

LEFT-TO-RIGHT BINARY SIGNED-DIGIT RECODING FOR ELLIPTIC CURVE CRYPTOGRAPHY

Rajendra Katti and Xiaoyu Ruan
 Department of Electrical and Computer Engineering
 North Dakota State University, Fargo, ND 58105, USA

ABSTRACT

In this paper we present a new left-to-right (i.e., from the most significant bit to the least significant bit) algorithm to compute the binary signed-digit representations of two integers g and h such that their joint weight is optimal. In [1] Solinas left this as an open problem. Such an algorithm is useful in simplifying the circuits for the implementation of elliptic curve cryptosystems (ECC).

1. INTRODUCTION

1.1 Signed-Digit Representation

A signed-digit representation of integers contains the digits 0, 1 and -1 as opposed to just 0 and 1 in the binary expansion. For instance, integer n is denoted as:

$$n = (u_L u_{L-1} \dots u_1 u_0),$$

which means that

$$n = \sum_{i=0}^L u_i 2^i$$

where,

$$u_i = 0, 1 \text{ or } -1 \text{ for all } i.$$

It is not difficult to see that the signed-digit representation of an integer uses exactly one more bit than its binary expansion. The signed-digit representation of a given integer is not unique.

For a signed-digit representation, define the *weight* to be the number of nonzero (either 1 or -1) bits. If the signed-digit representations of a pair of integers are written one below another, the number of nonzero columns is defined as the *joint weight*. The non-uniqueness of signed-digit representations provides the possibility of reducing the weight or joint weight compared to using the binary expansion.

1.2 The JSF Algorithm

Solinas introduced [1] an algorithm for computing signed-digit representations of a pair of integers, which is called Joint Sparse Form (JSF). We give three important properties of the JSF algorithm below.

- The JSF of two integers has the minimal joint weight among any signed-digit representations of the given pair of integers.

- The JSF algorithm operates from right to left.
- The average joint weight among all JSF representations of two L-bit binary integers is $L/2$.

1.3 The ECC and Shamir's Method

An elliptic curve cryptosystem (ECC) [2] is known to have high security because it depends on the discrete logarithm problem needing fully exponential time for solution. ECC also has lower key sizes and hence has the potential for low hardware overhead. The main operation in ECC is the computation of kP , where k is an integer and P is a point on an elliptic curve. Computing kP can also be expressed as

$$kP = gP + hQ,$$

where,

$$k = g + h2^s$$

and

$$Q = 2^s P.$$

Digital signatures are verified using a computation of the form $gP + hQ$, where g and h are integers and P and Q are points on an elliptic curve.

Shamir suggested [3] a simple method to compute $gP + hQ$. The method is illustrated in Example 1.

Example 1.

Suppose $g = 57$ and has a signed-digit representation of $(100\bar{1}001)$; $h = 22$ and has a signed-digit representation of $(10\bar{1}0\bar{1}0)$. Table 1 shows the process of computing $gP + hQ$ using Shamir's Method.

Table 1.
Computation of $gP + hQ$ using Shamir's Method.

| g | 1 | 0 | 0 | -1 | 0 | 0 | 1 |
|--------|---|------|-----|-----|------|------|------|
| h | 0 | 1 | 0 | -1 | 0 | -1 | 0 |
| Double | 0 | 2P | 4P+ | 8P+ | 14P+ | 28P+ | 56P+ |
| | | | 2Q | 4Q | 6Q | 12Q | 22Q |
| +P | P | | | | | | 57P+ |
| | | | | | | | 22Q |
| -P | | | | 7P+ | | | |
| | | | | 4Q | | | |
| +Q | | 2P+Q | | | | | |
| -Q | | | | 7P+ | | 28P+ | |
| | | | | 3Q | | 11Q | |

The JSF of g and h 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 g and h in that column. If the bit of g 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 g 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 h and Q . The bottommost entry in the rightmost column contains the desired computation “ $gP+hQ$ ”.

It is clear that the number of additions required is dependent on the joint weight of g and h and the number of doublings required is one less than the number of bits in g or h . 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.

Using Shamir’s Method to compute $gP+hQ$, the integers g and h are scanned from left to right (i.e., from the most significant bit to the least significant bit). Therefore obtaining new representations for g and h by scanning them from left to right is advantageous because computing the new representations of g and h and computing $gP+hQ$ can be combined. The amount of memory required to perform the computation is reduced because the new representations of g and h do not need to be stored. Since g and h are usually large numbers (greater than 160 bits) this is a very important consideration in resource-constrained environments like smart cards.

Although the JSF of two integers results in an optimal joint weight of $L/2$, where L is the number of bits needed to express the integers in binary form, it consumes more memory because it is a right-to-left method. This implies that the JSF of g and h has to be stored in memory. To avoid this extra use of memory we propose our new method to compute the binary signed-digit representations of g and h . Our method operates from left to right and also results in the optimal joint weight.

The rest of this paper is organized as follows. In Section 2 we describe our new recoding of two integers g and h . In Section 3 we compare our method to the JSF method. Finally Section 4 concludes the paper.

2. NEW RECODING

There are several known methods to obtain an optimal weight signed-digit representation of a single integer. Refer to [4] for a recent publication in this area. However the JSF algorithm [1] is the only method that results in the optimal joint weight for a pair of integers. Recently this algorithm has been extended to the case of more than two integers [5].

In this section we give our new method for computing the binary signed-digit representation of a pair of integers,

g and h . Our method is based on the following observation used commonly in Booth multiplication [6] of 2’s complement binary numbers.

Let d be an L -bit binary number which is 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.

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

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

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

In this expansion of d , each digit can be 0, 1 or -1.

Our method for computing the signed-digit representations of a pair of integers is based on the fact that the following two replacements on bits do not change the value of the integer.

- Consecutive bits $(x \bar{x})$ can be replaced by $(0 x)$.
- Consecutive bits $(x 0 \bar{x})$ can be replaced by $(0 x x)$.

In both cases x denotes either 1 or -1.

Algorithm 1 below gives our method for computing the new binary signed-digit representations.

Algorithm 1.

Let g and h be two L -bit binary numbers:

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} g_{L-1} & g_{L-2} & \dots & g_1 & g_0 \\ h_{L-1} & h_{L-2} & \dots & h_1 & h_0 \end{bmatrix}$$

1. Convert the binary expansions of g and h into

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} (g_{L-1}-0) & (g_{L-2}-g_{L-1}) & \dots & (g_0-g_1) & (0-g_0) \\ (h_{L-1}-0) & (h_{L-2}-h_{L-1}) & \dots & (h_0-h_1) & (0-h_0) \end{bmatrix}.$$

2. Make replacements on above expansions of g and h by going from left to right and

- Replacing $\begin{bmatrix} x \bar{x} \\ 0 z \end{bmatrix}$ by $\begin{bmatrix} 0 x \\ 0 z \end{bmatrix}$ (Replacement I)
- Replacing $\begin{bmatrix} x \bar{x} \\ y \bar{y} \end{bmatrix}$ by $\begin{bmatrix} 0 x \\ 0 y \end{bmatrix}$ (Replacement II)
- Replacing $\begin{bmatrix} x 0 \bar{x} \\ 0 y 0 \end{bmatrix}$ by $\begin{bmatrix} 0 x x \\ 0 y 0 \end{bmatrix}$ (Replacement III)
- Replacing $\begin{bmatrix} x 0 \bar{x} \\ y \bar{y} 0 \end{bmatrix}$ by $\begin{bmatrix} 0 x x \\ 0 y 0 \end{bmatrix}$ (Replacement IV)

Note that in Step 2,

- 1) x and y can be either 1 or -1; z can be 0, 1 or -1.
- 2) The order of g and h has nothing to do with the operations. For instance, replacing $\begin{bmatrix} x \bar{x} \\ 0 z \end{bmatrix}$ by $\begin{bmatrix} 0 x \\ 0 z \end{bmatrix}$

and replacing $\begin{bmatrix} 0 z \\ x \bar{x} \end{bmatrix}$ by $\begin{bmatrix} 0 z \\ 0 x \end{bmatrix}$ are considered the same

type of replacement, which is marked as Replacement I.

- 3) In step 2 the scanning from left to right should be done bit by bit. However, if a replacement is applied, then skip the replaced bits and continue to scan.

The two steps in Algorithm 1 can be combined by scanning the binary expansions of the two integers from left to right only once. The method is shown in Algorithm 2, which is the pseudo code of Algorithm 1 except for performing the sweeping only once.

Algorithm 2.

Input: Binary representations of g and h.

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} g_{L-1} & g_{L-2} & \dots & g_1 & g_0 \\ h_{L-1} & h_{L-2} & \dots & h_1 & h_0 \end{bmatrix}$$

Output: Binary signed-digit representations of g and h, given by array u[2][L+1]:

$$\begin{bmatrix} u[0] \\ u[1] \end{bmatrix} = \begin{bmatrix} u[0][L] & u[0][L-1] & \dots & u[0][1] & u[0][0] \\ u[1][L] & u[1][L-1] & \dots & u[1][1] & u[1][0] \end{bmatrix}$$

1. $u[0][L] \leftarrow g_{L-1} - 0$
2. $u[1][L] \leftarrow h_{L-1} - 0$
3. For j from L-1 down to 0 do
 - 3.1. If j > 0 then
 - 3.1.1. $u[0][j] \leftarrow g_{j-1} - g_j$
 - 3.1.2. $u[1][j] \leftarrow h_{j-1} - h_j$
 - 3.2. Else
 - 3.2.1. $u[0][0] \leftarrow 0 - g_0$
 - 3.2.2. $u[1][0] \leftarrow 0 - h_0$
 - 3.3. For i from 0 to 1 do
 - 3.3.1. If $u[i][j+1] \times u[i][j] = -1$ and $(u[1-i][j+1] = 0 \text{ or } u[1-i][j+1] \times u[1-i][j] = -1)$ 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
 - 3.4.1.2.6. $u[1-i][j+1] \leftarrow -u[1-i][j+1]$

In Algorithm 2 we scan g and h only once from left to right 3-bits at a time. Algorithm 2 can be combined with Shamir's method for computing $gP+hQ$ thereby

eliminating the necessity for storing the intermediary representations of g and h.

The JSF algorithm is given below for completeness. The function "mods" indicates that the modular reduction is to return the smallest residue in absolute value.

The JSF Algorithm.

Input: Non-negative integers, g and h, not both zero.

Output: The Joint Sparse Form of g and h, given by

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

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

Set $k_0 \leftarrow g$ and $k_1 \leftarrow h$; $j \leftarrow 0$; $d_0 \leftarrow 0$; $d_1 \leftarrow 0$

While $k_0 + d_0 > 0$ or $k_1 + d_1 > 0$ do

Set $n_0 \leftarrow d_0 + k_0$; $n_1 \leftarrow d_1 + k_1$

For i from 0 to 1 do

If n_i is even then $u \leftarrow 0$

Else $u \leftarrow n_i \text{ mods } 4$

If $n_i \equiv \pm 3 \pmod{8}$ and $n_{1-i} \equiv 2 \pmod{4}$ then $u \leftarrow -u$

Set $u_{i,j} \leftarrow u$

Next i

For i from 0 to 1 do

If $2d_i = 1 + u_{i,j}$ then $d_i \leftarrow 1 - d_i$

Set $k_i \leftarrow \lfloor k_i/2 \rfloor$

Next i

Set $j \leftarrow j+1$

EndWhile

Example 2 illustrates the operation of our method.

Example 2.

Suppose $g = 6699$ and $h = 4846$. The binary expansions of g and h are given by

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 1 & 0 \end{bmatrix}$$

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

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 1 & 0 & \bar{1} & 1 & \bar{1} & 0 & 0 & 1 & \bar{1} & 1 & \bar{1} & 1 & 0 & \bar{1} \\ 1 & \bar{1} & 0 & 1 & \bar{1} & 1 & 0 & 0 & \bar{1} & 1 & 0 & 0 & \bar{1} & 0 \end{bmatrix}$$

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

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \bar{1} & 1 & 0 & 0 & \bar{1} & 0 \end{bmatrix}$$

Now the joint weight is only 9. Applying the JSF algorithm to 6699 and 4846, the output is given by

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & \bar{1} & \bar{1} & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & \bar{1} & 0 & 0 & \bar{1} & 0 \end{bmatrix}$$

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

We state the theorems, which characterize our new recoding. No proof is given because of space limitations.

Theorem 1. (Uniqueness)

Given a pair of integers, the signed-digit representations obtained from our method are unique.

Theorem 2. (Optimality)

The signed-digit representations obtained from our method have the minimal joint weight among any signed-digit representations of the given pair of integers.

Theorem 3. (Average Joint Weight)

The average joint weight among all signed-digit representations of a pair of L-bit binary integers obtained from our method is L/2.

3. COMPARISON WITH JSF

3.1 Simplicity

Compared to the JSF algorithm, our algorithm is extremely easy to understand since it has a simple structure and works based on replacements on bits, rather than complex mathematical computations.

3.2 Software Implementation

In order to find the efficiency of the JSF algorithm and our method, we have simulated both algorithms. Table 2 gives the results. Each row of the table is obtained by randomly generating a pair of one million L-bit binary numbers and computing the average joint weight obtained from the JSF algorithm and Algorithm 2. The first column gives the length of the binary expansion, L. The second column gives the average joint weight obtained using the JSF algorithm and the third column gives the average joint weight obtained using Algorithm 2. The last two columns give execution time of the JSF algorithm and Algorithm 2 on a Pentium 4 Mobile, 1.8 GHz processor. The rows in Table 2 correspond to the size of field elements for the elliptic curves defined by NIST [7].

Table 2.
Joint Weights and Timings of 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) |
|-----|--|------------------------------------|--|----------------------------------|
| 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 | 205.1 | 205.0 | 83.2 | 70.8 |
| 571 | 285.9 | 285.9 | 114.8 | 96.1 |

The entries in the second and third columns in Table 2 show that the joint weights obtained from the JSF algorithm and our method are both approximately L/2. The

entries in the fourth and fifth columns in Table 2 show that when implementing in software our method takes less time to execute compared to the JSF algorithm.

3.3 Hardware Implementation

Using our left-to-right method to compute $gP+hQ$, the memory required is only 3 bits for each integer. While using the JSF algorithm the memory required is (L+1) bits for each integer, where L is usually larger than 160. Thus our method is superior to the JSF algorithm because it results in a great decrease in memory usage.

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

4. CONCLUSION

We have presented an algorithm to obtain the signed-digit representations of two integers that results in optimal joint weight. Our method has lower complexity compared to the best-known method, the JSF algorithm. A major advantage of our method is that it scans the integers from left to right, using only 3 bits of memory for each integer, thus making it compatible with Shamir’s method for computing $gP+hQ$. This property leads to lower hardware overhead as well as better performances in both hardware and software. Our method can be easily extended to the case of finding the signed-digit representations of more than two integers.

5. REFERENCES

1. J. A. Solinas, “Low-weight Binary Representations for pairs of Integers,” Technical Report CORR 2001-41, Center for Applied Cryptographic Research, University of Waterloo, Canada, 2001.
2. J. Lopez and R. Dahab, “An overview of elliptic curve cryptography,” Technical report, Institute of Computing, State University of Campinas, Brazil, May 2000.
3. T. ElGamal, “A public-key cryptosystem and signature scheme based on discrete logarithms,” The IEEE Transactions on Information Theory, Vol. 31, pp 469-472, 1985.
4. M. Joye and S. Yen, “Optimal Left-to-right Binary Signed-Digit Recoding,” IEEE Transactions on Computers 49(7): 740-748, 2000.
5. J. Proos, “Joint Sparse Forms and Generating Zero Columns when Combing,” Technical Report CORR 2003-23, Center for Applied Cryptographic Research, University of Waterloo, Canada, 2003.
6. A.D. Booth, “A Signed Binary Multiplication Technique,” Q. J. Mech. Appl. Math., 4(2): 236-240, 1951.
7. National Institute of Standards and Technology, Digital Signature Standard, FIPS publication 186-2, Feb. 2000.