

# Efficient Left-to-Right Multi-Exponentiations

Erik Dahmen,<sup>1</sup> Katsuyuki Okeya,<sup>2</sup> and Tsuyoshi Takagi<sup>3</sup>

<sup>1</sup> Technische Universität Darmstadt, Fachbereich Informatik,  
Hochschulstr.10, D-64289 Darmstadt, Germany  
`dahmen@rbg.informatik.tu-darmstadt.de`

<sup>2</sup> Hitachi, Ltd., Systems Development Laboratory,  
1099, Ohzenji, Asao-ku, Kawasaki-shi, Kanagawa-ken, 215-0013, Japan  
`ka-okeya@sdl.hitachi.co.jp`

<sup>3</sup> Future University - Hakodate,  
116-2 Kamedanakano-cho Hakodate Hokkaido, 041-8655, Japan  
`takagi@fun.ac.jp`

November 1, 2005

**Abstract.** Multi-exponent algorithms are frequently used in cryptographic protocols, for example, digital signature algorithms, some commitment schemes, etc. The previously known efficient algorithms use special exponent recording algorithms like  $w$ NAF or the joint sparse form. Those algorithms can only be applied starting at the least significant bit. Therefore, they require additional memory for the recoded exponent, and they are not suitable for the implementation on memory constraint devices. In this paper, we propose two multi-exponent algorithms which can be applied starting at the most significant bit. The first algorithm performs a special joint sliding window conversion over a mutual opposite form of signed binary strings. We need to scan only  $k + 1$  joint bits for generating the efficient chains with  $k$  bases. The second algorithm uses an existing exponent recording algorithm for single exponentiation, namely  $w$ MOF that which be applied starting at most significant bit. This proposed algorithm scans only  $w$  bits for each exponent. Therefore, the proposed schemes can achieve on-the-fly multi-exponent scalar multiplication for elliptic curve cryptosystems. We also explicitly calculate the joint non-zero density for the first scheme and show that for several numbers of exponents it is optimal among all joint signed forms using digits  $\{-1, 0, 1\}$ , e.g.,  $1/2$  for  $k = 2$  and  $23/39$  for  $k = 3$ . The average density of multiplications for the second proposed algorithm equals to  $k/(w + 1)$ . Based on these formula we compare the cost for the multi exponentiation including the pre-computation.

**Keywords:** *elliptic curve cryptosystem, multi-exponent, joint sparse form, left-to-right, mutual opposite form*

## 1 Introduction

Nowadays cryptography is often implemented on smartcards. There is large number of applications for this devices, e.g. authentication and digital signatures.

The problem with smartcards is that they have high constraints on memory and computational power. The elliptic curve cryptosystem (ECC) [Kob87,Mil86] is an efficient cryptosystem, which can attain high security with very short key length. Therefore, ECC is suitable for implementation on memory-constraint devices.

The most fundamental operation used for the ECC is a multi exponentiation, which computes

$$\prod_{j=1}^k g_j^{d_j},$$

for given integers  $d_j$  and  $g_j \in G$  with  $j = 1 \dots k$ , where  $G$  is a group. The research goal from practical requirement is to efficiently compute the multi exponentiation by minimizing both memory usage and computational costs (See, for example, [Ava02,Gor98,Möl01]).

Typically a multi-exponent algorithm consists of the following three stages: (1) The exponent recording stage, (2) the pre-computation stage and (3) the evaluation stage. Currently used methods for the exponent recording stage are the width- $w$  non adjacent form ( $w$ NAF) [Sol00,BSS99,MOC97] and the joint sparse form (JSF) [Pro03,Sol01]. For the pre-computation and the evaluation stage common used methods are the Interleave Method [Möl01] and the Shamir Method [Gor98]. In the exponent recording stage, the exponents are rewritten to speed up the evaluation stage. This is achieved by decreasing the number of non-zero entries in the exponents. The actual exponentiation is computed in the remaining two stages.

The strategies described above have the following problem.  $w$ NAF and JSF can only be applied starting at the least significant bit (right-to-left) while the methods for the evaluation stage parse the exponents starting at the most significant bit (left-to-right). The disadvantage of using both directions is that the exponent recording has to be finished before the exponentiation can begin. Therefore all recorded exponents have to be stored in extra memory. There are methods to perform the evaluation stage from right-to-left but those methods are much slower than their left-to-right counterparts [CMO98]. Therefore those strategies are not suitable for smartcards.

A possible solution to this problem is an exponent recording algorithm which can be applied from left-to-right. The benefit of such an algorithm is, that the exponent recording stage becomes dispensable because the recording of the exponents can be performed on-the-fly in the evaluation stage.

For recording one exponent at a time there already exists a left-to-right analogue to  $w$ NAF. It is called the width- $w$  mutual opposite form ( $w$ MOF) [OSST04]. It can be generated by applying a width- $w$  sliding window method on a mutual opposite form (MOF) [OSST04] of an integer. Currently  $w$ MOF is used only for single exponentiation and there is no left-to-right analogue to JSF.

## 1.1 Contribution of this Paper

In this paper we propose two new algorithms for multi exponentiation with an arbitrary number of exponents. Both algorithms are applied solely from left-to-

right, therefore they require no exponent recording stage and the conversion is performed during the evaluation stage.

The first algorithm is an exponent recording algorithm which simultaneously records all exponents from left-to-right. This is achieved by applying a joint conversion on the MOF representations of the exponents. The main idea is that after scanning at most  $k \cdot (k + 1)$  digits simultaneously, we can always create a zero column. In most cases it is sufficient to scan less bits to create a zero column. In the second algorithm we show how  $w$ MOF can be used with the Interleave Method to perform a multi exponentiation. We need to scan only  $w$  bits for each exponent at a time and then use a special conversion to get the  $w$ MOF representation of the  $w$  bits.

We precisely estimate the efficiency of the proposed schemes. For the first algorithm, we develop an explicit formula to calculate the average density of multiplications needed for multi exponentiation. The probability that a zero column can be created in a certain step is calculated. The computational costs for the second algorithm are gathered from the properties of  $w$ MOF and the Interleave Method.

We will also compare the new methods with the previously known methods. It will turn out that the computational costs equate, but the required memory can be dramatically reduced. A comparison for arbitrary  $k$  can be found in table 2.

This paper is organized as follows: In section 2 we give an overview of multi exponentiation. We will explain the three stages of an multi-exponent algorithm in detail and review currently known methods, e.g. the Interleave Method, the Shamir Method,  $w$ NAF,  $w$ MOF and JSF. In section 3 the new left-to-right algorithms for computing multi exponentiations are described and their computational costs are estimated. Then we compare them with the right-to-left methods. In section 4 conclusion remark is stated.

## 2 Overview of multi exponentiation

In this section, we review some efficient multi-exponent algorithms.

At first we introduce some frequently used terms. A binary representation of an  $n$ -bit integer  $d$  is a vector  $(d[n - 1], \dots, d[0])$  such that

$$d = \sum_{i=0}^{n-1} d[i] \cdot 2^i$$

where  $d[i] \in D, i = 0, \dots, n - 1$ .  $D$  is called the digit set. For the standard binary representation we have  $D = \{0, 1\}$ . If  $D$  also contains negative elements, we call  $(d[n - 1], \dots, d[0])$  a signed binary representation of the integer  $d$ .  $d$  is called an  $n$ -bit integer, if  $2^{n-1} \leq d \leq 2^n - 1$ . Note that we can represent any  $n$ -bit integer with  $(n + m)$ -bits by setting  $(d[n + m - 1], \dots, d[n]) = (0, \dots, 0)$ . The Hamming weight of a signed binary representation is its number of non zero entries. The non zero density (nzd) of a signed binary representation is its

Hamming weight divided by its bitlength. If we consider more than one signed binary representation simultaneously, i.e. dealing with more than one exponent, it is useful to examine non zero columns rather than non zero entries. The number of non zero columns is given by the joint Hamming weight. According to this we define the joint non zero density (joint nzd) as the quotient of the joint Hamming weight and the maximum bitlength of the exponents. All exponents with lesser bitlength are padded with zeroes on the left side.

Let  $K = GF(p)$  be a finite field, where  $p > 3$  is a prime. Let  $E$  be an elliptic curve over  $K$ . Since the elliptic curve is an additive group, multiplication becomes addition and exponentiation becomes multiplication. Hence, when using elliptic curves, a multi exponentiation becomes a multi scalar multiplication. In this paper we consider only algorithms which are suitable for elliptic curves, nevertheless we write them in a multiplicative fashion because in our opinion this enhances the readability. The elliptic curve  $E$  has an Abelian group structure with identity element  $\mathcal{O}$  called the point of infinity. A point  $P \in E$  is represented as  $P = (x, y)$ . The inverse of point  $P = (x, y)$  is equal to  $-P = (x, -y)$ , hence it can be computed virtually free. Because of this, it is advisable to use a signed binary representation of the exponents [MO90]. The elliptic curve additions  $P+Q$  and  $2P$  are denoted by ECADD and ECDBL, respectively, where  $P, Q \in E$ .

## 2.1 Multi Exponent Algorithms

Window-based algorithms are well-established methods for efficiently computing multi exponentiations. The exponents  $d_j$  are usually passed to the algorithm in their binary representation. In this section we describe the three stages of a multi-exponent algorithm and explain some common used variants.

*Exponent Recording Stage* The purpose of the exponent recording stage is to decrease the (joint) Hamming weight of the exponents. In order to do this a exponent recording algorithm is applied either to each exponent separately (to decrease the Hamming weight) or to all exponents simultaneously (to decrease the joint Hamming weight). In section 2.2 we will shortly describe the properties of the following three exponent recording algorithms: The width- $w$  non-adjacent form ( $w$ NAF), the width- $w$  mutual opposite form ( $w$ MOF) and the joint sparse form (JSF).

For the pre-computation and the evaluation stage there are two common methods. The Interleave Method [Möl01] and the Shamir Method [Gor98]. We will compare both stages of these methods separately.

*Pre-Computation Stage* In the pre-computation stage the Interleave and the Shamir Method computes

$$g_j^e, \forall e \in D \setminus \{0\}, j = 1, \dots, k, \quad \prod_{j=1}^k g_j^e, \forall e \in D^k \setminus \{0, \dots, 0\}$$

respectively. Since we limit our considerations to elliptic curves the element  $g_i^{-e}$  can be computed from  $g_i^e$  on-the-fly in the evaluation stage and doesn't have to

be precomputed. Note that the points  $g_j^1$  are trivially given as input and also don't have to be precomputed. The number of precomputed points is

$$k \cdot \frac{|D| - 1}{2} - k, \quad \frac{|D|^k - 1}{2} - k$$

for the Interleave, Shamir Method respectively. The precomputed points are stored in a table to be accessed later.

*Evaluation Stage* In the evaluation stage the  $k$  exponents  $d_1, \dots, d_k$  are parsed bitwise, starting at the most significant bit (left-to-right). At first an accumulator  $X$  is initialized with the neutral group element. Let  $d_j[i]$  denote the  $i$ -th bit of exponent  $j$ . The following two algorithms describe the evaluation stage of both methods.

Interleave Method	Shamir Method
<ol style="list-style-type: none"> <li>1. for <math>i = n - 1</math> to <math>0</math> do               <ol style="list-style-type: none"> <li>1.1. <math>X \leftarrow X^2</math></li> <li>1.2. for <math>j = 1</math> to <math>k</math> do                   <ol style="list-style-type: none"> <li>1.2.1. if <math>d_j[i] \neq 0</math> <ol style="list-style-type: none"> <li>1. <math>X \leftarrow X \cdot g_j^{d_j[i]}</math></li> </ol> </li> </ol> </li> </ol> </li> </ol>	<ol style="list-style-type: none"> <li>1. for <math>i = n - 1</math> to <math>0</math> do               <ol style="list-style-type: none"> <li>1.1. <math>X \leftarrow X^2</math></li> <li>1.2. if <math>(d_1[i], \dots, d_k[i]) \neq (0, \dots, 0)</math> <ol style="list-style-type: none"> <li>1.2.1. <math>X \leftarrow X \cdot \prod_{j=1}^k g_j^{d_j[i]}</math></li> </ol> </li> </ol> </li> </ol>

Note that  $g_j^{d_i}$  and  $\prod_{j=1}^k g_j^{d_i}$  are not actually computed, but obtained by a lookup in the table prepared in the pre-computation stage. Now we will analyse the average density of the operation used in both methods. We distinguish between squarings and multiplications. The average density of squarings is 1 for both methods, since a squaring is always performed. For the Shamir Method a multiplication is performed every time a non-zero column is found. Therefore the average density of multiplications is the joint nzd of the exponents. For the Interleave Method a multiplication is performed for every single non zero entry in the exponents. Therefore the average density of multiplications is  $k$  times the nzd of the exponents.

From the computational costs we read that the Shamir Method should be used with an exponent redording algorithm which minimizes the joint nzd, while the Interleave Method should be used with an exponent redording algorithm which minimizes the nzd of each exponent seperately.

## 2.2 Known exponent recording algorithms

In this section we describe the properties of three common used exponent recording algorithms.

**wNAF and wMOF** First we review the definition of wNAF as stated in [Sol00].

**Definition 1 (wNAF).** *A sequence of signed digits is called wNAF if the following three properties hold:*

1. *The most significant non-zero bit is positive.*

2. Among any  $w$  consecutive digits, at most one is non-zero.
3. Each non-zero digit is odd and less than  $2^{w-1}$  in absolute value.

An algorithm for the generation of  $w$ NAF was proposed by Solinas [Sol00]. The algorithm parses the integer starting at the least significant bit, i.e. right-to-left.  $w$ NAF uses the digit set  $D_w = \{0, \pm 1, \dots, \pm 2^{w-1} - 1\}$ . The average non-zero-density of  $w$ NAF is asymptotically  $1/(w+1)$  for  $n \rightarrow \infty$ .

We now explain the mutual opposite form (MOF) as stated in [OSST04], which is the canonical form of signed binary representation.

**Definition 2 (MOF).** *The  $n$ -bit mutual opposite form (MOF) is an  $n$ -bit signed binary string that satisfies the following properties:*

1. The signs of adjacent non-zero bits (without considering zero bits) are opposite.
2. The most non-zero bit and the least non-zero bit are 1 and  $\bar{1}$ , respectively, unless all bits are zero.

The MOF representation  $\mu$  of a given integer  $d$  can be computed by  $\mu = 2d \ominus d$ , where  $\ominus$  denotes the bitwise subtraction. This operation generates no carry over, therefore the conversion can be performed from left-to-right. MOF uses the digit set  $D = \{0, \pm 1\}$ . The MOF representation of an integer is unique determined and its non-zero-density is  $1/2$ . Next we describe the natural extension to MOF which is called  $w$ MOF as stated in [OSST04].

**Definition 3 ( $w$ MOF).** *A sequence of signed digits is called  $w$ MOF if the following three properties hold:*

1. The most significant non-zero bit is positive.
2. All but the least significant non-zero digit  $x$  are adjoint by  $w-1$  zeros as follows:
  - in case of  $2^{k-1} < |x| < 2^k$  for an integer  $2 \leq k \leq w-1$  the pattern equals  $\underbrace{0 \dots 0}_k x \underbrace{0 \dots 0}_{w-k-1}$ ,
  - in case of  $|x| = 1$  either the pattern equals  $x \underbrace{0 \dots 0}_{w-1}$  and the next lower non-zero digit has opposite sign from  $x$  or the pattern equals  $0x \underbrace{0 \dots 0}_{w-2}$

*and the next lower non-zero digit has the same sign as  $x$ .*

*If  $x$  is the least significant non-zero digit, it is possible that the number of right-hand adjacent zeros is smaller than stated above. In addition it is not possible that the last non-zero digit is a 1 following any non-zero digit.*

3. Each non-zero digit is odd and less than  $2^{w-1}$  in absolute value.

$w$ MOF can be constructed by applying width- $w$  sliding window method on MOF [OSST04]. The window method is applied starting at the most significant bit, i.e. left-to-right.  $w$ MOF uses the digit set  $D_w = \{0, \pm 1, \dots, \pm 2^{w-1} - 1\}$  and its nzd is asymptotically  $1/(w+1)$  for  $n \rightarrow \infty$ .

Obviously, when applied on multiple exponents, MOF,  $w$ MOF and  $w$ NAF minimizes the nzd of each exponent separately. Therefore, when performing a multi exponentiation one should use the Interleave Method.

**Joint Sparse Form** We shortly describe JSF for  $k$   $n$ -bit signed binary strings as defined in [Pro03]. Let  $d_j[i]$  denote the  $i$ -th bit of string  $j$ ,  $j = 0, \dots, k-1$  and  $d.[i]$  the  $i$ -th column of all strings,  $i = 0, \dots, n-1$ .

**Definition 4 (JSF).**  $k$   $n$ -bit signed binary strings  $d_j$ ,  $j = 0, \dots, k-1$  are called a JSF if the following three properties hold:

1. For each non zero column  $d.[i]$  there exists an  $j \in \{0, \dots, k-1\}$  such that
  - $d_j[i] \neq 0$  and
  - either  $i = 0$ , or there exists an integer  $l < i$  such that
 
$$d_j[i-1] = d_j[i-2] = \dots = d_j[l] = 0$$
 and either  $l = 0$  or  $d.[l]$  is a zero column
2. No two consecutive bits in a row are  $1\bar{1}$  or  $\bar{1}1$
3. If there exists an  $j \in \{0, \dots, k-1\}$  and integers  $i, l$  with  $l < i$  such that  $d_j[i+1] \neq d_j[i]$  and  $d_j[i] = d_j[i-1] = \dots = d_j[l] \neq 0$  then  $d.[i+1]$  is a zero column

JSF uses the digit set  $D = \{0, \pm 1\}$ . The joint nzd can be calculated using the formula  $1 - 1/c_k$ , where

$$c_k = \frac{1}{2^k} \left( 3 + \sum_{i=1}^{k-1} \binom{k}{i} (c_i + 1) \right)$$

and  $c_1 = 3/2$ . JSF is a exponent recording algorithm which minimizes the joint nzd of the exponents. Hence, one should use the Shamir Method when performing an multi exponentiation.

### 2.3 Left-to-Right vs. Right-to-Left

The Shamir and the Interleave Method are so called left-to-right binary methods, because they start the exponentiation at the most significant bit. There are also methods which start the exponentiation at the least significant bit, those are called right-to-left binary methods. In this section we explain why the left-to-right methods are preferable for the ECC. At first we describe the left-to-right and the right-to-left binary methods in their fundamental versions, namely for one  $n$ -bit scalar which is given in its binary representation, i.e.  $D = \{0, 1\}$  holds. Further we assume that  $d_{n-1} = 1$  holds.

Binary Method, l-t-r	Binary Method, r-t-l
input: $P \in E$ , scalar $d$ output: scalar multiplication $dP$ <ol style="list-style-type: none"> <li>1. <math>Q \leftarrow P</math></li> <li>2. for <math>i = n-2</math> to <math>0</math> do               <ol style="list-style-type: none"> <li>2.1. <math>Q \leftarrow \text{ECDBL}(Q)</math></li> <li>2.2. if <math>d[i] = 1</math> <ol style="list-style-type: none"> <li>2.2.1. <math>Q \leftarrow \text{ECADD}(Q, P)</math></li> </ol> </li> </ol> </li> <li>3. return <math>Q</math></li> </ol>	input: $P \in E$ , scalar $d$ output: scalar multiplication $dP$ <ol style="list-style-type: none"> <li>1. <math>Q_1 \leftarrow P, Q_2 \leftarrow \mathcal{O}</math></li> <li>2. for <math>i = 0</math> to <math>n-1</math> do               <ol style="list-style-type: none"> <li>2.1. if <math>d[i] = 1</math> <ol style="list-style-type: none"> <li>2.1.1. <math>Q_2 \leftarrow \text{ECADD}(Q_2, Q_1)</math></li> </ol> </li> <li>2.2. <math>Q_1 \leftarrow \text{ECDBL}(Q_1)</math></li> </ol> </li> <li>3. return <math>Q_2</math></li> </ol>

Though in general both methods provide the same efficiency, the left-to-right method is preferable due to the following reasons:

- The left-to-right method can be adjusted for representations using a larger digit sets  $D$ , like  $wNAF$  or  $wMOF$ , in a more efficient way than the right-to-left method.
- The ECADD step in the left-to-right method has the fixed input  $tP$ ,  $t \in D \setminus \{0\}$ . Therefore it is possible to speed up these steps if  $tP$  is expressed in affine coordinates for each  $t \in D \setminus \{0\}$ , since some operations are negligible in this case. The improvement for a 160-bit scalar multiplication is about 15% with NAF over right-to-left scheme in the Jacobian coordinates [CMO98].
- The right-to-left method needs an auxiliary register for storing  $2^i P$ .

Since we use a left-to-right binary method, it is desirable to perform the exponent recording also from left-to-right, because in that case we don't have to record the exponents prior to the exponentiation. They are rather recorded during the evaluation stage on-the-fly. This will save us the memory for the recorded exponents which is important on memory constraint devices like smartcards. From the algorithms stated in section 2.2, only MOF and  $wMOF$  can be generated from left-to-right.  $wNAF$  and JSF need an exponent recording stage and extra memory. While  $wMOF$  represents the left-to-right analogue of  $wNAF$  there is currently no algorithm to generate JSF from left-to-right.

### 3 Proposed Schemes

In this section we will describe the two new algorithms for multi exponentiation. In section 3.1 we present a new left-to-right exponent recording algorithm for an arbitrary number of MOF strings. It will turn out that concerning the joint nzd it is the left-to-right analogue of JSF. In section 3.2 we show how  $wMOF$  can be used in conjunction with the Interleave Method to perform a multi exponentiation. In section 3.3 we compare both new methods and the methods from section 2.

#### 3.1 New left-to-right exponent recoding algorithm

At first we state some properties which are important for the algorithm. Then we explicitly write down the algorithm and at last we show how the joint nzd can be calculated.

**Properties and Algorithm** In order to design the new recoding algorithm, we construct two lemmas on the property of MOF.

*Notation:* We define a function  $eval(\cdot)$  from a signed binary representation to an integer as  $eval(s) = \sum_{i=0}^{n-1} s[i] \cdot 2^i$  for a signed binary representation  $s = (s[n-1], s[n-2], \dots, s[0])$ . For a MOF  $\mu$ , we define  $Z(\mu)$  as

$$Z(\mu) = \{z \mid \text{there exists } \delta \text{ such that } \delta[z] = 0 \text{ and } eval(\delta) = eval(\mu)\}.$$

*Example 1.*  $\mu = 10\bar{1}$ . Then,  $Z(10\bar{1}) = \{2, 1\}$ . In fact, we have  $\delta_1 = 011$  and  $\delta_2 = 10\bar{1}$ .

Regarding  $Z(\mu)$ , we have the following lemmas:

**Lemma 1.** *Assume that an  $n$ -bit MOF  $\mu$  is not 0. Then  $Z(\mu) = \{0, \dots, n - 1\} \setminus \{f\}$ , where  $f$  is the index of the least non-zero digit.*

**Lemma 2.** *For any  $k$  ( $k + 1$ )-bit MOF  $\mu_j$ ,*

$$\bigcap_{j=1}^k Z(\mu_j) \neq \emptyset.$$

*In other words, there exists an index  $z$  such that, for any  $j$ , there exists a signed binary representation  $\delta_j$  with  $\delta_j[z] = 0$  and  $\text{eval}(\delta_j) = \text{eval}(\mu_j)$ .*

Let  $\mu_j[u..l] = (\mu_j[u], \mu_j[u-1], \dots, \mu_j[l+1], \mu_j[l])$ . The new recoding algorithm for general  $k$  terms is as follows:

**Left-to-Right Recording for General Terms** \_\_\_\_\_

input:  $k$   $n$ -digit MOF strings:  $\mu_1, \dots, \mu_k$

output: Conversion  $\delta_1, \dots, \delta_k$  of  $\mu_1, \dots, \mu_k$  with minimal Joint-Hamming-Weight

1. For  $u = n - 1$  downto 0 do
  - 1.1. For  $b = 1$  to  $k + 1$  do
    - 1.1.1. Let  $l \leftarrow u - b + 1$
    - 1.1.2. Compute  $Z \leftarrow \bigcap_{j=1}^k Z(\mu_j[u..l])$
    - 1.1.3. If  $Z \neq \emptyset$ 
      1.  $z \leftarrow \max\{\tilde{z} : \tilde{z} \in Z\}$ ,
      2.  $\delta_j[u..l] \leftarrow \text{convert}(\mu_j[u..l], z), \forall j = 1, \dots, k$
      3.  $u \leftarrow u - b + 1$
      4. break for-loop
2. return  $\delta_1, \delta_2, \dots, \delta_k$

The algorithm parses the MOF strings, and converts them in windows of length 1 to  $k + 1$ . Because of Lemma 2, the iteration at Step 1.1 terminates when  $b$  becomes  $k + 1$ .

The *convert* routine is based on the observation that  $x0\dots0\bar{x} = 0x\dots xx$ , where  $x$  is non-zero and  $\bar{x} = -x$  holds. It works as follows:

**Conversion routine** \_\_\_\_\_

input:  $(u - l + 1)$ -digit MOF string:  $\mu$  and the digit to convert  $z$

output: Conversion  $\delta$  of  $\mu$  where  $\delta[z] = 0$

1. if  $\mu[z] \neq 0$ 
  - 1.1.  $j \leftarrow z - 1$
  - 1.2. while  $\mu[j] = 0$ 
    - 1.2.1.  $j \leftarrow j - 1$
  - 1.3.  $\delta[z] \leftarrow 0$
  - 1.4.  $\delta[z - 1], \dots, \delta[j] \leftarrow \mu[z], \dots, \mu[z]$
  - 1.5.  $\delta[u], \dots, \delta[z + 1] \leftarrow \mu[u], \dots, \mu[z + 1]$
  - 1.6.  $\delta[j - 1], \dots, \delta[l] \leftarrow \mu[j - 1], \dots, \mu[l]$
2. else
  - 2.1.  $\delta \leftarrow \mu$
3. return  $\delta$

Note that in Step 1.2. such a digit always exists since  $z \neq f$ . From the MOF attributes we also know that  $\mu[z] = -\mu[j]$  holds. The conversion  $x0\dots0\bar{x} \rightarrow 0x\dots xx$ , is applied only on the substring  $\mu[z..j]$  (step, 1.3., 1.4.), the remaining bits stay the same (step 1.5., 1.6.).

**Average Joint Non-Zero Density** Next, we estimate the average joint non-zero density of the proposed algorithm. In order to do this, we first define the number  $C_b$ .

$C_b$  is the number of string combinations converted at the  $b$ -th iteration of Step 1.1 of Left-to-Right Recording for General Terms.

$C_b$  is a proportion of some convertible strings. At the  $b$ -th iteration we examine  $b$  bits of each of the  $k$  integers. Therefore, there is a total number of  $2^{kb}$  combinations of strings which have to be considered for conversion.  $C_b$  is then the proportion of these strings which can be converted, i.e.  $2^{kb} - C_b$  is the number of strings which cannot be converted. Using above  $C_b$ , the average joint non-zero density is as follows:

**Proposition 1.** *The average joint non-zero density of Left-to-Right Recording for General Terms with  $k$  terms is:*

$$nzd_k = 1 - \frac{1}{\sum_{b=1}^{k+1} \frac{C_b}{2^{kb}} \cdot b}.$$

Next, we estimate  $C_b$ . If a string combination can be converted in iteration  $b$  it clearly cannot be converted in iteration  $b - 1$ . Therefore at first, we have to calculate the number of string combinations which cannot be converted in iteration  $b - 1$ . From Lemma 1 we know that a string combination cannot be converted, if for each of the  $b - 1$  columns, at least one of the  $k$  strings has its first non zero bit in this column. A fnz combination is a matrix  $F \in \{0, 1\}^{k \times b-1}$ , where  $(f_{ij}) = 1$  if and only if the first non zero bit of string  $i$  is in column  $j$ .

**Lemma 3.** *There are  $\binom{k}{b-1}$  fnz combinations for given  $k$  and  $b$  such that the corresponding string combinations cannot be converted.*

Now that we know how many fnz combinations there are, we have to classify them. Let  $D \in \mathbb{N}^{\binom{k}{b-1} \times b-1}$ , where  $(d_{ij})$  is the number of first non zero bits in column  $j$  for fnz combination  $i$ .  $D$  can be generated using the following method:

Generation of  $D$

---

1.  $i \leftarrow 1$
  2. for  $j_1 = 1$  to  $k - (b - 1) + 1$  do
    - 2.1. for  $j_2 = 1$  to  $k - (b - 1) + 1 - (j_1 - 1)$  do
      - $\vdots$
      - 2.1.1. for  $j_{b-1} = 1$  to  $k - (b - 1) + 1 - \sum_{l=1}^{b-2} (j_l - 1)$  do
        1.  $D_i \leftarrow (j_1, j_2, \dots, j_{b-1})$
        2.  $i \leftarrow i + 1$
  3. return  $D$
-

Since Lemma 3 only gives us the number of fnz combinations with respect to row permutations, we need the number of possible row permutations for each fnz combination. We also need the number of string combinations which correspond to a given fnz combination, i.e. all string combinations with the same fnz combination. Then we have the number of all string combinations which cannot be converted in iteration  $b - 1$ . Next we want to know which of them can be converted in the next,  $b$ -th iteration. Since the algorithm scans one more column each iteration we need the number of columns, which can be appended to a certain fnz combination such that a joint zero column can be created in iteration  $b$ .

**Lemma 4.**

$$C_b = \sum_{i=1}^{\binom{k}{b-1}} P_i \cdot M_i \cdot N_i$$

where  $P_i$  is the number of row permutations,  $M_i$  the number of strings corresponding and  $N_i$  the number of columns which can be appended to fnz combination  $i$ .

**Lemma 5.** For  $i = 1, \dots, \binom{k}{b-1}$  and  $D = (d_{ij})$  as above, we have the following:

- i)  $P_i = \binom{k}{d_{i1}} \binom{k-d_{i1}}{d_{i2}} \dots \binom{k-(d_{i1}+\dots+d_{i,b-2})}{d_{i,b-1}}$
- ii)  $M_i = \prod_{j=1}^{b-1} 2^{(j-1) \cdot d_{ij}}$
- iii)  $N_i = 1 + \sum_{c=1}^{b-1} \left( (-1)^{c-1} \sum_{1 \leq j_1 < \dots < j_c \leq b-1} 2^{k-(d_{ij_1}+\dots+d_{ij_c})} \right)$

### 3.2 Interleave using $w$ MOF

This section is organized as the previous one. At first we state some properties and the explicit algorithm. Then we estimate the computational costs.

**Properties and Algorithm** The goal of the Interleave with  $w$ MOF Method is to perform the exponent recording on-the-fly during the evaluation stage, rather than in a separate step. This is possible since  $w$ MOF can be generated from left-to-right. So obviously we don't need an exponent recording stage.

The precomputation stage is the same as in section 2, namely the points

$$g_j^e, \forall e \in \{\pm 3, \dots, \pm 2^{w-1} - 1\}, j = 1, \dots, k$$

are computed. For the elliptic curve case the negative exponents are of course omitted.

The evaluation stage is modified such that the conversion to  $w$ MOF is applied on-the-fly. Let  $\ll$  denote a right shift. The evaluation stage is then

---

**Interleave Method with  $w$ MOF**

---

input: Bases  $g_1, \dots, g_k$  and  $n$  bit exponents  $d_1, \dots, d_k$  in their binary representationoutput:  $\prod_{j=1}^k g_j^{d_j}$ 

1.  $d_j[n] \leftarrow 0, d_j[-1], \dots, d_j[-w] \leftarrow 0, \dots, 0, \forall j = 1, \dots, k$
  2.  $X \leftarrow 1$
  3. for  $i = n$  to 0 do
    - 3.1.  $X \leftarrow X^2$
    - 3.2. for  $j = 1$  to  $k$  do
      - 3.2.1. if  $\delta_j = 0$  and  $d_j[i] \neq d_j[i - 1]$ 
        - 3.2.1.1  $\delta_j \leftarrow \text{convert}(d_j[i - 1] - d_j[i], d_j[i - 2] - d_j[i - 1], \dots, d_j[i - w] - d_j[i - w + 1])$
      - 3.2.2. if  $\delta_j[w - 1] \neq 0$ 
        - 3.2.2.1  $X \leftarrow X \cdot g_j^{\delta_j[w - 1]}$
      - 3.2.3.  $\delta_j \ll 1$
  4. return  $X$
- 

In step 1.2.1.1. the conversion to  $w$ MOF is applied to the current  $w + 1$  bits of exponent  $d_j$  and the result is stored in the  $w$ -bit temporary variable  $\delta_j$ . This variable is used in  $w$  succeeding steps of the exponentiation. In step 1.2.3.  $\delta_j$  is shifted to the right by one, therefore we always have to consider the most significant bit of  $\delta_j$  for the exponentiation. After  $w$  steps  $\delta_j = 0$  clearly holds and a new conversion is applied. The usage of the second constraint in step 1.2.1. is the following: If  $\delta_j = 0$  and  $d_j[i] = d_j[i - 1]$  holds, the converted  $w$ MOF string will have a zero entry at the most significant bit. Therefore no exponentiation is performed whatsoever and we can save us the conversion. The conversion from MOF to  $w$ MOF is the following

---

**Conversion routine from MOF to  $w$ MOF**

---

input:  $w$ -digit MOF string  $\mu$ output:  $w$  digit  $w$ MOF representation  $\delta$  of  $\mu$ 

1.  $i \leftarrow 0$
  2. while  $\mu[i] = 0$ 
    - 2.1.  $\delta[i] \leftarrow 0$
    - 2.2.  $i \leftarrow i + 1$
  3.  $\delta[i] \leftarrow \sum_{j=i}^{w-1} \mu[j] \cdot 2^{i-j}$
  4. while  $i \leq w$ 
    - 4.1.  $i \leftarrow i + 1$
    - 4.2.  $\delta[i] \leftarrow 0$
  5. return  $\delta$
- 

Note that this algorithm is only used to convert  $w$  bits at a time. Therefore it doesn't matter that it is performed in a right-to-left fashion.

**Computational costs** The computational costs of the Interleave with  $w$ MOF method are fairly easy to compute. From section 2.1 and 2.2 it immediately follows that the average density of squarings and multiplications needed is 1 and  $k/(w + 1)$  respectively.

### 3.3 Comparison

In this section at first we compare the Shamir and the Interleave Method in conjunction with the adequate exponent recording algorithm according to section 2.1. We compare the number of precomputed points and the average density of multiplications. In tables 1 and 2 we compare those numbers for different values of  $k$ . Here newLTR denotes the algorithm of section 3.1.

Method	# precomputed points	average density of multiplications
$k = 2$		
Shamir + JSF	2	1/2
Shamir + newLTR	2	1/2
Interleave + 2NAF	0	2/3
Interleave + 2MOF	0	2/3
Interleave + 3NAF	2	1/2
Interleave + 3MOF	2	1/2
$k = 3$		
Shamir + JSF	10	23/39
Shamir + newLTR	10	23/39
Interleave + 2NAF	0	1
Interleave + 2MOF	0	1
Interleave + 3NAF	3	3/4
Interleave + 3MOF	3	3/4

**Table 1.** Comparison for  $k = 2$  and  $k = 3$

Method	# precomputed points	average density of multiplications
Shamir + JSF	$(3^k - 1)/2 - k$	$1 - 1/c_k$
Shamir + newLTR	$(3^k - 1)/2 - k$	$1 - 1/nzd_k$
Interleave + $w$ NAF	$k \cdot 2^{w-2} - k$	$k/(w + 1)$
Interleave + $w$ MOF	$k \cdot 2^{w-2} - k$	$k/(w + 1)$

**Table 2.** Comparison for arbitrary  $k$

The notation  $c_k$  and  $nzd_k$  appearing in table 2 can be found in the definition of JSF in section 2.2 and in proposition 1 of section 3.1, respectively.

According to these tables, the Shamir and the Interleave Methods are a tradeoff between the number of precomputed points and the average density of multiplications. For the Shamir Method the number of precomputed points raises exponentially, while the average density of multiplications stays small. For the Interleave Method the number of precomputed points is comparatively small, but the number of multiplications is large. In other words the two methods are a tradeoff between memory usage for the precomputed points and time for the multi exponentiation.

One can also see that there is no efficiency disadvantage in using left-to-right exponent recording algorithms. In fact the number of precomputed points and the average density of multiplications equals the right-to-left case. But there is one great benefit when using left-to-right exponent recording. For the conversion, the memory usage is  $k \cdot (k + 1)$  bits for the algorithm of section 3.1 and  $k \cdot w$  bits for the algorithm of section 3.2. This is a great improvement since an exponent recording stage requires  $k \cdot n$  bits of memory, where  $n$  is the bitlength of the exponents.

In the case of arbitrary  $k$ , the equivalence of the formulas for the joint nzd of JSF and the proposed algorithm of section 3.1 is not obvious. Hence we included some more examples for the joint nzd of the proposed algorithm in table 3. Those values were calculated according to proposition 1

$k$	$nzd_k$	$\approx nzd_k$
2	$1/2$	0.5000
3	$23/39$	0.5897
4	$115/179$	0.6424
5	$4279/6327$	0.6763
6	$152821/218357$	0.6998
7	$21292819/29681427$	0.7173
8	$729686995/998122451$	0.7310
9	$395575908331/533014861803$	0.7421
10	$212669883207319/283038627384983$	0.7513

**Table 3.** Some values for  $nzd_k$

According to this table, the joint nzd of the proposed recoding algorithm equals Proos' method of section 2.2. Proos proved in [Pro03] that this joint nzd is minimal among all joint signed binary representations using digit set  $D = \{0, \pm 1\}$ .

## 4 Conclusion

We proposed a novel left-to-right exponent recording algorithm for multi exponentiation with  $k$  bases. The proposed algorithm scans only  $k + 1$  consecutive joint bits at most, and it is suitable for the implementation on memory-constraint devices. We precisely estimated the average density of the joint non-zero bits for general  $k$  bases, e.g.,  $1/2$  for  $k = 2$  and  $23/39$  for  $k = 3$ . These density is minimal among all representations using digit set  $\{-1, 0, 1\}$ . Further we showed explicitly how the existing techniques  $w$ MOF and the Interleave Method can be used to perform multi exponentiation scanning only  $w$  bits for each exponent. The average density of multiplications for this method equals  $k/(w + 1)$ . Moreover, the proposed schemes can be used for on-the-fly scalar multiplications. Therefore no exponent recording stage is required anymore and the memory usage is reduced dramatically. This is desirable for the implementation on smartcards. A typical

application of the proposed scheme is the verification algorithm of the digital signature standard (DSA) using elliptic curves.

## References

- [Ava02] Avanzi, R., *On multi-exponentiation in cryptography*, Cryptology ePrint Archive: Report 2002/154, 2002. <http://eprint.iacr.org/2002/154/>
- [BSS99] Blake, I., Seroussi, G., and Smart, N., *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.
- [CMO98] Cohen, H., Miyaji, A., Ono, T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), 51-65.
- [Gor98] Gordon, D., *A survey of fast exponentiation methods*, Journal of Algorithms, vol.27, (1998), 129-146.
- [Kob87] Koblitz, N., *Elliptic Curve Cryptosystems*, Math. Comp. 48, (1987), 203-209.
- [Mil86] Miller, V.S., *Use of Elliptic Curves in Cryptography*, Advances in Cryptology - CRYPTO '85, LNCS218, (1986), 417-426.
- [MOC97] Miyaji, A., Ono, T., and Cohen, H., *Efficient Elliptic Curve Exponentiation*, Information and Communication Security, ICICS 1997, LNCS 1334, (1997), 282-291.
- [MO90] Morain, F., Olivos, J., *Speeding Up the Computations on an Elliptic Curve using Addition-Subtraction Chains*, Informa. Theor. Appl., 24, (1990), pp.531-543.
- [Möl01] Möller, B., *Algorithms for Multi-exponentiation*, Selected Areas in Cryptography - SAC 2001, LNCS 2259, pp. 165-180
- [OSST04] Okeya, K., Schmidt-Samoa, K., Spahn, C., Takagi, T., *Signed Binary Representations Revisited*, Advances in Cryptology - CRYPTO 2004, LNCS 3152, (2004), pp.123-139. Full version is available at Cryptology ePrint Archive 2004/195. <http://eprint.iacr.org/2004/195/>
- [Pro03] Proos, J., *Joint Sparse Forms and Generating Zero Columns when Combining*, Technical Report of CACR, CORR 2003-23, 2003.
- [Sol00] Solinas, J.A., *Efficient Arithmetic on Koblitz Curves*, Design, Codes and Cryptography, 19, (2000), 195-249.
- [Sol01] Solinas, J.A., *Low-weight binary representations for pairs of integers*, Tech.Report CORR 2001-41, University of Waterloo, 2001, available at <http://www.cacr.math.uwaterloo.ca>

## A Several Proofs

**Lemma 1** *Assume that an  $n$ -bit MOF  $\mu$  is not 0. Then  $Z(\mu) = \{0, \dots, n-1\} \setminus \{f\}$ , where  $f$  is the index of the least non-zero digit.*

*Proof.* For an index  $i$ , there are three cases: (1)  $\mu[i] = 0$ , (2)  $\mu[i]$  is non-zero but it's not the least non-zero digit, and (3)  $\mu[i]$  is the least non-zero digit.

The case (1) is trivial. For the case (2), we can utilize the conversion  $x0\dots0\bar{x} \rightarrow 0x\dots xx$  for obtaining such a  $\delta$ , where  $x = \mu[i]$ . For the case (3),  $eval(\mu)$  is divisible by  $2^i$  but not divisible by  $2^{i+1}$ . This means that such a  $\delta$  does not exist.  $\square$

**Lemma 2** For any  $k$   $(k + 1)$ -bit MOF  $\mu_j$ ,

$$\bigcap_{j=1}^k Z(\mu_j) \neq \emptyset.$$

In other words, there exists an index  $z$  such that, for any  $j$ , there exists a signed binary representation  $\delta_j$  with  $\delta_j[z] = 0$  and  $\text{eval}(\delta_j) = \text{eval}(\mu_j)$ .

*Proof.* Because of Lemma 1, there are only  $k$  possibilities to place the least non-zero-digit for  $k$  MOFs. Thus there always exists such an index since the bit-length of the MOFs is  $k + 1$ .  $\square$

**Proposition 1** The average joint non-zero density of Left-to-Right Recording for General Terms with  $k$  terms is:

$$nzd_k = 1 - \frac{1}{\sum_{b=1}^{k+1} \frac{C_b}{2^{kb}} \cdot b}.$$

*Proof.* Recall that the  $nzd$  is the joint Hamming weight divided by the total number of bits, i.e. columns. Since a conversion creates one joint zero column independent of the iteration, there are  $(b - 1)$  non zero columns in the strings converted in step  $b$ . Therefore  $C_b \cdot (b - 1)$  is the joint Hamming weight and  $C_b \cdot b$  is the number of columns of the string combinations which can be converted in iteration  $b$ . From Lemma 2 we know that for  $k$  strings, we have to scan at most  $(k+1)$  columns to create a joint zero column. This leads to a total number of  $2^{k(k+1)}$  string combinations. Then

$$p_b = \frac{C_b \cdot 2^{k(k+1-b)}}{2^{k(k+1)}}$$

is the percentage of all these strings which are converted in iteration  $b$ . The numbers  $p_b \cdot (b - 1)$  and  $p_b \cdot b$  reflect the contribution of iteration  $b$  to the total joint Hamming weight and the total number of columns respectively. Therefore the total joint Hamming weight and the total number of columns are given as

$$\sum_{b=1}^{k+1} p_b \cdot (b - 1), \quad \sum_{b=1}^{k+1} p_b \cdot b$$

respectively. Using the formula for the  $nzd$  we get

$$nzd_k = \frac{\sum_{b=1}^{k+1} p_b \cdot (b - 1)}{\sum_{b=1}^{k+1} p_b \cdot b} = 1 - \frac{\sum_{b=1}^{k+1} p_b}{\sum_{b=1}^{k+1} p_b \cdot b}$$

Since  $p_b$  is the percentage of all  $2^{k(k+1)}$  string combinations which are converted in iteration  $b$  it is obvious that  $\sum_{b=1}^{k+1} p_b = 1$  holds. We can then simplify the formula to

$$nzd_k = 1 - \frac{1}{\sum_{b=1}^{k+1} p_b \cdot b} = 1 - \frac{1}{\sum_{b=1}^{k+1} \frac{C_b \cdot 2^{k(k+1-b)}}{2^{k(k+1)}} \cdot b} = 1 - \frac{1}{\sum_{b=1}^{k+1} \frac{C_b}{2^{kb}} \cdot b}$$

□

**Lemma 3** *There are  $\binom{k}{b-1}$  fnz combinations for given  $k$  and  $b$  such that the corresponding string combinations cannot be converted.*

*Proof.* For each fnz combination, we can represent it as

$$F = \begin{bmatrix} E_{b-1} \\ T \end{bmatrix},$$

where  $E_{b-1}$  is the identity matrix of size  $b-1$ , and  $T$  is an arbitrary binary matrix of  $(k+1-b) \times (b-1)$  in which each row has at most one non-zero entry. Thus, for each row of  $T$  there are  $b$  possibilities. We identify two fnz combinations if they are equal under a suitable permutation of rows. Hence the total number of fnz combinations is equal to the repeated combinations of  $(k+1-b)$  out of  $b$ . That is,

$$\binom{(k+1-b)+b-1}{k+1-b} = \binom{k}{k+1-b} = \binom{k}{b-1}$$

.

□

**Lemma 4**

$$C_b = \sum_{i=1}^{\binom{k}{b-1}} P_i \cdot M_i \cdot N_i$$

where  $P_i$  is the number of row permutations,  $M_i$  the number of strings corresponding and  $N_i$  the number of columns which can be appended to fnz combination  $i$ .

*Proof.* Assume we obtained the numbers  $P_i$ ,  $M_i$  and  $N_i$  for a given fnz combination  $i$ . For a row permutation  $\sigma_i$  of the fnz combination  $i$ , the number of strings corresponding to  $\sigma_i$ , namely  $M_{\sigma_i}$  obviously equals  $M_i$ . The same holds for  $N_i$ , the number of columns which can be appended. If a column  $q$  can be appended to fnz combination  $i$ ,  $\sigma_q$  can also be appended to  $\sigma_i$ . Therefore  $N_i$  is the same for all permutations. □

**Lemma 5** *For  $i = 1, \dots, \binom{k}{b-1}$  and  $D = (d_{ij})$  as above, we have the following:*

- i)  $P_i = \binom{k}{d_{i1}} \binom{k-d_{i1}}{d_{i2}} \dots \binom{k-(d_{i1}+\dots+d_{i,b-2})}{d_{i,b-1}}$
- ii)  $M_i = \prod_{j=1}^{b-1} 2^{(j-1) \cdot d_{ij}}$
- iii)  $N_i = 1 + \sum_{c=1}^{b-1} \left( (-1)^{c-1} \sum_{1 \leq j_1 < \dots < j_c \leq b-1} 2^{k-(d_{ij_1}+\dots+d_{ij_c})} \right)$

*Proof.*

- i) At first we calculate the number of possible row permutations for the first column, namely  $\binom{k}{d_{i1}}$ . For the second column there are only  $k - d_{i1}$  rows left for applying a row permutation, namely all rows where the first non zero bit is not in column 1. Generally speaking for a certain column  $J$  a row permutation can only be applied on the rows  $I = \{i : d_{ij} = 0, j = 1, \dots, J - 1\} \subset \{1, \dots, k\}$ , which are exactly  $k - (d_{i1} + \dots + d_{i,J-1})$ . Since we consider the  $(b - 1)$ -th step the last column to process is the  $(b - 1)$ -th.
- ii) If a certain string  $s$  has its first non zero bit on position  $j$  we have

$$s[l] \begin{cases} \in \{0, \pm 1\}, l = 1, \dots, j - 1 \\ \in \{\pm 1\}, l = j \\ = 0, l = j + 1, \dots, b - 1 \end{cases}$$

Therefore the number of strings which have their first non zero bit in column  $j$  is  $2^{j-1}$ . For fnz combination  $i$  there are  $d_{ij}$  strings which have their first non zero bit in column  $j$  which leads to  $2^{(j-1) \cdot d_{ij}}$  strings for column  $j$ . Since we are interested in the number of strings for all columns  $j = 1, \dots, k$  we get the formula

$$\prod_{j=1}^{b-1} 2^{(j-1) \cdot d_{ij}}$$

- iii) For each column  $j$  of fnz combination  $i$  there is a certain number of columns which can be appended in order to create a joint zero column at column  $j$ . Those are exactly the columns which are non zero in the rows where column  $j$  is non zero, i.e. a string has its first non zero bit in column  $j$ . By appending such a column the first non zero bit of this string moves to the appended column and column  $j$  can be converted. For each column  $j$  the number of columns which can be appended is  $2^{k-d_{ij}}$ , namely  $d_{ij}$  rows have to be non zero, the remaining  $k - d_{ij}$  can be either zero or non zero. The above formula counts the columns which can be appended for each column  $j = 1, \dots, b - 1$ , but each of the columns only once. This is achieved by successively subtracting the columns which were added to often and adding the columns which were subtracted to often.

Note that in ii) and iii) the non zero entries must be chosen according to the MOF attributes, i.e. the signs of adjacent non zero bits are opposite. Therefore each entry is uniquely determined and there is always only one possibility, namely  $-1$  or  $1$ .  $\square$