Fast Elliptic Curve Point Multiplication using Double-Base Chains

V. S. Dimitrov[1], L. Imbert[1,2], and P. K. Mishra[1]

[1] University of Calgary, 2500 University drive NW
Calgary, AB, T2N 1N4, Canada

[2] CNRS, LIRMM, UMR 5506
161 rue Ada, 34392 Montpellier cedex 5, France

Abstract

Among the various arithmetic operations required in implementing public key cryptographic algorithms, the elliptic curve point multiplication has probably received the maximum attention from the research community in the last decade. Many methods for efficient and secure implementation of point multiplication have been proposed. The efficiency of these methods mainly depends on the representation one uses for the scalar multiplier. In the current work we propose an efficient algorithm based on the so-called double-base number system. We introduce the new concept of double-base chains which, if manipulated with care, can significantly reduce the complexity of scalar multiplication on elliptic curves. Besides we have adopted some other measures to further reduce the operation count. Our algorithm compares favorably against classical and other similar approaches.

Keywords: Elliptic curve cryptography, scalar multiplication, double-base number system

1 Introduction

Elliptic curve cryptography (ECC) was proposed independently in 1985 by N. Koblitz [13] and V. Miller [15]. Since then, an immense amount of research has been dedicated to securing and accelerating its implementations. ECC has quickly received a lot of attention because of smaller key-length and increased theoretical robustness (there is no known sub-exponential algorithm to solve the ECDLP problem, which is the foundation of ECC). The relatively small key-size (a 160-bit key provides the same security as an 80-bit symmetric-key for block ciphers or a 1024-bit RSA modulus) is a major advantage for devices with limited hardware resources such as smartcards, cell phones or PDAs.

ECC algorithms for encryption, signature or key-exchange belong to the class of group-based protocols, such as ElGamal and Diffie-Hellman protocols [14]. which base their security on the hardness of the discrete logarithm problem (DLP) over a finite group. In ECC, we consider $E(K)$, the group of rational points on an elliptic curve, E, defined over a finite field K. The group is written additively (we talk about doublings and additions), mainly because we reserve the use of

squarings and multiplications (as well as inversions) for the arithmetic operations in the underlying field. Given a point $P \in E(K)$ and a positive integer k, the computation of the new point, $kP \in E(K)$, is called point multiplication or scalar multiplication. There has been extensive research to compute the scalar multiplication efficiently and several methods have been proposed in the literature (see [12] for a thorough presentation). In these methods the representation of the scalar, k, have a direct influence on the complexity. Efficient solutions have been proposed using binary, ternary, non-adjacent form (NAF), window methods (w-NAF), etc. Point multiplication is usually computed by a sequence of point doublings (triplings, halvings) and additions, whose numbers depend on the length and the number of non-zero digits of k. Reducing the number of curve operations is the main motivation of this paper.

As pointed out by D. Gordon in [10], it is often possible to find a sporadic number system which outperforms other representation schemes used in implementing exponentiation algorithms. Since 1996, a new number representation scheme, called double-base number system (DBNS) has been investigated, mainly due to its applicability in digital signal processing area [6]. It uses a representation of the integers as the sum of mixed powers of two and three. Section 2.2 is devoted to the main properties of the DBNS. In this paper we demonstrate an application of the DBNS in elliptic curve cryptography and we look for a particular representation of the scalar as the sum of numbers of the form 2^b3^t, where b, t are non-negative integers. The main theorem about DBNS states that one needs $\mathcal{O}\left(\frac{\log k}{\log\log k}\right)$ mixed powers of two and three to represent the positive integer, k, in DBNS (see [7]). The rigorous determination of the constant hidden in the big-$\mathcal{O}$ notation leads to tremendously difficult problems in transcendental number theory and exponential Diophantine equations. However, the computational experiments and some probabilistic arguments suggest that this constant is most probably equal to one. This means that if k is a randomly chosen 160-bit integer, then one needs only about 22 summands to represent k, as opposed to 80 in standard binary representation and 53 in the non-adjacent form (NAF), sometimes referred to as signed-digit binary representation. Although this sparseness does not immediately lead to algorithmic improvements, it outlines one of the main features of this number system and serve as a good starting point for potential applications in cryptography.

In the current work we propose a new scalar multiplication algorithm based on double-base number system and double-base chains. We assume that the scalar multiplier is in DBNS and we propose an algorithm which requires a very small number of point additions. Moreover, to further reduce the overall complexity, all the additions are combined with doublings, triplings or quadruplings operations.

The double-base number system is highly redundant. In our scalar multiplication algorithm, we use a particular DBNS form for the scalar k, by imposing restrictions on the binary and ternary exponents. We propose an algorithm to transform any given integer into such a DBNS representation. Our point multiplication algorithm compares favorably against classical solutions and a recently proposed binary/ternary approach.

The paper is organized as follows: In the next section, we recall some basic facts about elliptic

curve and introduce the double-base number system. In Section 3 we introduce the concept of double-base chain and we propose a new scalar multiplication algorithm based on the double-base number system. We present experimental results and compare our algorithm with other similar approaches in Section 4. We briefly discuss issues about side channels and conclude the paper in Section 5.

2 Background

In this section, we give a brief overview of elliptic curve cryptography and the double-base number system. The reader is encouraged to check the given references for more complete descriptions.

2.1 Elliptic curve cryptography

In this section we briefly introduce the main concepts of elliptic curve cryptography which are relevant to this work. The interested reader can refer to [12] for details.

Definition 1 (Elliptic curve) *An elliptic curve E over a field K is defined by an equation*

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{1}$$

where $a_1, a_2, a_3, a_4, a_6 \in K$, and $\Delta \neq 0$, where Δ is the discriminant of E.

Equation (1) is called a Weierstrass form of an elliptic curve. In ECC, curves without any singular points are used. The Weierstrass equation of an elliptic curve can be transformed into simpler equations depending upon the characteristic of the underlying field. Binary and prime fields are two popular classes of fields used for implementations. In binary fields, if $a_1 \neq 0$, then (1) rewrites as

$$y^2 + xy = x^3 + ax^2 + b, \tag{2}$$

where $a, b \in K$. In this case, if $\Delta = b \neq 0$, the curve is said to be *non-supersingular*. Over prime fields, the Weierstrass equation takes the simpler form

$$y^2 = x^3 + ax + b, \tag{3}$$

where $a, b \in K$, and the curve is free of any singularity if $\Delta = 4a^3 + 27b^2 \neq 0$.

Let E be an elliptic curve defined over a field K and let P, Q be two points in $E(K)$. There exists a rule for *adding* P and Q, which generates a third point in $E(K)$. A geometrical interpretation is given by the fact that the three points P, Q and $-(P + Q) \in E(K)$ form a straight line. Together with this addition law and a special point $\mathcal{O}$, called the *point at infinity* playing the role of the identity, the set of points, $E(K)$, forms an abelian group. Given $P \in E(K)$ and $k \in \mathbb{N}$, the operation of computing the new point $k \times P$ is called point multiplication or scalar multiplication. In any implementation of ECC primitives, scalar multiplication is the computationally dominant operation. Several methods have been proposed in the literature to speed-up point multiplication,

which use various representations of the base point (affine coordinates, projective coordinates, ...), various representations of the scalar (binary, ternary, NAF, w-NAF, ...), and various curve operations (additions, doublings, halvings, triplings). The computational cost (timing) of these curve operations depends on the cost of the arithmetic operations that have to be performed in the underlying field. In general, addition and subtraction in the underlying field are operations of negligible cost. In the current article we will only take into account the number of inversions, squarings and multiplications, that we shall denote $[i], [s]$ and $[m]$ respectively. The choice of parameters used for a specific implementation very often depends upon the ratio, $[i]/[m]$, between one inversion and one multiplication. In binary fields it is assumed to be between 3 and 8 [11], whereas in prime fields it is between 30 and 50 [9]. Also, a squaring, $[s]$, is generally assumed to be roughly equal to $0.8[m]$ in prime fields, and almost free in binary field (see [11] for more details). We will take these considerations into account in the next sections, when we explain our algorithm and discuss its efficiency.

In the following Table 1, we give the costs of the curve operations we shall use in our analysis. The details of these operations can be found in [8, 2].

Curve operation	Prime field	Binary field
$P+Q$	$1[i]+1[s]+2[m]$	$1[i]+1[s]+2[m]$
$2P$	$1[i]+2[s]+2[m]$	$1[i]+1[s]+2[m]$
$2P+Q$	$1[i]+2[s]+9[m]$	$1[i]+2[s]+9[m]$
$3P$	$1[i]+4[s]+7[m]$	$1[i]+4[s]+7[m]$
$3P+Q$	$2[i]+4[s]+9[m]$	$2[i]+3[s]+9[m]$
$4P$	$1[i]+9[s]+9[m]$	$1[i]+5[s]+8[m]$
$4P+Q$	$2[i]+4[s]+11[m]$	$2[i]+6[s]+10[m]$

Table 1: Number of inversions $[i]$, squarings $[s]$ and multiplications $[m]$, for different curve operations over $\mathbb{F}_p$ and $\mathbb{F}_{2^m}$ using affine coordinates

2.2 Double-Base number system

For the last decade, a new number representation scheme has been a subject of intensive studies, due to its applicability in digital signal processing. This number representation is called double-base number system. We will need the following definition from B. M. M. de Weger [3].

Definition 2 (s-integer) *An s-integer is a positive integer, whose largest prime factor does not exceed the s-th prime number.*

Definition 3 (double-base number system) *The double-base number system (DBNS) is a representation scheme in which every positive integer, n, is represented as the sum or difference of 2-integers, that is, numbers of the form $2^a 3^b$.*

$$n = \sum_{i=1}^{m} s_i \, 2^{b_i} 3^{t_i}, \quad \text{with } s_i \in \{-1, 1\}, \text{ and } b_i, t_i \geq 0. \tag{4}$$

Clearly, this number representation scheme is highly redundant. If one considers the DBNS with only positive signs ($s_i = 1$), then certain interesting numerical and theoretical results can be proved. For instance, 10 has exactly five different DBNS representations; 100 has exactly 402 different DBNS representations and 1 000 has exactly 1 295 579 different DBNS representations. In this paper, we shall use the number 314159 as an example. It has exactly $49\,305\,013\,890\,836\,539\,593\,285 > 10^{22}$ different DBNS representations! Probably, the most important theoretical result about the double-base number system is the following theorem from [7].

Theorem 1 *Every positive integer, n, can be represented as the sum of at most $\mathcal{O}\left(\frac{\log n}{\log\log n}\right)$ 2-integers.*

The proof is based on Baker's theory of linear forms of logarithms and, more specifically on the following result by R. Tijdeman [16].

Theorem 2 *There exists an absolute constant, C, such that there is always a number of the form 2^b3^t between n and $n - \frac{n}{(\log n)^C}$.*

Some of these representations are of special interest, most notably the ones that require the minimal number of 2-integers; i.e., an integer can be represented as the sum of m 2-integers, but cannot be represented as the sum of $(m-1)$ 2-integers. We call these representations *canonic*. The canonic representations are extremely sparse. Some numerical facts provide a good impression about the sparseness of the DBNS: The smallest integer requiring three 2-integers in its canonic DBNS representations is 23. The next smallest integers requiring m 2-integers are 431, 18 431, 3 448 733 and 1 441 896 119 for $m = 4, 5, 6$ and 7. The next record-setter would be most probably bigger than one trillion. In all of the above results we have assumed only positive (+1) values for the s_i's. If one considers both signs, then the theoretical difficulties in establishing the properties of this number system dramatically increase. To wit, it is possible to prove that the smallest integer that cannot be represented as the sum or difference of two 2-integers is 103. The next limit is most probably 4985, but to prove it rigorously, one has to show that none of the following exponential Diophantine equations have a solution.

Conjecture 1 *The Diophantine equations*

$$\pm 2^a 3^b \pm 2^c 3^d \pm 2^e 3^f = 4985 \tag{5}$$

do not have solutions in integers.

Finding one of the canonic DBNS representations, especially for very large integers, seems to be a very difficult task. Fortunately, one can apply a greedy algorithm to find a fairly sparse representation very quickly. Given $n > 0$, the following algorithm computes an unsigned DBNS representation (i.e., with $s_i = +1$ only). It can be easily extended to signed DBNS representations (see Algorithm 3 in Section 4). Although the greedy algorithm sometimes fails in finding a canonic

Algorithm 1 A greedy algorithm to convert integers into DBNS

Input A positive integer n

Output the sequence of exponents (b_n, t_n) leading to one DBNS representation of n

1: **while** $n > 0$ **do**
2: Find $z = 2^b 3^t$, the largest 2-integer less than or equal to n
3: **print** (b, t)
4: $n \leftarrow n - z$

representation,[1] it is very easy to implement and, more to the point, it guarantees a representation requiring $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ 2-integers.

The whole point of the above mentioned numerical results is to provide the readers with some intuitive ideas about this number representation scheme and its potential use in various applications. In this paper we demonstrate how to use it in order to speed up the performance of the elliptic curve scalar multiplication.

3 New scalar multiplication algorithm

Let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and let $P \neq \mathcal{O}$ be a point on $E(\mathbb{F}_q)$. Assuming k is represented in DBNS, our goal is to compute the new point $kP \in E(\mathbb{F}_q)$. Before we present our new scalar multiplication algorithm, we define *double-base chains*, that will be used to evaluate the complexity of our algorithm.

Definition 4 (Double-Base Chain) *Given $k > 0$, a sequence $(K_n)_{n>0}$, of positive integers satisfying:*

$$K_1 = 1, \quad K_{n+1} = 2^u\, 3^v\, K_n + s, \text{ with } s \in \{-1, 1\} \tag{6}$$

for some $u, v \geq 0$, and such that $K_m = k$ for some $m > 0$, is called a double-base chain for k. The length, m, of a double-base chain is equal to the number of 2-integers in (4), used to represent k.

Now, suppose that $P \in E(\mathbb{F}_q)$ and $k > 0$ is represented in DBNS as

$$k = \sum_{i=1}^{m} s_i\, 2^{b_i} 3^{t_i}, \quad \text{with } s_i \in \{-1, 1\}, \tag{7}$$

and assume that the sequences $(b_i)_{i>0}, (t_i)_{i>0}$, of binary and ternary exponents decrease, i.e., $b_1 \geq b_2 \geq \cdots \geq b_m \geq 0$ and $t_1 \geq t_2 \geq \cdots \geq t_m \geq 0$. Algorithm 2 below computes the new point $kP \in E(\mathbb{F}_q)$. Note that for a given integer, such a representation always exists (e.g., the binary representation is a special case). In fact, this particular DBNS representation is also highly redundant. Counting the exact number of DBNS representations which satisfy those conditions is indeed a very interesting problem. But the only partial results we have at the moment are beyond the scope of this paper.

[1] The smallest example is 41; the canonic representation is $32 + 9$, whereas the greedy algorithm returns $41 = 36 + 4 + 1$

Algorithm 2 Double-Base Scalar Multiplication

Input An integer $k = \sum_{i=1}^{m} s_i 2^{b_i} 3^{t_i}$, with $s_i \in \{-1, 1\}$, and such that $b_1 \geq b_2 \geq \cdots \geq b_m \geq 0$, and $t_1 \geq t_2 \geq \cdots \geq t_m \geq 0$; and a point $P \in E(\mathbb{F}_q)$

Output the point $kP \in E(\mathbb{F}_q)$

1: $Z \leftarrow s_1 P$
2: **for** $i = 1, \ldots, m-1$ **do**
3: $\quad u \leftarrow b_i - b_{i+1}$
4: $\quad v \leftarrow t_i - t_{i+1}$
5: $\quad$ **if** $u = 0$ **then**
6: $\quad\quad Z \leftarrow 3(3^{v-1} Z) + s_{i+1} P$
7: $\quad$ **else**
8: $\quad\quad Z \leftarrow 3^v Z$
9: $\quad\quad Z \leftarrow 4^{\lfloor (u-1)/2 \rfloor} Z$
10: $\quad\quad$ **if** $u \equiv 0 \pmod 2$ **then**
11: $\quad\quad\quad Z \leftarrow 4Z + s_{i+1} P$
12: $\quad\quad$ **else**
13: $\quad\quad\quad Z \leftarrow 2Z + s_{i+1} P$
14: **Return** Z

Although $m-1$ additions are required to compute kP, we never actually use the curve operation, $P + Q$, because we combine each addition with either a doubling (Step 13), a tripling (Step 6) or a quadrupling (Step 11), by using algorithms for $2P + Q$, $3P + Q$ and $4P + Q$ respectively (see Table 1). We shall denote these operations using the abbreviations DA, TA, QA, with evident meanings. Note also that the TA operation for computing $3P + Q$ is only used in Step 6, when $u = 0$. Moreover, depending the underlying field and the ratios $[i]/[m]$ and $[s]/[m]$, we implement the quadruple of a point either as $4P$ or $2(2P)$. Over prime fields, Table 1 tells us that computing $4P$ using the quadrupling operation (Q) requires $1[i] + 9[s] + 9[m]$, whereas computing $2(2P)$ using two doublings (D) costs $2[i] + 4[s] + 4[m]$. If we assume that $[s] = 0.8[m]$, then computing $4P$ is cheaper as soon as $[i]/[m]$ is larger than 9. Over binary fields, assuming that squarings are free, the breaking point is $[i]/[m] = 4$; when $[i]/[m] < 4$, then computing $2(2P)$ is cheaper than $4P$. Thus, over prime fields, when $[i]/[m]$ is usually larger than 9 we compute $4Z$ and $4Z + 1$ using the quadrupling (Q) and quadruple-and-add (QA) curve operations whenever it is possible. Over binary fields, however, when $[i]/[m]$ is usually between 3 and 8, the choice between doublings and quadruplings operations has to be made according to the exact value of $[i]/[m]$.

In order to evaluate the complexity of Algorithm 2 we have to count the number of curve operations (i.e., the number of D, DA, T, TA, Q, QA), which clearly depend on the DBNS representation of k. In fact, Algorithm 2 gives us K_m, a double-base chain for k, that we can use to determine the number of curve operations required to evaluate kP. We define W_n as the number of curve operations required to compute $K_n P$ from $K_{n-1} P$. We have $K_1 = 1$ and $W_1 = 0$ (in Step

1, we set Z to P or $-P$ at no cost). Then, for $n > 1$ we have

$$W_{n+1} = \delta_{u,0}\left((v-1)\,T + TA\right) + (1-\delta_{u,0})\left(v\,T + \left\lfloor \frac{u-1}{2} \right\rfloor Q + \delta_{|u|_2,0}\,QA + \delta_{|u|_2,1}\,DA\right), \qquad (8)$$

where $\delta_{i,j}$ is the Kronecker delta such that $\delta_{i,j} = 1$ if $i = j$ and $\delta_{i,j} = 0$ if $i \neq j$, and $|u|_2$ denotes $u \bmod 2$ (the remainder of u in the division by 2). When $[i]/[m] \leq 9$ (which is usually assumed to be the case over binary fields), we have seen that performing two consecutive doublings is more efficient than one quadrupling. Thus, in this case, we express the cost as

$$W_{n+1} = \delta_{u,0}\left((v-1)\,T + TA\right) + (1-\delta_{u,0})\left(v\,T + (u-1)D + DA\right), \qquad (9)$$

The total cost for computing kP from the input point P is thus given by

$$W_m = \sum_{i=1}^{m} W_i. \qquad (10)$$

As an example, let us compute $314159P$, using the following DBNS representation:

$$314159 = 2^{12}3^4 - 2^{11}3^2 + 2^8 3^1 + 2^4 3^1 - 2^0 3^0. \qquad (11)$$

Starting from P, Algorithm 2 successively computes $17P$, $409P$, $6545P$ and $314159P$. The sequence $(1, 17, 6545, 314159)$ is the corresponding double-base chain for 314159. It gives us the number of curve operations required to compute kP (see Table 2). Assuming we work over prime fields, then

i	K_i	s	u	v	W_i from (8)
1	1	1	0	0	0
2	$18K_1 - 1 = 17$	-1	1	2	$2T + DA$
3	$24K_2 + 1 = 409$	$+1$	3	1	$T + Q + DA$
4	$16K_3 + 1 = 6545$	$+1$	4	0	$Q + QA$
5	$48K_4 - 1 = 314159$	-1	4	1	$T + Q + QA$
Total cost:					$2DA + 4T + 3Q + 2QA$

Table 2: Total number of curve operations for computing $314159P$ using the double base representation of $k = 314159$ given by (11)

from Table 1, the total cost in terms of field operations is $13[i] + 40[s] + 95[m]$, which represent a saving of $4[i] + 1[s] + 2[m]$ over the 2-NAF method and of $2[i] + 1[s]$ over a recent binary/ternary approach proposed in [2]. The next section is dedicated to comparisons for prime and binary fields.

4 Comparisons

The idea of using a mixed binary/ternary approach have been recently proposed by M. Ciet et al. in [2]. In this section we analyze their solution using the double-base terminology and we give

some experimental results for randomly chosen numbers of cryptographic interest, which confirm the advantages of our solution.

The algorithm proposed in [2] is based on the following recursive decomposition of k. If $k \equiv 0$ or $3 \pmod 6$, return $3((k/3)P)$; if $k \equiv 2$ or $4 \pmod 6$, return $2((k/2)P)$; if $k \equiv 1 \pmod 6$, i.e., $k = 6m + 1$, return $2((3m)P) + P$; if $k \equiv 5 \pmod 6$, i.e., $k = 6m - 1$, return $2((3m)P) - P$. The recursion stops whenever $k = 1$ and returns P. Basically, this procedure can be considered as a binary/ternary tree decomposition of k. Almost the same technique has been considered for the case of classical modular exponentiation in [5], and in the case of computing the matrix polynomial, $G(N, A) = \sum_{i=0}^{N-1} A^i$, for a given square matrix A in [4]. Following [5], some interesting facts can be deduced to analyze the algorithm proposed in [2]. If one considers a randomly chosen scalar, k, then its remainder modulo 6 is equal to 0 and 3 about 23.08 % of the time; it is equal to 2 and 4 about 38.46 % of the time; to 1 about 15.38 % of the time and to 5 about 23.08 % of the time. We give these figures to show that the solution proposed in [2] can be more carefully analyzed. However, we believe it is beyond the scope of this paper.

If we apply the recursive algorithm mentioned above to $k = 314159$, we obtain the following DBNS representation: $314159 = 2^9 3^6 - 2^8 3^5 + 2^6 3^4 - 2^5 3^3 - 2^0 3^0$. This particular representation, of course, does satisfy the requirements that the ternary and binary exponents form two decreasing sequences. But, thanks to the huge redundancy of the DBNS, it is possible to seek for better representations, having the same property on the exponents and, at the same time, leading to better double-base chains and reduced complexity. However, the immense redundancy also has a back side, since it seems impossible to determine an optimal double-base chain for a given integer k.

Let us consider the following heuristic solution which leads to very good result on average. If $k = k_{n-1} \dots k_1 k_0$ is a randomly chosen n-bit integer (i.e., with $k_{n-1} \neq 0$), we compute the DBNS representation of k using a slightly modified version of the greedy algorithm presented in Section 2.2, by imposing some restrictions on the largest possible binary and ternary exponents. In Algorithm 3 below, we initially set $b_{max} = x$ and $t_{max} = y$, where $2^x 3^y$ is a very good (possibly the best one) non-trivial (with $y \neq 0$) approximation of 2^n. Then, in order to get decreasing sequences for the b_i's and t_i's, the new largest exponents are updated according to the values of b and t obtained in Step 3. Note that Algorithm 3 also takes into account positive and negative signs in order to generate a DBNS representation which satisfy Definition 3.

In order to compare our algorithm with other solutions, we have performed several experiments for many randomly chosen integers of different cryptographic sizes. We present the results for prime fields in Table 3, and binary fields in Table 4. For each scalar size, ranging from 160 to 256 bits for prime fields and from 163 bits to 283 bits for binary fields, we have counted the number of curve operations (D, DA, T, TA, Q, QA) and the corresponding number of field operations ($[i]$, $[s]$, $[m]$). We present the results obtained by setting the initial largest exponents, $b_{max} > t_{max}$, which correspond to the best non-trivial approximations of 2^n, where n is the size of k in bits.[2] We

[2] In some interesting cases, we also give the costs for some other very good approximation of 2^n.

Algorithm 3 Conversion to DBNS with restricted exponents

Input k, a n-bit positive integer; $b_{max}, t_{max} > 0$, the largest allowed binary and ternary exponents respectively

Output The sequence $(s_i, b_i, t_i)_{i>0}$ such that $k = \sum_{i=1}^{m} s_i\, 2^{b_i}\, 3^{t_i}$, with $b_1 \geq \cdots \geq b_m \geq 0$ and $t_1 \geq \cdots \geq t_m \geq 0$

1: $s \leftarrow 1$
2: **while** $k > 0$ **do**
3: define $z = 2^b 3^t$, the best approximation of k with $0 \leq b \leq b_{max}$ and $0 \leq t \leq t_{max}$
4: **print** (s, b, t)
5: $b_{max} \leftarrow b$
6: $t_{max} \leftarrow t$
7: **if** $k < z$ **then**
8: $s \leftarrow -s$
9: $k \leftarrow |k - z|$

also give the operation counts for the binary/ternary algorithm presented in [2].

If we assume that k is a random n-bit integer, the binary algorithm requires n doublings and about $n/2$ additions on average. In the 2-NAF method, the average density of non-zero digits is reduced to $n/3$. In general, if k is represented in a w-NAF form, the average number of additions is about $n/(w+1)$ (without taking into account the necessary precomputations). For 160-bit integers, and over prime fields, the corresponding costs can be estimated to $160[i] + 320[s] + 880[m]$ with the binary method, $106[i] + 320[s] + 691[m]$ with the 2-NAF method, and $160[i] + 192[s] + 544[m]$ with the 4-NAF method. In Table 3, we remark that our algorithm requires fewer inversions than any other solution, at the extra cost of a few number of squarings and multiplications.

According to our simulations, if we consider 160-bit integers and a ratio $[i]/[m] = 30$, our algorithm represents a gain of 20 % compared to the classical binary method; 17.4 % compared to the 2-NAF method; and 15.25 % compared with the 4-NAF method. It also surpasses the binary/ternary approach proposed in [2] by about 2.8% for 160-bit numbers; 5.3 % for 192-bit numbers; 5.1 % for 224-bit numbers; and 4.7% for 256-bit numbers. Even if we consider a smaller ratio, say $[i]/[m] = 10$ (which seems more realistic if one uses affine coordinates), our algorithm performs better than the other classical solutions. Note also that it does not require any precomputation.

Over binary fields, the ratio $[i]/[m]$ is much smaller; between 3 and 8. As mentioned earlier, we compute $4P$ as $2(2P)$ using two consecutive doublings rather than one quadrupling. The results are presented in Table 4. Since squarings are almost free over binary fields [11], we have inserted the number of squaring in Table 4 for completeness only, but we have omitted them for the evaluation of the overall cost. Again, our algorithm requires fewer operations than the other approaches. The 4-NAF method is more efficient when the ratio $[i]/[m] = 3$, but for 163-bit integers and $[i]/[m] = 8$, we obtain a gain of 16.5 % over the classical binary method; 8.6 % over the 2-NAF method; and about 1.5 % over the 4-NAF method. Compared to the binary/ternary solution proposed in [2],

b_{max}, t_{max}	D	DA	T	TA	Q	QA	$[i]$	$[s]$	$[m]$
$\|\boldsymbol{k}\| = \mathbf{160}$									
$76, 53$	-	15	43	10	18	12	119	453	814
[2]	37	36	55	-	-	-	127	364	782
$\|\boldsymbol{k}\| = \mathbf{192}$									
$108, 53$	-	20	46	6	27	16	138	562	983
[2]	43	44	66	-	-	-	153	438	945
$\|\boldsymbol{k}\| = \mathbf{224}$									
$140, 53$	-	24	47	6	36	21	161	670	1157
[2]	51	51	77	-	-	-	179	511	1096
$\|\boldsymbol{k}\| = \mathbf{256}$									
$172, 53$	-	30	47	6	44	27	186	771	1342
$153, 65$	-	27	58	7	39	24	185	758	1320
[2]	57	58	89	-	-	-	204	585	1257

Table 3: Average number of curve operations and arithmetic operations over $\mathbb{F}_p$ for different sizes of k

b_{max}, t_{max}	D	DA	T	TA	$[i]$	$[s]$	$[m]$
$\|\boldsymbol{k}\| = \mathbf{163}$							
$98, 41$	65	32	36	5	143	357	712
$79, 53$	51	27	42	10	141	367	735
[2]	38	37	55	-	130	371	795
$\|\boldsymbol{k}\| = \mathbf{233}$							
$149, 53$	99	49	47	6	207	507	1019
[2]	54	52	80	-	186	531	1138
$\|\boldsymbol{k}\| = \mathbf{283}$							
$199, 53$	131	66	47	6	256	606	1239
$115, 106$	75	39	77	28	248	651	1298
[2]	67	64	96	-	227	644	1380

Table 4: Average number of curve operations and arithmetic operations over $\mathbb{F}_{2^m}$ for different sizes of k

our algorithm saves up to 3.7% for 163-bit numbers; about 3.3% for 233-bit numbers; and about 2.6% for 283-bit numbers. The relatively small improvements obtained in the case of binary fields can be explained by the fact that we have chosen to compute $4P$ using two consecutive doublings always, although quadrupling becomes cheaper as soon as $[i]/[m]$ is larger than 4. Note that those percentages are obtained for $[i]/[m] = 3$. If we consider quadruplings and $[i]/[m] = 8$, then the gain are more important.

5 Discussions and Conclusions

In this paper, we have presented an original and efficient elliptic curve point multiplication algorithm based on the double-base number system. We hope, the reader has understood the very deep theoretical questions hidden behind the DBNS, particularly the impossibility to define precisely the average number of 2-integers required to represent any given integer in the signed DBNS representation. We introduced the novel concept of double-base chains and we use it in order to evaluate the complexity of our algorithm. We have presented experimental results which show the efficiency of our solution. Our algorithm, which generalizes and improves upon the binary/ternary approach proposed in [2], also compares favorably against all other classic methods (binary, w-NAF).

The next steps will be the extension of this algorithm to projective coordinates and hyper-elliptic curves; as well as a thorough analysis of its side-channel properties. Actually, not only we believe the DBNS must be considered as an alternative for someone willing to implement ECC efficiently, we also think that it offers some natural, elegant, protections against some side-channel attacks. We believe that the huge redundancy of the representation, can be advantageously exploited in order to randomly change the order of the operations, such that the same scalar does not give the same trace if it is processing several times. Other ideas like side-channel atomicity [1] can be adapted to counteract simple side-channel attacks. These are just very brief comments about side-channels, that we are now going to investigate more deeply. Of course, classical counter-measures, such as randomization of the message and the exponent can be applied.

References

[1] B. Chevalier-Mames, M. Ciet, and M. Joye. Low-cost solutions for preventing simple side-channel analysis: Side-channel atomicity. *IEEE Transactions on Computers*, 53(6):760–768, June 2004.

[2] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery. Trading inversions for multiplications in elliptic curve cryptography. Cryptology ePrint Archive, Report 2003/257, 2003.

[3] B. M. M. de Weger. *Algorithms for Diophantine equations*, volume 65 of *CWI Tracts*. Centrum voor Wiskunde en Informatica, Amsterdam, 1989.

[4] V. S. Dimitrov and T. V. Cooklev. Hybrid algorithm for the computation of the matrix polynomial $I + A + \cdots + A^{N-1}$. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 42(7):377–380, July 1995.

[5] V. S. Dimitrov and T. V. Cooklev. Two algorithms for modular exponentiation based on non-standard arithmetics. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Science*, E78-A(1):82–87, January 1995. Special issue on cryptography and information security.

[6] V. S. Dimitrov and G. A. Jullien. Loading the bases: A new number representation with applications. *IEEE Circuits and Systems Magazine*, 3(2):6–23, 2003.

[7] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. An algorithm for modular exponentitation. *Information Processing Letters*, 66(3):155–159, May 1998.

[8] K. Eisenträger, K. Lauter, and P. L. Montgomery. Fast elliptic curve arithmetic and improved weil pairing evaluation. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *LNCS*, pages 343–354. Springer-Verlag, 2003.

[9] K. Fong, D. Hankerson, J. Lòpez, and A. Menezes. Field inversion and point halving revisited. *IEEE Transactions on Computers*, 53(8):1047–1059, Aug. 2004.

[10] D. M. Gordon. A survey of fast exponentiation methods. *Journal of Algorithms*, 27(1):129–146, April 1998.

[11] D. Hankerson, J. Lòpez Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2000*, volume 1965 of *LNCS*, pages 1–24. Springer-Verlag, 2000.

[12] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2004.

[13] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[14] A. Menezes, P. C. Van Oorschot, and S. A. Vanstone. *Handbook of applied cryptography*. CRC Press, 1997.

[15] V. S. Miller. Uses of elliptic curves in cryptography. In H. C. Williams, editor, *Advances in Cryptology – CRYPTO '85*, volume 218 of *LNCS*, pages 417–428. Springer-Verlag, 1986.

[16] R. Tijdeman. On the maximal distance between integers composed of small primes. *Compositio Mathematica*, 28:159–162, 1974.