-2w)(2M+S+I)\\ &+(2w-2)(2M+2S+I)-(\#PreComp)I\\ =\ &(4\#PreComp+4f(w)-4)M+(2\#PreComp+2f(w)+2w-4)S\\ &+(\#PreComp+2f(w)-2)I\\ \to\ &(4\#PreComp+4f(w)-4)M+(2\#PreComp+2f(w)+2w-4)S\\ &+3\,(\#PreComp+2f(w)-2-(w+1))\,M+(w+1)I\\ =\ &(7\#PreComp+10f(w)-3w-13)M\\ &+(2\#PreComp+2f(w)+2w-4)S+(w+1)I.\end{aligned}$$

Here, $\#PreComp=f(w)^2-\left(\lfloor\frac{f(w)}{2}\rfloor\right)^2$. In the case of $w=2$, the computational cost is $25M+10S+2I$.

5.2 Evaluation Stage

$\mathcal{AD}, \mathcal{DA}$ and $\mathcal{DD}$ respectively denote addition prior to doubling, doubling prior to addition and doubling prior to doubling in the evaluation stage. We use the following coordinate systems for fast computation: $\mathcal{AD}: J+A\to J^m$ $(9M+5S)$, $\mathcal{DA}: J^m\to J$ $(3M+4S)$, $\mathcal{DD}: J^m\to J^m$ $(4M+4S)$.

The white space of $(0,0)$ between two consecutive windows has expected length of $\frac{4/9}{1-(4/9)}(=0.8)$, since the probability that a NAF bit is equal to 0 is $\frac{2}{3}$. As a result, an average of $\frac{t-w}{w+0.8}$ additions have to be computed in the multi-scalar multiplication. Thus, the computational cost[3]is

$$\frac{t-w}{w+0.8}(\mathcal{AD}+\mathcal{DA})+\left(t-w-\frac{t-w}{w+0.8}\right)\mathcal{DD}$$

[3] Since 0 and non-zero bits are not randomly distributed, we need an estimate of the number of computations in which the bit dependence is considered. The smaller the window width, the larger the error in the estimate. In the case of $w=2$, the computational cost for a 160-bit scalar is about $1869.5M$.

$$= \frac{t-w}{w+0.8}(\mathcal{A}D + DA + (w-0.2)\mathcal{D}D)$$
$$= \frac{t-w}{w+0.8}((4w+11.2)M + (4w+8.2)S).$$

Table 1. Computational cost for point multiplication $kP + lQ$

Method	Precomputation	Evaluation	Total	Points
	Proposed method			Stored
Simultaneous $(w = 1)$	$2.8M + I$	$2575.8M$	$2608.6M$	3
	$2.8M + I$		$2608.6M$	
Simultaneous $(w = 2)$	$38.0M + 13I$	$2039.2M$	$2467.2M$	15
	$68.0M + 3I$		$2197.2M$	
Simultaneous $(w = 3)$	$172.4M + 61I$	$1770.9M$	$3773.3M$	63
	$345.0M + 4I$		$2235.9M$	
Simultaneous NAF $(w = 1)$	$5.6M + I$	$2204.8M$	$2240.4M$	8
	$5.6M + I$		$2240.4M$	
Simultaneous NAF $(w = 2)$	$29.6M + 6I$	$1910.4M$	$2120.0M$	24
	$41.6M + 2I$		$2012.0M$	
Simultaneous NAF $(w = 3)$	$164.0M + 33I$	$1725.0M$	$2879.0M$	120
	$252.6M + 4I$		$2097.6M$	
Simul. slid. window NAF$(w = 2)$	$24.0M + 5I$	$1869.5M$	$2043.5M$	16
	$33.0M + 2I$		$1962.5M$	
Simul. slid. window NAF$(w = 3)$	$141.6M + 29I$	$1626.2M$	$2637.8M$	96
	$159.2M + 4I$		$1905.4M$	(33.6)
Interleaving $(w = 4)$	$35.2M + 12I$	$1740.3M$	$2135.5M$	20
	$52.4M + 4I$		$1912.7M$	
Interleaving $(w = 5)$	$68.8M + 24I$	$1642.4M$	$2431.2M$	44
	$119.0M + 5I$		$1911.4M$	

"Precomputation", "Proposed method" and "Evaluation" indicate the computational cost of the conventional precomputation method, our proposed method of precomputation, and the evaluation stage, respectively "Total" means the overall computational cost of the precomputation and evaluation stages. "Points Stored" means the number of points stored in the precomputed table. This number determines the consumption of memory. We assume that $t = 160$, $S = 0.8M$, and $I = 30M$, where t is the number of bits in the input integers k and l.

5.3 Comparison

Firstly, we compare the computational cost of the respective precomputation stages. The computational cost of Algorithm 1 is $16M + 10S + 5I$ for $w = 2$ and $100M + 52S + 29I$ for $w = 3$, while the corresponding figures for Algorithm 4, our proposed method, are $25M + 10S + 2I$ and $175M + 54S + 4I$[4], respectively.

[4] When we take the elimination of unused points into consideration, this computational cost is about $159.2M + 4I$ and the number of the points stored in the precomputation table is about 33.6.

Table 2. Computational cost for point multiplication $kP + lQ$, with P fixed

Method	Precomputation / Proposed method	Evaluation	Total
Simultaneous ($w = 1$)	$2.8M + I$	$2575.8M$	$2608.6M$
	$2.8M + I$		$2608.6M$
Simultaneous ($w = 2$)	$31.6M + 11I$	$2039.2M$	$2400.8M$
	$58.6M + 2I$		$2157.8M$
Simultaneous ($w = 3$)	$155.0M + 55I$	$1770.9M$	$3575.9M$
	$311.6M + 3I$		$2172.5M$
Simultaneous NAF ($w = 1$)	$5.6M + I$	$2204.8M$	$2240.4M$
	$5.6M + I$		$2240.4M$
Simultaneous NAF ($w = 2$)	$26.0M + 5I$	$1910.4M$	$2086.4M$
	$35.0M + 2I$		$2005.4M$
Simultaneous NAF ($w = 3$)	$152.0M + 29I$	$1725.0M$	$2747.0M$
	$230.0M + 3I$		$2045.0M$
Simul. slid. window NAF($w = 2$)	$20.4M + 4I$	$1869.5M$	$2009.9M$
	$26.4M + 2I$		$1955.9M$
Simul. slid. window NAF($w = 3$)	$129.6M + 25I$	$1626.2M$	$2505.8M$
	$137.4M + 3I$		$1853.6M$
Interleaving ($w = 4$)	$14.8M + 5I$	$1740.3M$	$1905.1M$
	$31.0M + 4I$		$1891.3M$
Interleaving ($w = 5$)	$31.6M + 11I$	$1642.4M$	$2004.0M$
	$63.6M + 5I$		$1856.0M$

"Precomputation", "Proposed method" and "Evaluation" indicate the computational cost of the conventional precomputation method, our proposed method of precomputation, and the evaluation stage, respectively. "Total" means the overall computational cost of the precomputation and evaluation stages. We assume that $t = 160$, $S = 0.8M$ and $I = 30M$, where t is the number of bits in the input integers k and l.

Secondly, we compare the respective results for total computational cost. Assume that $t = 160$, where t is the number of bits in the input integers k and l. In the actual implementation [LH00], $S/M = 0.8$ and $I/M = 30$ are assumed[5]. The computational cost of Algorithm 2 is $1869.5M$ for $w = 2$ and $1626.2M$ for $w = 3$, respectively. Thus, the total computational cost with Algorithm 1 is $2043.5M$ for $w = 2$ and $2637.8M$ for $w = 3$, and the corresponding figures with Algorithm 4 are $1962.5M$ and $1905.4M$, respectively. The comparative results on computational cost that we have covered thus far are covered in Table 1.

We see from Table 1, the simultaneous sliding window NAF method is the fastest in terms of total computational cost. In the precomputation stage for the simultaneous sliding window NAF method, Algorithm 4, our proposed method, is 1.9 times faster than Algorithm 1 when $w = 2$, and 3.6 times faster when $w = 3$. The best choice of window width for the simultaneous sliding window

[5] If the ratio I/M is larger than 30, the proposed method is much more efficient than the conventional method. If not, the proposed method is not so efficient.

NAF method with our proposed method is 3, whereas the best width with the conventional method is 2.

On the other hand, in the multi-scalar multiplication $kP + lQ$ of the ECDSA scheme's signature-verification procedure, the point P is assumed to be fixed. Thus, the computation of points uP for $u \in [0, 2^w - 1]$ or $[0, f(w)]$ in advance of the precomputation stage removes the cost of computing these points from the precomputation stage. Moreover, the simultaneous computation of vQ and $uP + vQ$ reduces the number of applications of Montgomery trick by one, that is, the number of inversions is reduced by one. In a case where P is fixed, we obtain Table 2 in the same way[6].

Acknowledgements. The authors would like to thank the anonymous referees for their useful comments.

References

[Aki01] Akishita, T., *Fast Simultaneous Scalar Multiplication on Elliptic Curve with Montgomery Form*, Eighth Annual Workshop on Selected Area in Cryptography (SAC2001), (2001).

[ANSI] ANSI X9.62, Public Key Cryptography for the Financial Services Industry, *The Elliptic Curve Digital Signature Algorithm (ECDSA)*, (1998).

[BHLM01] Brown, M., Hankerson, D., López, J., Menezes, A., *Software Implementation of the NIST Elliptic Curves over Prime Fields*, Topics in Cryptology - CT-RSA 2001, LNCS2020, (2001), 250-265.

[BP98] Bailey, D.V., Paar, C., *Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms*, Advances in Cryptology - CRYPTO'98, LNCS1462, (1998), 472-485.

[BP99] Bailey, D.V., Paar, C., *Inversion in Optimal Extension Fields*, Conference on The Mathematics of Public Key Cryptography, (1999).

[BSS99] Blake, I.F., Seroussi, G., Smart, N.P., *Elliptic Curves in Cryptography*, Cambridge University Press,(1999).

[CC87] Chunnovsky, D., Chundovsky, G., *Sequences of numbers generated by addition in formal groups and new primality and factoring tests*, Advances in Applied Mathematics, 7 (1987), 385-434.

[CMO98] Cohen, H., Miyaji, A., Ono, T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), 51-65.

[Coh93] Cohen, H., *A course in computational algebraic number theory*, GTM138, Springer-Verlag, New York, (1993).

[DSA] National Institute of Standards and Technology (NIST), *Digital Signature Standard (DSS)*, FIPS PUB 186-2, (2000).

[6] The combination of a fixed-base comb method and a sliding window method might provide good efficiency for the ECDSA verification. The fixed-base comb method ($w = 8$) and the sliding window method ($w = 4$) using the technique of Montgomery trick is an optimal choice in terms of speed and memory consumption. The computational cost with 160-bit scalars is $1862.6M$. Thus, the simultaneous sliding window NAF ($w = 3$) is faster.

[Elg85] ElGamal, T., *A public-key cryptosystem and a signature scheme based on discrete logarithm*, IEEE Transactions on Information Theory 31 (1985), 469-472.

[Eng99] Enge, A., *Elliptic Curves and their applications to Cryptography*, Kluwer Academic publishers,(1999).

[GLV01] Gallant, R.P., Lambert, R.J., Vanstone, S.A., *Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms*, Advances in Cryptology - CRYPTO 2001, LNCS2139, (2001), 190-200.

[HHM00] Hankerson, D., Hernandez, J.L., Menezes, A., *Software Implementation of Elliptic Curve Cryptography over Binary Fields*, Cryptographic Hardware and Embedded Systems (CHES2000), LNCS1965, (2000), 1-24.

[Kob87] Koblitz, N., *Elliptic curve cryptosystems*, Math. Comp. 48, (1987), 203-209.

[Knu81] Knuth, D.E., *The Art of Computer Programming, 2 - Semi-numerical Algorithms*, Addison-Wesley, 2nd edition, (1981).

[IEEEp1363] IEEE P1363 Standard Specifications for Public Key Cryptography (1999). Available at http://grouper.ieee.org/groups/1363/

[LH00] Lim, C.H., Hwang, H.S., *Fast implementation of Elliptic Curve Arithmetic in $GF(p^m)$*, Proc. PKC'00 LNCS1751, (2000), 405-421.

[LL94] Lim, C.H., Lee, P.J., *More Flexible Exponentiation with Precomputation*, Advances in Cryptology - CRYPTO '94, LNCS839, (1994), 95-107.

[Men93] Menezes, A.J., *Elliptic Curve Public Key Cryptosystems*, Kluwer Academic Publishers, (1993).

[Mil86] Miller, V.S., *Use of elliptic curves in cryptography*, Advances in Cryptology - CRYPTO '85, LNCS218,(1986),417-426.

[MO90] Morain, F., Olivos, J., *Speeding up the computations on an elliptic curve using addition-subtraction chains*, Theoretical Informatics and Applications 24 No.6, (1990), 531-544.

[Möl01] Möller, B., *Algorithm for multi-exponentiation*, Eighth Annual Workshop on Selected Area in Cryptography (SAC2001), (2001), 179-194.

[Mon87] Montgomery, P.L., *Speeding the Pollard and Elliptic Curve Methods of Factorizations*, Math. Comp. 48, (1987), 243-264

[RSA78] Rivest, R.L., Shamir, A., Adleman, L., *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*, Communications of the ACM, Vol.21, No.2, (1978), 120-126.

[SEC-1] Standards for Efficient Cryptography, *Elliptic Curve Cryptography Ver.1.0*, (2000), Available at http://www.secg.org/secg_docs.htm

[Sil86] Silverman, J.H., The Arithmetic of Elliptic Curves, GMT106, Springer-Verlag, New York, (1986).