An Efficient Algorithm for Simultaneous Elliptic Curve Scalar Multiplication

KiHyung Kim*, JaeCheol Ha** and SangJae Moon*

* School of Electrical Engineering and Computer Science, Kyungpook National Univ.

** Division of Information Science, Korea Nazarene Univ.

Abstract

This paper introduces a new joint signed expansion method for computing simultaneous scalar multiplication on an elliptic curve and a modified binary algorithm for efficient use of the new expansion method. The proposed expansion method can be also be used in cryptosystems such as RSA and ElGamal cryptosystems.

I. Introduction

Because elliptic curve cryptosystems can provide a higher level of security with smaller key sizes, they have become the focus of intensive research in various fields and several standardizing bodies, such as the IEEE, ANSI etc, have already issued standards for elliptic curve cryptosystems. To use cryptosystems based on an elliptic curve, computing kP and $kP+lQ$ are essential, where P and Q are elliptic curve points. Thus several algorithms [3,7,9,10] have introduced to compute scalar multiplication fast and efficiently and it becomes known that it is efficient to process $kP+lQ$ simultaneously rather than computing it separately [1,11,13,16]. Thus we focus on simultaneous scalar multiplication algorithms. Methods to compute $kP+lQ$ simultaneously are introduced in [5,6,15,17].

In this paper, we describe the previous simultaneous scalar multiplication methods and then introduce a new joint signed expansion method that can speed up simultaneous scalar multiplication. We know that the number of additions depends on the joint Hamming weight. Thus we can decrease the joint Hamming weight by representing a bit with a large number. Of course, if we use a large number as bit value, a lot of pre-computations will have to be stored. If we can afford to store them, we can decrease the joint Hamming weight efficiently by using our algorithm. Our proposed algorithm can be used in elliptic curve cryptosystems and also in RSA and ElGamal cryptosystems.

II. Previous Simultaneous Scalar Multiplication Method

Several algorithms have been introduced for simultaneous scalar multiplication, among which we focus on Shamir's trick and the joint sparse form (JSF).

1. Algorithm using Shamir's Trick

Input: $k_0 = (k_{0,t-1}k_{0,t-2}...k_{0,1}k_{0,0})$, $k_1 = (k_{1,t-1}k_{1,t-2}...k_{1,1}k_{1,0})$
Output: $Q = k_0P + k_1G$

Pre-compute $R = P + G$

$Q = 0$
For $i = t-1$ to 0 do :
  $Q \leftarrow 2Q$
  if k_0 and k_1 are 1 $Q \leftarrow Q + R$
  else if $k_{0,i}$ is 1 and $k_{1,i}$ is 0 $Q \leftarrow Q + P$
  else if $k_{0,i}$ is 0 and $k_{1,i}$ is 1 $Q \leftarrow Q + G$
End For

Return Q

Fig 1. Simultaneous Scalar Multiplication : Shamir's Trick

Shamir suggested a simple trick, referred to as Shamir's trick [5], to speed up an exponentiation. It can be also used in elliptic curve scalar multiplication. The algorithm using Shamir trick in an elliptic curve is shown in Fig 1. k_0 and k_1 are represented in unsigned binary form. To using the algorithm shown in Fig 1, one pre-computed point $P+Q$ and two points , P and Q, are required. Shamir's trick is also used with the non adjacent form [12]. In this case, point $P-Q$ must also be pre-computed.

2. The Joint Sparse Form

Although Shamir's trick is relatively efficient, it can also be improved. For example, In 2001, Solinas introduced the joint sparse form and also provided proofs for the existence and uniqueness of the joint sparse form [17], which can be applied to cryptosystems based on an elliptic curve and also a discrete logarithm where the inversion is free.

The joint sparse form has three properties as follows :

Property 1: Of any three consecutive positions, at least one is a double zero.

Property 2: Adjacent terms do not have opposite signs.

Property 3: If $u_{i,j+1}u_{i,j} \neq 0$, then $u_{1-i,j+1} = \pm 1$ and $u_{1-i,j} = 0$

We can see the JSF algorithm and the explanation in [17]. The JSF has the lowest joint Hamming weight among all signed joint binary expansions. For example, the average joint Hamming weight of the JSF is about 1/2. Whereas the average joint Hamming weight of the joint unsigned binary representation and of the joint NAF is 3/4 , 5/9 respectively. Thus the performance of the JSF is clearly better.

Another form of the JSF is the simple joint sparse form, which has the same joint Hamming weight as the JSF, yet is simpler to implement and can be decrease the number of additions using multi-dimensions.

Ⅲ. New Joint Signed Expansion Algorithm

1. Proposed Expansion Algorithm

Now we describe a fast algorithm for elliptic curve scalar multiplication. The proposed algorithm increases the speed of elliptic curve scalar multiplication by decreasing the joint Hamming weight based on representing a bit with a number larger than 1, thereby also decreasing the number of additions.

This algorithm has one property as follows :

property 1: Of any three consecutive positions, at least one is a double zero.

This is the same as the first property of the JSF. The proposed algorithm is not satisfied with the second property of the JSF and the third property is only satisfied in case of *wsize*=2.

Example~1 : Let $k = 403_{(10)}$, $l = 334_{(10)}$ and window size wsize = 3 . the jsf is on the left, while the our proposed algorithm is on the right

$$k = k_0 = (1\,0\,\bar{1}\,0\,0\,1\,0\,0\,1\,1) = (3\,0\,0\,1\,0\,0\,0\,3)$$

$$l = k_1 = (1\,0\,\bar{1}\,\bar{1}\,0\,1\,0\,0\,\bar{1}\,0) = (3\,0\,0\,\bar{3}\,0\,0\,\bar{1}\,0)$$

In this example, joint Hamming weights of the JSF is 6, whereas , with the proposed algorithm, when the *wsize* is 3 , the joint Hamming weight is 4.

Input: k , l , $wsize$
Output: $k_0 = (k_{0,t-1}k_{0,t-2}...k_{0,1}k_{0,0})$, $k_1 = (k_{1,t-1}k_{1,t-2}...k_{1,1}k_{1,0})$

1. $j = 0$
2. While ($k \neq 0$ or $l \neq 0$) do:
   $w_0 = k \bmod 2^{wsize}$, $w_1 = l \bmod 2^{wsize}$.
   Selection Function (w_0, w_1, j).
   $k = (k - k_{0,j})/2$, $l = (l - k_{1,j})/2$.
   $j = j + 1$.
End While.

Fig 2. New Joint Signed Expansion Algorithm

Input: w_0, w_1, j
Output: $k_{0,j}$, $k_{1,j}$

$Mask_1 \leftarrow 1 << wsize - 1$, $Mask_2 \leftarrow 1 << (wsize - 1)$

For i *from* 0 *to* 1 do :

```
if (w_i == Mask_1)
        if (w_{1-i} == Mask_2)      k_{i,j} = w_i mod Mask_2
        else                        k_{i,j} = -1

else if (w_i mod 2 == 1 and (w_i ≠ 1))
    if (w_i mod Mask_2 == Mask_2 - 1)
        if (w_{1-i} == Mask_2)      k_{i,j} = -1
        else                        k_{i,j} = w_i mod Mask_2
    else
        if (w_{1-i} == Mask_2)      k_{i,j} = w_i mod Mask_2
        else
            if(w_i ≥ Mask_2)        k_{i,j} = w_i - (Mask_1 + 1)
            else                    k_{i,j} = w_i

else
  check_0 ← w_i mod 2 , check_1 ← w_{1-i} mod 2
  check_2 ← (w_i >> 1) mod 2 , check_3 ← (w_{1-i} >> 1) mod 2
  if (check_0 ≠ check_1 and check_2 ≠ check_3)   k_{i,j} = -w_i mod 2
  else                                           k_{i,j} = w_i mod 2
End For
```

Fig 3. The Selection Function Algorithm

Our proposed expansion algorithm is divided into two algorithms , algorithm 1 and algorithm 2, which are shown in Fig 2 and Fig 3 respectively. An explanation is given later.

2. Scalar Multiplication Algorithm

This section introduces an efficient scalar multiplication algorithm that can be applicable to the above expansion method.

Fig 4 shows the algorithm applied to the new joint signed expansion method. The algorithm is like binary algorithm and facilitates the efficient storage of pre-computed points and effective use of the memory in which pre-computed

Input: k , l
Output: $R = kP + lQ$

```
Pre-computation :
        compute all cases of aP + bG
          a,b ∈ {0 , ±1 , ..., 1 << (w-1)-1} , a,b are odd , i ∈ {0 , 1}

R ← O
Expansion Algorithm (k , l , wsize)

For i = t-1 to 0 do:
   R = 2R
   If k_{0,i} is equal to 0
      check_1 = ⌊|k_{1,i}|⌋
      If k_{1,i} > 0        R = R + B[check_1]
      Else if k_{1,i} < 0   R = R - B[check_1]
   Else
      check_1 = ⌊|k_{0,i}|⌋
      check_2 = k_{1,i} + (1 << (wsize - 1) + 1)
      If k_{0,i} > 0
            If k_{1,i} < 0      R = R + A[check_1][check_2]
            Else                R = R + A[check_1][k_{1,i}]
      Else
            If k_{1,i} < 0      R = R - A[check_1][⌊k_{1,i}⌋]
            Else                R = R - A[check_1][check_2]
Return R
```

그림 4. Scalar Multiplication Algorithm using the New Expansions

points are stored. The scalar multiplication algorithm uses pre-computed points that are not consecutive like $P,2P,3P\cdots$. Thus if points are not stored efficiently, unused memory space may be included. Thus a method is used to store pre-computed points. In Fig 4, the pre-computation part can be implemented as follows , where $\lfloor x \rfloor$ and sft imply lower bound of x and 1<<(wsize-1)+1 respectively :

case 1 (a==0) : B[$\lfloor b/2 \rfloor$] = bG

case 2 (b $\geq$0) : A[$\lfloor a/2 \rfloor$][b] =$aP+bG$

case 3 (b<0) : A[$\lfloor a/2 \rfloor$][b+sft] = $aP+bG$

3. Analysis

This section analyzes the algorithms shown in Fig 2 and 3. The proposed expansion algorithm uses the window size *wsize*, which is used for modular operation, $Mask_1$ and $Mask_2$. The extent of an output bit is also decided according to the *wsize*. For example, if *wsize* is 3, the extent is ±3. The window size ,*wsize*, can be changed. That is, any number can be used as the *wsize*. However, if *wsize* is increased, the number of the pre-computed points must be increased. For example, when the *wsize* is 4, the number of pre-computed points is 40 and if the *wsize* is 5 , the number of pre-computed points over 100. Therefore, the

good choice for the *wsize* is determined to be 2 , 3 or 4.

We can see one important conditional operation , if(w_{l-i}==$Mask_2$), in Fig 3. In almost every case, this is operated. If w_{l-i} is equal to $Mask_2$, a Hamming weight will certainly occur in *wsize*-1 round. Thus, if a Hamming weight occurs at the same time, the correlation between k_0 and k_l will increase. As a result, the joint Hamming weight will be decreased.

Ⅳ. Comparison

Scalar multiplication algorithm		Points stored	The length of k and l			
			160 bits	192 bits	256 bits	512 bits
Joint Sparse Form		4	79.8	96.7	126.8	255.7
Joint windo w-w NAF	w=2	4	88.3	107.4	139.9	283.8
	w=3	12	69.9	84.4	111.2	223.9
Propos ed algorit hm	wsize =2	4	79.8	96.7	126.8	255.7
	wsize =3	12	62.5	75.4	98.9	200.2
	wsize =4	40	52.8	63.5	84	168.1

Table 1. Comparison of the average number of additions

.This section compares our proposed algorithm with other expansion algorithms for computing $kP+lQ$. Ten thousand simulations were conducted in each case using the C program and rand() as a random number generator. The average number is shown in Table 1. We can see that our new joint signed expansion method has the minimal joint Hamming weight among all the measured methods. Of course, when the *wsize* is 2, the joint Hamming weight of our proposed expansion algorithm was the same as that of the JSF. Yet because the JSF has the minimal joint Hamming weight, our method also has the minimal joint Hamming weight. When *wsize* was 3 and 4, the joint Hamming weight was about 22% and 34% lower respectively than that of the joint sparse form and when *wsize* was 3 , the joint Hamming weight was about 10% lower than that of the joint window-w NAF method with w=2 .

Ⅴ. Conclusion

We introduced a new joint signed expansion method and scalar multiplication algorithm for the efficient application of the new expansion method and then compared our expansion method with other expansion methods. Table 1 shows that our expansion method has the lowest number of additions among the compared methods. The number of additions with an increase in the *wsize* was also analyzed and an appropriate value for the *wsize* suggested. Because the large *wsize* is used, the number of pre-computed points increases, yet it is considered that the higher number of pre-computations is still acceptable. If we can't afford to store a lot of pre-computed points, we can use multi-exponentiation by interleaved exponentiation algorithm or other algorithms with our proposed expansion algorithm.

References

[1] T. Akishita, "Fast simultaneous sclar multiplicationon elliptic curves with montgomery form", Selected Areas in Cryptography, SAC 2001 LNCS vol.2259 pp.255-267, Springer-Verlag, 2001.

[2] R.M. Avanzi, "On multi-exponentiation in cryptography",Cryptology ePrint Archive,2002

[3] M. Brown, D. Hankerson, J. Lopez, and A. Menzes, " Software Implementation of the NIST Elliptic Curves Over Prime Field " , CT-RSA 2001, LNCS 2020, pp.250-265, Springer-Verlag, 2001.

[4] Mathieu Ciet, Tanja Lange, Francesco Sica and Jean-Jacques Quisquater, " Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms", Advances in Cryptology, EUROCRYPT 2003 LNCS vol.2656, pp.388-400, 2003

[5] T.Elgamal, "A public key cryptosystem and

a signature scheme based on discrete logarithms", IEEE Tran. Infomation Theory, vol.31, no.4, pp.469-472, 1985.

[6] R. Gallant, R. Lambert and S. Vanstone, "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms", Advances in Cryptology - CRYPTO 2001, LNCS vol.2139, pp.190-200, Springer-Verlag, 2001.

[7] Daniel. M. Gordon, "A survey of fast exponentiation methods", Journal of Algorithms 27, pp.129-146, 1998 .

[8] Peter J. Grabner, Clemens Heuberger and Helmut Prodinger, "Distribution results for low-weight binary representations for pairs of integers", Theoretical Computer Science, to appear 2003.

[9] Darrel Hankerson, J. Lopez and A. Menezes, " Software Implementation of Elliptic Curve Cryptography Over Binary Fields ", CHES2000, LNCS, pp.1-24, Springer-Verlag, 2000.

[10] Koyama and T. Tsuruoka, "Speeding up elliptic cryptosystems by using a signed binary window method" Advances in Cryptology - CRYPTO 92, LNCS vol.740, pp.345-357, 1993.

[11] C.H.Lim and P.J.Lee, " More flexible exponentiation with precomputation" Analysis for Elliptic Curve Cryptography", Advances in Cryptology-CRYPTO 94, LNCS vol.839, pp.95-107, Springer-Verlag, 1994.

[12] F. Morain and J. Olivos, "Speeding up the computations on an elliptic curve using addition-subtraction chains", Theoretical Informatics and Applications, vol.24, pp.531-543, 1990.

[13]B. Möller, "Algorithms for multi-exponentiation." , Selected Areas in Cryptography, SAC 2001, LNCS vol.2259, pp.165-180, Springer-Verlag, 2001.

[14] B. Möller, "Improved Techniques for Fast Exponentiation. ", Information Security and Cryptology - ICISC 2002 , LNCS vol.2587. pp.298-312, Springer-Verlag, 2003.

[15] Yasuyyki Sakai and Kouichi Sakurai, "An Efficient Representation of Scalars for Simultaneous Elliptic Scalar Multiplication", IEICE Trans, Fundamentals, vol E86-A, no.5 May 2003.

[16] J. Solinas, "Some computational speedups and bandwidth improvements for curves over prime fields", 5th workshop on Elliptic Curve Cryptography (ECC 2001) , 2001.

[17] J. Solinas, "Low-Weight Binary Representations for Pairs of Integers.", Tecnical Report, CORR 2001-41 , 2001.