Fast Multi-computations with Integer Similarity Strategy *

Wu-Chuan Yang[1], D. J. Guan[2], and Chi-Sung Laih[1]

[1] Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan 701, R.O.C.
[2] Department of Computer Science, National Sun Yat Sen University, Kaohsiung, Taiwan 804, R.O.C.
wcyang77@ms32.hinet.net, guan@cse.nsysu.edu.tw, laihcs@eembox.ncku.edu.tw

Abstract. Multi-computations in finite groups, such as multiexponentiations and multi-scalar multiplications, are very important in ElGamal-like public key cryptosystems. Algorithms to improve multi-computations can be classified into two main categories: precomputing methods and recoding methods. The first one uses a table to store the precomputed values, and the second one finds a better binary signed-digit (BSD) representation. In this article, we propose a new integer similarity strategy for multi-computations. The proposed strategy can aid with precomputing methods or recoding methods to further improve the performance of multi-computations. Based on the integer similarity strategy, we propose two efficient algorithms to improve the performance for BSD sparse forms. The performance factor can be improved from 1.556 to 1.444 and to 1.407, respectively.
Keywords: ElGamal-like public key cryptosystems, binary signed-digit (BSD) representations, sparse forms, multi-computations, multiexponentiations, multi-scalar multiplications

1 Introduction

Multi-computations in finite groups, such as multiexponentiations, e.g. $c = a^x b^y$, and multi-scalar multiplications, e.g. $C = xA + yB$ (A, B, and C denote points in one elliptic curve), are very important in many ElGamal-like public key cryptosystems [8] [21] [9]. In addition to the algorithms for single computations (some good surveys can be found in [13] [5] [10]), the performance of multi-computations can be improved by the concept of multiexponentiation [8, Section V.B]. This concept was generalized to the small window methods by Yen, Laih, and Lenstra [23].

Based on the concept of multiexponentiations, many algorithms have been proposed to improve the performance of multi-computations. In general, these algorithms can be classified into two categories: precomputing methods and recoding methods. Precomputing methods use a large table to store the precomputed values, such as the BGMW method [4] and the Lim-Lee method [14].

* This work was supported by the National Science Council, Taiwan, under contract NSC 92-2213-E-232-002

Precomputing methods are very suitable for memory sufficient environment and have the better performance indeed. Since the binary signed-digit (BSD) representation of an integer is not unique, recoding methods try to recode the BSD representations of x and y such that their joint Hamming weight $\omega(x, y)$ is as minimal as possible [7] [22]. The joint Hamming weight can be defined by the number of digit pairs, at least one of which is nonzero. Recoding methods are very useful in memory limited environments, such as IC cards or smart consumer electronic devices. Recently, this topic has been discussed in many articles [15] [18] [2] [3] [16] [19].

In this article, we focus on the memory limited environment and introduce a new integer similarity strategy to improve the performance of multi-computations. When computing $c = a^x b^y$ or $C = xA + yB$, the recoding methods match the zeros or nonzeros as possible by **recoding** x and y in advance, therefore the performance of multi-computations can be improved. Instead of recoding x and y, the new strategy is by **deleting** or **inserting** some digits in x and y, such that x and y have as much similarity as possible. For example, if $x = 010101011_2$ and $y = 101010101_2$, we can match the zeros by deleting the first zero in x and inserting a zero before the last digit in y as follows.

$$\begin{array}{rllll}
 & x & 0\,1\,0\,1\,0\,1\,0\,1 & 1 & \\
\text{Original computation} & y & 1\,0\,1\,0\,1\,0\,1\,0 & 1 & \omega(x,y)=9 \\
 & & & & \\
 & x & 0\,1\,0\,1\,0\,1\,0\,1 & 1 & \\
\text{adjusted computation} & y & 1\,0\,1\,0\,1\,0\,1 & 01 & \omega(x,y)=5 \\
 & & \uparrow & \uparrow & \\
 & & \text{deleted} & \text{inserted} &
\end{array}$$

Obviously, the computation must be modified for evaluating the correct result if some digits in x or y were deleted or inserted. As the above example, we only compute the deleted digit which is the beginning digit of x. Afterwards the digit with the same value can be computed simultaneously. Finally, the inserted digit in y should be computed with the last digit pair. Different from the recoding methods, our proposed methods improve the performance by **shifting** the digits. Thus our methods are very promising ones to improve performance in memory limited environments.

Since the performance of the multi-computation algorithms is determined by the computations of nonzero columns, we use a performance factor, ρ, to evaluate the performance of multi-computations. The performance factor can be defined as follows, note that "1" refers to the necessary computations of square (in multiexponentiation) or double (in multi-scalar multiplication).

$$\rho = 1 + \frac{\text{number of nonzero digit pairs}}{\text{number of total digit pairs}}.$$

The performance of multi-computations by BSD representations can be described as follows: $\rho = 1.556$ by using sparse forms directly [13], $\rho' = 1.534$

[7] by using the Dimitrov-Jullien-Miller method, and $\rho'' = 1.500$ [22] by using joint sparse forms, respectively. The proposed integer similarity strategy has practical applications for the above BSD methods. Based on the integer similarity strategy, we propose two efficient algorithms to improve the performance for BSD sparse forms; the performance factor can be reduced from 1.556 to 1.444 and to 1.407, respectively. The proposed strategy can also be used in binary representations since it does not recode the representation. Based on the proposed strategy, ρ can be reduced from 1.75 to 1.667 in binary method.

The rest of this article is organized as follows. In Section 2, we first review the basic multi-computation algorithms. The concept of integer similarity strategy and the proposed algorithms are illustrated in Section 3. And we also prove the performance of the proposed algorithms. In Section 4, we compare the performances of some well-known recoding methods and our proposed methods. Besides, the application of the proposed strategy to binary representations is also discussed in Section 4. Finally, our conclusion is presented in Section 5.

2 Preliminaries of Multi-Computations

To simplify the description, the integer similarity strategy is described by multi-scalar multiplication, $C = xA + yB$, with BSD representations only. Note that our strategy can also be applied to multiexponentiation, $c = a^x b^y$, with binary representations [11]. The notations used in this article are described as follows. The uppercase alphabet, such as A or B, denotes the discrete point in elliptic curve public key cryptosystems. The lowercase alphabet, such as x or y, denotes an n-bit integer. Because the minimal weight BSD representations need an extra BSD, x can be represented by $n+1$ BSDs as follows ($\bar{1}$ denotes -1).

$$x = \sum_{i=0}^{n} x_i 2^i = (x_n x_{n-1} \cdots x_1 x_0)_2, \text{ where } x_i \in \{\bar{1}, 0, 1\}.$$

Symbol $|x|$ represents the bit-length of x, $\omega(x)$ represents the Hamming weight of x, i.e. the number of nonzero digits. In multi-computation, we put our emphasis on whether the digit is zero or not. Therefore we use "o" to denote zero value, and "ι" to denote the nonzero values. Hence the digits can be classified into two sets: the zero set S_o and the nonzero set S_ι. $x_i \sim y_i$ denotes $x_i, y_i \in S_o$ or $x_i, y_i \in S_\iota$. The expression $x_i \not\sim y_i$ denotes $x_i \in S_o$, $y_i \in S_\iota$ or $x_i \in S_\iota$, $y_i \in S_o$.

For integer pairs, $|(x, y)| = \max(|x|, |y|)$, the joint Hamming weight $\omega(x, y)$ is defined by the total number of $(x_i, y_i) \neq (0, 0)$, for all i. Thus, the performance factor ρ can be simplified to $\rho = 1 + \frac{\omega(x,y)}{|(x,y)|}$.

2.1 The Basic BSD Method for Multi-Scalar Multiplications

The expected $\omega(x)$ in minimal weight BSD representations is $\frac{1}{3}n$ [1]. Many algorithms can be used to recode the binary representation or any BSD representation to minimum weight BSD representation [20] [11] [12]. Notice that an integer may have many minimal weight BSD representations, the most famous one is called the sparse form since no two consecutive digits are both nonzeros. Sparse forms are also called canonical forms or non-adjacent forms [10]. Minimal weight BSD representations are especially suitable for elliptic curve scalar multiplications since the inverse of a point is easy to compute. The basic BSD method for multi-scalar multiplications is shown in *Algorithm 1.* Symbol O denotes the identity element of the elliptic curve, this point is also called "point at infinity." The value of all possible $x_iA + y_iB$ must be precomputed in Line 6 of *Algorithm 1.* Therefore it needs 5 registers to store the value of A, B, $A + B$, $A - B$, and C. The inverse value $-A$, $-B$, $-A - B$ and $-A + B$ are easily to obtain from the precomputed table, so we do not need to precompute these value. The performance factor of *Algorithm 1*, ρ_1, is equal to 1.556. The proof is shown in Theorem 1.

Algorithm 1. The Basic BSD Method for multi-computations

I/P: A, B, x, y
O/P: $C = xA + yB$

1: Recode x and y to the minimum weight BSD representations;
2: Prepare the following values: $A, B, A \pm B$;
3: $C = O$;
4: **for** $i = n$ **downto** 0 **do** {
5: $C = 2C$;
6: **if** $(x_i, y_i) \neq (0, 0)$ **then** $C = C + (x_iA + y_iB)$;
7: }

Theorem 1. *The performance factor of Algorithm 1 is* $\rho_1 = 1\frac{5}{9} \simeq 1.556$.

Proof. In Line 6, the probability of $(x_i, y_i) \neq (0, 0)$ is $1 - (\frac{2}{3})^2 = \frac{5}{9}$. Therefore the performance factor $\rho_1 = 1 + \frac{5}{9} \simeq 1.556$. □

2.2 The Recoding Methods for BSD Representations

Since there are many minimal weight BSD representations, the result of *Algorithm 1* can be improved by recoding the representations. Dimitrov, Jullien, and Miller proposed 8 reduction rules to recode x and y (called the DJM method in this article) [7]. In their method, if the scanned segment of three consecutive digits matches one of the upper part of Table 1, the algorithm recode the segment to the corresponding lower part. The performance factor can be reduced from 1.556 to 1.534 by using the DJM method.

On the view of sparse form for the single integer, Solinas proposed the concept of joint sparse form (JSF) for pairs of integers, the properties of JSF are illustrated as follows [22] :

1. Of any 3 consecutive digits, at least one is double zeros.
2. Adjacent digits do not have opposite signs.

Table 1. The DJM reduction rules

Original	$x_{i+2}x_{i+1}x_i$	010	010	$0\bar{1}0$	$0\bar{1}0$	$10\bar{1}$	$10\bar{1}$	$\bar{1}01$	$\bar{1}01$
digits	$y_{i+2}y_{i+1}y_i$	$10\bar{1}$	$\bar{1}01$	$10\bar{1}$	$\bar{1}01$	010	$0\bar{1}0$	010	$0\bar{1}0$
After	$x'_{i+2}x'_{i+1}x'_i$	010	010	$0\bar{1}0$	$0\bar{1}0$	011	011	$0\bar{1}\bar{1}$	$0\bar{1}\bar{1}$
Adjusted	$y'_{i+2}y'_{i+1}y'_i$	011	$0\bar{1}\bar{1}$	011	$0\bar{1}\bar{1}$	010	$0\bar{1}0$	010	$0\bar{1}0$

3. If $x_{i+1}x_i \neq 0$, then $y_{i+1} \neq 0$ and $y_i = 0$.
   If $y_{i+1}y_i \neq 0$, then $x_{i+1} \neq 0$ and $x_i = 0$.

Solinas also proposed two efficient recoding algorithms to generate the joint sparse form from binary representation and sparse form, respectively. The performance factor can be improved to 1.500 when n approaches infinite, and this value is the minimum of all the recoding methods.

3 The Integer Similarity Strategy

By observing above recoding methods, we find two major limitations in those method. First, they can not recode the binary representations since the binary representation for an integer is unique. Second, they cannot recode the digits with the same signs, such as 101 or $\bar{1}0\bar{1}$, because they are unique minimum weight form. For example, if $x = (10101010)_2$ and $y = (01010101)_2$, all recoding methods cannot improve the computation. Based on the observation, we propose a totally new strategy, the integer similarity strategy, to improve the performance of multi-computations. Our idea is to shift some digits by **deleting** and **inserting** so that two different integers can be as much similarity as possible. For example $x = (01010101)_2$ and $y = (10101010)_2$, x can be adjusted by deleting the first zero and inserting a zero in the end. When the digit of x or y is deleted or inserted, the corresponding computation must be defined for evaluating the correct result. In order to use the proposed strategy for multi-computations, the following items must be taken into consideration.

1. *The condition for deleting or inserting.*
   For improving the performance, we have to define the condition to let the integers be as much similarity as possible. The condition depends on both integer representations and memory space.
2. *The corresponding computation of deletion or insertion.*
   In computing $C = xA + yB$, $C = 2C + x_iA$ is computed when deleting x_i and $C = 2(C + y_{i+1}B) + (x_iA + y_iB)$ when inserting y_i.
3. *The computation after deletion or insertion.*
   After deletion, the corresponding digits of x and y will be shifted, that is the corresponding digits of x_{i-1} is y_i after deleting x_i. The corresponding computation after deletion is $C = 2C + (x_{i-1}A + y_i2B)$.

The simplest case of the integer similarity strategy is that one insertion in an integer follows one deletion in another integer, we name it the single-stage version. The deletion can be acted on only one integer, called the single-integer version, and it can be also acted on both the integers, call double-integer version. In this article, in order to point out the essence of the integer similarity strategy, two basic methods are taken into consideration. The first one, called the **single-stage single-integer (1S1I)** method, is to delete one digit in x then to insert another digit in y at an appropriate position. The second one, called the **single-stage double-integer (1S2I)** method, is to delete one digit in x or y and insert another digit in its opposite integer. The single stage can be generalized to multi-stage. However, we do not discuss the generalization of 1S1I and 1S2I method in this article due to the page limitation.

3.1 The 1S1I Method for Sparse Forms

The BSD sparse form has an important property – of any 2 consecutive digits, at least one is zero. According to this property, if we want to match the zeros and nonzeros, $x_i \nsim y_i$ is a suitable condition to delete one digit in x. When one digit in x is deleted, the computation should be modified, which is called "Delete x" state, denoted by *Dx*. On the contrary, if the computation is the same with the original algorithm, the state can be called the "Normal" state, denoted by *Nr*. Thus the state diagram of the 1S1I method is shown in Fig. 1.

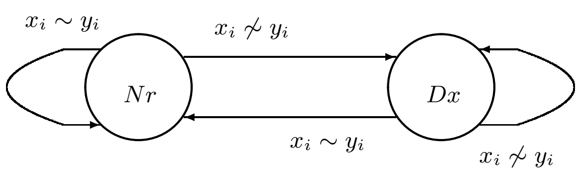

Fig. 1. The state diagram of the 1S1I method

Consider the following condition, x_u is deleted in x and y_v is inserted in y.

$$\begin{aligned} x &= (x_n \cdots \cancel{x_u}\ x_{u-1} \cdots \quad x_v\ x_{v-1} \cdots x_0)_2 \\ y &= (y_n \cdots \quad y_u \quad \cdots y_{v+1} y_v\ y_{v-1} \cdots y_0)_2 \end{aligned}$$

Before deleting x_u and after inserting y_v ($i > u$ or $i < v$), the computation is $2C + (x_iA + y_{i+1}B)$. It is the same as *Algorithm 1.* When deleting x_u ($i = u$), the computation is $2C + x_uA$. After deleting x_u and before inserting y_v ($u > i > v$), the computation is $2C + (x_iA + y_{i+1}2B)$ and the state is transferred into *Dx*. When inserting y_v $(i = v)$, the computation is $C = 2(C+y_{v+1}B)+(x_vA+y_vB)$. The corresponding digits are (x_i, y_i) in *Nr*, and (x_i, y_{i+1}) in *Dx*. Therefore, the corresponding computations can be illustrated in Table 2. To summarize the above, *Algorithm 1* can be modified to the following *Algorithm 2*.

Table 2. The corresponding computations of the proposed algorithm

State	Corresponding computation
Nr	$C = 2C + (x_iA + y_iB)$
$Nr \to Dx$	$C = 2C + x_iA$
Dx	$C = 2C + (x_iA + y_{i+1}2B)$
$Dx \to Nr$	$C = 2(C + y_{i+1}B) + (x_iA + y_iB)$

Algorithm 2. The 1S1I method for sparse forms

I/P: A, B, $x = (x_{n-1}, \cdots, x_1, x_0)_2$, $y = (y_{n-1}, \cdots, y_1, y_0)_2$
O/P: $C = xA + yB$

1: Prepare the value of A, B, $A \pm B$, $A \pm 2B$;
2: $C = O$, $State = Nr$;
3: **for** $i = n$ **downto** 0 **do** {
4:   **if** ($State = Nr$) {
5:     **if** ($x_i \sim y_i$) **then** $C = 2C + (x_iA + y_iB)$;
6:     **else** $State = Dx$, $C = 2C + x_iA$;
7:   }
8:   **else** {
9:     **if** ($x_i \not\sim y_i$) **then** $C = 2C + (x_iA + y_{i+1}2B)$;
10:     **else** $State = Nr$, $C = 2(C + y_{i+1}B) + (x_iA + y_iB)$;
11:   }
12: }

The rules in Fig. 1 are very simple and efficient. Theorem 2 proves that *Algorithm 2* is guaranteed to further improvement of the performance of *Algorithm 1* with BSD sparse forms.

Theorem 2. *Let ρ_1 and ρ_2 be the performance factor of Algorithm 1 and Algorithm 2, respectively. If x and y are both sparse forms, then $\rho_2 \leq \rho_1$.*

Proof. The performance factor is analyzed by considering the computation of *Nr* and of *Dx*.

First, we consider the computation in state *Nr* of *Algorithm 2*. In Line 5, it is the same with *Algorithm 1*. In Line 6, if y_i is zero, ρ will be decreased by 1, otherwise ρ remains the same with *Algorithm 1*.

Then we consider the computation of state *Dx*. if $x_i \not\sim y_i$ for $u \geq i \geq v$, therefore *Dx* is occurred for $(u-1) \geq i \geq (v-1)$, thus the corresponding computation digits are shown as follows.

Nr	$Nr \rightarrow Dx$	Dx	$\cdots$	Dx	Dx	$Dx \rightarrow Nr$
x_{u+1}	$\not x_u$	x_{u-1}	$\cdots$	x_{v+1}	x_v	x_{v-1}
y_{u+1}		y_u	$\cdots$	y_{v+2}	y_{v+1}	$y_v y_{v-1}$

Suppose the length of the above interval of *Dx* is k, then $k = u - v + 1$. We can get $x_i \sim y_{i+1}$ for $(u-1) \geq i \geq v$, because of the property of sparse forms and $x_i \not\sim y_i$ for $u \geq i \geq v$. Thus ρ can be considered into the following 4 conditions:

1. Led by deleting o and ended by inserting ι: $\rho = \frac{3k+1}{2}$, $k = 1, 3, 5, \cdots$.
2. Led by deleting ι and ended by inserting o: $\rho = \frac{3k+1}{2}$, $k = 1, 3, 5, \cdots$.
3. Led by deleting o and ended by inserting o: $\rho = \frac{3k}{2}$, $k = 2, 4, \cdots$.
4. Led by deleting ι and ended by inserting ι: $\rho = \frac{3k}{2} + 1$, $k = 2, 4, \cdots$.

ρ will be decreased by $\frac{k-1}{2}$, $\frac{k-1}{2}$, $\frac{k}{2}$, and $\frac{k}{2} - 1$ for the above 4 conditions, respectively, because $\rho = 2k$ in *Algorithm 1*. Thus ρ will never be increased either in *Dx*.

For the above discussion, $\rho_2 \leq \rho_1$. □

According to the proof of Theorem 2, the computation cost will not be increased even if in the worst case. The average performance of *Algorithm 2* is analyzed as follows. We now concern the conditional probability of x_i when x_{i+1} is given. We know $P_o = \frac{2}{3}$ and $P_\iota = \frac{1}{3}$ in sparse forms have been proved in [20]. Lemma 1 illustrates the conditional probability $P_{x_i|x_{i+1}}$, and it can be extended to pairs of integers, $P_{x_i y_i|x_{i+1} y_{i+1}}$, as described in Lemma 2.

Lemma 1. *Let* $P_{x_i|x_{i+1}}$ *be the conditional probability of* x_i *given* x_{i+1}. *Then* $P_{o|o} = P_{\iota|o} = \frac{1}{2}$, $P_{o|\iota} = 1$, *and* $P_{\iota|\iota} = 0$ *in BSD sparse forms.*

Proof. Since no two consecutive digits are nonzeros, $P_{o|\iota} = 1$ and $P_{\iota|\iota} = 0$.

Let $P_{o|o} = p$ and $P_{\iota|o} = 1 - p$.

$P_o = P_o \cdot P_{o|o} + P_\iota P_{o|\iota}$, therefore $\frac{2}{3} = \frac{2}{3} \cdot p + \frac{1}{3} \cdot 1 \rightarrow p = \frac{1}{2}$.

We can get $P_{o|o} = p = \frac{1}{2}$ and $P_{\iota|o} = 1 - p = \frac{1}{2}$. □

Lemma 2. *Let* $P_{x_i y_i|x_{i+1} y_{i+1}}$ *be the conditional probability of* $x_i y_i$ *given* $x_{i+1} y_{i+1}$. *Then* $P_{oo|\iota\iota} = 1$, $P_{oo|o\iota} = P_{\iota o|o\iota} = P_{oo|\iota o} = P_{o\iota|\iota o} = \frac{1}{2}$, $P_{oo|oo} = P_{o\iota|oo} = P_{\iota o|oo} = P_{\iota\iota|oo} = \frac{1}{4}$, $P_{o\iota|\iota\iota} = P_{\iota o|\iota\iota} = P_{\iota\iota|\iota\iota} = P_{o\iota|o\iota} = P_{\iota\iota|o\iota} = P_{\iota o|\iota o} = P_{\iota\iota|\iota o} = 0$.

Proof. Because the digits in x and y are independent, the probability $P_{x_i y_i|x_{i+1} y_{i+1}} = P_{x_i|x_{i+1}} \times P_{y_i|y_{i+1}}$. Thus the proof of this Lemma is completed. □

According to Lemma 3, the corresponding computations and their probabilities of *Algorithm 2* are illustrated in Table 3, where the symbols "P.S." and "N.S." stand for "Present state" and "Next state", respectively. The items "computations," "$n_{x_i y_i}$," "$P_{x_{i+1} x_i y_{i+1} y_i}$," and "Line" denote the corresponding computations, the number of additions, the probability of the computation of this row, and the corresponding line number in *Algorithm 2*. In Theorem 3, we show that the performance factor ρ_2 of *Algorithm 2* is 1.444. In comparison with 5 registers in *Algorithm 1*, *Algorithm 2* needs 2 extra registers to store the value of $A \pm 2B$.

Lemma 3. *Among the 16 possible $x_{i+1}x_iy_{i+1}y_i$, there are 9 nonzero $P_{x_{i+1}x_iy_{i+1}y_i}$, i.e. P_{oooo}, $P_{oooι}$, $P_{oιoo}$, $P_{oιoι}$, $P_{ιoιo}$, $P_{ooιo}$, $P_{oιιo}$, $P_{ιooo}$, and $P_{ιooι}$, and all of them are all equal to $\frac{1}{9}$.*

Proof. The value of P_{x_i} is $P_o = \frac{2}{3}$ and $P_\iota = \frac{1}{3}$, then the value of $P_{x_iy_i}$ is $P_{oo} = \frac{4}{9}$, $P_{o\iota} = P_{\iota o} = \frac{2}{9}$, and $P_{\iota\iota} = \frac{1}{9}$.

The value of $P_{x_{i+1}x_iy_{i+1}y_i}$ is equal to $P_{x_{i+1}y_{i+1}} \times P_{x_iy_i|x_{i+1}y_{i+1}}$. Thus, according to Lemma 2 and the above fact, all the nonzero $P_{x_{i+1}x_iy_{i+1}y_i}$ are as shown in this Lemma, and the values are all $\frac{1}{9}$. □

Table 3. Performance Analysis of *Algorithm 2*

P.S.	$x_{i+1}y_{i+1}$	x_iy_i	N.S. computations	$n_{x_iy_i}$	$P_{x_{i+1}x_iy_{i+1}y_i}$	Line
Nr	oo	$\underline{oo}$	*Nr* $C = 2C$	1	1/9	5
Nr	oo	$\underline{o}\iota$	*Dx* $C = 2C$	1	1/9	6
Nr	oo	$\underline{\iota}o$	*Dx* $C = 2C \pm A$	2	1/9	6
Nr	oo	$\underline{\iota\iota}$	*Nr* $C = 2C \pm (A \pm B)$	2	1/9	5
Nr	$\iota\iota$	$\underline{oo}$	*Nr* $C = 2C$	1	1/9	5
Dx	$o\underline{\iota}$	$\underline{oo}$	*Nr* $C = 2(C \pm B)$	2	1/9	10
Dx	$o\underline{\iota}$	$\underline{\iota}o$	*Dx* $C = 2C \pm (A \pm 2B)$	2	1/9	9
Dx	$\iota\underline{o}$	$\underline{oo}$	*Nr* $C = 2C$	1	1/9	10
Dx	$\iota\underline{o}$	$\underline{o}\iota$	*Dx* $C = 2C$	1	1/9	9

Theorem 3. *The performance factor of Algorithm 2 is $\rho_2 = 1\frac{4}{9} \simeq 1.444$.*

Proof. The performance factor is computed by $\sum(n_{x_iy_i} \times P_{x_{i+1}x_iy_{i+1}y_i})$.

According to Table 3,

$\rho_2 = \frac{1\cdot1+1\cdot1+1\cdot2+1\cdot2+1\cdot1+1\cdot2+1\cdot2+1\cdot1+1\cdot1}{9} = \frac{13}{9} \simeq 1.444.$ □

3.2 The 1S2I Method for Sparse Forms

When the deleted digit is equal to zero, it only needs one computation. Therefore if $x_i = \iota$ and $y_i = o$, it is more suitable to delete y_i instead of x_i. When we delete y_i, the state is transferred into the state "Delete y," denote by *Dy*. In this subsection, we propose a method which deletes one digit of x_i or y_i rather than deletes x_i only. The method is called the 1s2I method. The corresponding computation is illustrated in Table 4 and the state diagram of the 1S2I method is shown in Fig. 2. Thus *Algorithm 2* can be modified in the 1S2I method, as shown in *Algorithm 3*.

Table 4. The corresponding computations of the 1S2I algorithm

State	Corresponding computations
Nr	$C = 2C + (x_i A + y_i B)$
$Nr \rightarrow Dx$	$C = 2C$
$Nr \rightarrow Dy$	$C = 2C$
Dx	$C = 2C + (x_i A + y_{i+1} 2B)$
$Dx \rightarrow Nr$	$C = 2(C + y_{i+1} B) + (x_i A + y_i B)$
Dy	$C = 2C + (x_{i+1} 2A + y_i B)$
$Dy \rightarrow Nr$	$C = 2(C + x_{i+1} A) + (x_i A + y_i B)$

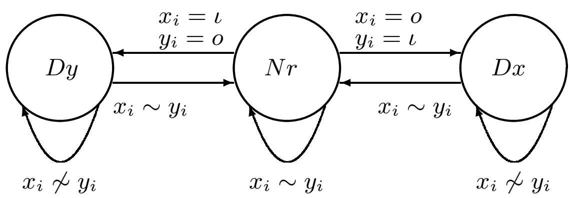

Fig. 2. The state diagram for the 1S2I method

Algorithm 3. The 1S2I method for sparse forms

I/P: A, B, $x = (x_{n-1}, \cdots, x_1, x_0)_2$, $y = (y_{n-1}, \cdots, y_1, y_0)_2$
O/P: $C = xA + yB$

Prepare the value of A, B, $A \pm B$, $A \pm 2B$, $2A \pm B$;
$C = O$, $State = Nr$;
for $i = n$ **downto** 0 **do** {
  if $(State = Nr)$ {
    if $(x_i \sim y_i)$ **then** $C = 2C + (x_i A + y_i B)$;
    else if $(x_i = o$ **and** $y_i = \iota)$ **then** $State = Dx$, $C = 2C$;
    else $State = Dy$, $C = 2C$;
  }
  else if $(State = Dx)$ {
    if $(x_i \not\sim y_i)$ **then** $C = 2C + (x_i A + y_{i+1} 2B)$;
    else $State = Nr$, $C = 2(C + y_{i+1} B) + (x_i A + y_i B)$;
  }
  else {
    if $(x_i \not\sim y_i)$ **then** $C = 2C + (x_{i+1} 2A + y_i B)$;
    else $State = Nr$, $C = 2(C + x_{i+1} A) + (x_i A + y_i B)$;
  }
}

The performance analysis of *Algorithm 3* is similar to *Algorithm 2*. In order to get the performance analysis table like Table 3, we compute the probability of all the state beforehand. We first find that the deleted digit is always zero and the corresponding digit is always nonzero. Therefore, the state Dx is separated into Dx' ($x_i = o$ and $y_i = \iota$) and Dx'' ($x_i = \iota$ and $y_i = o$); the state Dy is separated into Dy' ($x_i = \iota$ and $y_i = o$) and Dy'' ($x_i = o$ and $y_i = \iota$). Then according to Lemma 1 and Lemma 2, the probability of the state diagram is illustrated as Fig. 3. The probability of state Nr, Dx', Dx'', Dy', and Dy'' are illustrated in Lemma 4. Thus the performance analysis is illustrated in Table 5. In Theorem 4, the performance factor ρ_3 is proved to be 1.407. In comparison with 5 registers in *Algorithm 1*, *Algorithm 3* needs 4 extra registers to store the value of $A \pm 2B$ and $2A \pm B$.

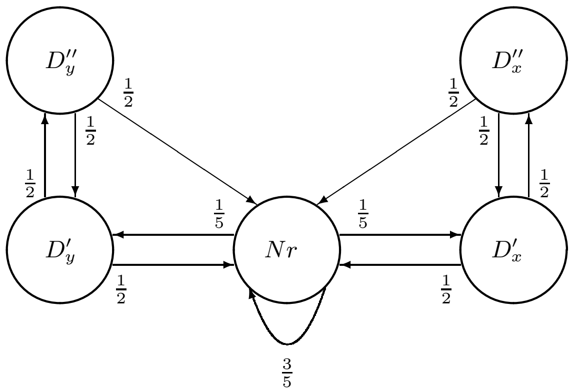

Fig. 3. The detail state probability of *Algorithm 3*

Lemma 4. *Suppose* p_0, p'_1, p''_1, p'_2, *and* p''_2 *denote the probabilities of state* Nr, Dx′, Dx″, Dy′, *and* Dy″, *respectively. Then* $p_0 = \frac{5}{9}$, $p'_1 = \frac{4}{27}$, $p''_1 = \frac{2}{27}$, $p'_2 = \frac{4}{27}$, *and* $p''_2 = \frac{2}{27}$

Proof. Consider the probability in Fig. 2, we can get

$p''_2 = \frac{1}{2}p'_2 \rightarrow p'_2 = 2p''_2$,
$p'_2 = \frac{1}{5}p_0 + \frac{1}{2}p''_2 \rightarrow p_0 = \frac{15}{2}p''_2$,
$p''_1 = \frac{1}{2}p'_1 \rightarrow p'_1 = 2p''_1$,
$p'_1 = \frac{1}{5}p_0 + \frac{1}{2}p''_1 \rightarrow p_0 = \frac{15}{2}p''_1$,
Suppose $p''_2 = p''_1 = a$, and $p'_2 = p'_1 = 2a, p_0 = \frac{15}{2}a$,
We can get $(1 + 1 + 2 + 2 + \frac{15}{2})a = 1 \rightarrow a = \frac{2}{27}$,
Therefore $p_0 = \frac{5}{9}$, $p'_1 = \frac{4}{27}$, $p''_1 = \frac{2}{27}$, $p'_2 = \frac{4}{27}$, and $p''_2 = \frac{2}{27}$. □

Table 5. Performance analysis of *Algorithm 3*

P.S.	$x_{i+1}y_{i+1}$	x_iy_i	N.S.	computations	$n_{x_iy_i}$	$P_{x_{i+1}x_iy_{i+1}y_i}$	Line
Nr	oo	$\underline{oo}$	Nr	$C = 2C$	1	1/9	5
Nr	oo	$\underline{o}\iota$	Dx	$C = 2C$	1	1/9	6
Nr	oo	$\iota\underline{o}$	Dx	$C = 2C$	1	1/9	7
Nr	oo	$\underline{\iota\iota}$	Nr	$C = 2C \pm (A \pm B)$	2	1/9	5
Nr	$\iota\iota$	$\underline{oo}$	Nr	$C = 2C$	1	1/9	5
Dx	$o\underline{\iota}$	$\underline{oo}$	Nr	$C = 2(C \pm B)$	2	2/27	11
Dx	$o\underline{\iota}$	$\underline{\iota}o$	Dx	$C = 2C \pm (A \pm 2B)$	2	2/27	10
Dx	$\iota\underline{o}$	$\underline{oo}$	Nr	$C = 2C$	1	1/27	11
Dx	$\iota\underline{o}$	$\underline{o}\iota$	Dx	$C = 2C$	1	1/27	10
Dy	$\underline{o}\iota$	$\underline{oo}$	Nr	$C = 2C$	1	1/27	15
Dy	$\underline{o}\iota$	$\iota\underline{o}$	Dy	$C = 2C$	1	1/27	14
Dy	$\underline{\iota}o$	$\underline{oo}$	Nr	$C = 2(C \pm A)$	2	2/27	15
Dy	$\underline{\iota}o$	$o\underline{\iota}$	Dy	$C = 2C \pm (2A \pm B)$	2	2/27	14

Theorem 4. *The performance factor of Algorithm 3 is* $\rho_3 = 1\frac{11}{27} \simeq 1.407$.

Proof. The performance factor is computed by $\sum (n_{x_iy_i} \times P_{x_{i+1}x_iy_{i+1}y_i})$.
According to Table 5,
$\rho_3 = \frac{1\cdot1+1\cdot1+1\cdot1+1\cdot2+1\cdot1}{9} + \frac{2\cdot2+2\cdot2+1\cdot1+1\cdot1+2\cdot2+2\cdot2+1\cdot1+1\cdot1}{27} = \frac{38}{27} \simeq 1.407.$ □

4 Comparison and Discussion

The performance of multi-computations can be improved by integer similarity strategy. Consider the 1S1I and 1S2I methods with sparse forms, $\rho_1 = 1.556$ is improved to $\rho_2 = 1.444$ and $\rho_3 = 1.407$. The performance of the proposed algorithm seems to be further improve by combining with recoding methods. Thus, using recoding methods in *Algorithm 2* and *Algorithm 3* is an interesting approach. As described in proof of Theorem 2, the computation in Dx can be divided into 4 conditions, and the performance factor can be increased in each condition. Thus our proposed methods will also improve the performance when combined with recoding methods. Unfortunately, the performance is poorer than directly using sparse forms. The reason is that zeros (or nonzeros) have been aligned between x and y in recoding methods. If we try to apply our method to the recoded BSD representations, the ratio of the improvement is less than the ratio that we apply the method on sparse forms. In our simulation (10000 pairs of 1024-bit integers generated by `java.security.SecureRandom` object in Java 2 platform), the performance factor is shown in Table 6. Thus the proposed strategy is suitable for spars forms especially. We illustrate improvement of the 1S1I method of the by given instance in Example 1. Furthermore, the proposed strategy seems to be similar to the width-w nonadjacent form (w-NAF) encoding method [6][17]. In order to achieve the unique w-NAF, the digits in w-NAF should be zero or odds. If the digits is in $\{-2,-1,0,1,2\}$, the effect is very near to the proposed integer similarity strategy, but the integer will be many representations. It does not exist an exact method to find a good "w-NAF(-2,-1,0,1,2)" for multi-computations. Based on the proposed strategy, *Algorithm 2* and *Algorithm 3* exactly define the rules of deleting or inserting digits. However, the w-NAF encoding is a very interesting research topic in multi-computations.

Table 6. The comparison of some algorithms

Performance Factor	Original	with 1S1I	with 1S2I
Sparse Forms	1.556	1.444	1.407
recode by DJM	1.534	1.453	1.414
recode to JSF	1.500	1.469	1.438

Example 1. Let $x = (10\bar{1}0\bar{1}01010101 0\bar{1}0)_2$ and $y = (01010\bar{1}0\bar{1}0\bar{1}010\bar{1}00)_2$. The performance factor of the combination with recoding methods and the 1s1I method is shown as follows. In this example, we first find that $\rho_1 = 1.938$ is improved to $\rho'_1 = \rho''_1 = 1.563$ by using the DJM method and JSF, respectively. Second, we find that $\rho_1 = 1.938$ is improved to $\rho_2 = 1.500$ by using the 1S1I method. Finally, ρ_2 can not be improved by using the DJM method and JSF.

Sparse forms	$1\,0\ \bar{1}\ \ 0\ \bar{1}\ \ 0\,1\,0\,1\ \ 0\ \ 1\,0\,\bar{1}\,0\ \ \bar{1}\ \ 0$	
	$0\,1\ \ 0\ \ 1\ \ 0\ \ \bar{1}\,0\,\bar{1}\,0\ \ \bar{1}\ \ 0\,1\,0\,\bar{1}\ \ 0\ \ 0$	$\rho_1 = 1.938$
with 1S1I	$\cancel{1}\,0\ \bar{1}\ \ 0\ \bar{1}\ \ 0\,1\,0\,1\ \ 0\ \ 1\,0\,\bar{1}\,0\ \ \bar{1}\ \ 0$	
	$0\ \ 1\ \ 0\ \ 1\ \ 0\,\bar{1}\,0\,\bar{1}\ \ 0\ \ \bar{1}\,0\,1\,0\ \ \bar{1}\,00$	$\rho_2 = 1.500$
recode by DJM	$0\,\bar{1}\ \ \bar{1}\ \ 0\ \bar{1}\ \ 0\,1\,0\,1\ \ 0\ \ 0\,1\,0\,1\ \ 1\ \ 0$	
	$0\,1\ \ 0\ \ 0\ \ 1\ \ 0\,1\,0\,1\ \ 1\ \ 0\,1\,0\,\bar{1}\ \ 0\ \ 0$	$\rho'_1 = 1.563$
with 1S1I	$0\,\bar{1}\ \ \cancel{\bar{1}}\ \ 0\ \bar{1}\ \ 0\,1\,0\,1\ \cancel{0}\ \ 0\,1\,0\,1\ \cancel{1}\ \ 0$	
	$0\,1\ \ \ 00\ 1\ \ 0\,1\,0\,1\ \ \ 10\,1\,0\,\bar{1}\ \ \ 00$	$\rho'_2 = 1.563$
recode to JSF	$0\,1\ \ 0\ \ 1\ \ 1\ \ 0\,1\,0\,1\ \ 0\ \ 0\,1\,0\,1\ \ 1\ \ 0$	
	$0\,1\ \ 0\ \ 1\ \ 0\ \ 0\,1\,0\,1\ \ 1\ \ 0\,1\,0\,\bar{1}\ \ 0\ \ 0$	$\rho''_1 = 1.563$
with 1S1I	$0\,1\ \ 0\ \ 1\ \cancel{1}\ \ 0\,1\,0\,1\ \cancel{0}\ \ 0\,1\,0\,1\ \cancel{1}\ \ 0$	
	$0\,1\ \ 0\ \ 1\ \ \ 00\,1\,0\,1\ \ \ 10\,1\,0\,\bar{1}\ \ \ 00$	$\rho''_2 = 1.563$

Besides, our proposed strategy can also be applied to multiexponentiation with binary representations. With regard to state Dx, the corresponding digits are (x_i, y_{i+1}), $x_i \neq y_{i+1}$ is suitable to insert y_i. But the value of $(c \times b^{y_{i+1}})^{\beta} \times a^{x_i} b^{y_i}$ must be computed by inserting y_i. The computation needs 1 square and 2 multiplications. Therefore, the condition, $x_i = y_i$ and $y_i \neq y_{i+1}$ (denoted by

$x_i = y_i \neq y_{i+1}$) is more suitable. Since the number of multiplication can be reduced by 1. The conditions of deletion and insertion for binary representations are shown in Fig. 4. Apply the strategy to the binary method (the square-and-multiply method), the modified algorithm is shown in *Algorithm 4*. The performance factor can be reduced from 1.75 to 1.667 and only increase one extra register to store ab^2.

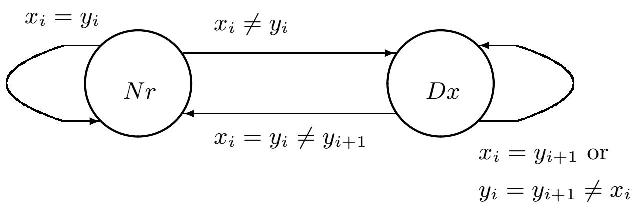

Fig. 4. The state diagram for binary representations

Algorithm 4 Apply the integer similarity strategy to binary methods

I/P: a, b, $x = (x_{n-1} \cdots x_1 x_0)_2$, $y = (y_{n-1} \cdots y_1 y_0)_2$
O/P: $c = a^x b^y$

Precompute and store the values of a, b, ab, and ab^2.
$c = 1$, $state = Nr$;
for $i = n-1$ **downto** 0 **do** {
  if ($state = Nr$) {
    if ($x_i \neq y_i$) **then** $state = Dx$, $c = c^2 \times a^{x_i}$;
    else $c = c^2 \times (a^{x_i} b^{y_i})$;
  }
  else {
    if ($x_i \neq y_{i+1}$) **then** {
      if ($x_i = y_i$) **then** $state = Nr$, $c = (c \times b^{y_{i+1}})^2 \times (a^{x_i} b^{y_i})$;
      else $c = (c \times b^{y_{i+1}})^2 \times a^{x_i}$;
    }
    else $c = c^2 \times (a^{x_i} b^{2y_i})$;
  }
}

5 Conclusion

In this article, we propose a totally new strategy, the integer similarity strategy, for multi-computations. In order to match zeros and nonzeros in multi-computation, the proposed strategy modifies the computing sequences by deleting and inserting some digits. According to the strategy, we propose two efficient algorithms, named the 1S1I and 1S2I method for multi-scalar multiplications with BSD sparse forms. The performance factor is improved from 1.556 to 1.444 and to 1.407, respectively. The memory space only required 2 and 4 extra registers, respectively. Thus the proposed algorithms is suitable for memory limited environments.

Our proposed algorithms can also be combined with recoding methods, including the DJM method and joint sparse forms. However, this way turns out to be far from desirable. Besides, the proposed strategy can be still used in binary representations. In binary methods for multiexponentiation, the performance factor can be improved form 1.75 to 1.667 with only one extra register.

Based on the integer similarity strategy, all the proposed methods are all single stage in this article, that is one insertion must appear after one deletion. In general case, the deletion and insertion should be appeared without any limitations. The multi-stage version of the proposed strategy is an interesting work in the future.

References

1. S. Arno and F. S. Wheeler. Signed digit representations of minimal hamming weight. *IEEE Trans. Computers*, 42(8):1007–1010, 1993.
2. R. M. Avanzi. On multi-exponentiation in cryptography. *IACR Cryptology ePrint Archive 2002/154, http://eprint.iacr.org*, 2002.
3. D. J. Bernstein. Pippenger's exponentiation algorithm. *http://cr.yp.to/antiforgery.html*, 2002.
4. E. F. Brickelland, D. M. Gordon, K. S. McCurley, and D. Wilson. Fast exponentiation with precomputation. *Advances in Cryptology-EUROCRYPT'92, LNCS 658, Springer-Verlag*, pages 200–207, 1992.
5. Ç. K. Koç. High-speed RSA implementations. *RSA Laboratories, Technique Notes TR201, http://www.rsasecurity.com/rsalabs*, pages 9–32, Nov. 1994.
6. H. Cohen, A. Miyagi, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. *Advances in Cryptology-AISACRYPT'98, LNCS 1514, Springer-Verlag*, pages 51–65, 1998.
7. V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Complexity and fast algorithms for multiexponentiation. *IEEE Trans. Computers*, 49(2):141–147, Feb. 2000.
8. T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Trans. Information Theory*, 31(4):469–472, Jul. 1985.
9. FIPS186-2. Digital signature standard(DSS). *NIST Computer Security FIPS page, http://csrc.nist.gov/publications/fips/*, 2001.
10. D. M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27:129–146, 1998.
11. J. Jedwab and C. J. Mitchell. Minimum weight modified signed-digit representations and fast exponentiation. *Electronics Letters*, 25(17):1171–1172, 1989.

12. M. Joye and S. M. Yen. Optimal left-to-right binary signed-digit recoding. *IEEE Trans. Computers*, 49(7):740–748, 2000.
13. D. E. Knuth. *The Art of Computer Programming, Seminumerical Algorithms*, volume 2. Addison-Wesley, 3^{rd} edition, 1998.
14. C. H. Lim and P. J. Lee. More flexible exponentiation with precomputation. *Advances in Cryptology-CRYPTO'94, LNCS 839, Springer-Verlag*, pages 95–107, 1994.
15. B. Möller. Algorithms for multi-exponentiations. *8th Annual Workshop on Selected Areas in Cryptography -SAC 2001, LNCS 2259, Springer-Verlag*, pages 165–180, 2001.
16. P. K. Mishra. Scalar multiplication in elliptic curve cryptosystems: Pipelining with pre-computations. *IACR Cryptology ePrint Archive 2004/191, http://eprint.iacr.org*, 2004.
17. J. Muir and D. Stinson. Minimality and other properties of the width-w nonadjacent form. *Technique Report CORR 2004-08, http://www.cacr.math.uwaterloo.ca*, 2004.
18. K. Okeya and K. Sakurai. Fast multi-scalar multiplication methods on elliptic curves with precomputation using montgomery trick. *4th International Workshop on Cryptographic Hardware and Embedded Systems - CHES 2002, LNCS 2523, Springer-Verlag*, pages 564–578, 2003.
19. K. Okeya, K. Schmidt-Samoa, C. Spahn, and T. Takagi. Signed binary representations revisited. *IACR Cryptology ePrint Archive 2004/195, http://eprint.iacr.org*, 2004.
20. G. W. Reitwiesner. Binary arithmetic. *Advance in computers*, pages 231–308, 1960.
21. C. P. Schnorr. Efficient identification and signatures for smart cards. *Advances in Cryptology-CRYPTO'89, LNCS 435, Springer-Verlag*, pages 239–252, 1989.
22. J. A. Solinas. Low-weight binary representations for pairs of integers. *Technique Report CORR 2001-41, http://www.cacr.math.uwaterloo.ca*, 2001.
23. S. M. Yen, C. S. Laih, and A. K. Lenstra. Multiexponentiation. *IEE Proc., Computers and Digital Techniques*, 141(6):325–326, 1994.