Improved Fixed-base Comb Method for Fast Scalar Multiplication

Nashwa A. F. Mohamed[1], Mohsin H. A. Hashim[1], and Michael Hutter[2]

[1] Faculty of Mathematical Sciences, University of Khartoum,
P.O.Box 321, Khartoum, Sudan
nafarah@uofk.edu(or nashwaabbas@gmail.com), mhashim@uofk.edu
[2] TU Graz, Institute for Applied Information Processing and Communications
Inffeldgasse 16a, 8010 Graz, Austria
michael.hutter@iaik.tugraz.at

Abstract. Computing elliptic-curve scalar multiplication is the most time consuming operation in any elliptic-curve cryptosystem. In the last decades, it has been shown that pre-computations of elliptic-curve points improve the performance of scalar multiplication especially in cases where the elliptic-curve point P is fixed. In this paper, we present an improved fixed-base comb method for scalar multiplication. In contrast to existing comb methods such as proposed by Lim and Lee or Tsaur and Chou, we make use of a width-ω non-adjacent form representation and restrict the number of rows of the comb to be greater or equal ω. The proposed method shows a significant reduction in the number of required elliptic-curve point addition operation. The computational complexity is reduced by 33 to 38 % compared to Tsaur and Chou method even for devices that have limited resources. Furthermore, we propose a constant-time variation of the method to thwart simple-power analysis attacks.

Keywords: Elliptic-curve cryptosystem, scalar multiplication, Lim-Lee method, Tsaur-Chou method, non-adjacent form, width-ω NAF.

1 Introduction

In 1985, N. Koblitz [13] and V. Miller [19] introduced elliptic curves for their use in cryptography. The difficulty of solving the elliptic curve discrete logarithm problem is mathematically hard so that Elliptic Curve Cryptography (ECC) can be efficiently applied in modern cryptosystems. Among the most time consuming operation of ECC is the scalar multiplication. A secret scalar k is multiplied with a point P on an elliptic curve $E(\mathbb{F}_q)$ resulting in the point $Q \in E(\mathbb{F}_q)$. Over the last years, there have been many publications that propose new methods to efficiently calculate $Q = kP$, e.g., [5] or [8].

When the elliptic-curve point P is fixed, suggestion to pre-compute some data that depend only on P was first made by Brickell, Gordon, McCurley, and Wilson (BGMW) in 1992 [4]. They observed that if the multiplier k is expressed in a

A. Mitrokotsa and S. Vaudenay (Eds.): AFRICACRYPT 2012, LNCS 7374, pp. 342–359, 2012.
The final publication is available at Springer via http://dx.doi.org/10.1007/978-3-642-31410-0_21
© Springer-Verlag Berlin Heidelberg 2012

base b, more time may be saved by adding together powers with like coefficients first.

Another improvement was proposed by Lim and Lee [16] in 1994. They proposed a more flexible pre-computation technique for speeding up the computation of exponentiation. Later in 2005, Tsaur and Chou [28] proposed a new fixed-base comb method by applying a NAF representation of the scalar k and Sakai and Sakurai [23] method for direct doubling.

In this paper, we propose an efficient method for scalar multiplication by combining the ideas of Lim-Lee [16] and Tsaur-Chou [28]. Our method makes use of a fixed-base comb technique and represents the scalar k in a width-ω NAF representation. Furthermore, we restrict the number of rows of the comb to be greater or equal ω. As a result, our proposed method provides a significant reduction in the number of required elliptic-curve point addition operation. In practice, a speed improvement by 33 to 38 % is achieved.

The rest of this paper is organized as follows. In Section 2, we give an introduction to elliptic curves and review some of existing scalar-multiplication methods. In Section 3, we review the methods of Lim-Lee and Tsaur-Chou. In Section 4, we propose efficient method for speeding up elliptic curve scalar multiplication. In Section 5, we show that our proposed method can accelerate simultaneous scalar multiplication. In Section 6, we discuss the resistance against side-channel attacks. In Section 7, we give results of our method compared with Tsaur and Chou method. In Section 8 we draw conclusions. Finally, in the appendix, we give an example to illustrate our method.

2 Preliminaries

This section introduces some elementary background on elliptic curves. We refer the reader to [5], [9], and [25] for further details.

An elliptic curve E over a finite field $\mathbb{F}_q$ of characteristic $\neq 2, 3$ can be given by the short Weierstrass equation

$$E : y^2 = x^3 + ax + b$$

where $a, b \in \mathbb{F}_q$, for which $4a^3 + 27b^2 \neq 0$. Elliptic-curve points $E(\mathbb{F}_q)$ are defined to be

$$E(\mathbb{F}_q) = \{(x, y) \in \mathbb{F}_q \times \mathbb{F}_q : y^2 = x^3 + ax + b\} \cup O$$

where O is the point at infinity. $E(\mathbb{F}_q)$ forms an additively abelian group with the point at infinity O which serves as the identity element. Adding two points in $E(\mathbb{F}_q)$ is defined by the *chord-and-tangent* rule.

The most time consuming operation in elliptic curve cryptography is the scalar multiplication, i.e., $Q = kP$, where P and Q are points on the curve E and k is a scalar such that $0 \leq k < ord_E(P)$. The Elliptic Curve Discrete Logarithm Problem (ECDLP) is to find the scalar k given points P and Q.

Since scalar multiplication largely determines the execution time of ECC-based protocols, it is attractive to provide efficient methods that reduce the computational complexity by applying different multiplication techniques. There are many proposals given in literature which provide improvements for different kind of scenarios: (1) both the scalar and the base point are unknown, (2) the scalar is fixed, and (3) the base point is fixed [5,9]. In the following sections, we will describe generic methods and methods where the base point is fixed. For methods where the scalar is fixed, addition chains can be used to improve the performance of scalar multiplication. We refer the reader to [5,9] for more details.

2.1 Generic Methods

One of the most easiest way to perform a scalar multiplication where both the scalar and the base point are unknown is the binary method (or often referred as *double-and-add*). A point doubling operation is performed at every loop iteration whereas point addition is only performed if the scalar bit value k_i is 1, where $i \in [0, l-1]$ denotes the bit index of the scalar k with size l. It therefore achieves a density of approximately 1/2 which results in a computational complexity of $\frac{l}{2}A + lD$, where A and D represent the costs for addition and doubling, respectively. Note that the binary method does not need any pre-computations but does not provide resistance against timing [14] or Simple Power Analysis [15] attacks.

Windowing Techniques. A generalization of the binary method has been proposed by Brauer [3] in 1939 (also often referred as 2^r−ary or *window* method). The idea is to slice the representation of the scalar k into pieces and to process ω digits at a time. For this, k is represented in a base 2^ω where $\omega > 1$. The method scans the bits either from left-to-right or from right-to-left (like for the binary method). Note that windowing techniques require extra memory but they significantly improve the speed of scalar multiplication.

An efficient variant of the 2^r−ary method is the *sliding window* method introduced by Thurber [27] in 1973. By pre-computing iP for $i \in \{1, 3, 5, 7, ..., 2^\omega - 1\}$, one can move a width-$\omega$ window across the scalar k and search for the first non-zero bit. After finding the bit, the window is placed such that the value of the window is odd.

Non-Adjacent Form (NAF) Representations. The density of the prior described methods can be further reduced by using a signed-digit representation. The advantage of this representation is that the cost of computing the inverse of elliptic-curve points, e.g., $-P$, comes almost for free. Booth [1] proposed in 1951 to expand the coefficients in the representation of the scalar k to $\{0, \pm 1\}$. However, the disadvantage of his proposal has been that the representation is not unique. Thus, Reitwiesner [22] proposed to apply a Non-Adjacent Form (NAF) representation in 1960. A NAF of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$ where $k_i \in \{0, \pm 1\}, k_{l-1} \neq 0$, and no two consecutive digits k_i are nonzero. The length of the NAF is l. It is a canonical representation with the fewest number of non-zero digits for a given scalar k. The expected number of non-zero bits in a NAF is $l/3$ as shown by Morain and Olivos [21]. The runtime

Algorithm 1 Width-ω NAF method for a positive integer k

Require: Window width w and a positive integer k.
Ensure: Width-ω NAF(k).
$i = 0$.
While $k > 0$ **do**
    If k is odd **then**
        $b \equiv k \bmod 2^{\omega}$.
        If $b \geq 2^{\omega-1}$ **then**
            $b = b - 2^{\omega}$;
        $k = k - b$.
    Else $b = 0$.
    $k_i = b$; $i = i + 1$; $k = k/2$.
Return $(k_{i-1}, k_{i-2}, ..., k_1, k_0)$.

complexity of a binary NAF method is therefore approximately $\frac{l}{3}A + lD$ (cf. [2] and [21]).

A generalization of NAF is the width-ω NAF, proposed by Solinas [26] in 2000. For the width-ω NAF, the scalar k is represented by

$$k = \sum_{i=0}^{l-1} k_i 2^i \tag{1}$$

where each nonzero coefficient k_i is odd, $|k_i| < 2^{\omega-1}$, $k_{l-1} \neq 0$, and at most one of any ω consecutive digits is nonzero. Algorithm 1 can be used to obtain the width-ω NAF of a positive integer k and is denoted by $\mathrm{NAF}_{\omega}(k)$.

In order to perform the scalar multiplication using width-ω NAF, the points $P, 3P, ..., (2^{\omega-1} - 1)P$ are pre-computed and the scalar multiplication is performed in the evaluation phase as shown in Algorithm 2. The average density of non-zero bits among all width-ω NAFs is asymptotically $1/(1+\omega)$ [21]. The expected runtime of Algorithm 2 is therefore

$$[1D + [2^{\omega-2} - 1]A] + [\frac{l}{\omega+1}A + lD]. \tag{2}$$

Algorithm 2 Width-ω NAF method for scalar multiplication

Require: Window width-ω, positive integer k and $P \in E(\mathbb{F}_q)$.
Ensure: $Q = kP$.
Use Algorithm 1 to compute $\mathrm{NAF}_{\omega}(k) = \sum_{i=0}^{l-1} k_i 2^i$.
Compute $P_i = iP$ for all $i \in \{1, 3, 5, 7, ..., 2^{\omega-1} - 1\}$.
$Q = O$.
For $i = l - 1$ **downto** 0 **do**
    $Q = 2Q$.
    If $k_i \neq 0$ **then**
        If $k_i > 0$ **then** $Q = Q + P_{k_i}$.
        Else $Q = Q - P_{k_i}$.
Return (Q).

Note that most of the described methods above do not *per se* provide resistance against side-channel attacks [15,18]. The methods have to provide at least a constant runtime and (even better) a regular structure to resist against most of these attacks, for example, as provided by the Montgomery powering ladder [12,20]. There, a point addition and doubling is performed in every loop iteration achieving a density of 1 ($lA + lD$).

2.2 Fixed Base-Point Methods

When the base point P is fixed, the efficiency of scalar multiplication can be improved by pre-computations. The idea is to pre-compute every multiple $2^i P$ where $0 < i < l$. If theoretically all $2^i P$ points are pre-computed, the complexity of scalar multiplication is reduced to only $\frac{l}{2}A$ (without the need of any doublings).

Similar to generic methods, fixed base-point methods can be mainly separated into windowing, NAF windowing, and fixed-base comb techniques. One of the first who proposed a fixed-base windowing technique has been due to Brickell, Gordon, McCurley, and Wilson (BGMW) [4]. They proposed to split the scalar k into d slices, where $d = \lceil l/\omega \rceil$. The runtime complexity is then reduced to $(2^\omega + d - 3)A$. Similarly, a NAF windowing technique can be applied which further reduces the complexity to approximately $(\frac{2^{\omega+1}}{3} + d - 2)A$, where $d = \lceil (l+1)/\omega \rceil$.

In the following, we will introduce two common techniques proposed by Lim and Lee [16] as well as Tsaur and Chou [28]. Both methods are based on a fixed-base comb technique. Afterwards, we will present our proposed method that makes use of both ideas to reduce the complexity.

3 Fixed-Base Comb Methods

The main idea of fixed-base comb methods is to represent the scalar k as a binary matrix of h rows and v columns. The matrix is then processed column-wise from right-to-left or from left-to-right.

3.1 Lim and Lee Method

In 1994, Lim and Lee [16] introduced a comb technique that divides the scalar k into h blocks K_i from right-to-left, for $0 \le i \le h-1$, of equal size $a = \lceil \frac{l}{h} \rceil$ (we pad zeros if necessary). Then, subdivide each block K_i from up-to-down into v subblocks $k_{i,j}$ of equal size $b = \lceil \frac{a}{v} \rceil$, where $0 \le j \le v-1$. We can rewrite the h blocks of k in terms of a binary matrix, i.e.,

$$k = \begin{bmatrix} K_0 \\ \vdots \\ K_i \\ \vdots \\ K_{h-1} \end{bmatrix} = \begin{bmatrix} k_{0,v-1} & \cdots & k_{0,j} & \cdots & k_{0,0} \\ \vdots & & \vdots & & \vdots \\ k_{i,v-1} & \cdots & k_{i,j} & \cdots & k_{i,0} \\ \vdots & & \vdots & & \vdots \\ k_{h-1,v-1} & \cdots & k_{h-1,j} & \cdots & k_{h-1,0} \end{bmatrix} = \sum_{j=0}^{v-1} \sum_{i=0}^{h-1} k_{i,j} 2^{jb} 2^{ia}.$$

Let $P_0 = P$ and $P_i = 2^a P_{i-1} = 2^{ia} P$ for $0 < i < h$. Then, we can rewrite kP as

$$\begin{aligned} kP &= \sum_{j=0}^{v-1} \sum_{i=0}^{h-1} k_{i,j} 2^{jb} 2^{ia} P \\ &= \sum_{j=0}^{v-1} \sum_{i=0}^{h-1} k_{i,j} 2^{jb} P_i. \end{aligned} \tag{3}$$

Since K_i is of size a, we can let $K_i = e_{i,a-1}...e_{i,1}e_{i,0}$ be the binary representation of K_i for all $0 \leq i < h$, and hence $k_{i,j} = e_{i,jb+b-1}...e_{i,jb+1}e_{i,jb}$ is the binary representation of $k_{i,j}$, therefore

$$kP = \sum_{t=0}^{b-1} 2^t (\sum_{j=0}^{v-1} \sum_{i=0}^{h-1} e_{i,jb+t} 2^{jb} P_i). \tag{4}$$

Suppose that the following values are pre-computed and stored for all $1 \leq s < 2^h$ and $1 \leq j \leq v-1$,

$$\begin{aligned} &G[0][s] = e_{h-1}P_{h-1} + e_{h-2}P_{h-2} + ... + e_0 P_0, \\ &G[j][s] = 2^b(G[j-1][s]) = 2^{jb} G[0][s], \end{aligned}$$

where the index s is equal to the decimal value of $e_{h-1}...e_1e_0$. Therefore, we can rewrite kP as follows

$$kP = \sum_{t=0}^{b-1} 2^t (\sum_{j=0}^{v-1} G[j][I_{j,t}]) \tag{5}$$

where $I_{j,t}$ is the decimal value of $e_{h-1,jb+t}...e_{0,jb+t}$.

Now we can use the left-to-right binary method to compute kP using these pre-computed values. The number of elliptic-curve operations in the worst case is $a + b - 2$, and since $I_{j,t}$ is of size h, we may assume that the probability of $I_{j,t}$ being zero is $\frac{1}{2^h}$ and $I_{j,t}$ occurs a times, thus the expected number of elliptic-curve operations is reduced to $(1 - \frac{1}{2^h})a + b - 2$.

3.2 Tsaur and Chou Method

In 2005, Tsaur and Chou [28] proposed a new fixed-base comb method by applying a NAF representation of the scalar k. Furthermore, they divided k into $h \times v$ blocks from up-to-down and then from right-to-left. Moreover, they used a special doubling operation proposed by Sakai and Sakurai [23] which increases the performance in addition.

Let k be an l-bit scalar represented in NAF. First, we divide k from up-to-down into a blocks of equal size $h = \lceil \frac{l}{a} \rceil$. Thus we can write k as follows

$$k = c_{a-1}c_{a-2}...c_1c_0 = \sum_{l=0}^{a-1} c_l 2^{lh}, \tag{6}$$

Then, from right-to-left we divide the $h \times a$ blocks into $h \times v$ blocks, each of size $b = \lceil \frac{a}{v} \rceil$.

Let $P_0 = P$ and $P_j = 2^{hb}P_{j-1} = 2^{jhb}P$ for $0 < j < v$. Therefore, we can rewrite kP as follows

$$kP = c_{a-1}c_{a-2}...c_1c_0P = \sum_{l=0}^{a-1} c_l 2^{lh}P = \sum_{t=0}^{b-1} 2^{th}(\sum_{j=0}^{v-1} c_{jb+t}2^{jhb}P), \tag{7}$$

where $c_{jb+t} = e_{h-1,jb+t}...e_{1,jb+t}e_{0,jb+t}$ is the NAF representation. Suppose that the following values are pre-computed and stored for all $1 \le s \le \sum_{i=1}^{\lceil \frac{h}{2} \rceil} 2^{h-2i+1}$ and $0 \le j \le v-1$

$$G[0][s] = e_{h-1}2^{h-1}P + e_{h-2}2^{h-2}P + ... + e_0P,$$
$$G[j][s] = 2^{hb}(G[j-1][i]) = 2^{jhb}G[0][s],$$

where the index s is equal to the decimal value of $e_{h-1}...e_1e_0$. Therefore, we can rewrite kP as follows

$$kP = \sum_{t=0}^{b-1} 2^{th}(\sum_{j=0}^{v-1} G[j][I_{j,t}]), \tag{8}$$

where $I_{j,t}$ is the decimal number of $e_{h-1,jb+t}...e_{1,jb+t}e_{0,jb+t}$.

We know that NAF is always sparse, hence the probability of $I_{j,t}$ being zero is $\frac{1}{2^h}$ and $I_{j,t}$ occurs a times, thus the number of elliptic-curve operations in the worst case is $(1 - \frac{1}{2^h})a + b - 2$. And the expected number of elliptic-curve operations required is $(1 - (\frac{2}{3})^h)a + b - 2$ on average.

Direct doubling method. When the multiplier k is a power of 2, Sakai and Sakurai [23] introduced an efficient method to compute $kP = 2^rP (r \ge 1)$ on elliptic curves over $\mathbb{F}_p$. Given a point $P = (x_1, y_1) \in \mathbb{F}_p$, their method compute 2^rP directly. Algorithm 3 from [23] illustrates their method.

In Table 1, we give a comparison of required numbers of multiplication (M), squaring (S), and inversion (I) required to performed the scalar multiplication $k = 2^rP$ between direct doubling method and separate r doubling.

Algorithm 3 Sakai-Sakurai method for direct doubling

Require: A positive integer r such that $k = 2^r$ and $P \in E(\mathbb{F}_q)$.
Ensure: $k = 2^r P$
1: $A_1 = x_1$, $B_1 = 3x_1^2 + a$ **and** $C_1 = -y_1$.
2: **For** $i = 2$ **to** r.
3: $A_i = B_{i-1}^2 - 8A_{i-1}C_{i-1}^2$.
4: $B_i = 3A_i^2 + 16^{i-1}a(\prod_{j=1}^{i-1} C_j)^4$.
5: $C_i = -8C_{i-1}^4 - B_{i-1}(A_i - 4A_{i-1}C_{i-1}^2)$.
6: Compute $D_r = 12A_rC_r^2 - B_r^2$.
7: Compute $x_{2^r} = \frac{B_r^2 - 8A_rC_r^2}{(2^r \prod_{i=1}^{r} C_i)^2}$.
8: Compute $y_{2^r} = \frac{8C_r^4 - B_rD_r}{(2^r \prod_{i=1}^{r} C_i)^3}$.
9: **Return** x_{2^r} and y_{2^r}.

Table 1. Complexity Comparison

Method	**S**	**M**	**I**
Direct Doubling	$4r+1$	$4r+1$	1
Separate r Doubling	$2r$	$2r$	r

4 Our Proposed Method

We propose a new method for elliptic-curve scalar multiplication based on the methods of Lim-Lee [16] and Tsaur-Chou [28]. In our method, the scalar k is represented in width-ω NAF. Furthermore, it is divided into $\omega \times v$ blocks from up-to-down and then from right-to-left as in the method of Tsaur-Chou. In order to illustrate our method, let k be represented in width-ω NAF with size l. First, we divide k into $a = \lceil \frac{l}{\omega} \rceil$ blocks of equal size ω (we pad the last block with $a\omega - l$ zeros if necessary), therefore, we can write k as follows

$$k = K_{a-1}K_{a-2}...K_1K_0 = \sum_{d=0}^{a-1} K_d 2^{d\omega}, \tag{9}$$

where $0 \le d < a$.

Then, each block K_d is a column of ω bits (K_0 represents the first ω bits, K_1 the second ω bits, ... , and K_{a-1} the last ω bits), i.e.,

$$k = \left[K_{a-1}...K_d...K_0\right] = \begin{bmatrix} k_{a-1,(a-1)\omega} & \cdots & k_{d,d\omega} & \cdots & k_{0,0} \\ \vdots & & \vdots & & \vdots \\ k_{a-1,(a-1)\omega+i} & \cdots & k_{d,d\omega+i} & \cdots & k_{0,i} \\ \vdots & & \vdots & & \vdots \\ k_{a-1,(a-1)\omega+(\omega-1)} & \cdots & k_{d,d\omega+(\omega-1)} & \cdots & k_{0,\omega-1} \end{bmatrix}.$$

Note that, for each element $k_{d,d\omega+i}$ in the matrix the first subscript d indicates the column, whereas the second subscript $d\omega + i$ indicates the exact bit index from width-ω NAF(k). To simplify the notation in the following we write $k_{d,d\omega+i}$ as $k_{d,i}$.

From right-to-left we divide the $\omega \times a$ blocks into $\omega \times v$ blocks, each of size $b = \lceil \frac{a}{v} \rceil$, i.e.,

$$k = \left[K_{a-1}...K_{a-b} \cdots K_{jb+b-1}...K_{jb} \cdots K_{b-1}...K_0\right]$$
$$= \begin{bmatrix} k_{a-1,0}...k_{a-b,0} & \cdots & k_{jb+b-1,0}...k_{jb,0} & \cdots & k_{b-1,0}...k_{0,0} \\ \vdots & & \vdots & & \vdots \\ k_{a-1,i}...k_{a-b,i} & \cdots & k_{jb+b-1,i}...k_{jb,i} & \cdots & k_{b-1,i}...k_{0,i} \\ \vdots & & \vdots & & \vdots \\ k_{a-1,\omega-1}...k_{a-b,\omega-1} & \cdots & k_{jb+b-1,\omega-1}...k_{jb,\omega-1} & \cdots & k_{b-1,\omega-1}...k_{0,\omega-1} \end{bmatrix}.$$

$$kP = K_{a-1}K_{a-2}...K_1K_0P = \sum_{j=0}^{v-1}\sum_{t=0}^{b-1}(K_{jb+t}2^{t\omega})2^{jb\omega}P = \sum_{t=0}^{b-1}2^{t\omega}\sum_{j=0}^{v-1}K_{jb+t}2^{jb\omega}P,$$

where $K_{jb+t} = k_{jb+t,\omega-1}...k_{jb+t,0}$ is in width-ω NAF representation. The maximum value of K_{jb+t} is $(2^{\omega-1}-1)2^{\omega-1}$.

Suppose that the following values are all pre-computed and stored for all $s \in \{1, 2, 2^2, 2^3, ..., 2^{\omega-1}\}, 0 < j \leq v-1$ and $d \in \{1, 3, ..., 2^{w-1}-1\}$

$$\begin{aligned} G[0][sd] &= e_{\omega-1}2^{\omega-1}P + e_{\omega-2}2^{\omega-2}P + ... + e_0P = sdP, \\ G[j][sd] &= 2^{\omega b}(G[j-1][sd]) \\ &= 2^{j\omega b}G[0][sd] = 2^{j\omega b}sdP, \end{aligned}$$

where the index sd is equal to the decimal value of $(e_{\omega-1}...e_1e_0)$. Therefore, we can rewrite kP as follows

$$kP = \sum_{t=0}^{b-1}2^{t\omega}(\sum_{j=0}^{v-1}G[j][I_{j,t}]) \tag{10}$$

where $I_{j,t}$ is the decimal value of $k_{jb+t,\omega-1}...k_{jb+t,0}$. Algorithm 4 can be used to compute kP using the proposed method.

From [21], we know that the average density of non-zero digits among all width-ω NAF of length l is approximately $1/(\omega+1)$, therefore, we can assume that the probability of $I_{j,t}$ being zero on average is $(\omega/(\omega+1))^{\omega}$, hence the average cost of our proposed method is

$$(1 - (\frac{\omega}{\omega+1})^{\omega})a + b - 2. \tag{11}$$

On the other hand, the density of non-zero digits among all width-ω NAF of length l in the worst case is $1/\omega$, therefore, we can assume that the probability

Algorithm 4 Proposed width-ω NAF method for scalar multiplication

Require: Positive integers ω, v, $k = (k_{l-1}, ..., k_1, k_0)_{NAF_\omega}$ and $P \in E(\mathbb{F}_q)$.
Ensure: $Q = kP$.

```
a = ⌈l/ω⌉ and b = ⌈a/v⌉.
Compute G[0][sd] and G[j][sd] for all s ∈ {1, 2, 2^2, 2^3, ..., 2^(ω−1)}, 0 < j ≤ v − 1 and
    d ∈ {1, 3, 5, ..., 2^(w−1) − 1}.
Q = O.
For t = b − 1 downto 0 do
    If ω = 1 then
        Q = 2Q.
    Else
        Use Algorithm 3 to compute Q = 2^ω Q.
    For j = v − 1 downto 0 do
        I_{j,t} = (k_{jb+t,ω−1}...k_{jb+t,0})_{NAF_ω}.
        If I_{j,t} > 0 then
            Q = Q + G[j][I_{j,t}].
        Else if I_{j,t} < 0
            Q = Q − G[j][−I_{j,t}].
Return (Q).
```

of $I_{j,t}$ being zero is at most $((\omega-1)/\omega)^\omega$, hence the cost of our proposed method in the worst case is

$$(1 - (\frac{\omega-1}{\omega})^\omega)a + b - 2. \tag{12}$$

5 Simultaneous Scalar Multiplication

In elliptic curve cryptosystems, like in ECDSA, we need to perform the computation of multiple scalar multiplication, i.e., the computation of $kP + rQ$, where $P, Q \in E(\mathbb{F}_q)$ are two elliptic-curve points and k, r are two large integers such that, $0 \leq k < ord_E(P)$ and $0 \leq r < ord_E(Q)$. The direct way is to perform two single scalar multiplications kP, rQ and then one point addition, but, since scalar multiplication is the most time consuming operation in ECC, it is advisable to perform two scalar multiplications simultaneously. There are many proposals given in literature to perform multiple scalar multiplications, we refer the reader to [5,9] for further details.

Our proposed method in Section 4 can be used to accelerate the computation of simultaneous scalar multiplication. In order to illustrate that, assume that we want to compute $kP + rQ$, let k and r be represented in width-ω NAF be l-bit multipliers, then k and r can be represented as follows

$$k = K_{a-1}...K_0 = \sum_{d=0}^{a-1} K_d 2^{d\omega} \quad \text{and} \quad r = R_{a-1}...R_0 = \sum_{d=0}^{a-1} R_d 2^{d\omega}. \tag{13}$$

Let $P_0 = P, Q_0 = Q, P_j = 2^{\omega b} P_{j-1} = 2^{j\omega b} P$, and $Q_j = 2^{\omega b} Q_{j-1} = 2^{j\omega b} Q$ for $0 < j < v$. Therefore, as in Section 4 we can write kP and rQ as follows

$$kP = \sum_{t=0}^{b-1} 2^{t\omega} \sum_{j=0}^{v-1} K_{jb+t} 2^{jb\omega} P \quad \text{and} \quad rQ = \sum_{t=0}^{b-1} 2^{t\omega} \sum_{j=0}^{v-1} R_{jb+t} 2^{jb\omega} Q, \tag{14}$$

where $K_{jb+t} = k_{jb+t,\omega-1}...k_{jb+t,0}$ and $R_{jb+t} = r_{jb+t,\omega-1}...r_{jb+t,0}$ are in width-ω NAF representations.

Therefore we can write $kP + rQ$ as follows

$$kP + rQ = \sum_{t=0}^{b-1} 2^{t\omega} \sum_{j=0}^{v-1} (K_{jb+t} 2^{jb\omega} P + R_{jb+t} 2^{jb\omega} Q), \tag{15}$$

Suppose that the following values are pre-computed and stored for all $s \in \{1, 2, 2^2, 2^3, ..., 2^{\omega-1}\}, 0 < j \le v - 1$ and $d \in \{1, 3, ..., 2^{w-1} - 1\}$

$$\begin{aligned}
G_p[0][sd] &= e_{\omega-1} 2^{\omega-1} P + e_{\omega-2} 2^{\omega-2} P + ... + e_0 P = sdP, \\
G_p[j][sd] &= 2^{\omega b}(G_p[j-1][sd]) \\
&= 2^{j\omega b} G_p[0][sd] = 2^{j\omega b} sdP \\
G_q[0][sd] &= e_{\omega-1} 2^{\omega-1} Q + e_{\omega-2} 2^{\omega-2} Q + ... + e_0 Q = sdQ, \\
G_q[j][sd] &= 2^{\omega b}(G_q[j-1][sd]) \\
&= 2^{j\omega b} G_q[0][sd] = 2^{j\omega b} sdQ,
\end{aligned}$$

where the index sd is equal to the decimal value of $(e_{\omega-1}...e_1 e_0)$. Therefore, we can rewrite $kP + rQ$ as follows

$$kP + rQ = \sum_{t=0}^{b-1} 2^{t\omega} \sum_{j=0}^{v-1} (G_p[j][M_{j,t}] + G_q[j][N_{j,t}]), \tag{16}$$

where $M_{j,t}$ is the decimal value of $k_{jb+t,\omega-1}...k_{jb+t,0}$ $(0 \le t < b)$, and $N_{j,t}$ is the decimal value of $r_{jb+t,\omega-1}...r_{jb+t,0}$ $(0 \le t < b)$. Algorithm 5 can be used to compute kP using the proposed method.

The expected runtime of Algorithm 5 is

$$2(1 - (\frac{\omega}{\omega+1})^{\omega})a + b - 2. \tag{17}$$

6 Resistance to Side Channel Attacks

The method of Lim-Lee, Tsaur-Chou, and our proposed method are *per se* not resistant to side-channel attacks. Side-Channel Analysis (SCA) attacks have been first introduced by Kocher et al. [14,15,18] in 1996. By monitoring physical characteristics of a given implementation, e.g., the power consumption or the timing

Algorithm 5 Proposed width-ω NAF method for multiple scalar multiplication

Require: Positive integers ω, v, $P, Q \in E(\mathbb{F}_q)$, $k = (k_{l-1}, ..., k_1, k_0)_{NAF_\omega}$, $r = (r_{l-1}, ..., r_1, r_0)_{NAF_\omega}$.
Ensure: $kP + rQ$.

```
a = ⌈l/ω⌉ and b = ⌈a/v⌉.
Compute G_p[0][sd] and G_p[j][sd] for all s ∈ {1, 2, 2^2, 2^3, ..., 2^(ω−1)}, 0 < j ≤ v − 1
    and d ∈ {1, 3, 5, ..., 2^(w−1) − 1}.
Compute G_q[0][sd] and G_q[j][sd] for all s ∈ {1, 2, 2^2, 2^3, ..., 2^(ω−1)}, 0 < j ≤ v − 1
    and d ∈ {1, 3, 5, ..., 2^(w−1) − 1}.
R = O.
For t = b − 1 downto 0 do
  If ω = 1 then
    R = 2R.
  Else
    Use Algorithm 3 to compute R = 2^ω R.
  For j = v − 1 downto 0 do
    M_{j,t} = (k_{jb+t,ω−1}...k_{jb+t,0})_{NAF_ω}.
    If M_{j,t} > 0 then
      R = R + G_p[j][M_{j,t}].
    Else if M_{j,t} < 0
      R = R − G_p[j][−M_{j,t}].
  For j = v − 1 downto 0 do
    N_{j,t} = (r_{jb+t,ω−1}...r_{jb+t,0})_{NAF_ω}.
    If N_{j,t} > 0 then
      R = R + G_q[j][N_{j,t}].
    Else if N_{j,t} < 0
      R = R − G_q[j][−N_{j,t}].
Return (R).
```

behavior, an attacker is able to extract secret information such as the ephemeral key or private key in asymmetric-key cryptography. One simple countermeasure to prevent an attacker from being able to recover the bit values of the scalar k by timing attacks and Simple Power Analysis (SPA) [14], is to execute the same code independently of the value of the scalar k, i.e., to make the algorithm have constant runtime. By having a look at the given fixed-base comb methods of Lim-Lee, Tsaur-Chou, and our proposed method, it shows that the scalar is leaking by implementations because the runtime of the algorithms depends on the number of non-zero digits of the secret scalar, cf. [24]. Therefore, one can (regularly) add the point at infinity O when $I_{j,t}$ is equal to zero and even add O to the pre-computed values. In this case, a point addition is executed in every loop iteration and a constant runtime is obtained with complexity of $a+b-2$. A more sophisticated approach is to guarantee that all values of $I_{j,t}$ are non-zero, as for example proposed by Hedabou et al. [10,11] or Feng et al. [7]. Another solution would be to use highly regular (exponent-recoding) techniques such as proposed by Joye and Tunstall [17]. In order to provide resistance against Differential Power Analysis (DPA), we also recommend to include randomization techniques as proposed by Coron [6].

Table 2. Number of non-zero columns for different block sizes h of Tsaur-Chou and our proposed method for a 160-bit scalar multiplication.

h	Tsaur-Chou worst	Tsaur-Chou average	Proposed worst	Proposed average
2	61	45	61	45
3	48	38	38	32
4	39	33	29	25
5	32	29	23	20
6	27	25	18	17
7	23	22	16	14
8	21	21	14	13
9	18	18	12	12
10	17	17	12	11
11	15	15	10	10
12	14	14	10	9
13	13	13	9	9
14	12	12	8	8
15	11	11	8	7

7 Discussion and Results

In Table 3, we give a runtime-complexity comparison of Tsaur-Chou and our proposed method. We compare the runtime in terms of worst-runtime cost, average-runtime cost, and memory-storage cost. By having a look at the table, one can notice that when $h = \omega = 2$, our proposed method and Tsaur-Chou method are identical. When $h = \omega = 3$, the worst cost of our proposed method is equal to the average cost of Tsaur-Chou method. Furthermore, when $h = \omega > 3$, the worst cost of our proposed method is less than the average cost of Tsaur-Chou method. For fixed values of h and v, the term $b-2$ is fixed for both methods for fixed key-bit size of the scalar in average cost and worst cost.

In Table 2 and Figure 1, we analyze the number of non-zero columns for Tsaur-Chou and our proposed method. Values are given for different block sizes h and a 160-bit scalar multiplication. For our method and in order to simplify the comparison, we have chosen $h = \omega$. It shows that our method performs best in both the average-cost and the worst-cost scenario. In particular, by evaluating the performance for all possible block sizes $2 \leq h \leq 15$, we obtain an improvement by 33 to 38 % (for the worst and average case).

For a fixed value of h, we noticed that the number of pre-computations (storage cost) is increased in our proposed method. In devices with limited resources (memory), in most cases we found a suitable choice of h, the window size ω and

Table 3. Runtime complexity of Tsaur-Chou and our proposed method.

Method	Worst cost	Average cost	Storage cost
Tsaur-Chou [28]	$(1-(\frac{1}{2})^h)a+b-2$	$(1-(\frac{2}{3})^h)a+b-2$	$\sum_{i=1}^{\lceil \frac{h}{2} \rceil} 2^{h-2i+1}v$
Proposed	$(1-(\frac{\omega-1}{\omega})^\omega)a+b-2$	$(1-(\frac{\omega}{\omega+1})^\omega)a+b-2$	$\omega 2^{\omega-2}v$

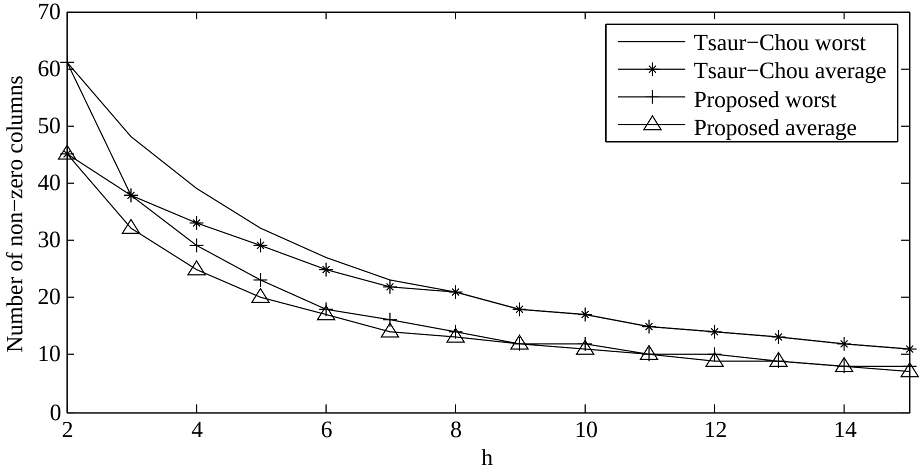

Fig. 1. Comparison of Tsaur-Chou [28] and our proposed method for a 160-bit scalar multiplication.

v, which makes our method best. In order to illustrate this, we assume that the scalar has a bit size of 160 bits. First, we will fix the window size ω to be equal h and then, depending on the available memory, we choose h and v. For example, if storage is available for 5 elements and if we apply the Tsaur-Chou method, we have two choices: (1) $h = 2$ and $v = 2$ (the cost[3] is 84), or (2) $h = 3$ and $v = 1$ (the cost is 90). Now, using our proposed method, we have only one choice, i.e., $h = 2$ and $v = 2$ (the cost is 84). This coincides with what we previously noted.

If storage is available for 18 elements and if we use the Tsaur-Chou method, one can choose between three choices: (1) $h = 2$ and $v = 9$ (the cost is 52), (2) $h = 3$ and $v = 3$ (the cost is 49), or (3) $h = 4$ and $v = 1$ (the cost is 72). For our proposed method, there are only two choices, i.e., (1) $h = 2$ and $v = 9$ (the cost is 52) or (2) $h = 3$ and $v = 3$ (the cost is 48). Thus, we will choose $h = 3$ and $v = 3$ which has a minimum cost of 48. In Table 4, we give the suitable choices of h and v when the available storage vary from 2 to 50 elements.

8 Conclusion

In this paper, we proposed an efficient method for scalar multiplication by combining the ideas of Lim-Lee [16] and Tsaur-Chou [28]. Our proposed method makes a significant reduction in terms of number of elliptic-curve point addition operations. By comparing our method with previous work, it shows that when $h = \omega = 2$ our proposed method and Tsaur-Chou method are identical, when $h = \omega = 3$ the worst cost of our proposed method is equal to the average

[3] The cost is measured in terms of number of elliptic-curve point addition operations.

Table 4. Runtime complexity of Tsaur-Chou and our proposed method for different available storage elements (2-50) and suitable choices of h and v for 160-bit key size.

Available storage elements	Tsaur-Chou method				Proposed method			
	h	***v***	***costs***	***AUS***[a]	***h***	***v***	***costs***	***AUS***[a]
2-3	2	1	124	2	2	1	124	2
4-5	2	2	84	4	2	2	84	4
6-7	2	3	70	6	2	3	70	6
8-9	2	4	64	8	2	4	64	8
10-11	2	5	60	10	2	5	60	10
12-13	2	6	57	12	2	6	57	12
					3	2	57	12
14	2	7	55	14	2	7	55	14
15	3	3	54	15	2	7	55	14
16-17	2	8	54	16	2	8	54	16
18-19	2	9	52	18	3	3	48	18
20-23	3	4	50	20	3	3	48	18
24	3	4	50	20	3	4	44	24
25-29	3	5	47	25	3	4	44	24
30-34	3	6	45	30	3	5	41	30
35	3	7	44	35	3	5	41	30
36-39	3	7	44	35	3	6	39	36
40-41	4	4	42	40	3	6	39	36
42-47	4	4	42	40	3	7	38	42
45	3	9	42	45				
48-49	4	4	42	40	3	8	37	48
50	4	5	40	50	3	8	37	48

[a] The term AUS is referred to the number of elements actually used to store.

cost of Tsaur-Chou method, and when $h = \omega > 3$ the worst cost of our proposed method is less than the average cost of Tsaur-Chou method.

Also we showed that our proposed method can be used for speeding up simultaneous scalar multiplication of elliptic curves which is interesting, for example, in many digital signature verification algorithms.

For pre-computations, if storage space is disregarded, our proposed method is the best choice and we can define $h \geq \omega$, otherwise $\omega = h$. There is always a suitable choice for h and v which make our method best. In Table 4, we gave the suitable choices of h and v when the available storage vary from 2 to 50 elements.

Acknowledgements. This work has been supported by the Faculty of Mathematical Sciences of the University of Khartoum (Sudan) and the European Commission through the ICT program under contract ICTSEC- 2009-5-258754 (Tamper Resistant Sensor Node - TAMPRES).

References

1. Booth, A.D.: A signed binary multiplication technique. Q. J. Mech. Applied Math pp. 236–240 (1951)
2. Bosma, W.: Signed bits and fast exponentiation. Jornal de théorie des nombers de Bordeaux 13, 27–41 (2001)
3. Brauer, A.: On addition chains. Bull. Amer. Math. Soc. 45, 736–139 (1939)
4. Brickell, E.F., Gordon, D.M., McCurley, K.S., Wilson, D.B.: Fast exponentiation with precomputation (extended abstract). In: EUROCRYPT. Lecture Notes in Computer Science, vol. 658, pp. 200–207. Springer (1992)
5. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of elliptic and hyperelliptic curve cryptography. Taylor and Francis Group, LLC (2006)
6. Coron, J.S.: Resistance against differential power analysis for elliptic curve cryptosystems. In: Ç.K. Koç, I., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems (CHES '99). Lecture Notes in Computer Science, vol. 1717, pp. 292–302. Springer-Verlag (1999)
7. Feng, M., Zhu, B.B., Xu, M., Li, S.: Efficient comb elliptic curve multiplication methods resistant to power analysis. IACR Cryptology ePrint Archive 2005, 222 (2005)
8. Gordan, D.M.: A survey of fast exponentiation methods. Journal of Algorithms 27, 129–146 (1998)
9. Hankerson, D., Menezes, A., Vanstone, S.: Guide to elliptic curve cryptography. Springer Verlag New York (2004)
10. Hedabou, M., Pinel, P., Bénéteau, L.: A comb method to render ecc resistant against side channel attacks (2004), paper submitted only to the Cryptology ePrint Archive. hedabou@insa-toulouse.fr 12754 received 2 Dec 2004
11. Hedabou, M., Pinel, P., Bnteau, L.: Countermeasures for preventing comb method against sca attacks. In: ISPEC'05. pp. 85–96 (2005)
12. Joye, M., Yen, S.M.: The Montgomery Powering Ladder. In: Goos, G., Hartmanis, J., van Leeuwen, J. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2002. Lecture Notes in Computer Science, vol. 2523, pp. 291–302. Springer (2003)
13. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48, 203–20 (1987)
14. Kocher, P.C.: Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In: Koblitz, N. (ed.) Advances in Cryptology - CRYPTO '96. Lecture Notes in Computer Science, vol. 1109, pp. 104–113. Springer (1996)
15. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: CRYPTO. pp. 388–397 (1999)
16. Lim, C.H., Lee, P.J.: More flexible exponentiation with precomputation. In: Desmedt, Y. (ed.) Advances in Cryptology - CRYPTO '94. Lecture Notes in Computer Science, vol. 839, pp. 95–107. Springer (1994)
17. M. Joye and M.Tunstall: Exponent recoding and regular exponentiation algorithms. In: Preneel, B. (ed.) Progress in Cryptology-AFRICACRYPT 2009. Lecture Notes in Computer Science, vol. 5580, pp. 334–349. Springer (2009)
18. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks – Revealing the Secrets of Smart Cards. Springer (2007), iSBN 978-0-387-30857-9
19. Miller, V.S.: Uses of elliptic curves in cryptography. Lecture Notes in Computer Science, vol. 218, pp. 417–426. Springer-Verlag (1987)

20. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Mathematics of Computation 48, 243–264 (1987)
21. Morain, F., Olivos, J.: Speeding up the computations on an elliptic curve using addition-subtraction chains. Theor. Inform. Appli. 24, 531–543 (1989)
22. Reitwiesner, G.W.: Binary arithmetic. Advances in Computers 1, 231–308 (1960)
23. Sakai, Y., Sakurai, K.: Speeding up elliptic scalar multiplication using multidoubling. IEICE Transactions Fundamentals E85-A(5) pp. 1075–1083 (2002)
24. Sakai, Y., Sakurai, K.: A new attack with side channel leakage during exponent recoding computations. In: Joye, M., Quisquater, J.J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004. Lecture Notes in Computer Science, vol. 3156, pp. 298–311. Springer (2004)
25. Silverman, J.H.: The arithmetic of elliptic curves, vol. 106. Springer-Verlag, Berlin (1986)
26. Solinas, J.A.: Effiecient arithmetic on koblitz curves. Designs, Codes and Cryptography 19, 195–249 (2000)
27. Thurber, E.G.: On addition chains $l(mn) \leq l(n) - b$ and lower bounds for $c(r)$. Duke Mathematical Journal 40, 907913 (1973)
28. Tsaur, W.J., Chou, C.H.: Efficient algorithm for speeding up the computations of elliptic curve cryptosystem. Applied Mathematics and Computation 168, 1045–1064 (2005)

A Example

In order to illustrate our method, we select at random a positive integer $k = 1065142573068$ and choose $\omega = 3$. First, we represent k in width-3 NAF,

$$k = (010000\bar{1}00000000000\bar{1}00\bar{1}00100\bar{1}00300000000300).$$

Then, we divide from up-to-down to $a = \lceil \frac{41}{3} \rceil = 14$ blocks of size 3, such that

$$k = \begin{bmatrix} 0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 1\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0\,0 \\ 0\,0\,\bar{1}\,0\,0\,0\,\bar{1}\,\bar{1}\,1\,\bar{1}\,3\,0\,0\,3 \end{bmatrix}. \tag{18}$$

Then, from right-to-left we divide the 3×14 blocks to 3×7 blocks, each of size $b = \lceil \frac{14}{7} \rceil = 2$, such that

$$k = \begin{bmatrix} 00\;00\;00\;00\;00\;00\;00 \\ 10\;00\;00\;00\;00\;00\;00 \\ 00\;\bar{1}0\;00\;\bar{1}\bar{1}\;1\bar{1}\;30\;03 \end{bmatrix}. \tag{19}$$

Next, we compute and store the following values. For pre-computed values $G[0][sd]$, where $s \in \{1, 2, 4\}$ and $d \in \{1, 3\}$:

$$G[0][1] = P, G[0][2] = 2P, G[0][4] = 4P, G[0][3] = 3P, G[0][6] = 6P, G[0][12] = 12P.$$

For pre-computed values $G[j][sd]$, where $s \in \{1, 2, 4\}, d \in \{1, 3\}$ and $0 \leq j \leq 6$:

$$G[j][1] = 2^{6j}P, G[j][2] = 2^{6j}(2P), G[j][4] = 2^{6j}(4P),$$
$$G[j][3] = 2^{6j}(3P), G[j][6] = 2^{6j}(6P), G[j][12] = 2^{6j}(12P).$$

Next, compute $I_{j,t} = (e_{2,2j+t}e_{1,2j+t}e_{0,2j+t})_2$ for all $0 \le t \le 1$ and $0 \le j \le 6$ as follows:

$$
\begin{aligned}
I_{0,0} &= (e_{2,0}e_{1,0}e_{0,0})_{NAF_3} = (300)_{NAF_3} = 12,\\
I_{1,0} &= (e_{2,2}e_{1,2}e_{0,2})_{NAF_3} = 0,\\
I_{2,0} &= (e_{2,4}e_{1,4}e_{0,4})_{NAF_3} = (\bar{1}00)_{NAF_3} = -4,\\
I_{3,0} &= (e_{2,6}e_{1,6}e_{0,6})_{NAF_3} = (\bar{1}00)_{NAF_3} = -4,\\
I_{4,0} &= (e_{2,8}e_{1,8}e_{0,8})_{NAF_3} = 0,\\
I_{5,0} &= (e_{2,10}e_{1,10}e_{0,10})_{NAF_3} = 0,\\
I_{6,0} &= (e_{2,12}e_{1,12}e_{0,12})_{NAF_3} = 0,\\
I_{0,1} &= (e_{2,1}e_{1,1}e_{0,1})_{NAF_3} = 0,\\
I_{1,1} &= (e_{2,3}e_{1,3}e_{0,3})_{NAF_3} = (300)_{NAF_3} = 12,\\
I_{2,1} &= (e_{2,5}e_{1,5}e_{0,5})_{NAF_3} = (100)_{NAF_3} = 4,\\
I_{3,1} &= (e_{2,7}e_{1,7}e_{0,7})_{NAF_3} = (\bar{1}00)_{NAF_3} = -4,\\
I_{4,1} &= (e_{2,9}e_{1,9}e_{0,9})_{NAF_3} = 0,\\
I_{5,1} &= (e_{2,11}e_{1,11}e_{0,11})_{NAF_3} = (\bar{1}00)_{NAF_3} = -4,\\
I_{6,1} &= (e_{2,13}e_{1,13}e_{0,13})_{NAF_3} = (010)_{NAF_3} = 2.
\end{aligned}
$$

Finally, we can compute kP by using above values as follows:

$$
\begin{aligned}
kP = {} & G[0][12] + G[1][0] - G[2][4] - G[3][4] + G[4][0] + G[5][0] + G[6][0] + \\
& 2^3(G[0][0] + G[1][12] + G[2][4] - G[3][4] + G[4][0] - G[5][4] + G[6][2]).
\end{aligned}
$$