

# Left-to-Right Optimal Signed-Binary Representation of a Pair of Integers

Xiaoyu Ruan, *Student Member, IEEE*, and Rajendra S. Katti, *Member, IEEE*

**Abstract**—The common computation in elliptic curve cryptography (ECC),  $aP + bQ$ , is performed by extending Shamir's method for the computation of the product of powers of two elements in a group. The complexity of computing  $aP + bQ$  is dependent on the joint weight of the binary expansion of positive integers  $a$  and  $b$ . In this paper, we give a method of finding a minimum joint weight signed-binary representation of a pair of integers. Our method examines the integers  $a$  and  $b$  from left to right, thereby making the conversion to signed-binary form compatible with Shamir's method. This reduces the memory required to perform the computation of  $aP + bQ$ .

**Index Terms**—Public-key cryptography, elliptic curve cryptography, elliptic scalar multiplication, recoding of integers, joint sparse form.

## 1 INTRODUCTION

IN [2], Joye and Yen described a left-to-right method to find an optimal weight signed-binary representation of an integer  $k$ . The signed-binary representation of an integer contains the digits 0, 1, or -1 (also referred to as  $\bar{1}$ ) in its expansion. The signed-binary representation of an integer  $k$  is used to compute  $kP$ , where  $P$  is a point on an elliptic curve. The signed-binary representation of  $L$ -bit integers can have lower average weight (the number of nonzero digits) than a binary expansion, which has an average weight of  $L/2$ . The computation of  $kP$ , also known as the elliptic scalar multiplication operation [3], [4], [5], is performed using Shamir's method [6]. The complexity of Shamir's method depends on the weight of the signed-binary representation of  $k$ . Hence, finding a signed-binary representation that reduces the weight of  $k$  is very important. Shamir's method scans the binary expansion of  $k$  from the most significant digit to the least significant digit (left-to-right). Finding a left-to-right method to convert the binary expansion of  $k$  to a signed-binary form implies that this conversion and Shamir's method can be performed in tandem. However, a right-to-left algorithm to convert the binary expansion of  $k$  to signed-binary would imply first converting  $k$  to signed-binary and then using Shamir's method for computing  $kP$ . The NAF method (Non Adjacent Form [3]), which is a right-to-left method to convert the binary expansion of an integer  $k$  to a signed-binary form, results in an average weight of  $L/3$ . Joye and Yen's method [2] results in the same weight but is a left-to-right method.

The digital signature algorithm requires the computation of  $aP + bQ$ , where  $a$  and  $b$  are integers and  $P$  and  $Q$  are points on an elliptic curve. Computing  $kP$  (commonly used in all ECC protocols) can also be expressed as  $aP + bQ$ , where  $k = a + b2^s$  and  $Q = 2^sP$ . Shamir's method is again used to compute  $aP + bQ$ . The joint weight of the integers  $a$

and  $b$  determines the speed of the computation. The joint weight is the number of nonzero columns when the binary expansion of two integers is written one below the other. For example, the joint weight of 57 and 22 in their binary expansion is 6, as seen below.

$$\begin{array}{r} 57 = 111001 \\ 22 = 010110 \end{array}$$

**Example 1.** Suppose  $a = 57$  and has a signed-binary representation of  $(100\bar{1}001)$ ;  $b = 22$  and has a signed-binary representation of  $(10\bar{1}0\bar{1}0)$ . Table 1 shows the process of computing  $aP + bQ$  using Shamir's Method.

The JSF recoding (Joint Sparse Form [1] is described below) of  $a$  and  $b$  are shown in the first two rows of the table. In order to compute  $57P + 22Q$ , we start from the leftmost column and go to the rightmost column. The first entry in the row labeled "Double" is 0 (this is the entry in the third row of the leftmost nonzero column). The entry in the third row doubles the bottommost entry in the column to its left. The entries in the other rows of a column are formed based on the bits of  $a$  and  $b$  in that column. If the bit of  $a$  in a column is 1, then the entry in the row labeled "Double" of that column is added to  $P$  and this is entered in the row labeled "+ $P$ " of that column. If the bit of  $a$  in a column is  $-1$ , then the entry in the row labeled "Double" of that column is added to  $-P$  and this is entered in the row labeled " $-P$ " of that column. Similar operations apply to  $b$  and  $Q$ . The bottommost entry in the rightmost column contains the desired computation " $aP + bQ$ ."

It is clear that the number of additions required is dependent on the joint weight of  $a$  and  $b$  and the number of point-doublings required is one less than the number of bits in  $a$  or  $b$ . Thus, minimizing the joint weight would speed up the computation. Note that obtaining  $-P$  from  $P$  in the elliptic curve group can be done at negligible cost. Furthermore, a left-to-right method of finding a signed-binary representation reduces the amount of memory required to compute  $aP + bQ$  since it's compatible with

• The authors are with the Department of Electrical and Computer Engineering, North Dakota State University, Fargo, ND 58105.  
E-mail: {xiaoyu.ruan, rajendra.katti}@ndsu.nodak.edu.

Manuscript received 27 Jan. 2004; revised 11 July 2004; accepted 2 Sept. 2004; published online 15 Dec. 2004.

For information on obtaining reprints of this article, please send e-mail to: tc@computer.org, and reference IEEECS Log Number TC-0025-0104.

TABLE 1  
Computation of  $aP + bQ$  Using Shamir's Method

$a$	1	0	0	-1	0	0	1
$b$	0	1	0	-1	0	-1	0
Double	0	$2P$	$4P+2Q$	$8P+4Q$	$14P+6Q$	$28P+12Q$	$56P+22Q$
$+P$	$P$						$57P+22Q$
$-P$				$7P+4Q$			
$+Q$		$2P+Q$					
$-Q$				$7P+3Q$		$28P+11Q$	

Shamir's method; thus the new representation of  $a$  or  $b$  does not need to be stored. Since  $a$  and  $b$  are large numbers (greater than 160 bits), this is a very important consideration in resource-constrained environments like smart cards.

Solinas [1] gave a right-to-left algorithm to compute the Joint Sparse Form (JSF) of two integers. The JSF of two integers is unique and results in an optimal joint weight. Solinas stated that it is impossible to find the JSF of two integers using a left-to-right method. However, he suggested that there may be a left-to-right method to obtain a signed-binary representation of two integers that has the same joint weight as JSF. In this paper, we give such a method and show that the resulting joint weight is optimal.

The rest of the paper is organized as follows: Section 2 gives our algorithm for computing a signed-binary representation of two integers. Section 3 gives a proof for the fact that the joint weight of the signed-binary representation obtained from our algorithm is optimal. Properties of our algorithm are studied in Section 4. Section 5 compares the time taken to run our algorithm with the time taken to obtain the JSF.

## 2 ALGORITHM

There are several known methods to obtain an optimal-weight signed-binary representation of a single integer. See [2] for a recent publication in this area. For a pair of integers, the JSF algorithm [1] is a right-to-left method that results in optimal joint weight. Recently, this algorithm has been generalized to the case of more than two integers [7]. In this section, we give a left-to-right method for computing a signed-binary representation of two integers  $a$  and  $b$ . Another left-to-right solution is given by Grabner et al. in [10]. However, the method in [10] differs from our method in that it is more complicated and does not minimize the number of replacements. The method proposed by us is more practical, much easier to understand, and requires much less code to implement.

Our method is based on the following observation used commonly in Booth multiplication [8] of 2's complement binary numbers. Let  $d$  be an  $L$ -bit binary number  $(d_{L-1}, d_{L-2}, \dots, d_1, d_0)$ . Then,  $d$  can be written as

$$d = d_{L-1}2^{(L-1)} + d_{L-2}2^{(L-2)} + \dots + d_12 + d_0.$$

Since  $2^x = 2^{x+1} - 2^x$ , we can express  $d$  as follows:

$$d = (d_{L-1} - 0)2^L + (d_{L-2} - d_{L-1})2^{(L-1)} + \dots + (d_1 - d_2)2^2 + (d_0 - d_1)2 + (0 - d_0).$$

$d$  can now be expressed as an  $(L+1)$ -bit number

$$d = ((d_{L-1} - 0), (d_{L-2} - d_{L-1}), \dots, (d_0 - d_1), (0 - d_0)).$$

Each digit can be 0, 1, or -1. Using the above observation, we obtain our new recoding for  $a$  and  $b$ . Let  $a$  and  $b$  be two  $L$ -bit binary numbers expressed as follows:

$$a = (a_{L-1}, a_{L-2}, \dots, a_1, a_0).$$

$$b = (b_{L-1}, b_{L-2}, \dots, b_1, b_0).$$

Algorithm 1 gives our method for computing the new signed-binary representation.

### Algorithm 1

1. Convert the binary expansion of  $a$  and  $b$  into the intermediate signed-binary (ISB) representation (defined later)  $P_1$  and  $P_2$ :

$$P_1 = ((a_{L-1} - 0), (a_{L-2} - a_{L-1}), \dots, (a_0 - a_1), (0 - a_0))$$

$$P_2 = ((b_{L-1} - 0), (b_{L-2} - b_{L-1}), \dots, (b_0 - b_1), (0 - b_0)).$$

2. Convert  $P_1$  and  $P_2$  into  $Q_1$  and  $Q_2$  by going from left to right and performing any of the following replacements. In the replacements shown below, the top row of digits belongs to  $P_1$  and the bottom row of digits belongs to  $P_2$  and  $x, y \in \{1, -1\}$ . If a replacement is applied, then discard the columns that have been replaced and consider the next two or

three columns for replacement. If no replacement is possible, then discard one column and consider the next two or three columns for replacement.

$$\begin{aligned}
 \text{A1. } & \begin{bmatrix} 0x \\ y\bar{y} \end{bmatrix} \rightarrow \begin{bmatrix} 0x \\ 0y \end{bmatrix} \text{ or } \begin{bmatrix} 00 \\ y\bar{y} \end{bmatrix} \rightarrow \begin{bmatrix} 00 \\ 0y \end{bmatrix} \\
 \text{A2. } & \begin{bmatrix} x\bar{x} \\ y\bar{y} \end{bmatrix} \rightarrow \begin{bmatrix} 0x \\ 0y \end{bmatrix} \\
 \text{A3. } & \begin{bmatrix} x0\bar{x} \\ 0y0 \end{bmatrix} \rightarrow \begin{bmatrix} 0xx \\ 0y0 \end{bmatrix} \\
 \text{A4. } & \begin{bmatrix} x0\bar{x} \\ y\bar{y}0 \end{bmatrix} \rightarrow \begin{bmatrix} 0xx \\ 0y0 \end{bmatrix}.
 \end{aligned}$$

Note that if  $x = 1$ , then  $\bar{x} = -1$ . Also note that if

$$\begin{bmatrix} c_1c_2c_3 \\ d_1d_2d_3 \end{bmatrix} \rightarrow \begin{bmatrix} e_1e_2e_3 \\ f_1f_2f_3 \end{bmatrix}$$

is a replacement, then so is

$$\begin{bmatrix} d_1d_2d_3 \\ c_1c_2c_3 \end{bmatrix} \rightarrow \begin{bmatrix} f_1f_2f_3 \\ e_1e_2e_3 \end{bmatrix}.$$

This is because it does not matter whether we write  $P_1$  on top or  $P_2$ .

The two steps of Algorithm 1 can be combined together by scanning the binary input from left to right only once. This is illustrated by Algorithm 2, the pseudocode for Algorithm 1.

### Algorithm 2

Input:  $L$ -bit binary expansion of  $a$  and  $b$ :

$$a = (a_{L-1}, a_{L-2}, \dots, a_1, a_0).$$

$$b = (b_{L-1}, b_{L-2}, \dots, b_1, b_0).$$

Output:  $(L + 1)$ -bit signed-binary representation of  $a$  and  $b$  given by  $u_0$  and  $u_1$ :

$$u_0 = (u_{0,L}, u_{0,L-1}, \dots, u_{0,1}, u_{0,0})$$

$$u_1 = (u_{1,L}, u_{1,L-1}, \dots, u_{1,1}, u_{1,0}).$$

$$1. u_{0,L} \leftarrow a_{L-1} - 0$$

$$2. u_{1,L} \leftarrow b_{L-1} - 0$$

3. for  $j$  from  $L - 1$  down to 0 do

3.1. if  $j > 0$  then

$$3.1.1. u_{0,j} \leftarrow a_{j-1} - a_j$$

$$3.1.2. u_{1,j} \leftarrow b_{j-1} - b_j$$

3.2. else

$$3.2.1. u_{0,0} \leftarrow 0 - a_0$$

$$3.2.2. u_{1,0} \leftarrow 0 - b_0$$

3.3. for  $i$  from 0 to 1 do

3.3.1. if  $u_{i,j+1} \times u_{i,j} = -1$  and  $(u_{1-i,j+1} = 0$  or  $u_{1-i,j+1} \times u_{1-i,j} = -1)$  then

$$3.3.1.1. u_{i,j+1} \leftarrow 0$$

$$3.3.1.2. u_{i,j} \leftarrow -u_{i,j}$$

next  $i$

3.4. if  $j < L - 1$  then

3.4.1. for  $i$  from 0 to 1 do

3.4.1.1. if  $u_{i,j+2} \times u_{i,j} = -1$  and  $u_{i,j+1} = 0$  and  $u_{1-i,j+2} = 0$  and  $u_{1-i,j+1} \neq 0$  and  $u_{1-i,j} = 0$  then

$$3.4.1.1.1. u_{i,j+2} \leftarrow 0$$

$$3.4.1.1.2. u_{i,j+1} \leftarrow -u_{i,j}$$

$$3.4.1.1.3. u_{i,j} \leftarrow -u_{i,j}$$

3.4.1.2. else if  $u_{i,j+2} \times u_{i,j} = -1$  and  $u_{i,j+1} = 0$  and  $u_{1-i,j} = 0$  and  $u_{1-i,j+2} \times u_{1-i,j+1} = -1$  then

$$3.4.1.2.1. u_{i,j+1} \leftarrow u_{i,j+2}$$

$$3.4.1.2.2. u_{i,j} \leftarrow u_{i,j+2}$$

$$3.4.1.2.3. u_{i,j+2} \leftarrow 0$$

$$3.4.1.2.4. u_{1-i,j+1} \leftarrow -u_{1-i,j+1}$$

$$3.4.1.2.5. u_{1-i,j+2} \leftarrow 0$$

next  $i$

next  $j$

Algorithm 2 can be combined with Shamir's method for computing  $aP + bQ$ , thereby eliminating the necessity for storing  $Q_1$  and  $Q_2$ , the output of Algorithm 2.

**Example 2.** Suppose  $a = 6,699$  and  $b = 4,846$ . The binary expansion of  $a$  and  $b$  is given by

$$\begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} 1101000101011 \\ 1001011101110 \end{bmatrix}.$$

The joint weight is 10. Applying Step 1 of Algorithm 1, we have

$$\begin{bmatrix} P_1 \\ P_2 \end{bmatrix} = \begin{bmatrix} 10\bar{1}\bar{1}1001\bar{1}\bar{1}\bar{1}0\bar{1}\bar{1} \\ \bar{1}\bar{1}0\bar{1}\bar{1}100\bar{1}100\bar{1}0 \end{bmatrix}.$$

Now, the joint weight is 13. Applying Step 2 of Algorithm 1, we have

$$\begin{bmatrix} Q_1 \\ Q_2 \end{bmatrix} = \begin{bmatrix} 01101000110\bar{1}0\bar{1}\bar{1} \\ 01001100\bar{1}100\bar{1}0 \end{bmatrix}.$$

Now the joint weight is 9. The leftmost three columns of

$$\begin{bmatrix} P_1 \\ P_2 \end{bmatrix}$$

have been replaced using replacement A3, the two columns after that have used replacement A2, and so on. Note that the JSF of 6,699 and 4,846 is given by

$$\begin{bmatrix} a \\ b \end{bmatrix}_{JSF} = \begin{bmatrix} 011010010\bar{1}\bar{1}011 \\ 010011000\bar{1}00\bar{1}0 \end{bmatrix}.$$

The joint weight is also 9. Although the JSF may differ from the output of our method, both methods result in minimum joint weight.

## 3 PROOF OF OPTIMALITY

In this section, we prove the optimality of Algorithm 1 (optimality in this case implies minimum joint weight). The outline of the proof is as follows: We first take any signed-binary representation of two integers with joint weight  $w_0$ . We convert this representation to an intermediate signed-binary (ISB) representation (defined later) with a joint weight of  $w_1$ . This ISB representation is then the input to Step 2 of Algorithm 1, the output of which has a joint weight of  $w_2$ . We will prove that  $w_2 \leq w_0$ . In other words, the joint weight of the two integers resulting from our method is always less than or equal to the joint weight of the original signed-binary representation that we started with. However, the ISB representation of two integers and the output of Algorithm 1 are unique regardless of the signed-binary representation

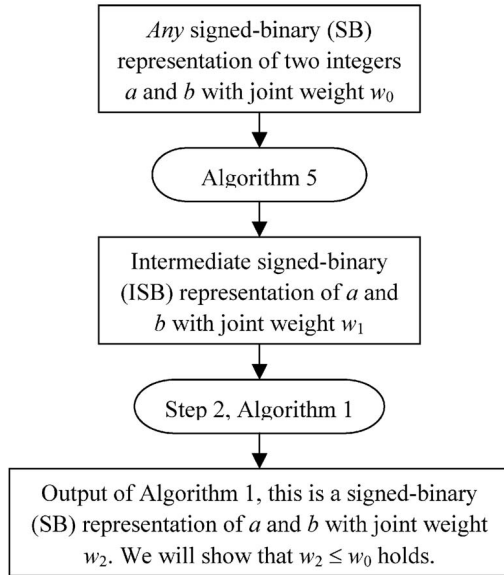


Fig. 1. Outline of the proof.

that we started with. Note that, for a fixed set of two integers, there are several signed-binary representations and the ISB representation is one of them. Therefore, if we started with a JSF representation converted this to ISB, which is unique for the two integers, and then input this to Step 2 of Algorithm 1, the output would have a joint weight that is less than or equal to the JSF input. This proves that Algorithm 1 produces an output with optimal joint weight because the JSF representation has optimal joint weight. The outline of the proof is shown in Fig. 1.

In what follows, we first define the signed-binary (SB) and the intermediate signed-binary (ISB) representations. We then prove the existence and uniqueness of the ISB representation. We then give an algorithm to convert any SB representation to ISB. Finally, we prove that  $w_2 \leq w_0$ .

### 3.1 Intermediate Signed-Binary (ISB) Representation

**Definition 1.** A signed-binary (SB) representation of an integer  $k \geq 0$  is a representation which contain the digits 0, 1 or -1.

The SB representation of a nonnegative integer is not unique. We write the SB representation of one integer below another to form a table with two rows. The SB table for these two integers is not unique either. Among all the SB representations of two integers, we are looking for the one with minimum joint weight.

One kind of SB representation is the Intermediate SB. This is defined below.

**Definition 2.** An Intermediate SB (ISB) representation of an integer  $k \geq 0$  satisfies the following properties:

1. The sequence  $10 \dots 01$  does not occur anywhere in the ISB representation of  $k$ , where the number of zeros is greater than or equal to 0.
2. The sequence  $\bar{1}0 \dots 0\bar{1}$  does not occur anywhere in the ISB representation of  $k$ , where the number of zeros is greater than or equal to 0.

3. The leftmost nonzero digit is 1 and the rightmost nonzero digit is  $\bar{1}$ .

**Example 3.** Let  $k = 53$ , the binary expansion of  $k$  is (110101). The ISB representation of  $k$  is (10 $\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$ ). Another SB representation of 53 is (100 $\bar{1}$ 1 $\bar{1}$ 1 $\bar{1}$ ). However, this is not the ISB representation as it contains two consecutive  $\bar{1}$ s, thus not satisfying Property 2 of Definition 2.

Algorithm 3 converts the binary expansion to the ISB representation, while Algorithm 4 converts the ISB representation to the binary expansion.

#### Algorithm 3

**Input:**  $L$ -bit binary expansion  $(k_{L-1}, k_{L-2}, \dots, k_1, k_0)$  of an integer  $k \geq 0$ .

**Output:** The ISB representation  $(K_L, K_{L-1}, \dots, K_1, K_0)$  of  $k$ .

```

 $k_L \leftarrow 0$ 
 $k_{-1} \leftarrow 0$ 
for  $j$  from  $L$  down to 0 do
     $K_j \leftarrow k_{j-1} - k_j$ 
next  $j$ 
    
```

Note that Step 1 of Algorithm 1 is equivalent to Algorithm 3. Thus, the output of Step 1 of Algorithm 1,  $P_1$  and  $P_2$ , is the ISB representation of  $a$  and  $b$ .

#### Algorithm 4

**Input:**  $(L + 1)$ -bit ISB representation  $(K_L, K_{L-1}, \dots, K_1, K_0)$  of an integer  $k \geq 0$ .

**Output:** The binary expansion  $(k_{L-1}, k_{L-2}, \dots, k_1, k_0)$  of  $k$ .

```

 $k_L \leftarrow 0$ 
for  $j$  from  $L - 1$  down to 0 do
     $k_j \leftarrow k_{j+1} + K_{j+1}$ 
next  $j$ 
    
```

**Lemma 1.** The ISB representation of an integer  $k \geq 0$  exists and is unique.

**Proof.** The existence of the ISB representation is proven by Algorithm 3. Algorithm 4 is the inverse algorithm of Algorithm 3 and vice versa. Both algorithms are deterministic algorithms. This implies that they yield a bijection between the binary expansion and the ISB representation. This immediately implies uniqueness of the ISB representation and correctness of Algorithm 4.  $\square$

Algorithm 5 converts any SB representation of an integer  $k \geq 0$  to its ISB representation.

#### Algorithm 5

**Input:** Any SB representation of an integer  $k \geq 0$ .

**Output:** The ISB representation of  $k$ .

Scan the SB representation of  $k$  from left to right. After each replacement given below, remove the leftmost digit and consider the remaining representation for further replacements.

**R1.** Replace any subsequence  $S = 0x00 \dots 0x$  by the subsequence  $R = x\bar{x}00 \dots 0x$ . Both  $S$  and  $R$  have  $n$  bits and  $n \geq 3$  and  $x \in \{1, -1\}$ .

**R2.** Replace any subsequence  $S = \bar{x}x00 \dots 0x$  by the subsequence  $R = 0\bar{x}00 \dots 0x$ . Both  $S$  and  $R$  have  $n$  bits and  $n \geq 3$  and  $x \in \{1, -1\}$ .

**R3.** If the rightmost nonzero bit is 1, then perform one of the following two replacements on the rightmost nonzero bit and the bit to its left.

**R3a.** Replace  $S = 01$  by  $R = 1\bar{1}$ .

**R3b.** Replace  $S = \bar{1}1$  by  $R = 0\bar{1}$ .

If Algorithm 5 produces a representation which contains  $10\dots 01$  or  $\bar{1}0\dots 0\bar{1}$ , then one could use replacements R1 through R3 repeatedly to remove any occurrence of such sequences, thus resulting in the ISB representation. Therefore, Algorithm 5 must produce the ISB representation of its input. Note that the replacements R1 and R3a could result in an increase in the joint weight. This is because a zero in the original subsequence  $S$  is replaced by a 1 or -1 in these replacements. The replacements R2 and R3b will never result in an increase in the joint weight.

**Example 4.** One SB representation of 45 is  $(\bar{1}\bar{1}0\bar{1}\bar{1}101)$ . This can be converted to the ISB representation  $(01\bar{1}10\bar{1}\bar{1}\bar{1})$  using the following steps. The underlined bits indicate a replacement.

Step 1:

$$(\bar{1}\bar{1}\underline{0}\bar{1}\bar{1}101) \xrightarrow{R2} (0\underline{1}0\bar{1}\bar{1}101).$$

Step 2:

$$(01\underline{0}\bar{1}\bar{1}101) \xrightarrow{R1} (01\bar{1}\underline{0}\bar{1}101).$$

Step 3:

$$(01\bar{1}\underline{0}1101) \xrightarrow{R1} (01\bar{1}\bar{1}\underline{0}101).$$

Step 4:

$$(01\bar{1}\bar{1}0\underline{1}01) \xrightarrow{R3a} (01\bar{1}\bar{1}0\bar{1}\underline{1}).$$

### 3.2 Optimality

Any SB representation of two integers (referred to as "original SB" below) with joint weight  $w_0$  is first converted to their ISB representation (referred to as "ISB" below) with joint weight  $w_1$  and then input to Step 2 of Algorithm 1. Step 2 of Algorithm 1 then scans the ISB representation of the two integers from left to right, performs appropriate replacements A1 through A4, and outputs an SB representation (referred to as "output SB" below) with joint weight  $w_2$ .

We first introduce some lemmas and then show that  $w_2 \leq w_0$ .

**Lemma 2.** If some bit is 0 in the original SB representation and Algorithm 5 converts it to  $x$  ( $x \in \{1, -1\}$ ), then, in the ISB representation, there exists an  $\bar{x}$  to its right.

**Proof.** Recall that R1 and R3a are the only replacements that convert 0 to  $x$  ( $x \in \{1, -1\}$ ). By observing R1 and R3a, one can see that, in either case, the sequence  $R$  contains an  $\bar{x}$  to the right of the bit which is originally 0 and replaced by  $x$ . Now, we consider R2, which may be applied after applying R1. Suppose the leftmost nonzero bit of a sequence  $S$  is  $x$ . If  $S$  is input to R2, then the leftmost nonzero bit of the sequence  $R$  remains  $x$ . This proves Lemma 2.  $\square$

TABLE 2  
3-Bit ISB and Output SB

	$ISB \xrightarrow{\text{Step 2, Algorithm 1}} \text{Output SB}$
1	$\begin{array}{ccc} x00 & \xrightarrow{A1} & x00 \\ y\bar{y}y & & y0\bar{y} \end{array}$
2	$\begin{array}{ccc} x0\bar{x} & \xrightarrow{A4} & 0xx \\ y\bar{y}0 & & 0y0 \end{array}$
3	$\begin{array}{ccc} x0\bar{x} & \xrightarrow{A1} & x0\bar{x} \\ y\bar{y}y & & y0\bar{y} \end{array}$
4	$\begin{array}{ccc} x00 & \xrightarrow{A1} & x00 \\ 0y\bar{y} & & 00y \end{array}$
5	$\begin{array}{ccc} x0\bar{x} & \xrightarrow{A3} & 0xx \\ 0y0 & & 0y0 \end{array}$
6	$\begin{array}{ccc} x0\bar{x} & \xrightarrow{A1} & x0\bar{x} \\ 0y\bar{y} & & 00y \end{array}$

**Lemma 3.** If any three consecutive columns of ISB, with at least one row of the form  $x0\bar{x}$  or  $x00$  ( $x \in \{1, -1\}$ ), are input to Step 2 of Algorithm 1, then, among the three consecutive columns of the output SB, there is at least one zero column.

**Proof.** If there is at least one zero column in the three consecutive columns of ISB, then the conclusion holds since Step 2 of Algorithm 1 never converts a zero column to nonzero. Thus, we consider the cases where all three consecutive columns are nonzero. All such input combinations are listed in Table 2 ( $x, y \in \{1, -1\}$ ).

In each case, the output SB contains at least one zero column.  $\square$

**Lemma 4.** If any two consecutive columns of ISB, with at least one row of the form  $x\bar{x}$  and the other row not of the form  $y0$  ( $x, y \in \{1, -1\}$ ), are input to Step 2 of Algorithm 1, then, among the two consecutive columns of the output SB, there is at least one zero column.

**Proof.** All such input combinations are listed in Table 3 ( $x, y \in \{1, -1\}$ ). Note that the cases where either row is of the form  $y0$  are discussed in Lemma 3.

In each case, the output SB contains at least one zero column.  $\square$

**Theorem 1.** The SB representation of two nonnegative integers resulting from Algorithm 1 is unique.

**Proof.** Algorithm 1 is a deterministic algorithm. It applies fixed operations to certain sequences of digits and does not include any alternative operation for the same input.  $\square$

We now state the crucial result of this paper.

TABLE 3  
2-Bit ISB and Output SB

	$ISB \xrightarrow{\text{Step 2, Algorithm 1}} \text{Output SB}$
1	$\begin{array}{ccc} x\bar{x} & \xrightarrow{A2} & 0x \\ y\bar{y} & & 0y \end{array}$
2	$\begin{array}{ccc} x\bar{x} & \xrightarrow{A1} & 0x \\ 0y & & 0y \end{array}$
3	$\begin{array}{ccc} x\bar{x} & \xrightarrow{A1} & 0x \\ 00 & & 00 \end{array}$

**Theorem 2.**  $w_2 \leq w_0$ .

**Proof.** We compare the original SB with ISB column-by-column from left to right. If one column is a zero column in the original SB but, after applying Algorithm 5, becomes a nonzero column in ISB (this is named an “increased column”), then we examine ISB and

1. Mark the row(s) with the bit in the increased column converted from 0 to 1/-1. This is named an “increased bit.”
2. For each marked row, find the next nonzero bit to the right of the increased bit. By Lemma 2, the desired nonzero bit must exist. Among the marked rows, let the largest distance between the increased bit and next nonzero bit be  $(p - 1)$ , i.e.,

$$x \underbrace{0 \dots 0}_{(p-1)0s} \bar{x},$$

where  $x \in \{1, -1\}$  and  $p \geq 1$ .

Let the joint weight of the  $(p + 1)$  columns of the original SB be  $w_{p0}$ , the joint weight of the  $(p + 1)$  columns of ISB be  $w_{p1}$ , and the joint weight of the  $(p + 1)$  columns of the output SB be  $w_{p2}$ . We have  $w_{p2} \leq w_{p1}$  because Step 2 of Algorithm 1 never converts a zero column to nonzero. Since a subsequence with the form  $x00\dots0\bar{x}$  in ISB, where  $x$  is originally a zero before converting, must come from either  $0xx\dots xx$  (by first applying R1 once and then repeatedly applying R2) or  $0x$  (by applying R3a once), we have  $w_{p0} = p$  and the joint weight increases by at most 1 within the  $(p + 1)$  columns of ISB. i.e.,  $w_{p1} \leq w_{p0} + 1$ .

We now examine the  $(p + 1)$  columns of ISB.

If there exists at least one zero column in the  $(p + 1)$  columns of ISB, i.e.,  $w_{p1} \leq p$ , then we have  $w_{p2} \leq w_{p1} \leq p = w_{p0}$ .

If all the  $(p + 1)$  columns of ISB are nonzero, then let us consider two cases.

Case 1:  $p \geq 2$ , i.e., at least one row is of the form  $x0\bar{x}\dots$  or  $x00\dots$ .

By Lemma 3, Step 2 of Algorithm 1 results in at least one zero column out of three consecutive columns of ISB. Therefore,  $w_{p1} \leq p$ . So, we have  $w_{p2} \leq w_{p1} \leq p = w_{p0}$ .

Case 2:  $p = 1$ , i.e., at least one row is of the form  $x\bar{x}\dots$  and the other row is not of the form  $y0\dots$ .

By Lemma 4, Step 2 of Algorithm 1 results in at least one zero column out of two consecutive columns of ISB. Therefore,  $w_{p1} \leq p$ . So, we have  $w_{p2} \leq w_{p1} \leq p = w_{p0}$ .

From the above discussions, we have shown that  $w_{p2} \leq w_{p0}$  holds in all cases. Recall that we are comparing the original SB with ISB column by column from left to right and looking for those columns that are zero in the original SB, but nonzero in ISB. For each such increased column and the  $p$  columns to its right,  $w_{p2} \leq w_{p0}$  holds, therefore  $w_2 \leq w_0$  holds.  $\square$

**Corollary.** *The SB representation of a pair of nonnegative integers resulting from Algorithm 1 has the minimum joint weight among any SB representations of the given pair of integers.*

**Proof.** This is a direct consequence of Theorem 2.  $\square$

## 4 PROPERTIES

In this section, we study some properties of the SB representation resulting from our method.

**Theorem 3.** *The average joint weight among all SB representations of a pair of  $L$ -bit nonnegative integers resulting from Algorithm 1 is  $L/2$ .*

**Proof.** Solinas [1] proved that the JSF has an average joint weight of  $L/2$ . Since the output SB of Algorithm 1 always has the same joint weight as the JSF, the average joint weight of the SB representations resulting from Algorithm 1 is also  $L/2$ .  $\square$

**Lemma 5.** *If any three consecutive columns of ISB are input to Step 2 of Algorithm 1, then, among the three consecutive columns of the output SB, there is at least one zero column.*

**Proof.** This is a direct consequence of Lemma 3 and Lemma 4.  $\square$

**Theorem 4.** *Among every five consecutive columns of the SB representation resulting from Algorithm 1, there is at least one zero column.*

**Proof.** By Lemma 5, in the worst case, Algorithm 1 makes the leftmost column of three consecutive nonzero columns of ISB a zero column. Then, the remaining two nonzero columns of ISB that have been replaced are skipped. If we are unlucky enough, then the next two columns of ISB that will be looked at might be all nonzero and impossible to be replaced at all. Thus, in the worst case, only one zero column results out of five consecutive columns.  $\square$

## 5 SIMULATION AND CONCLUSION

Our method scans the SB representation at most three columns at a time from left to right, thus, to compute  $aP + bQ$ , the memory required is only three SB bits for each integer. While using the JSF algorithm, the whole representation needs to be stored in memory before computing  $aP + bQ$ . Thus, the memory required is  $(L + 1)$  SB bits for each integer, where  $L$  is usually larger than 160. Concerning the hardware overhead, our method is superior to the

TABLE 4  
Joint Weights and Timings by the JSF Algorithm and Algorithm 2

$L$	Avg. Joint Weight from the JSF Algorithm	Avg. Joint Weight from Algorithm 2	Timings by the JSF Algorithm (seconds)	Timings by Algorithm 2 (seconds)
80	40.8	40.8	17.9	15.5
120	60.6	60.6	27.3	21.7
250	125.5	125.5	52.9	44.5
350	175.6	175.6	72.4	60.5
450	225.3	225.3	92.6	76.5
163	82.0	82.1	34.9	29.4
233	116.3	116.3	50.0	40.6
283	142.0	142.0	58.9	49.9
409	205.0	205.0	83.2	70.8
571	285.9	285.9	114.8	96.1

JSF algorithm because it results in a significant decrease in memory usage.

Our method can easily be implemented in hardware as a sequential circuit with the bits of  $a$  and  $b$  as input (the most significant bits are input first). However, this is not the case with the JSF algorithm.

In order to find the efficiency of the JSF algorithm and our method, we have simulated both methods. Table 4 gives the results. Each row of the table is obtained by randomly generating one million pairs of  $L$ -bit binary numbers and computing the average joint weights obtained from the JSF algorithm and from Algorithm 2. The first column gives the length of the binary expansion,  $L$ . The second column gives the average joint weights obtained from the JSF algorithm and the third column gives the average joint weights obtained from Algorithm 2. The last two columns give the execution time of the JSF algorithm and Algorithm 2 on a Pentium 4 Mobile, 1.8 GHz processor. The last five rows in Table 4 correspond to the size of field elements for the elliptic curves defined by NIST [9].

From Table 4, we see that the joint weights obtained from both the JSF algorithm and Algorithm 2 are approximately  $L/2$ . The entries in the fourth and fifth columns in Table 4 show that Algorithm 2 takes less operating time than the JSF algorithm does.

We have presented a left-to-right method to obtain the signed-binary representation of a pair of integers that results in optimal joint weight. Our algorithm leads to lower hardware overhead as well as better performance in

both hardware and software. Our method can be easily extended to the case of finding an optimal signed-binary representation of more than two integers.

#### ACKNOWLEDGMENTS

This work is partially supported by the US National Science Foundation under grant CCR-0429523. The authors would like to thank the editors and referees for their insightful comments.

#### REFERENCES

- [1] J.A. Solinas, "Low-Weight Binary Representations for Pairs of Integers," Technical Report CORR 2001-41, Center for Applied Cryptographic Research, Univ. of Waterloo, Canada, 2001.
- [2] M. Joye and S. Yen, "Optimal Left-to-Right Binary Signed-Digit Recoding," *IEEE Trans. Computers*, vol. 49, no. 7, pp. 740-748, July 2000.
- [3] J. Lopez and R. Dahab, "An Overview of Elliptic Curve Cryptography," technical report, Inst. of Computing, State Univ. of Campinas, Brazil, May 2000.
- [4] A. Menezes, *Elliptic Curve Public Key Cryptosystems*. Kluwer Academic, 1993.
- [5] I. Blake, G. Serroussi, and N. Smart, *Elliptic Curves In Cryptography*. London Math. Soc., Lecture Note Series 265, Cambridge Univ. Press, 1999.
- [6] T. ElGamal, "A Public-Key Cryptosystem and Signature Scheme Based on Discrete Logarithms," *IEEE Trans. Information Theory*, vol. 31, pp. 469-472, 1985.
- [7] J. Proos, "Joint Sparse Forms and Generating Zero Columns When Combining," Technical Report CORR 2003-23, Center for Applied Cryptographic Research, Univ. of Waterloo, Canada, 2003.

- [8] A.D. Booth, "A Signed Binary Multiplication Technique," *Quarterly J. Mechanical Applications Math.*, vol. 4, no. 2, pp. 236-240, 1951.
- [9] Nat'l Inst. of Standards and Technology, "Digital Signature Standard," FIPS publication 186-2, Feb. 2000.
- [10] P.J. Grabner, C. Heuberger, H. Prodinger, and J. Thuswaldner, "Analysis of Linear Combination Algorithms in Cryptography," preprint, [http://www.wits.ac.za/helmut/abstract/abs\\_202.htm](http://www.wits.ac.za/helmut/abstract/abs_202.htm), 2004.



**Xiaoyu Ruan** received the BS degree in electronics engineering from Fudan University, Shanghai, China, in 2003. He is now pursuing the PhD degree in the Department of Electrical and Computer Engineering at North Dakota State University. His current research interests include cryptography, error correcting codes, and digital systems. He is a student member of the IEEE and IEEE Computer Society.



**Rajendra S. Katti** received the BTech degree from the Indian Institute of Technology (Bombay), India, in 1983. He received the MS degree in mechanical engineering from the University of Idaho in 1985 and the MS degree in electrical engineering from Washington State University in 1987, where he also received the PhD degree in electrical engineering in 1991. He teaches courses related to digital systems and computer architecture. He has received funding from the US National Science Foundation in the area of performance modeling of computer architectures and cryptography. His interests are in smart sensors, cryptographic hardware, finite field arithmetic, fault-tolerant computing, and computer architecture. He has published more than 20 journal and conference papers on these topics. He was a senior design engineer at Intel Corporation in 2000 and 2001, where he worked in the Design for Testability Group. He has also taught at Wichita State University in Kansas. He has collaborated with the IBM Almaden Research Center for the development of unidirectional error correcting codes. He is currently an associate professor in the Department of Electrical and Computer Engineering at North Dakota State University. He is a member of the IEEE.

▷ For more information on this or any other computing topic, please visit our Digital Library at [www.computer.org/publications/dlib](http://www.computer.org/publications/dlib).