

# On the Signed-Binary Window Method

Xiaoyu Ruan and Rajendra Katti

Department of Electrical and Computer Engineering

North Dakota State University, Fargo, ND 58105

E-mail: {xiaoyu.ruan,rajendra.katti}@ndsu.edu

**Abstract**—We propose a left-to-right (i.e., from the most significant column to the least significant column) algorithm for computing a signed-binary (SB) representation of pairs of integers with minimum joint weight as well as maximum average length of zero-column runs. The proposed method speeds up the scalar multiplication of elliptic curve cryptosystems (ECC) by all known strategies, while at the same time reduces the hardware overhead. Furthermore, we present the necessary and sufficient condition that an SB representation of  $n$  integers has minimum joint weight.

## I. INTRODUCTION

The signed-binary (SB) representation uses  $\{0,1,-1\}$  ( $-1$  is also referred to as  $\bar{1}$  afterwards) to recode integers. i.e., the SB representation of an integer  $d$  is denoted as

$$d = (\cdots u_3 u_2 u_1 u_0).$$

Here

$$d = \cdots + u_3 2^3 + u_2 2^2 + u_1 2^1 + u_0$$

where  $u_i \in \{0,1,\bar{1}\}$  for all  $i$ 's. It is not difficult to see that the SB representation of a given non-negative integer is not unique. For an SB representation, define the *weight* to be the number of non-zero (either 1 or  $\bar{1}$ ) bits. If the SB representation of  $n$  integers is written one below another to form a table, the number of non-zero columns is defined as the *joint weight*. The non-uniqueness of an SB representation provides the possibility of resulting in lower weight or joint weight than using the normal binary expansion. An  $n$ -row SB representation can be split to *windows*. A *window* is a combination of consecutive columns such that the number of columns is less than or equal to  $w$ .

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  $P$  a point on an elliptic curve. This is called a scalar multiplication. Note that  $kP$  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. The representation of  $g$  and  $h$  with low joint weight can speed up the scalar multiplication in ECC as the zero-columns can be ignored. Maximizing the average length of zero-column runs is also useful for speeding up the

operation since the values of each *window* can be precomputed and stored in memory. Scanning  $g$  and  $h$  from left to right (i.e., from the most significant column to the least significant column) can reduce the amount of memory required thus is important in memory-constrained systems like smart cards.

Several papers have been involved in the matter of speeding up the scalar multiplication in ECC. In [3] Solinas presents a right-to-left method, called the Joint Sparse Form (JSF), for computing the SB representation of a pair of integers which results in minimum joint weight. The JSF has been generalized by Proos to the case of  $n$  numbers in [7]. In [4] and [10] we have introduced a method (referred to as KR method afterwards) for obtaining another type of SB representation of a pair of integers which also results in minimum joint weight but operates from left to right. These results have been extended to the case of  $n$  integers in [5]. In [8] and [9] another left-to-right method is proposed. This is similar to the KR method. However neither the JSF method nor the KR method has considered the concept of *window*. In [1] Koyama and Tsuruoka present an SB window method (KT method) which maximizes the average length of zero runs in the SB representation of a single non-negative integer.

In this paper, we extend the KT method [1] to a pair of non-negative integers. To achieve this, we improve the KR method [4] [10] by maximizing the average length of zero-column runs. This is so far the only algorithm that uses the concept of *window* to analyze more than one integer. Concerning the advantages, the improved KR method

- scans the two integers from left to right,
- uses only 9 SB bits of memory for each integer,
- results in minimum joint weight, and
- results in the maximum average length of zero-column runs.

These properties guarantee that the improved KR method speeds up the scalar multiplication of ECC by all known strategies, while at the same time reduces the hardware overhead.

There are different forms of SB representations of  $n$  integers. In this paper we propose a necessary and sufficient condition that an SB representation of  $n$  integers is optimal, i.e., is with minimum joint weight. Using the proposed condition one can easily verify whether a given SB representation is optimal or not.

The remainder of the paper is organized as follows. In Section II we give the algorithm of the improved KR method. A brief introduction to the SB window method is presented in

Section III. Section IV proposes the general necessary and sufficient condition that an SB representation of  $n$  non-negative integers is with minimum joint weight. Finally Section V concludes the paper.

## II. IMPROVED KR METHOD

In this section we propose the algorithm that computes an SB representation of two integers that has minimum joint weight and maximum average length of zero-column runs.

### A. Length of Zero-column Runs in an SB Representation

The average length of zero-column runs in the  $2 \times (L+1)$  SB table, denoted by  $Z(T)$ , is defined as follows:

$$Z(T) = \frac{1}{L+1} \sum_{i=1}^L z(i) \quad (1)$$

and

$$z(i) = \begin{cases} 1 + z(i-1) & \text{if } t_i = 0 \\ 0 & \text{if } t_i \neq 0 \text{ or } i = -1 \end{cases} \quad (2)$$

where  $0 \leq i \leq L$  and  $t_i = 0$  if both entries in column  $i$  are zero and  $t_i \neq 0$  if at least one entry in column  $i$  is non-zero.  $Z(T)$  evaluates the length of zero-column length in a 2-row SB representation table. Said another way, for the same joint weight of an SB representation, large value of  $Z(T)$  indicates that the non-zero columns are distributed more concentratively. Thus there exist fewer windows which need to be precomputed.

### B. Algorithm

The conversion from the normal binary expansion to the SB representation in our method is based on the observation used commonly in the Booth multiplication [6] of 2's complement binary numbers. Let  $d$  be an L-bit binary number which is written as

$$d = d_{L-1}2^{L-1} + d_{L-2}2^{L-2} + \dots + d_12 + d_0 \quad (3)$$

Since  $2^x = 2^{x+1} - 2^x$ ,  $d$  can also be expressed as follows:

$$d = (d_{L-1}-0)2^L + (d_{L-2}-d_{L-1})2^{L-1} + \dots + (d_1-d_2)2 + (0-d_0) \quad (4)$$

Now we have an (L+1)-bit SB representation of  $d$ :

$$d = ((d_{L-1}-0) (d_{L-2}-d_{L-1}) \dots (d_0-d_1) (0-d_0)) \quad (5)$$

Note that in the notation of  $d$  in (5) each bit can be 0, 1 or  $\bar{1}$ . This SB representation of  $d$  is defined as the *intermediate signed-binary (ISB) representation* [10]. An ISB representation has the following properties:

- 1) The sequence  $10\dots01$  does not occur anywhere in the SB representation (5) of  $d$ , where the number of zeros is greater than or equal to 0.
- 2) The sequence  $\bar{1}0\dots0\bar{1}$  does not occur anywhere in the SB representation (5) of  $d$ , where the number of zeros is greater than or equal to 0.
- 3) The leftmost non-zero digit is 1 and the rightmost non-zero digit is  $\bar{1}$ .

Our method for computing the SB representations of a pair of integers is based on the fact that the following three replacements on an ISB representation do not change the value of an integer:

- Consecutive bits  $x\bar{x}$  can be replaced by  $0x$ .
- Consecutive bits  $x0\bar{x}$  can be replaced by  $0xx$ .
- Consecutive bits  $x00\bar{x}$  can be replaced by  $0xx0x$ .

In each case  $x \in \{1, \bar{1}\}$ .

Algorithm 1 below gives the steps of our method for computing the SB representations of  $g$  and  $h$ .

### Algorithm 1

**Input:** Two L-bit binary numbers  $g$  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:** (L+1)-bit SB representation of  $g$  and  $h$  with minimum joint weight and maximum average length of zero-column runs.

1. Convert the binary expansions of  $g$  and  $h$  into:

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

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

- Replacing  $\begin{matrix} x\bar{x} \\ 0z \end{matrix}$  by  $\begin{matrix} 0x \\ 0z \end{matrix}$  . (R.I)

- Replacing  $\begin{matrix} x\bar{x} \\ y\bar{y} \end{matrix}$  by  $\begin{matrix} 0x \\ 0y \end{matrix}$  . (R.II)

- Replacing  $\begin{matrix} x0\bar{x} \\ 0y0 \end{matrix}$  by  $\begin{matrix} 0xx \\ 0y0 \end{matrix}$  . (R.III)

- Replacing  $\begin{matrix} x0\bar{x} \\ y\bar{y}0 \end{matrix}$  by  $\begin{matrix} 0xx \\ 0y0 \end{matrix}$  . (R.IV)

3. Make replacements on output of Step 2 by going from left to right and:

- Replacing  $\begin{matrix} 0x0\bar{x} \\ 000z \end{matrix}$  by  $\begin{matrix} 00xx \\ 000z \end{matrix}$  . (R.V)

- Replacing  $\begin{matrix} 0x0\bar{x} \\ 0y0\bar{y} \end{matrix}$  by  $\begin{matrix} 00xx \\ 00yy \end{matrix}$  . (R.VI)

- Replacing  $\begin{matrix} 00x00\bar{x}\bar{x} \\ 000000z \end{matrix}$  by  $\begin{matrix} 000xx0x \\ 000000z \end{matrix}$  . (R.VII)

- Replacing  $\begin{matrix} 00x00\bar{x}\bar{x} \\ 00y00\bar{y}\bar{y} \end{matrix}$  by  $\begin{matrix} 000xx0x \\ 000y0y \end{matrix}$  . (R.VIII)

Note that in Algorithm 1,

- 1)  $x \in \{1, \bar{1}\}$ ,  $y \in \{1, \bar{1}\}$  and  $z \in \{0, 1, \bar{1}\}$ .
- 2) The order of  $g$  and  $h$  has nothing to do with the operations. For instance, replacing  $\begin{matrix} x\bar{x} \\ 0z \end{matrix}$  by  $\begin{matrix} 0x \\ 0z \end{matrix}$  and replacing  $\begin{matrix} 0z \\ x\bar{x} \end{matrix}$  by  $\begin{matrix} 0z \\ 0x \end{matrix}$  are considered the same type of replacement, which is marked as *R.I*.
- 3) In both Step 2 and Step 3 the scanning from left to right should be done column by column. However, if a replacement is applied, then discard the replaced columns and continue to scan rightwards.
- 4) The first two steps are equivalent to the KR method [4].

Also note that the three steps in Algorithm 1 can be combined together by scanning  $g$  and  $h$  only once from left to right 9

columns at a time.

We state the following theorem that characterizes the improved KR method. The proof is not given in this manuscript because of space limitations.

**Theorem 1.** *The output of Algorithm 1 has minimum joint weight and maximum average length of zero-column runs among any signed-binary representations of minimum joint weight of two given integers.*

### C. Example

The following example illustrates the improved KR method and compares it with previous methods.

#### Example 1

Let  $g = 1, 930, 173, 207$  and  $h = 1, 929, 143, 809$ . The normal binary expansion of  $g$  and  $h$  is given by:

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 1110011000011000001101100010111 \\ 1110010111111000110011000000001 \end{bmatrix} \quad (6)$$

The joint weight is 22 and the average length of zero columns is 0.51613. Recall that (1) and (2) are used to evaluate the average length of zero-column runs in an SB representation. Applying Step 1 of Algorithm 1 to (6) results in:

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 100\bar{1}010\bar{1}000010\bar{1}000010\bar{1}10\bar{1}001\bar{1}100\bar{1} \\ 100\bar{1}01\bar{1}100000\bar{1}0010\bar{1}010\bar{1}00000001\bar{1} \end{bmatrix} \quad (7)$$

Applying Step 2 of Algorithm 1 to (7) results in (8). This is the output of the KR method [4] [10]:

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 100\bar{1}010\bar{1}000010\bar{1}0000100\bar{1}0\bar{1}0001100\bar{1} \\ 100\bar{1}010\bar{1}00000\bar{1}0010\bar{1}0011000000001 \end{bmatrix} \quad (8)$$

The joint weight is 14 and the average length of zero-column runs is 0.87500. Applying Step 3 of Algorithm 1 to (8) results in (9). This is the output of the improved KR method.

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 1000\bar{1}0\bar{1}0\bar{1}0000110000100\bar{1}0\bar{1}0001100\bar{1} \\ 1000\bar{1}0\bar{1}0\bar{1}00000\bar{1}000110011000000001 \end{bmatrix} \quad (9)$$

The joint weight is 14 and the average length of zero-column runs is 1.09375. On the other hand, the JSF [3] of  $g$  and  $h$  is given by:

$$\begin{bmatrix} g \\ h \end{bmatrix} = \begin{bmatrix} 100\bar{1}010\bar{1}000010\bar{1}000010\bar{1}0110010\bar{1}00\bar{1} \\ 100\bar{1}010\bar{1}00000\bar{1}0010\bar{1}010\bar{1}000000001 \end{bmatrix} \quad (10)$$

The joint weight is 14 and the average length of zero-column runs is 0.78125.

From Example 1 we see that all the three methods (KR, improved KR and JSF) result in the same joint weight which is minimum. However the improved KR method is the best among them because it results in the maximum average length of zero-column runs.

## III. SB WINDOW METHOD

In this section we analyze the SB representation tables of  $n$  non-negative integers using the *window method*. The main procedure is to split the table into several parts, namely, *windows*, with each window consisting of a certain number

of consecutive columns. Let  $w$  be the maximum width of the window. Let  $T$  denote the SB representation table to be split. Algorithm 2 [1] first splits the table into windows and then generates a list of all windows. The height of each window is  $n$  and the width is less than or equal to  $w$ .

### Algorithm 2

**Input:** SB representation table of  $n$  integers, denoted as  $T$ .

**Output:** Window list of  $T$ , denoted as  $S$ .

1. **while**  $Length(T) \geq w$
2.     **do** Let  $W$  be the left  $w$  columns of  $T$
3.         Let  $\tilde{R}$  be  $T$  excluding  $W$
4.         Let  $\tilde{W}$  be  $W$  excluding the right zero-columns
5.         Let  $\tilde{R}$  be  $R$  excluding the left zero-columns
6.         Add new window  $\tilde{W}$  to the window list  $S$
7.         Let  $T$  be  $\tilde{R}$
8.     **endwhile**
9. Add the last window  $T$  to the window list  $S$
10. **return**  $S$

The following example uses Algorithm 2 to split the SB tables in (9) and (10).

#### Example 2

Assume that the window size we choose is 3, i.e.,  $w = 3$ . The the right-hand-side of (9) can then be rewritten as

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 000 \\ 000 \end{bmatrix} \begin{bmatrix} \bar{1}\bar{1} \\ \bar{1}\bar{1} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} \bar{1} \\ \bar{1} \end{bmatrix} \begin{bmatrix} 0000 \\ 0000 \end{bmatrix} \begin{bmatrix} \bar{1}\bar{1} \\ 0\bar{1} \end{bmatrix} \begin{bmatrix} 000 \\ 000 \end{bmatrix} \begin{bmatrix} 01 \\ 11 \end{bmatrix} \begin{bmatrix} 00 \\ 00 \end{bmatrix} \begin{bmatrix} \bar{1}0\bar{1} \\ 110 \end{bmatrix} \begin{bmatrix} 000 \\ 000 \end{bmatrix} \begin{bmatrix} \bar{1}\bar{1} \\ 00 \end{bmatrix} \begin{bmatrix} 00 \\ 00 \end{bmatrix} \begin{bmatrix} \bar{1} \\ 1 \end{bmatrix}$$

where each block in the square brackets represents a window. There are totally eight windows in the output. Similarly, the the right-hand-side of (10) can be rewritten as

$$\begin{bmatrix} 1 \\ 1 \end{bmatrix} \begin{bmatrix} 00 \\ 00 \end{bmatrix} \begin{bmatrix} \bar{1}0\bar{1} \\ \bar{1}0\bar{1} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} \bar{1} \\ \bar{1} \end{bmatrix} \begin{bmatrix} 000 \\ 000 \end{bmatrix} \begin{bmatrix} \bar{1}\bar{1} \\ 00\bar{1} \end{bmatrix} \begin{bmatrix} 00 \\ 00 \end{bmatrix} \begin{bmatrix} 001 \\ 10\bar{1} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \begin{bmatrix} \bar{1}0\bar{1} \\ 10\bar{1} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \begin{bmatrix} 00 \\ 00 \end{bmatrix} \begin{bmatrix} \bar{1}0\bar{1} \\ 000 \end{bmatrix} \begin{bmatrix} 00 \\ 00 \end{bmatrix} \begin{bmatrix} \bar{1} \\ 1 \end{bmatrix}$$

Note that there are nine windows in the JSF output.

From the example we see that the improved KR method has increased the average length of zero-column runs in the gaps and reduced the number of windows that need to be precomputed.

## IV. SB REPRESENTATION OF $n$ NON-NEGATIVE INTEGERS WITH MINIMUM JOINT WEIGHT

We now consider the recoding of  $n$  non-negative integers instead of only two. We first define the *dense window* and then use it to analyze the SB representation of  $n$  integers.

**Definition 1.** A *dense window* consists of a group of consecutive non-zero columns in an SB representation. To the left or right of a dense window is either the end of the representation or a zero column.

Note that a *dense window* is different from the *window* we discussed in Section III. A dense window requires all columns be non-zero but does not limit the window width.

A *vertical separator* is a vertical line placed between two consecutive columns in a dense window. We define  $v$  to be the left column of the vertical separator. For example, if the vertical separator is placed between column  $j$  and  $j - 1$ , then

$v = j$ . Note that the rightmost column of an SB table is numbered as column 0.

Suppose the leftmost column of a dense window is column  $j_l$  and the rightmost column of the dense window is column  $j_r$ . The bit in row  $i$ , column  $j$  of the SB table is denoted as  $a_{i,j}$  ( $a_{i,j} \in \{0, 1, \bar{1}\}$ ).

**Theorem 2.** *A signed-binary representation of  $n$  non-negative integers has minimum joint weight if and only if the following condition is satisfied.*

*For each dense window in the representation, there exists at least one “vertical separator” between column  $v$  and column  $v - 1$  that simultaneously satisfies the following conditions:*

- 1)  $j_l + 1 \geq v \geq j_r$ .
- 2) If  $v \neq j_l + 1$ , then there exists at least one row  $i$  such that  $a_{i,j_l} = a_{i,j_l-1} = a_{i,j_l-2} = \dots = a_{i,v} \neq 0$ .
- 3) If  $v \neq j_l + 1$ , then for each column  $p$  with  $j_l > p > v$  there exists at least one row  $i$  such that  $a_{i,p} \neq 0$  and  $a_{i,p-1} = a_{i,p-2} = \dots = a_{i,v} = 0$ .
- 4) If  $v \neq j_r$ , then for each column  $p$  with  $v > p \geq j_r$  there exists at least one row  $i$  such that  $a_{i,p} \neq 0$  and  $a_{i,p-1} = a_{i,p-2} = \dots = a_{i,j_r} = 0$ .
- 5) If  $v \neq j_l + 1$  and  $v \neq j_r$ , then there exists at least one row  $i$  such that  $a_{i,v} \neq -a_{i,v-1}$ .

Due to its considerable length, the proof is not presented in this manuscript.

Theorem 2 serves as the necessary and sufficient condition that a given SB representation of  $n$  non-negative integers is with minimum possible joint weight. First we find all dense windows in an SB representation table. Then we try to find a decent vertical separator for each of these dense windows. If this could be done then the SB representation is optimal.

The following corollaries are straightforward conclusions from Theorem 2.

**Corollary 1.** *In a signed-binary representation of  $n$  integers that is with minimum joint weight, there is at least one zero column among any  $2n + 1$  consecutive columns.*

**Corollary 2.** *The average joint weight among all signed-binary representations of a pair of integers that have minimum joint weight is  $1/2$ .*

The following example illustrates Theorem 2.

### Example 3

Suppose an SB representation of 28 and 14 is given by:

$$\begin{bmatrix} 28 \\ 14 \end{bmatrix} = 0 \begin{bmatrix} 1110 \\ 100\bar{1} \end{bmatrix} 0$$

The joint weight of this representation is 4, which is minimum because the vertical separator can be placed between column 2 and column 3 as follows:

$$\begin{bmatrix} 28 \\ 14 \end{bmatrix} = 0 \begin{bmatrix} 11 & | & 10 \\ 10 & | & 0\bar{1} \end{bmatrix} 0$$

Note that this is not the only SB representation that is optimal. Another SB representation of 28 and 14, called the

JSF [3], is given by:

$$\begin{bmatrix} 28 \\ 14 \end{bmatrix} = \begin{bmatrix} 10 \\ 01 \end{bmatrix} 0 \begin{bmatrix} \bar{1}0 \\ 0\bar{1} \end{bmatrix} 0$$

For the two dense windows in the JSF of 28 and 14, the vertical separators can be respectively placed at any of the three possible positions. Thus the condition stated in Theorem 2 is satisfied.

However the following SB representation of 28 and 14 is not optimal.

$$\begin{bmatrix} 28 \\ 14 \end{bmatrix} = \begin{bmatrix} \bar{1}\bar{1}110 \\ 0100\bar{1} \end{bmatrix} 0$$

Because there does not exist a vertical separator that simultaneously satisfies the five conditions stated in Theorem 2.

## V. CONCLUSION

We have proposed a left-to-right algorithm for computing an SB representation of a pair of integers that has minimum joint weight as well as maximum average length of zero-column runs. This algorithm is advantageous to other known recoding methods in that it speeds up the computation in ECC by minimizing the number of operations and reducing the amount of memory required.

For  $n$  non-negative integers, there exist many different forms of SB representations that are of minimum joint weight. A necessary and sufficient condition of minimum-joint-weight SB representations is introduced. This is useful for determining whether a given SB representation is optimal or not.

## ACKNOWLEDGMENT

This work is partially supported by the US National Science Foundation under grant CCR-0429523.

## REFERENCES

- [1] K. Koyama and Y. Tsuruoka: Speeding up Elliptic Cryptosystems by Using a Signed-Binary Window Method, *Advances in Cryptology-Crypto 1992*, LNCS 740, Springer-Verlag, 1993.
- [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] 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.
- [4] R. S. Katti and Xiaoyu Ruan: Left-to-Right Signed-Digit Representation for Elliptic Curve Cryptography, *Proceedings of 2004 IEEE International Symposium on Circuits and Systems*, pp. II365-II368, May 2004.
- [5] Xiaoyu Ruan and R. S. Katti: Low-Weight Left-to-Right Binary Signed-Digit Representation for  $n$  Integers, *Proceedings of 2004 IEEE International Symposium on Information Theory*, pp. 517, June 2004.
- [6] A. D. Booth: A Signed Binary Multiplication Technique, *Q. J. Mech. Appl. Math.*, 4(2), pp. 236-240, 1951.
- [7] 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.
- [8] P. J. Grabner, C. Heubberger and H. Prodinger: Distribution Results for Low-Weight Binary Representations for Pairs of Integers, to appear in *Theoretical Computer Science*.
- [9] P. J. Grabner, C. Heubberger, H. Prodinger and J. Thuswaldner: Analysis of Linear Combination Algorithms in Cryptography, Preprint, available at <http://www.opt.math.tu-graz.ac.at/cheub/publications/windows.pdf>
- [10] Xiaoyu Ruan and R. S. Katti: Left-to-Right Signed-Binary Representation of a Pair of Integers, *IEEE Transactions on Computers*, Vol. 54, No. 2, pp. 124-131, Feb. 2005.