

A New Upper Bound for the Minimal Density of Joint Representations in Elliptic Curve Cryptosystems

Erik DAHMEN[†], *Nonmember*, Katsuyuki OKEYA^{††}, and Tsuyoshi TAKAGI^{†††}, *Members*

SUMMARY The most time consuming operation to verify a signature with the Elliptic Curve Digital Signature Algorithm is a multi-scalar multiplication with two scalars. Efficient methods for its computation are the Shamir method and the Interleave method, whereas the performance of those methods can be improved by using general base-2 representations of the scalars. In exchange for the speed-up, those representations require the precomputation of several points that must be stored. In the case of two precomputed points, the Interleave method and the Shamir method provide the same, optimal efficiency. In the case of more precomputed points, only the Interleave method can be sped-up in an optimal way and is currently more efficient than the Shamir method. This paper proposes a new general base-2 representation of the scalars that can be used to speed up the Shamir method. It requires the precomputation of ten points and is more efficient than any other representation that also requires ten precomputed points. Therefore, the proposed method is the first to improve the Shamir method such that it is faster than the Interleave method. **key words:** *elliptic curve cryptosystem, joint sparse form, left-to-right, multi-scalar multiplication, shamir method*

1. Introduction

Digital signatures are a key technology to secure IT infrastructures. They provide authenticity and integrity and are widely used in identification and authentication protocols. Digital signature schemes are often implemented on smart cards, due to their mobility and tamper resistance. However, the available memory and computational power of a smart card is very limited. Therefore, cryptosystems based on elliptic curves [10], [12] are often used in conjunction with smart cards. The basic operation to verify a signature with the Elliptic Curve Digital Signature Algorithm (ECDSA) is a multi-scalar multiplication $uP + vQ$, where u, v are the scalars and P, Q are points on an elliptic curve. The research goal here is to efficiently compute this multi-scalar multiplication by minimizing both memory usage and computational costs.

The Shamir method [6] and the Interleave method [14] are two efficient methods to compute a multi-scalar multiplication. The speed of those methods depends on the number of non-zero entries or columns in the binary representation of the scalars, respectively. A speed up of those methods is possible by deploying general base-2 representations of the scalars that provide less non-zero entries or columns. However, using general base-

2 representations also requires the precomputation of some points. In fact, there is a trade-off between the number of points to precompute and the speed up of a multi-scalar multiplication.

In the case of the Interleave method, there exist representations that offer an optimal speed up for any number of precomputed points [4], [13], [15]–[19]. In the case of the Shamir method, optimal speed up currently can only be achieved for two precomputed points [9], [20], using the digits $0, \pm 1$. The efficiency of the Interleave method and the Shamir method is the same for two precomputed points. When considering more than two precomputed points, the next logical step for the Shamir method is to use representations that require ten precomputed points as proposed in [1], [11]. However, the speed up of those representations is worse compared to the optimal representations for the Interleave method. When using ten precomputed points, the natural choice for the digits to use is $0, \pm 1, \pm 3$. Using more digits for the Shamir method, e.g. $0, \pm 1, \pm 3, \pm 5$ is impractical, since the effort for the precomputation is too great in this case.

This paper proposes a new general base-2 representation of the scalars to speed up the Shamir method. The proposed representation uses the digits $0, \pm 1, \pm 3$ and requires the precomputation of ten points. The main idea of the algorithm is to apply a sliding window method with variable width on both scalars simultaneously. The proposed representation provides an average density of non-zero columns of $239/661 \approx 0.3615$. The proposed representation is more efficient than any other representation that also requires ten precomputed points and is therefore the first method to improve the Shamir method such that it is faster than the Interleave method. Besides the practical improvement, the proposed method is very interesting from a theoretical point of view. Although the methods that are used to speed up the Interleave method are proven to be optimal, they are inferior to the proposed method. Further, the proposed scheme is generated starting at the most significant bit, which is more natural and memory saving. In detail, only 6 joint bits must be stored at once.

This paper is organized as follows: Section 2 gives an overview of multi-scalar multiplication. Section 3 reviews known general base-2 representations. Section 4 describes the proposed scheme. Section 5 gives a comparison and Section 6 states the authors conclusion.

[†]Technische Universität Darmstadt

^{††}Hitachi, Ltd., Systems Development Laboratory

^{†††}Future University - Hakodate

2. Preliminaries

This section introduces some background knowledge about general base-2 representations, elliptic curves and multi-scalar multiplications.

Let \mathbb{F}_p be a finite field, where $p > 3$ is prime. Let E be an elliptic curve over \mathbb{F}_p . The elliptic curve can be used to construct an abelian group $E(\mathbb{F}_p)$ with identity element \mathcal{O} called the point of infinity. A point $P \in E(\mathbb{F}_p)$ in affine coordinates is represented as $P = (x, y)$. The inverse of $P = (x, y)$ is to $-P = (x, -y)$, hence it can be computed virtually for free. Note that this is also true for elliptic curves over different fields, e.g. binary curves. The elliptic curve operations $P + Q$ and $2P$ are denoted by ECADD and ECDBL, respectively, where $P, Q \in E(\mathbb{F}_p)$.

A scalar d is a positive integer. A vector (d_{n-1}, \dots, d_0) is called a general base-2 representation of d , if $d = \sum_{i=0}^{n-1} d_i \cdot 2^i$ and $d_i \in \mathcal{D}, \forall i = 0, \dots, n-1$. The set \mathcal{D} is called digit set. The term "general" means, that \mathcal{D} may be an arbitrary subset of \mathbb{Z} and is not fixed to $\mathcal{D} = \{0, 1\}$ as it is for the common binary representation. If $\mathcal{D} = \{0, \pm 1\}$ the representation is called a signed binary representation. If $\mathcal{D} = \{0, \pm 1, \dots, \pm x\}$, the representation is called a signed representation.

Algorithms 1 and 2 show adjusted versions of the Interleave method [14] and the Shamir method [6], respectively. The algorithms are adjusted to compute $uP + vQ$, where u, v are the n bit scalars given in some signed representation and $P, Q \in E(\mathbb{F}_p)$. In the following, u_i and v_i denote the i th digit of u and v , respectively.

Algorithm 1 Interleave Method

Require: Points $P, Q \in E(\mathbb{F}_p)$, scalars u, v .

Ensure: Multi-scalar multiplication $uP + vQ$

```

1:  $R_t \leftarrow tP, \forall t \in \mathcal{D}_1, t > 1$ 
2:  $S_t \leftarrow tQ, \forall t \in \mathcal{D}_2, t > 1$ 
3:  $X \leftarrow \mathcal{O}$ 
4: for  $i = n - 1$  down to  $0$  do
5:    $X \leftarrow \text{ECDBL}(X)$ 
6:   if  $u_i \neq 0$  then
7:      $X \leftarrow \text{ECADD}(X, R_{u_i})$ 
8:   if  $v_i \neq 0$  then
9:      $X \leftarrow \text{ECADD}(X, S_{v_i})$ 
10: return  $X$ 

```

The Interleave method allows the scalars to be represented using different signed representations. Lines 1 and 2 precompute the required points. Since point inversions are virtually for free, it is sufficient to precompute and store only half of all possible points. For example, if $\mathcal{D}_1 = \{0, \pm 1, \pm 3\}$, $\mathcal{D}_2 = \{0, \pm 1, \pm 3, \pm 5, \pm 7\}$, the points to precompute are $3P, 3Q, 5Q, 7Q$. If a different point, e.g. $-3P$ is required, it is obtained by an on-the-fly point inversion. Lines 6 and 8 show, that the Interleave method performs an ECADD operation if the current digit in the scalar is non-zero. Call the number

of non-zero digits in a scalar u Hamming weight and the expected density of non-zero digits average Hamming density ($\text{AHD}(u)$). Since an ECDBL operation is performed in each iteration, the Interleave method on average requires $n(\text{AHD}(u) + \text{AHD}(v))$ ECADD + n ECDBL operations for the evaluation of $uP + vQ$ exclusive the costs for the precomputation.

Algorithm 2 Shamir Method

Require: Points $P, Q \in E(\mathbb{F}_p)$, scalars u, v .

Ensure: Multi-scalar multiplication $uP + vQ$

```

1:  $R_{t_1 t_2} \leftarrow t_1 P + t_2 Q, \forall t_1, t_2 \in \mathcal{D} \setminus \{0\}, t_1 > 0$ 
2:  $R_{t0} \leftarrow tP, \forall t \in \mathcal{D}, t > 1$ 
3:  $R_{0t} \leftarrow tQ, \forall t \in \mathcal{D}, t > 1$ 
4:  $X \leftarrow \mathcal{O}$ 
5: for  $i = n - 1$  down to  $0$  do
6:    $X \leftarrow \text{ECDBL}(X)$ 
7:   if  $(u_i, v_i) \neq (0, 0)$  then
8:      $X \leftarrow \text{ECADD}(X, R_{u_i v_i})$ 
9: return  $X$ 

```

The required points are precomputed in lines 1-3. For example, if $\mathcal{D} = \{0, \pm 1\}$, the two points to precompute are $P + Q, P - Q$. If $\mathcal{D} = \{0, \pm 1 \pm 3\}$, the ten points to precompute are $3P, 3Q, P + Q, P - Q, P + 3Q, P - 3Q, 3P + Q, 3P - Q, 3P + 3Q, 3P - 3Q$. The base points P and Q do not count as precomputed points. Line 7 shows, that the Shamir method performs an ECADD operation if both current digits in the scalars are non-zero, i.e. the current column is non-zero. Call the number of non-zero columns in two scalars u, v joint Hamming weight and the expected density of non-zero columns average joint Hamming density ($\text{AJHD}(u, v)$). Since an ECDBL operation is performed for each column, the Shamir method on average requires $n \text{AJHD}(u, v)$ ECADD + n ECDBL operations for the evaluation of $uP + vQ$ exclusive the costs for the precomputation.

2.1 A Special Signed Binary Representation

This section reviews a special signed binary representation required to generate the proposed representation. It was first proposed in [3] to speed up a binary multiplication and later used in [7] and [17] to generate signed representations of scalars. In the following, this representation will be referred to as the Mutual Opposite Form (MOF). Let (d_{n-1}, \dots, d_0) be the binary representation of an integer d . Define $\mu_i = d_{i-1} - d_i$ for $i = 0, \dots, n$, where $d_n = d_{-1} := 0$. The MOF of d is then given as $(\mu_{n-1}, \dots, \mu_0)$. Further, the MOF satisfies the following properties

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

The MOF uses the digit set $\mathcal{D} = \{0, \pm 1\}$, provides an AHD of $1/2$ and each n -bit integer has a unique representation as $(n + 1)$ -digit MOF [17].

3. Signed Representations to Speed Up $uP + vQ$

This section serves two purposes. It first reviews several known methods to produce signed representations of the scalars. Those methods are also called recoding algorithms. Second, it examines the size of the digit set to be used for the Shamir method. Note, that all representations reviewed in this section are uniquely determined and at most one bit longer than the corresponding binary representation (see the respective reference).

One recoding algorithm to decrease the AJHD is the joint sparse form (JSF) [20]. Its AJHD is $1/2$ and it uses the digit set $\mathcal{D} = \{0, \pm 1\}$. It requires the precomputation of the two points $\{P + Q, P - Q\}$. A similar method was proposed in [9]. It uses the same digit set and provides the same AJHD as the JSF, but unlike the JSF it can be applied to the scalars starting at the most significant bit. The AHD provided by both representations is minimal.

Remark 3.1: The direction in which the scalars are recoded is very important. Recall, that the Interleave method and the Shamir method perform the evaluation starting at the most significant bit, i.e. left-to-right (LtR). If the recoding algorithm performs the recoding starting at the least significant bit, i.e. right-to-left (RtL), the scalars must be recoded and stored completely in memory before the evaluation can begin. This requires $O(n)$ bits of memory for each scalar. If the scalars are recoded LtR, this can be done "on-the-fly" during the evaluation. Then it is not necessary to store the whole recoded scalars, but only a small part which leads to a significant saving of memory.

To decrease the AHD of the scalars, there is the width- w non adjacent form (w NAF) [4], [13], [19]. It uses the digit set $\mathcal{D} = \{0, \pm 1, \dots, \pm 2^{w-1} - 1\}$ and its AHD is $1/(w + 1)$, which is minimal. There also exists a LtR analogue of the w NAF called the width- w mutual opposite form (w MOF)[17]. Another LtR analogue was proposed in [2]. Both these methods use the same digit set and provide the same minimal AHD as w NAF. Choosing $w = 3$ for both scalars requires the two points $\{3P, 3Q\}$ to be precomputed. The resulting AHD of each scalar then is $1/4$. Therefore, the average density of ECADD operations used by the Interleave method is $1/2$, the same as for the Shamir method using the JSF or the scheme described in [9].

One way to decrease the AJHD using 10 precomputed points is the JSF₃ [11]. It uses the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$ and the resulting AJHD is $121/326 \approx 0.3712$. This methods requires the precomputation of the ten points $\{3P, 3Q, P + Q, P - Q, P + 3Q, P - 3Q, 3P + Q, 3P - Q, 3P + 3Q, 3P - 3Q\}$. Another ap-

proach is to apply a width-2 window method from LtR to the JSF of two scalars [1]. This method uses the digit set $\mathcal{D} = \{0, \pm 1, \pm 2, \pm 3\}$, but due to the properties of the JSF only the ten points $\{P + Q, P - Q, P + 2Q, P - 2Q, 2P + Q, 2P - Q, 2P + 3Q, 2P - 3Q, 3P + 2Q, 3P - 2Q\}$ have to be precomputed. The resulting AJHD of this method is $3/8 = 0.3750$. Both methods can only be generated from RtL, due to the carry over that is created during the conversion.

Using the w MOF with $w_1 = 4$ for the first and $w_2 = 5$ for the second scalar results in the digit sets $\mathcal{D}_1 = \{0, \pm 1, \dots, \pm 7\}$ and $\mathcal{D}_2 = \{0, \pm 1, \dots, \pm 15\}$. The resulting AHDs are $1/5 = 0.2$ for the first and $1/6 \approx 0.1667$ for the second scalar. The average density of ECADD operations required by the Interleave method is the given as $11/30 \approx 0.3666$. The points to precompute are $\{3P, 3Q, 5P, 5Q, 7P, 7Q, 9Q, 11Q, 13Q, 15Q\}$.

A better approach to decrease the AHD is to use the fractional window recoding method [15], [16], [18], which can be generated from LtR and RtL likewise. The resulting representation uses the digit set $\mathcal{D} = \{0, \pm 1, \dots, \pm 2^{w-1} + m\}$ and the AHD is $1/(w + \frac{m+1}{2^w} + 1)$ which is minimal. In order to obtain 10 precomputed points, chose $w = 4$ and $m = 3$ for both scalars. The resulting digit set is $\mathcal{D} = \{0, \pm 1, \dots, \pm 11\}$ and the AHD is $2/11 \approx 0.1818$ for each scalar. The average density of ECADD operations required by the Interleave method is given as $4/11 \approx 0.3636$. This method requires the points $\{3P, 3Q, 5P, 5Q, 7P, 7Q, 9P, 9Q, 11P, 11Q\}$ to be precomputed.

The costs for the precomputation and the total costs to compute $uP + vQ$ with the Shamir method, using the JSF, the JSF₃ and the method described in [1] when considering the customary 160-bit scalars are as follows.

	Precomputation	Total costs for $uP + vQ$
JSF	2 ECADD	82 ECADD + 160 ECDBL
[1]	12 ECADD + 2 ECDBL	72 ECADD + 162 ECDBL
JSF ₃	10 ECADD + 2 ECDBL	69 ECADD + 162 ECDBL

This shows that switching from two to ten precomputed points indeed improves the total performance. The problem of the Shamir method is, that it is rather inflexible when choosing the number of points to precompute, i.e. switching from $\mathcal{D} = \{0, \pm 1\}$ to $\mathcal{D} = \{0, \pm 1, \pm 3\}$ requires eight more points to be precomputed. This explains the comparison two points vs. ten points. Since the number of points to precompute grows quadratically, it is unlikely that choosing larger digit sets, such as $\mathcal{D} = \{0, \pm 1 \pm 3, \pm 5\}$ where 22 points must be precomputed, will further reduce the total costs as long as 160-bit scalars are used. Also, storing 22 points requires 880 bytes of memory, which might lead to problems on a smart card.

4. Proposed Scheme

The last section showed that both the Interleave method and the Shamir method provide the same, optimal efficiency when using two precomputed points. In the case of ten precomputed points, the Interleave method currently provides better performance than the Shamir method. Also, the AHD provided by the signed representations for the Interleave method is minimal. Using the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$ is optimal in case of the Shamir method, since larger digit sets require to much operations and memory for the precomputed points.

This section describes the proposed signed representation of the scalars to speed up the Shamir method. For the reasons explained above, the proposed method uses the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$ and therefore requires ten precomputed points. The proposed representation is generated by applying a sliding window method (SWM) with variable widths on the MOF of the scalars simultaneously. If certain conditions are satisfied, already converted columns are reused. The proposed algorithm is divided in three parts: the *Main Routine*, the *Calculation of Z* and the *Conversion Routine*. The proposed representation provides an AJHD of 239/661 and it can be generated from left-to-right by storing at most 6 bits of each scalar at once.

4.1 First Considerations

This section examines in what way the MOF representation can be used to decrease the joint Hamming weight (JHW). The first MOF property implies that the absolute value of any w consecutive MOF digits is at most $2^{w-1} - 1$. Therefore, any w consecutive MOF digits can be represented using $w - 1$ zero entries and 1 non-zero entry with absolute value of at most $2^{w-1} - 1$. Since the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$ is used, $w = 3$ holds and extending this to two scalars yields

Lemma 4.1: *Given two MOF representations, it is possible to create at most two consecutive zero columns without exceeding the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$. After that, at least one non-zero column must follow.*

The following lemma considers the positions of the columns that are candidates to become zero.

Lemma 4.2: *Let μ_0 and μ_1 be two k -bit MOF representations. Further, let f_0 and f_1 be the digit of the least non-zero entry of μ_0 and μ_1 respectively. The set*

$$Z := \{k - 1, \dots, 0\} \setminus \{f_0, f_1\}$$

contains the indices of the columns that are candidates to become zero columns.

Note that to create two zero columns, at least three and at most four columns of the MOF representation have to be scanned.

4.2 The Main Routine

The purpose of the main routine is to decide on the window width used in a certain step. The widths and the required number of zero columns to create are chosen such that the resulting joint Hamming density (JHD) of the recoded window increases from step to step. In other words, at first a width which results in a low JHD is tested and if that fails, the width is increased and a worse JHD is accepted. Table 1 shows the sequence in which the widths and the required zero columns are chosen.

Sequence of conversion	1.	2.	3.	4.
zero columns required	1	2	3	2
window width	1	3	5	4
resulting JHD	0	0.33	0.4	0.5

Table 1 Sequence of window widths

If a recoding with one of the first three widths is possible, the window is recoded, written out and the algorithm proceeds to the next column. Otherwise, after using the last width where a recoding is always possible, the following two conditions are checked to decide how to proceed.

1. If the last two columns remain unchanged after the recoding, the first two recoded columns are written out and the algorithm starts the next iteration with the third column.
2. If the last column has been changed, but does not contain any entries equal to ± 3 , the first three columns are written out and the algorithm starts the next iteration with the fourth column.

If none of those conditions is satisfied, all four columns are written out and the algorithm proceeds with the next column. If one of those conditions is satisfied, the algorithm reuses already converted columns in the next step. The problem is, that now it is no longer guaranteed that adjacent non-zero bits have opposite signs. Therefore, Lemma 4.1 doesn't hold anymore and has to be reduced as follows.

Lemma 4.3: *If an already converted column is reused, at most one zero column can be created without exceeding the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$. After that at least one non-zero column must follow.*

According to Lemma 4.2, in this case at least two and at most three columns must be scanned in order to create one zero column. Therefore, a different sequence of widths is used as shown in Table 2.

Sequence of conversion	1.	2.	3.	4.
zero columns required	1	1	2	1
window width	1	2	4	3
resulting JHD	0	0.5	0.5	0.66

Table 2 Sequence of window widths when reusing

Again, if a recoding using one of the first three widths is possible the window is recoded, written out and the algorithm proceeds to the next column. Otherwise the fourth width is used for the recoding and the above explained conditions are tested. Note, that if no already converted column is reused, Lemma 4.1 holds again in the next step.

Algorithm 3 shows the pseudo code of the main routine. It uses the following notation: The variables u and l denote the first and the last index of the current window, respectively. The variable c denotes the current case, namely $c = 0$ if an already converted column is reused and $c = 1$ otherwise. The notation $d_{i,j}$ denotes the j th bit of the i th scalar and substrings are denoted by $d_{0,u..l} := (d_{0,u}, d_{0,u-1}, \dots, d_{0,l})$. Also, \ominus denotes the bitwise subtraction which is used for the on-the-fly MOF generation.

Algorithm 3 The Main Routine

Require: two n -bit scalars d_0, d_1 in their binary representation
Ensure: recoded representation μ_0 and μ_1

- 1: $d_{0,-1} \leftarrow 0$; $d_{1,-1} \leftarrow 0$; $d_{0,n} \leftarrow 0$; $d_{1,n} \leftarrow 0$
- 2: $u \leftarrow n$; $c \leftarrow 1$
- 3: **while** $u > 0$
- 4: **while** $d_{0,u} = d_{0,u-1} \wedge d_{1,u} = d_{1,u-1} \wedge u > 0$
- 5: $\mu_{0,u} \leftarrow 0$; $\mu_{1,u} \leftarrow 0$
- 6: $u \leftarrow u - 1$; $c \leftarrow 1$
- 7: **end while**
- 8: $l \leftarrow u - 1 - c$
- 9: $\mu_{0,u+c-1..l} \leftarrow d_{0,u+c-1..l} \ominus d_{0,u+c-2..l-1}$
- 10: $\mu_{1,u+c-1..l} \leftarrow d_{1,u+c-1..l} \ominus d_{1,u+c-2..l-1}$
- 11: $z \leftarrow \text{calculateZ}(\mu_{0,u..l}, \mu_{1,u..l}, c)$
- 12: **if** $z_{u-l} + \dots + z_0 \geq 1 + c \vee l \leq 0$
- 13: $(\mu_{0,u..l}, \mu_{1,u..l}) \leftarrow \text{convert}(\mu_{0,u..l}, \mu_{1,u..l}, z)$
- 14: $u \leftarrow u - 2 - c$; $c \leftarrow 1$
- 15: **else**
- 16: $l \leftarrow u - 3 - c$
- 17: $\mu_{0,u+c-1..l} \leftarrow d_{0,u+c-1..l} \ominus d_{0,u+c-2..l-1}$
- 18: $\mu_{1,u+c-1..l} \leftarrow d_{1,u+c-1..l} \ominus d_{1,u+c-2..l-1}$
- 19: $z \leftarrow \text{calculateZ}(\mu_{0,u..l}, \mu_{1,u..l}, c)$
- 20: **if** $z_3 = 1 \wedge z_2 = 0 \wedge z_1 = 1 \wedge z_0 = 0$
- 21: $(\mu_{0,u..l}, \mu_{1,u..l}) \leftarrow \text{convert}(\mu_{0,u..l}, \mu_{1,u..l}, z)$
- 22: $u \leftarrow u - 4 - c$; $c \leftarrow 1$
- 23: **else**
- 24: $l \leftarrow u - 2 - c$
- 25: $\mu_{0,u+c-1..l} \leftarrow d_{0,u+c-1..l} \ominus d_{0,u+c-2..l-1}$
- 26: $\mu_{1,u+c-1..l} \leftarrow d_{1,u+c-1..l} \ominus d_{1,u+c-2..l-1}$
- 27: $z \leftarrow \text{calculateZ}(\mu_{0,u..l}, \mu_{1,u..l}, c)$
- 28: $(\mu_{0,u..l}, \mu_{1,u..l}) \leftarrow \text{convert}(\mu_{0,u..l}, \mu_{1,u..l}, z)$
- 29: **if** $\mu_{0,l+1..l} = d_{0,l+1..l} \ominus d_{0,l..l-1} \wedge$
 $\mu_{1,l+1..l} = d_{1,l+1..l} \ominus d_{1,l..l-1}$
- 30: $u \leftarrow u - 1 - c$; $c \leftarrow 1$
- 31: **else if** $\mu_{0,l} \neq \pm 3 \wedge \mu_{1,l} \neq \pm 3 \wedge$
 $(\mu_{0,l}, \mu_{1,l}) \neq (0, 0)$
- 32: $u \leftarrow u - 2 - c$; $c \leftarrow 0$
- 33: **else**
- 34: $u \leftarrow u - 3 - c$; $c \leftarrow 1$
- 35: **end if**
- 36: **end if**
- 37: **end if**
- 38: **end while**
- 39: **return** μ_0, μ_1 .

4.3 The Calculation Of Z

This method computes the number of zero columns that can be created in a certain window. Therefore it is used by the main routine to decide whether a certain width can be used or not. Further it computes the positions of the columns to become zero, which are needed by the conversion routine. First, the set Z is calculated according to Lemma 4.2. Next, a set $\tilde{Z} \subset Z$ is selected which represents the columns that will actually be converted to zero. This choice is performed according to Lemma 4.1 or Lemma 4.3. If there is more than one choice for \tilde{Z} , the leftmost candidates are picked first. In the following examples let $\bar{x} = -x$.

Example 4.4: Without reusing: Let $\mu_0 = \bar{1}01\bar{1}1$, $\mu_1 = 10\bar{1}00$. Therefore $f_0 = 0$ and $f_1 = 2$ holds. Lemma 4.2 implies $Z := \{4, 3, 2, 1, 0\} \setminus \{2, 0\} = \{4, 3, 1\}$ and Lemma 4.1 implies $\tilde{Z} = \{4, 3, 1\}$.

Example 4.5: With reusing: Let $\mu_0 = \bar{1}\bar{1}\bar{1}1$, $\mu_1 = \bar{1}0\bar{1}1$. Therefore $f_0 = 0$ and $f_1 = 0$ holds. Lemma 4.2 implies $Z := \{3, 2, 1, 0\} \setminus \{0\} = \{3, 2, 1\}$ and Lemma 4.3 implies $\tilde{Z} = \{3, 1\}$.

Algorithm 4 shows the pseudo code for the calculation of Z using the same notation as in Algorithm 3. The set Z is represented as a k -bit array z , where $z_j = 1$, if the j th column in the current window is to be converted and $z_j = 0$, otherwise. Here k is the width of the current window and $j = k - 1, \dots, 0$.

Algorithm 4 Calculation of z calculateZ

Require: two k -bit MOF strings μ_0 and μ_1 and the current case c
Ensure: the vector z

- 1: **for** $i = 0$ **to** 1
- 2: $f_i \leftarrow -1$
- 3: **for** $j = k - 1$ **down to** 0
- 4: **if** $\mu_{i,j} \neq 0$
- 5: $f_i \leftarrow j$
- 6: **end if**
- 7: **end for**
- 8: **end for**
- 9: $r \leftarrow 0$
- 10: **for** $j = k - 1$ **down to** 0
- 11: **if** $j = f_0 \vee j = f_1 \vee r = 2$
- 12: $z_j \leftarrow 0$; $r \leftarrow 0$
- 13: **else**
- 14: $z_j \leftarrow 1$; $r \leftarrow r + 1$
- 15: **end if**
- 16: **if** $c = 0 \wedge j = k - 1 \wedge z_{k-1} = 1$
- 17: $r \leftarrow 2$
- 18: **end if**
- 19: **end for**
- 20: **return** z .

4.4 The Conversion Routine

The conversion routine performs the actual recoding of the window. At this point the columns that shall become zero are known and therefore it is possible to recode each scalar separately. Each window is scanned from left-to-right and if a non-zero entry that should become zero is detected, the algorithm scans for the next non-zero entry to the right and applies one of the following conversions.

- (1) $100\dots 0\bar{1} \mapsto 011\dots 11$ / $1\bar{1} \mapsto 01$
- (2) $\bar{1}00\dots 01 \mapsto 0\bar{1}\bar{1}\dots \bar{1}\bar{1}$ / $\bar{1}1 \mapsto 0\bar{1}$
- (3) $100\dots 01 \mapsto 03\bar{1}\dots \bar{1}\bar{1}$ / $11 \mapsto 03$
- (4) $\bar{1}00\dots 0\bar{1} \mapsto 0\bar{3}1\dots 11$ / $\bar{1}\bar{1} \mapsto 0\bar{3}$

Note, that because of Lemma 4.2 it is always possible to find a non-zero entry to the right in the current window.

Example 4.6: Let $\mu_0 = \bar{1}01\bar{1}1$, $\mu_1 = 10\bar{1}00$, $\tilde{Z} = \{4, 3, 1\}$. Applying (1) – (4) yields

$$\begin{array}{ccccccc} \bar{1}01\bar{1}1 & \xrightarrow{(2)} & 0\bar{1}\bar{1}\bar{1}1 & \xrightarrow{(4)} & 00\bar{3}\bar{1}1 & \xrightarrow{(2)} & 00\bar{3}0\bar{1} \\ \uparrow & & \uparrow & & \uparrow & & \\ 10\bar{1}00 & \xrightarrow{(1)} & 01100 & \xrightarrow{(3)} & 00300 & \xrightarrow{(2)} & 00300 \\ \uparrow & & \uparrow & & \uparrow & & \end{array}$$

Example 4.7: Let $\mu_0 = \bar{1}\bar{1}\bar{1}1$, $\mu_1 = \bar{1}0\bar{1}1$, $\tilde{Z} = \{3, 1\}$. Applying (1) – (4) yields

$$\begin{array}{ccc} \bar{1}\bar{1}\bar{1}1 & \xrightarrow{(4)} & 0\bar{3}\bar{1}\bar{1} & \xrightarrow{(2)} & 0\bar{3}0\bar{1} \\ \uparrow & & \uparrow & & \\ \bar{1}0\bar{1}1 & \xrightarrow{(4)} & 0\bar{3}11 & \xrightarrow{(3)} & 0\bar{3}03 \\ \uparrow & & \uparrow & & \end{array}$$

Algorithm 5 shows the pseudo code for the conversion. The notation is the same as in Algorithms 3 and 4.

Algorithm 5 Conversion routine *convert*

Require: two k -bit MOF strings μ_0 and μ_1
and the columns to convert z

Ensure: recoded representation of μ_0 and μ_1

```

1: for  $i = 0$  to 1
2:   for  $j = k - 1$  down to 0
3:     if  $z_j = 1 \wedge \mu_{i,j} \neq 0$ 
4:        $s \leftarrow j - 1$ 
5:       while  $\mu_{i,s} = 0$ 
6:          $s \leftarrow s - 1$ 
7:       end while
8:       if  $\mu_{i,j} = -\mu_{i,s}$ 
9:         for  $t = j - 1$  down to  $s$  do  $\mu_{i,t} \leftarrow \mu_{i,j}$ 
12:         $\mu_{i,j} \leftarrow 0$ 
13:       else if  $\mu_{i,j} = \mu_{i,s}$ 
14:         for  $t = j - 2$  down to  $s$  do  $\mu_{i,t} \leftarrow -\mu_{i,j}$ 
17:         $\mu_{i,j-1} \leftarrow 3 \cdot \mu_{i,j}$ ;  $\mu_{i,j} \leftarrow 0$ 
18:       end if
19:     end if
20:   end for
21: end for
22: return  $\mu_0, \mu_1$ .
```

4.5 Average Joint Hamming Density

This section estimates the AJHD of the proposed representation using Markov Chains [8]. The Markov Chain of the proposed scheme consists of 14 states S_1, \dots, S_{14} .

Figure 1 shows the transition graph of the proposed scheme. Each state indicates the number of columns currently scanned, the number of columns which are reused (boxed) and the probability of the state changes. After a window is recoded, the algorithm jumps back to state 1. Those changes are indicated by arrows with a dot at the end.

The transition probabilities are given by the matrix $(p_{ij}) := P(S_i \mapsto S_j)$, where $S_i \mapsto S_j$ indicates that state S_i changes into S_j . Those numbers were obtained by checking all cases. Also, the matrix (t_{ij}) which contains the total number of columns written out by the algorithm if $S_i \mapsto S_j$ and the matrix (n_{ij}) which contains the number of non-zero columns written out if $S_i \mapsto S_j$ are required, $i, j = 1, \dots, 14$. The non-zero entries of those three matrices as well as the line in Algorithm 3 where the transitions occur are summarized in Table 4.

line	$S_i \mapsto S_j$	p_{ij}	t_{ij}	n_{ij}
4	$S_1 \mapsto S_1$	1/4	1	0
4	$S_6 \mapsto S_1$	17/37	2	1
4	$S_9 \mapsto S_1$	1/2	2	1
4	$S_{10} \mapsto S_1$	7/19	1	0
4	$S_{14} \mapsto S_1$	7/15	2	1
4/15	$S_6 \mapsto S_7$	20/37	0	0
4/15	$S_9 \mapsto S_7$	1/2	0	0
4/15	$S_{14} \mapsto S_7$	8/15	0	0
7	$S_1 \mapsto S_2$	3/4	0	0
7	$S_{10} \mapsto S_{11}$	12/19	0	0
12	$S_2 \mapsto S_1$	1/2	3	1
12	$S_5 \mapsto S_1$	1	3	1
12	$S_{11} \mapsto S_1$	9/16	3	1
15	$S_2 \mapsto S_3$	1/2	0	0
15	$S_{11} \mapsto S_{12}$	7/16	0	0
20	$S_3 \mapsto S_1$	17/48	5	2
20	$S_7 \mapsto S_1$	3/8	4	2
20	$S_{12} \mapsto S_1$	5/14	5	2
23	$S_3 \mapsto S_4$	31/48	0	0
23	$S_7 \mapsto S_8$	5/8	0	0
23	$S_{12} \mapsto S_{13}$	9/14	0	0
29	$S_4 \mapsto S_5$	3/31	2	1
29	$S_8 \mapsto S_5$	1/5	1	1
29	$S_{13} \mapsto S_5$	1/9	2	1
31	$S_4 \mapsto S_6$	37/62	3	1
31	$S_8 \mapsto S_9$	4/5	2	1
31	$S_{13} \mapsto S_{14}$	5/8	3	1
33	$S_4 \mapsto S_{10}$	19/62	4	2
33	$S_{13} \mapsto S_{10}$	19/72	4	2

Table 4 Non-zero entries in p_{ij} , t_{ij} and n_{ij}

For example, a change from state S_3 to state S_1 occurs with probability 17/48. In this case five columns are written out, two of them non-zero. A change from state S_3 to state S_4 occurs with probability 31/48. No columns are written out in this case.

not very convincing, since the overhead strongly depends on the implementation and the architecture of the platform. For example on a smart card, conditional branches are comparable to subtractions which have negligible cost compared to elliptic curve point additions. On other platforms this might be different. Hence the authors focused on what can be proved, the average joint Hamming density. Besides the practical improvement, the theoretical impact of the proposed scheme should not be neglected. It is provably better than the known methods which are provably minimal.

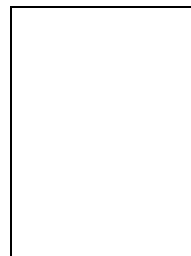
6. Conclusion

This paper proposed a new general representation of the scalars to speed up the calculation of $uP + vQ$ using the Shamir method. The proposed representation uses the digit set $\mathcal{D} = \{0, \pm 1, \pm 3\}$ and requires the precomputation of ten points. The resulting AJHD is $239/661 \approx 0.3615$, which is superior to any known method that also uses ten precomputed points. The proposed representation is the first representation with which the Shamir method outperforms the Interleave method in the case of ten precomputed points. This is especially interesting since the representations used for the Interleave method are provably minimal. Due to the left to right fashion of the proposed algorithm, the memory consumption by the recoding is reduced, i.e. only 6 bits of the binary representation of each scalar must be stored at once.

Future research in this topic includes a further reduction of the AJHD and an estimate of the total speed on different platforms, e.g. a smart card.

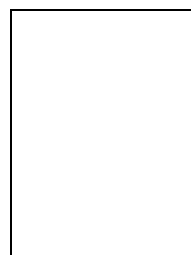
References

- [1] Avanzi, R., *On multi-exponentiation in cryptography*, Cryptology ePrint Archive: Report 2002/154, 2002
- [2] Avanzi, R., *A Note on the Signed Sliding Window Integer Recoding and a Left-to-Right Analogue*, Selected Areas in Cryptography - SAC 2004, LNCS 3357, pp. 130-143
- [3] Booth, A., *A signed binary multiplication technique*, Quarterly Journal of Mechanics and Applied Mathematics, vol. 4, no. 2, 1951, pp. 236-240.
- [4] Blake, I., Seroussi, G., and Smart, N., *Elliptic Curves in Cryptography*, Cambridge University Press, 1999.
- [5] Cohen, H., Miyaji, A., Ono, T., *Efficient Elliptic Curve Exponentiation Using Mixed Coordinates*, Advances in Cryptology - ASIACRYPT '98, LNCS1514, (1998), pp. 51-65.
- [6] ElGamal, T., *A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms*, IEEE Transactions on Information Theory, Vol. 31, IEEE 1985, pp. 469-472.
- [7] Grabner, P., Heuberger, C., Prodinger, H., Thuswaldner J., *Analysis of linear combination algorithms in cryptography*, ACM Transactions on Algorithms 1 (2005), pp. 123-142.
- [8] Häggström, O., *Finite Markov Chains and Algorithmic Applications*, London Mathematical Society Student Texts 52, Cambridge University Press, (2002).
- [9] Heuberger, C., Katti, R., Prodinger, H., Ruan, X., *The Alternating Greedy Expansion and Applications to Left-To-Right Algorithms in Cryptography* Theoret. Comput. Sci 341 (2005), pp. 55-72.
- [10] Koblitz, N., *Elliptic Curve Cryptosystems*, Math. Comp. 48, (1987), 203-209.
- [11] Kuang, B., Zhu, Y., Zhang, Y., *An Improved Algorithm for $uP+vQ$ using JSF_3* , Applied Cryptography and Network Security - ACNS 2005, LNCS 3089, pp. 467-478
- [12] Miller, V.S., *Use of Elliptic Curves in Cryptography*, Advances in Cryptology - CRYPTO '85, LNCS218, (1986), 417-426.
- [13] Miyaji, A., Ono, T., and Cohen, H., *Efficient Elliptic Curve Exponentiation*, Information and Communication Security, ICICS 1997, LNCS 1334, (1997), 282-291.
- [14] Möller, B., *Algorithms for Multi-exponentiation*, Selected Areas in Cryptography - SAC 2001, LNCS 2259, pp. 165-180
- [15] Möller, B., *Improved Techniques for Fast Exponentiation*, Information Security and Cryptology ICISC 2002. LNCS 2587, pp. 298312
- [16] Möller, B., *Fractional Windows Revisited: Improved Signed-Digit Representations for Efficient Exponentiation*, Information Security and Cryptology ICISC 2004, to appear.
- [17] Okeya, K., Schmidt-Samoa, K., Spahn, C., Takagi, T., *Signed Binary Representations Revisited*, Advances in Cryptology - CRYPTO 2004, LNCS 3152, pp.123-139, available at <http://eprint.iacr.org/2004/195/>
- [18] Schmidt-Samoa, K., Semay, O., Takagi, T., *Analysis of Some Efficient Window Methods and their Application to Elliptic Curve Cryptosystems*, Technical Report No. TI-3/04, 16. August 2004.
- [19] Solinas, J.A., *Efficient Arithmetic on Koblitz Curves*, Design, Codes and Cryptography, 19, (2000), 195-249.
- [20] Solinas, J.A., *Low-weight Binary Representations for Pairs of Integers*, University of Waterloo, Technical Report CORR 2001-41, 2001, available at <http://www.cacr.math.uwaterloo.ca>



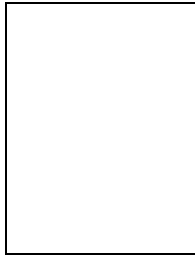
Erik Dahmen received the Dipl. math. degree from Technische Universität Darmstadt in 2006. He engaged in research on cryptography in 2003 as TA in Prof. Takagi's research group. In 2004/2005, he spent six months at Systems Development Laboratory, Hitachi Ltd. as intern. He is currently working as PhD student in the research group of Prof. J. Buchmann at Technische Universität Darmstadt. His research interests

include efficient elliptic curve cryptography and post-quantum signature schemes.



Katsuyuki Okeya received the B.S. degree from Toyama University in 1994 and the M.S. degree from Kyushu University in 1996. He had been engaged in development on information security at Software Division, Hitachi Ltd. from 1998 to 2001. He received the Dr. degree in engineering from Kyushu University in

2004. He has been engaged in research on cryptography and information security at Systems Development Laboratory, Hitachi Ltd. since 2001. He received the IPSJ Best Paper Award from Information Processing Society of Japan in 2004. Dr. Okeya is a member of the Information Processing Society of Japan, the Institute of Electronics, Information and Communication Engineers, the Japan Society for Industrial and Applied Mathematics, and the Mathematical Society of Japan.



Tsuyoshi Takagi received the B.Sc. and M.Sc. degrees in mathematics from Nagoya University in 1993 and 1995, respectively. He had engaged in the research on network security at NTT Laboratories from 1995 to 2001. He received the Dr.rer.nat degree from Technische Universität Darmstadt in 2001. He was an Assistant Professor in the Department of Computer Science at Technische Universität Darmstadt until 2005. He is currently an Associate Professor in the School of Systems Science Information at Future University-Hakodate. His current research interests are information security and cryptography. Dr. Takagi is a member of International Association for Cryptologic Research (IACR).