Efficient and Secure Elliptic Curve Point Multiplication using Double-Base Chains

Vassil Dimitrov[1,2], Laurent Imbert[1,2,3], and Pradeep Kumar Mishra[2]

[1] Advanced Technology Information Processing Systems laboratory,
University of Calgary, Canada
`dimitrov@atips.ca`

[2] Centre for Informations Security and Cryptography,
University of Calgary, Canada
`pradeep@math.ucalgary.ca`

[3] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier,
CNRS UMR 5506, Montpellier, France
`Laurent.Imbert@lirmm.fr`

Abstract. In this paper, we propose a efficient and secure point multiplication algorithm, based on double-base chains. This is achieved by taking advantage of the sparseness and the ternary nature of the so-called double-base number system (DBNS). The speed-ups are the results of fewer point additions and improved formulæ for point triplings and quadruplings in both even and odd characteristic. Our algorithms can be protected against simple and differential side-channel analysis by using side-channel atomicity and classical randomization techniques. Our numerical experiments show that our approach leads to speed-ups compared to windowing methods, even with window size equal to 4, and other SCA resistant algorithms.

1 Introduction

Elliptic curve cryptography (ECC) [24, 21] has rapidly received a lot of attention because of its small key-length and increased theoretical robustness (there is no known subexponential algorithm to solve the ECDLP problem, which is the foundation of ECC). The efficiency of an ECC implementation mainly depends on the way we implement the *scalar* or *point multiplication*; i.e., the computation of the point $kP = P + \cdots + P$ (k times), for a given point P on the curve. A vast amount of research has been done to accelerate and secure this operation, using various representations of the scalar k (binary, ternary, non-adjacent form (NAF), window methods (w-NAF) , Frobenius expansion,...), various systems of coordinates (affine, projective,...) and various randomization techniques. See [15, 4, 1] for complete presentations.

In this paper, we propose new scalar multiplication algorithms based on a representation of the multiplier as a sum of mixed powers of 2 and 3, called the *double-base number system* (DBNS). The inherent sparseness of this representation scheme leads to fewer point additions than other classical methods. For example, if k is a randomly chosen 160-bit integer, then one needs only about 22 summands to represent it, as opposed to 80 in standard binary representation and 53 in the non-adjacent form (NAF).

In order to best exploit the sparse and ternary nature of the DBNS, we also propose new formulæ for point tripling and quadrupling for curves defined over binary fields and points in affine coordinates; and for prime fields using Jacobian coordinates. Our algorithms can be protected against side-channel attacks (SCA) by using *side-channel atomicity* [5] for simple analysis, and, in the odd case, using a point randomization method proposed by Joye and Tymen [20] for differential analysis.

2 Background

In this section, we give a brief overview of elliptic curve cryptography (see [1, 3, 4, 15] for more details) and the double-base number system.

2.1 Elliptic Curve Cryptography

Definition 1. *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.

In practice, the Weierstrass equation (1) can be greatly simplified by applying admissible changes of variables. If the characteristic of K is not equal to 2 and 3, then (1) rewrites

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

where $a, b \in K$, and $\Delta = 4a^3 + 27b^2 \neq 0$.

When the characteristic of K is equal to 2, we use the *non-supersingular* form of an elliptic curve, given for $a \neq 0$ by

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

where $a, b \in K$ and $\Delta = b \neq 0$.

The set $E(K)$ of rational points on an elliptic curve E defined over a field K is an abelian group, where the operation (generally denoted additively) is defined by the well-known law of chord and tangent, and the identity element is the special point $\mathcal{O}$, called *point at infinity.*

If the points on the curve are represented using affine coordinates, as $P = (x, y)$, both the *point addition* (ADD) and *point doubling* (DBL) involve an expensive field inversion (to compute the slope of the chord or the tangent). To avoid these inversions, several projective systems of coordinates have been proposed in the literature. The choice of a coordinates system has to be made according to the so-called $[i]/[m]$ ratio between one field inversion and one field multiplication. It is generally assumed that $3 \leq [i]/[m] \leq 10$ for binary fields [8, 14] and 30 or more for prime fields [12]. In this paper we consider affine ($\mathcal{A}$) coordinates for curves defined over binary fields and Jacobian ($\mathcal{J}$) coordinates, where the point $P = (X, Y, Z)$ corresponds to the point $(X/Z^2, Y/Z^3)$ on the elliptic curve for curves defined over fields of odd characteristic.

As we shall see, our DBNS-based point multiplication algorithms use several primitives. In the following lines, we give a very brief description and the complexities of some previously published point arithmetic algorithms. We also propose improved primitives and new formulæ in Section 4.

In the following, we will use $[i]$, $[s]$ and $[m]$ to denote the cost of one inversion, one squaring and one multiplication respectively. We shall always leave out the cost of field additions. In binary fields, we assume that squarings are free (if normal bases are used) or of negligible cost (linear operation). Moreover, for curves defined over large prime fields, we will assume that $[s] = 0.8[m]$. Note that our algorithm can be protected against SCA (see Section 2.2) using side-channel atomicity [5], which we have shown in the case of prime fields. In this case, squarings and multiplications must be performed using the same multiplier in order to be indistinguishable, and we must consider $[s] = [m]$.

For fields of even characteristic, we use affine coordinates and we consider doublings (DBL), triplings (TPL) and quadruplings (QPL) as well as the combined double-and-add (DA), triple-and-add (TA) and quadruple-and-add (QA). It is easy to verify that ADD and DBL can be computed in $1[i] + 1[s] + 2[m]$. In [11], K. Eisenträger et al. have proposed efficient algorithms for DA, TPL and TA. By trading some inversions for a small number of multiplications, these results have been further improved when $[i]/[m] > 6$ in [6]. In Table 1 below, we give the complexities of each of these primitives. We also give the break-even points between the different formulæ.

Table 1. Costs comparisons and break-even points for DA, T and TA over binary fields using affine coordinates

Operation	[11]	[6]	break-even point
$2P \pm Q$	$2[i] + 2[s] + 3[m]$	$1[i] + 2[s] + 9[m]$	$[i]/[m] = 6$
$3P$	$2[i] + 2[s] + 3[m]$	$1[i] + 4[s] + 7[m]$	$[i]/[m] = 4$
$3P \pm Q$	$3[i] + 3[s] + 4[m]$	$2[i] + 3[s] + 9[m]$	$[i]/[m] = 5$

When Jacobian coordinates are used and the curve is defined over a prime field (or a field of odd characteristic > 3), the addition and doubling operations, that we will denote $\text{ADD}^{\mathcal{J}}$ and $\text{DBL}^{\mathcal{J}}$ in this paper, require $12[m] + 4[s]$ and $4[m] + 6[s]$ respectively. The cost of $\text{DBL}^{\mathcal{J}}$ can be reduced to $4[m] + 4[s]$ when $a = -3$ in (2). Also, if the base point is given in affine coordinates ($Z = 1$), then the cost of the so-called *mixed addition* ($\mathcal{J} + \mathcal{A} \rightarrow \mathcal{J}$) reduces to $8[m] + 3[s]$. When several doublings have to be computed, as for the computation of $2^w P$, the algorithm proposed by Itoh et al. in [16] is more efficient than w invocations of $\text{DBL}^{\mathcal{J}}$. In the general case ($a \neq -3$) it requires $4w[m] + (4w+2)[s]$. In Table 2, we summarize the complexity of these different elliptic curve primitives.

Table 2. Complexity of several elliptic curve operations in Jacobian coordinates for fields of odd characteristic $\neq 3$

Curve operation	Complexity	# Registers
$\mathrm{DBL}^{\mathcal{J}}$	$4[m] + 6[s]$	6
$\mathrm{DBL}^{\mathcal{J},\, a=-3}$	$4[m] + 4[s]$	5
$\mathrm{ADD}^{\mathcal{J}}$	$12[m] + 4[s]$	7
$\mathrm{ADD}^{\mathcal{J}+\mathcal{A}}$	$8[m] + 3[s]$	7
w-$\mathrm{DBL}^{\mathcal{J}}$	$4w[m] + (4w + 2)[s]$	7

2.2 Preventing side-channel analysis

Side-channel attacks (SCA) are one of the most serious threat to ECC implementations. Discovered by Kocher et al. [23, 22], these attacks can reveal a secret information by sampling and analyzing various side-channel information (e.g. timing, power consumption, electromagnetic radiations) of a device. SCA can be divided into two types: *simple attacks* which observe only one trace given by a single execution of the algorithm, and *differential attacks* which use many observations and try to reveal the secret using statistical tools. Protecting ECC implementations against SCA has itself become an interesting area of research and several countermeasures have been proposed. Interested readers can refer to [4, 1] for details.

In the current work we will use a solution proposed by Chavalier-Mames et al. in [5] to protect against simple attacks, called *side-channel atomicity.* The countermeasure is based on the simple observation that some elementary operations are side-channel equivalent in the sense that they are indistinguishable (or can be made so by clever software implementation) from the side-channel.

2.3 Double-Base Number System

The double-base number system (DBNS) [10] is a representation scheme in which every positive integer k is represented as the sum or difference of $\{2, 3\}$-integers (i.e., numbers of the form $2^b 3^t$) as

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

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 1000 has exactly 1 295 579 different DBNS representations. Probably, the most important theoretical result about the double-base number system is the following theorem from [9].

Theorem 1. *Every positive integer k can be represented as the sum of at most $O\left(\dfrac{\log k}{\log \log k}\right)$ $\{2, 3\}$-integers.*

The proof is based on Baker's theory of linear forms of logarithms and more specifically on a result by R. Tijdeman [25].

Some of these representations are of special interest, most notably the ones that require the minimal number of $\{2,3\}$-integers; i.e., an integer can be represented as the sum of m terms ($\{2,3\}$-integers), but cannot be represented as the sum of $m-1$ or less. These representations, called *canonic* representations, are extremely sparse. Some numerical facts provide a good impression about the sparseness of the DBNS. The smallest integer requiring three $\{2,3\}$-integers in its canonic DBNS representations is 23. The next smallest integers requiring 4-to-7 $\{2,3\}$-integers are 431, 18 431, 3 448 733 and 1 441 896 119 respectively. 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 $\{2,3\}$-integers is 103. The next limit is conjectured to be 4985, but to prove it rigorously, one has to prove that the Diophantine equations $\pm 2^a 3^b \pm 2^c 3^d \pm 2^e 3^f = 4985$ 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 $k > 0$, find the largest number of the form $z = 2^b 3^t$ less than or equal to k, and apply the same procedure with $k - z$ until reaching zero. Although the greedy algorithm sometimes fails in finding a canonic representation[4], it is very easy to implement and it guarantees a representation satisfying the asymptotic bound given by Theorem 1 (see [9]).

In this paper, we will use a slightly modified version of the greedy algorithm in order to find a DBNS representation of the scalar k of particular form, well adapted to fast and secure elliptic curve point multiplication. In the next section, we introduce the concept of double-base chains and the corresponding scalar multiplication algorithms.

3 Double-Base Chain and Point Multiplication

Let E be an elliptic curve defined over K, and let $P \neq \mathcal{O}$ be a point on $E(K)$. Assuming k is represented in DBNS, our new scalar multiplication algorithm computes the new point $kP \in E(K)$, by using the so-called *double-base chain* as defined below.

Definition 2 (Double-Base Chain). *Given $k > 0$, a sequence $(C_n)_{n>0}$ of positive integers satisfying:*

$$C_1 = 1, \quad C_{n+1} = 2^u 3^v C_n + s, \ \textit{with } s \in \{-1, 1\} \tag{5}$$

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

for some $u, v \geq 0$, and such that $C_m = k$ 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 $\{2,3\}$-integers in (4) used to represent k.

Let $k > 0$ be an integer represented in DBNS as $k = \sum_{i=1}^{m} s_i\, 2^{b_i} 3^{t_i}$, with $s_i \in \{-1, 1\}$, where the b_i's and t_i's form two decreasing sequences; i.e., $b_1 \geq b_2 \geq \cdots \geq b_m \geq 0$ and $t_1 \geq t_2 \geq \cdots \geq t_m \geq 0$. These particular DBNS representations allow us to expand k in a Horner-like fashion such that all partial results can be reused.

We first remark that 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 these conditions is indeed a very interesting problem, but the only partial results we have at the moment are beyond the scope of this paper.

If necessary, such a particular DBNS representation for k can be computed using Algorithm 1 below, which is a modified version of the greedy algorithm briefly described in Section 2.3. Two important parameters of this algorithm are

Algorithm 1 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
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:     $b_{max} \leftarrow b$,    $t_{max} \leftarrow t$
6:     if $k < z$ then
7:         $s \leftarrow -s$
8:     $k \leftarrow |k - z|$
```

the upper bounds for the binary and ternary exponents in the expansion of k, called b_{max} and t_{max} respectively. Clearly, we have $b_{max} < \log_2(k) < n$ and $t_{max} < \log_3(k) \approx 0.63n$. We noticed that using these utmost values for b_{max} and t_{max} do not result in short expansion. Instead, we consider the following heuristic which leads to very good results: if $k = (k_{n-1} \ldots k_1 k_0)_2$ is a randomly chosen n-bit integer (with $k_{n-1} \neq 0$), we initially set $b_{max} = x$ and $t_{max} = y$, where $2^x 3^y$ is a very good, non-trivial (with $y \neq 0$) approximation of 2^n. (Specific values are given in Table 7 for $n = 160$.) Then, in order to get decreasing sequences for 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.

The complexity of Algorithm 1 mainly depends on the way we implement Step 3; finding the best approximation of k of the form $z = 2^b 3^t$. If we can afford the storage of all the mixed powers of 2 and 3, this can be implemented very easily using a search over an ordered table of precomputed values. Otherwise, we can use an efficient solution recently proposed in [2] based on continued fractions and Ostrowski's number system. In both cases, the complexity of the conversion is negligible compared to the cost of the scalar multiplication. However, it is important to remark that, in most cases, the conversion into DBNS might not be needed. Indeed, in most ECC protocols, the multiplier k is a randomly chosen integer. We can thus directly generate a random DBNS number in the required form. Also, when k is part of a secret key, the conversion into DBNS can be done offline and even further optimized, when computation time is not an issue.

In the next sections, we present two versions of the DBNS-based point multiplication algorithm. We shall refer to the even case for curves defined over binary fields, when affine coordinates are used; and to the odd case for curves defined over large prime fields (or more generally any field of odd characteristic greater than 3), when Jacobian coordinates are preferred.

3.1 Point Multiplication in Even Characteristic

In even characteristic, i.e., with $P \in E(\mathbb{F}_{2^n})$ and k defined as above, Algorithm 2 below, computes the new point kP. We remark that although $m-1$ additions are

Algorithm 2 Double-Base Scalar Multiplication in even characteristic

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

required to compute kP, we never actually use the addition operation (ADD); simply because we combine each addition with either a doubling (Step 13), a tripling (Step 6) or a quadrupling (Step 11), using the DA, TA and QA primitives. Note also that the TA operation for computing $3P \pm Q$ is only used in Step 6, when $u = 0$. Another approach of similar cost is to start with all the quadruplings plus one possible doubling when u is odd, and then perform $v - 1$

triplings followed by one final triple-and-add. We present new algorithms for $4P$ and $4P \pm Q$ in Section 4.

In order to evaluate the complexity of Algorithm 2, we have to count the number of curve operations; i.e., the number of DBL, DA, TPL, TA, QPL, QA, which clearly depends on the DBNS representation of the scalar k. In fact, Algorithm 2 gives us a double-base chain for k, say K_m, that we can use to determine the number of curve operations required to evaluate kP. Let us define W_n as the number of curve operations required to compute K_nP 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

$$\begin{aligned} W_{n+1} = {} & \delta_{u,0}\left((v-1)\,T + TA\right) \\ & + (1-\delta_{u,0})\left(v\,T + \left\lfloor \frac{u-1}{2} \right\rfloor Q + \delta_{|u|_2,0}\,QA + \delta_{|u|_2,1}\,DA\right) , \end{aligned} \quad (6)$$

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

$$W_m = \sum_{i=1}^{m} W_i \ . \quad (7)$$

In Section 5, we illustrate the efficiency of this algorithm by providing comparisons with classical methods and a recently proposed ternary/binary approach [6].

3.2 Point multiplication in Odd Characteristic

For fields of odd characteristic > 3, when primitives in Jacobian coordinates are more efficient, Algorithm 3 below is used to compute kP. It takes advantage of the known w-DBL$^{\mathcal{J}}$ and ADD$^{\mathcal{J}+\mathcal{A}}$ formulæ recalled in Section 2.1 and the new TPL$^{\mathcal{J}}$, w-TPL$^{\mathcal{J}}$ and w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$ proposed in Section 4. Its complexity

Algorithm 3 Double-Base Scalar Multiplication in Odd Characteristic > 3

```
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 $P \in E(K)$
Output  the point $kP \in E(K)$
1: $Z \leftarrow s_1 P$
2: for $i = 1, \ldots, m-1$ do
3:     $u \leftarrow b_i - b_{i+1}$,    $v \leftarrow t_i - t_{i+1}$
4:     $Z \leftarrow 3^v Z$
5:     $Z \leftarrow 2^u Z$
6:     $Z \leftarrow Z + s_{i+1} P$
7: Return $Z$
```

depends on the number of doublings, triplings and mixed additions that have to be performed. Clearly, the total number of (mixed) additions is equal to the length m of the double-base chain for k, or equivalently the number of $\{2,3\}$-integers in its DBNS representation. Also, the number of doublings and triplings are equal to $b_1 \leq b_{max}$ and $t_1 \leq t_{max}$ respectively. However, the field cost can be more precisely evaluated if one considers the exact complexity of each iteration, by counting the exact number of field multiplications and squarings required in Steps 4 and 5 by the consecutive calls to v-TPL and u-DBL. In Section 5, we make this complexity analysis more precise and we compare our new approach with several previous algorithms recognized for their efficiency.

4 New Point Arithmetic Algorithms

In this section we present new formulæ for point quadrupling (QPL) and combined quadruple-and-add (QA) in even characteristic, and for triplings (TPL$^{\mathcal{J}}$, w-TPL$^{\mathcal{J}}$ and w-TPL$^{\mathcal{J}}/w'$-DBL$^{\mathcal{J}}$) in odd characteristic, to be used in conjunction with the proposed point multiplication algorithms.

4.1 New algorithms for $4P$ and $4P \pm Q$ in Even Characteristic

We remark that the trick used in [11] by Eisenträger et al., which consists in evaluating only the x-coordinate of $2P$ when computing $2P \pm Q$, can also be applied to speed-up the quadrupling (QPL) primitive. Indeed, given $P = (x_1, y_1)$, where $P \neq -P$, we have $2P = (x_3, y_3)$, where

$$\lambda_1 = x_1 + \frac{y_1}{x_1}, \quad x_3 = \lambda_1^2 + \lambda_1 + a, \quad y_3 = \lambda_1(x_1 + x_3) + x_3 + y_1,$$

and $4P = 2(2P) = (x_4, y_4)$, where

$$\lambda_2 = x_3 + \frac{y_3}{x_3}, \quad x_4 = \lambda_2^2 + \lambda_2 + a, \quad y_4 = \lambda_2(x_1 + x_4) + x_4 + y_1 \ .$$

We observe that the computation of y_3 can be avoided by evaluating λ_2 as

$$\lambda_2 = \frac{x_1^2}{x_3} + \lambda_1 + x_3 + 1 \ . \tag{8}$$

As a result, computing $4P$ over binary fields requires $2[i]+3[s]+3[m]$. Compared to two consecutive doublings, it saves one field multiplication at the extra cost of one field squaring. Note that we are working in characteristic two and thus squarings are free (normal basis) or of negligible cost (linear operation in binary fields).

For the QA operation, we evaluate $4P \pm Q$, as $2(2P) \pm Q$ using one doubling (DBL) and one double-and-add (DA), resulting in $3[i] + 3[s] + 5[m]$. This is always better than applying the previous trick one more time by computing $(((P+Q)+P)+P)+P)$ in $4[i]+4[s]+5[m]$; or evaluating $3P+(P+Q)$ which requires $4[i] + 4[s] + 6[m]$.

In [6], Ciet et al. have improved an algorithm by Guajardo and Paar [13] for the computation of $4P$; their new method requires $1[i] + 5[s] + 8[m]$. Based on their costs, QA is best evaluated as $(4P) \pm Q$ using one quadrupling (QPL) followed by one addition (ADD) in $2[i] + 6[s] + 10[m]$. In Table 3 below, we summarize the costs and break-even points between our new formulæ and the algorithms proposed in [6].

Table 3. Costs comparisons and break-even points for QPL and QA in even characteristic using affine coordinates

Operation	present work	[6]	break-even point
$4P$	$2[i] + 3[s] + 3[m]$	$1[i] + 5[s] + 8[m]$	$[i]/[m] = 5$
$4P \pm Q$	$3[i] + 3[s] + 5[m]$	$2[i] + 6[s] + 10[m]$	$[i]/[m] = 5$

4.2 New Point Tripling Formula in Odd Characteristic

In order to best exploit the ternary nature of the DBNS representation we also propose new point tripling algorithms in Jacobian coordinates, for curves defined over fields of odd characteristic ($\neq 3$).

To simplify, let us first consider affine coordinates. Let $P = (x_1, y_1) \in E(K)$ be a point on an elliptic curve E defined by (2). By definition, we have $2P = (x_2, y_2)$, where

$$\lambda_1 = \frac{3x_1^2 + a}{2y_1}, \quad x_2 = \lambda_1^2 - 2x_1, \quad y_2 = \lambda_1(x_1 - x_2) - y_1 \ . \tag{9}$$

We can compute $3P = 2P + P = (x_3, y_3)$, by evaluating λ_2 (the slope of the chord between the points $2P$ and P) as a function of x_1 and y_1 only. We have

$$\begin{aligned} \lambda_2 &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= -\lambda_1 - \frac{2y_1}{x_2 - x_1} \\ &= -\frac{3x_1^2 + a}{2y_1} - \frac{8y_1^3}{(3x_1^2 + a)^2 - 12x_1y_1^2} \ . \end{aligned} \tag{10}$$

We further remark that

$$\begin{aligned} x_3 &= \lambda_2^2 - x_1 - x_2 \\ &= \lambda_2^2 - x_1 - \lambda_1^2 + 2x_1 \\ &= (\lambda_2 - \lambda_1)(\lambda_2 + \lambda_1) + x_1, \end{aligned} \tag{11}$$

and

$$\begin{aligned} y_3 &= \lambda_2(x_1 - x_3) - y_1 \\ &= -\lambda_2(\lambda_2 - \lambda_1)(\lambda_2 + \lambda_1) - y_1 \ . \end{aligned} \tag{12}$$

Thus $3P = (x_3, y_3)$ can be computed directly from $P = (x_1, y_1)$, without evaluating the intermediate values x_2 and y_2.

By replacing x_1 and y_1 by X_1/Z_1^2 and Y_1/Z_1^3 respectively, we obtain the following point tripling formulæ in Jacobian coordinates. Given $P = (X_1, Y_1, Z_1)$, we compute $3P = (X_3, Y_3, Z_3)$ as

$$\begin{aligned} X_3 &= 8Y_1^2(T - ME) + X_1E^2 \\ Y_3 &= Y_1(4(ME - T)(2T - ME) - E^3) \\ Z_3 &= Z_1E, \end{aligned} \tag{13}$$

where $M = 3X_1^2 + aZ_1^4$, $E = 12X_1Y_1^2 - M^2$ and $T = 8Y_1^4$.

The complexity of this new point tripling algorithm is equal to $6[s] + 10[m]$. If one uses side-channel atomicity to resist simple SCA, then this is equivalent to $16[m]$. We express $\text{TPL}^{\mathcal{J}}$ in terms of atomic blocks Table 11 of Appendix A. In comparison, computing $3P$ using the doubling and addition algorithms from [5], expressed as a repetition of atomic blocks, costs $10[m] + 16[m] = 26[m]$.

As we have seen in Section 3.2, operation count of Algorithm 3 can be reduced by improving the computation of consecutive triplings; i.e., expressions of the form 3^wP. From (13), we remark that the computation of the intermediate value $M = 3X_1^2 + aZ_1^4$ requires $1[m] + 3[s]$ (we omit the multiplication by 3). If we need to compute $9P$, we have to evaluate $M' = 3X_3^2 + aZ_3^4$. Since $Z_3 = Z_1E$, we have $aZ_3^4 = aZ_1^4E^4$ (where $E = 12X_1Y_1^2 - M^2$), and aZ_1^4 and E^2 have already been computed in the previous iteration. Hence, using these precomputed subexpressions, we can compute $M' = 3X_3^2 + (aZ_1^4)(E^2)^2$, with $1[m] + 2[s]$. The same technique can be applied to save one multiplication for each subsequent tripling. Thus, we can compute 3^wP with $(15w + 1)[m]$, which is better than w invocation of the tripling algorithm. The atomic blocks version of w-$\text{TPL}^{\mathcal{J}}$ is given in Table 12 of Appendix A. Note that the idea of reusing aZ^4 for multiple doublings was first proposed by Cohen et al. in [7], where modified Jacobian coordinates are proposed. It is possible that a similar approach for repeated triplings can lead to further improvements.

From Table 2, $\text{DBL}^{\mathcal{J}}$ normally requires $4[m] + 6[s]$, or equivalently 10 blocks of computation if side-channel atomicity is used. However, in our scalar multiplication algorithm, we remark that we very often invoke w'-$\text{DBL}^{\mathcal{J}}$ right after a w-$\text{TPL}^{\mathcal{J}}$ (the only exceptions occur when $u = 0$, which correspond to a series of consecutive $\{2, 3\}$-integers in the expansion of k having the same binary exponents). Using subexpressions computed for the last tripling, we can save $1[s]$ for the first $\text{DBL}^{\mathcal{J}}$. The next $(w' - 1)$-$\text{DBL}^{\mathcal{J}}$ are then computed with $(4w' - 4)[m] + (4w' - 4)[s]$. (The details of these algorithms are given in Appendix A.) We summarize the complexities of these curve operations in Table 4.

5 Comparisons

In this section, we compare our algorithms to the classic double-and-add, NAF and 4-NAF methods, plus some other recently proposed algorithms. More pre-

Table 4. Costs of tripling algorithms in Jacobian coordinates for curves defined over fields of odd characteristic > 3

Curve operation	Complexity	# Registers
$\mathrm{TPL}^{\mathcal{J}}$	$6[s] + 10[m]$	8
w-$\mathrm{TPL}^{\mathcal{J}}$	$(4w+2)[s] + (11w-1)[m]$	10
w-$\mathrm{TPL}^{\mathcal{J}}/w'$-$\mathrm{DBL}^{\mathcal{J}}$	$(11w+4w'-1)[s] + (4w+4w'+3)[m]$	10

cisely, we consider the ternary/binary approach from [6] in even characteristic and two algorithms from Izu et al., published in [17] and [19] for curves defined over fields of odd characteristic. In the later case, we consider the protected version of our algorithm, combined with Joye's and Tymen's randomization technique to counteract differential attacks [20].

If we assume that k is a randomly chosen n-bit integer, it is well known that the double-and-add algorithm requires n doublings and $n/2$ additions on average. Using the NAF representation, the average density of non-zero digits is reduced to $1/3$. More generally, for w-NAF methods, the average number of non-zero digits is roughly equal to $1/(w+1)$. Unfortunately, it seems very difficult to give such an estimate for the particular DBNS representation we are considering in this paper. In [9], it is proved that the greedy algorithm (with unbounded exponents) returns a DBNS expansion which satisfies the asymptotic bound of $O(n/\log n)$ additions, but this is probably not valid with the restriction that the exponents form two decreasing sequences. The rigorous determination of this complexity leads to tremendously difficult problems in transcendental number theory and exponential Diophantine equations and is still an open problem.

Hence, in order to estimate the average number of $\{2,3\}$-integers required to represent k, and to precisely evaluate the complexity of our point multiplication algorithms, we have performed several numerical experiments, over 10000 randomly chosen 160-bit integers (163-bit integers for binary fields). Our results are presented in the next two sections.

5.1 Binary Fields

The average number of curve operations are presented in Table 5 for 163-bit numbers. The corresponding numbers of field operations are given in Table 6 for different ratios $[i]/[m]$, using the best complexities from Tables 1 and 3 in each case.

In Table 6, we remark that our algorithm requires fewer inversions and multiplications than the other methods, and because we are working over binary fields, squarings can be ignored. We can estimate the cost of each method, in terms of the equivalent number of field multiplications, by multiplying the number of inversions by the ratio $[i]/[m]$. By doing so, we obtain a speed-up of 21%, 13.5% and 5.4% over the binary, NAF and ternary/binary approaches respectively for $[i]/[m] = 8$; and 14.1%, 4.8% and 4.4% for $[i]/[m] = 4$.

Table 5. Average number of curve operations using the binary, NAF, ternary/binary and DB-chain approaches for $n = 163$ bits

Algorithm	D	DA	T	TA	Q	QA
binary	82	81	–	–	–	–
NAF	109	54	–	–	–	–
ternary/binary	38	37	55	–	–	–
DB-chain (Algo. 2)	–	17	35	5	25	14

Table 6. Average number of field operations using the binary, NAF, ternary/binary and DB-chain approaches for $n = 163$ bits, and $[i]/[m] = 4, 8$

Algorithm	$[i]/[m] = 4$			$[i]/[m] = 8$		
	$[i]$	$[s]$	$[m]$	$[i]$	$[s]$	$[m]$
binary	244	244	407	163	244	893
NAF	217	217	380	163	217	704
ternary/binary	222	222	353	130	333	795
DB-chain (Algo. 2)	215	240	327	117	405	798

5.2 Prime Fields

In this section, we report results for 160-bit integers. If the classic methods are used in conjunction with side-channel atomicity (which implies $[s] = [m]$), the average cost of the double-and-add method can be estimated to $159 \times 10 + 80 \times 11 + 41 = 2511[m]$; similarly, the NAF and 4-NAF methods require $2214[m]$ and $1983[m]$ respectively. The results of our numerical experiments are presented in Table 7.

Table 7. Average number of terms and the corresponding field complexity of our new scalar multiplication algorithm obtained using 10000 randomly chosen 160-bit integers and different largest binary and ternary exponents

b_{max}	t_{max}	m	Field cost	Complexity (#$[m]$)
57	65	44.52	$1[i] + 742.10[s] + 1226.92[m]$	1999.02
76	53	38.40	$1[i] + 740.59[s] + 1133.58[m]$	1904.17
95	41	36.83	$1[i] + 755.77[s] + 1077.48[m]$	1863.25
103	36	38.55	$1[i] + 772.42[s] + 1074.22[m]$	1876.25

In Table 7, we give the average number m of $\{2, 3\}$-integers used to represent a random 160-bit integer, and the average number of field operations performed by Algorithm 3 for different values of b_{max} and t_{max}. (This cost includes the fixed cost of Joye and Tymen's randomization.) In order to compare our algorithm with the side-channel resistant algorithms presented in [17, 19, 18], we also give the uniform cost in terms of the number of field multiplications. Note that, because we are using side-channel atomicity to prevent simple analysis, squarings cannot be optimized and must be computed using a general multiplier. We thus assume $[s] = [m]$ and $[i] = 30[m]$.

In Table 8, we summarize the complexities of these recognized methods. The figures for the algorithms from Izu, Möller and Takagi are taken from [17] and [19] assuming Coron's randomization technique which turns out to be more efficient in their case. The cost of our algorithm is taken from the third row of Table 7, with $b_{max} = 95$ and $t_{max} = 41$, which corresponds to the best non-trivial approximation to 2^{160} and leads to the best complexity.

Table 8. Comparison of different scalar multiplication algorithms protected against simple and differential analysis

Algorithm	Complexity (#$[m]$)
double-and-add	2511
NAF	2214
4-NAF	1983
Izu, Möller, Takagi 2002 [17]	2449
Izu, Takagi 2005 [19]	2629
Double-base chain (Algo. 3)	1863

We remark that our new algorithm outperforms all the previous recognized methods. It represents a gain of 25.8% over the double-and-add, 15.8% over the NAF, 6% over 4-NAF, 23.9% over [17] and 29.1% over [19].

6 Conclusions

In this paper, we have presented fast and secure scalar multiplication algorithms which take advantage of the sparseness and the ternary nature of the double-base number system. When Jacobian coordinates are used for curves defined over fields of odd characteristic (greater than 3), new formulæ for $\text{TPL}^{\mathcal{J}}$ and w-$\text{TPL}^{\mathcal{J}}$ have been proposed and expressed in atomic blocks to prevent simple analysis. Differential attacks are prevented using Joye and Tymen randomization method, but any countermeasure (allowing for mixed addition) can be integrated to our point multiplication algorithm. When working over binary fields, improved algorithms for point quadrupling and combined quadruple-and-add have been presented. Although many theoretical questions remain open about the double-base number system, e.g. the exact determination of the average number of $\{2, 3\}$-integer, or the number of DBNS representation with decreasing exponents of a given integer, we have produced a modified greedy algorithm to convert the multiplier k into the particular DBNS form required by our point multiplication algorithm. However, we want to make clear the point that in most cases, this conversion is not necessary. When k is randomly chosen, it suffices to generate directly a random, convenient DBNS number (with decreasing exponents); and when k is part of a secret key, the conversion process can be performed offline and even further optimized. We believe that the proposed point multiplication algorithms are very competitive contenders for fast and secure ECC implementations.

Acknowledgments

The authors would like to thank Tanja Lange for her invaluable comments and suggestions during the final preparation of this paper.

References

[1] R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* CRC Press, 2005.

[2] V. Berthé and L. Imbert. On converting numbers to the double-base number system. In F. T. Luk, editor, *Advanced Signal Processing Algorithms, Architecture and Implementations XIV*, volume 5559 of *Proceedings of SPIE*, pages 70–78. SPIE, 2004.

[3] I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography.* Number 256 in London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

[4] I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in Elliptic Curve Cryptography.* Number 317 in London Mathematical Society Lecture Note Series. Cambridge University Press, 2005.

[5] 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.

[6] 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.

[7] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In K. Ohta and D. Pei, editors, *Advances in Cryptology – ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 51–65. Springer-Verlag, 1998.

[8] E. De Win, S. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle. A fast software implementation for arithmetic operations in GF(2^n). In K. Kim and T. Matsumoto, editors, *Advances in Cryptology – ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 65–76. Springer-Verlag, 1996.

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

[10] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. Theory and applications of the double-base number system. *IEEE Transactions on Computers*, 48(10):1098–1106, Oct. 1999.

[11] 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 *Lecture Notes in Computer Science*, pages 343–354. Springer-Verlag, 2003.

[12] 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.

[13] J. Guajardo and C. Paar. Efficient algorithms for elliptic curve cryptosystems. In *Advances in Cryptology – CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 342–356. Springer-Verlag, 1997.

[14] 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 *Lecture Notes in Computer Science*, pages 1–24. Springer-Verlag, 2000.

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

[16] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara. Fast implementation of public-key cryptography on a DSP TMS320C6201. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES '99*, volume 1717 of *Lecture Notes in Computer Science*, pages 61 – 72. Springer-Verlag, 1999.

[17] T. Izu, B. Möller, and T. Takagi. Improved elliptic curve multiplication methods resistant against side channel attacks. In A. Menezes and P. Sarkar, editors, *Progress in Cryptology – INDOCRYPT 2002*, volume 2551 of *Lecture Notes in Computer Science*, pages 269–313. Springer-Verlag, 2002.

[18] T. Izu and T. Takagi. A fast parallel elliptic curve multiplication resistant against side channel attacks. In D. Naccache and P. Paillier, editors, *Public Key Cryptography*, volume 2274 of *Lecture Notes in Computer Science*, pages 280–296. Springer-Verlag, 2002.

[19] T. Izu and T. Takagi. Fast elliptic curve multiplications resistant against side channel attacks. *IEICE Transactions Fundamentals*, E88-A(1):161–171, Jan. 2005.

[20] M. Joye and C. Tymen. Protections against differential analysis for elliptic curve cryptography – an algebraic approach. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2001*, volume 2162 of *Lecture Notes in Computer Science*, pages 377 – 390. Springer-Verlag, 2001.

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

[22] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology – CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer-Verlag, Aug. 1999.

[23] P. C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In N. Koblitz, editor, *Advances in Cryptology - CRYPTO '96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer-Verlag, Aug. 1996.

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

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

A w-DBL$^{\mathcal{J}}$ and w-TPL$^{\mathcal{J}}$ Algorithms in Atomic Blocks

In this appendix, we give the algorithms for DBL$^{\mathcal{J}}$ (including the case when a doubling is performed right after a tripling), w-DBL$^{\mathcal{J}}$, TPL$^{\mathcal{J}}$ and w-TPL$^{\mathcal{J}}$, expressed in atomic blocks.

Table 9. The DBL$^{\mathcal{J}}$ algorithm in atomic blocks. When DBL$^{\mathcal{J}}$ is called right after w-TPL$^{\mathcal{J}}$, the blocks Δ_2, Δ_3 and Δ_4 can be replaced by the blocks Δ'_2 and Δ'_3 to save one multiplication

DBL$^{\mathcal{J}}$
Input: $P = (X_1, Y_1, Z_1)$
Output: $2P = (X_3, Y_3, Z_3)$
Init: $R_1 = X_1,\ R_2 = Y_1,\ R_3 = Z_1$

Δ_1	$R_4 = R_1 \times R_1$	(X_1^2)	Δ_6	$R_2 = R_2 \times R_2$	(Y_1^2)
	$R_5 = R_4 + R_4$	$(2X_1^2)$		$R_2 = R_2 + R_2$	$(2Y_1^2)$
	*			*	
	$R_4 = R_4 + R_5$	$(3X_1^2)$		*	
Δ_2	$R_5 = R_3 \times R_3$	(Z_1^2)	Δ_7	$R_5 = R_1 \times R_2$	(S)
	$R_1 = R_1 + R_1$	$(2X_1)$		*	
	*			$R_5 = -R_5$	$(-S)$
	*			*	
Δ_3	$R_5 = R_5 \times R_5$	(Z_1^4)	Δ_8	$R_1 = R_4 \times R_4$	(M^2)
	*			$R_1 = R_1 + R_5$	$(M^2 - S)$
	*			*	
	*			$R_1 = R_1 + R_5$	(X_3)
Δ_4	$R_6 = a \times R_5$	(aZ_1^4)	Δ_9	$R_2 = R_2 \times R_2$	$(4Y_1^4)$
	$R_4 = R_4 + R_6$	(M)		$R_7 = R_2 + R_2$	(T)
	*			*	
	$R_5 = R_2 + R_2$	$(2Y_1)$		$R_5 = R_1 + R_5$	$(X_3 - S)$
Δ_5	$R_3 = R_3 \times R_5$	(Z_3)	Δ_{10}	$R_4 = R_4 \times R_5$	$(M(X_3 - S))$
	*			$R_2 = R_4 + R_7$	$(-Y_3)$
	*			$R_2 = -R_2$	(Y_3)
	*			*	

Δ'_2	$R_5 = R_{10} \times R_{10}$	Δ'_3	$R_5 = R_5 \times R_9$
	$R_1 = R_1 + R_1$		$R_4 = R_4 + R_6$
	*		*
	*		*

Table 10. The w-DBL$^{\mathcal{J}}$ algorithm in atomic blocks. The 10 blocks (or 9 if executed after w-TPL$^{\mathcal{J}}$) of DBL$^{\mathcal{J}}$ (Table 9) must be executed once, followed by the blocks Δ_{11} to Δ_{18} which have to be executed $w-1$ times. After the execution of DBL$^{\mathcal{J}}$, the point of coordinates (X_t, Y_t, Z_t) correspond to the point $2P$; at the end of the $w-1$ iterations, $2^w P = (X_3, Y_3, Z_3) = (X_t, Y_t, Z_t)$

w-DBL$^{\mathcal{J}}$
Input: $P = (X_1, Y_1, Z_1)$
Output: $2^w P = (X_3, Y_3, Z_3)$
Init: (X_t, Y_t, Z_t) is the result of DBL$^{\mathcal{J}}(P)$, $R_6 = aZ_1^4$, $R_7 = 8Y_1^4$

Δ_{11}	$R_4 = R_1 \times R_1$	(X_t^2)	Δ_{15}	$R_5 = R_1 \times R_2$	(S)
	$R_5 = R_4 + R_4$	$(2X_t^2)$		$*$	
	$*$			$R_5 = -R_5$	$(-S)$
	$R_4 = R_4 + R_5$	$(3X_t^2)$		$*$	
Δ_{12}	$R_5 = R_6 \times R_7$	$(aZ_t^4 + 8Y_t^4)$	Δ_{16}	$R_1 = R_4 \times R_4$	(M^2)
	$R_6 = R_5 + R_5$	(aZ_t^4)		$R_1 = R_1 + R_5$	$(M^2 - S)$
	$*$			$*$	
	$R_4 = R_4 + R_6$	(M)		$R_1 = R_1 + R_5$	(X_{t+1})
Δ_{13}	$R_3 = R_2 \times R_3$	$(Y_t Z_t)$	Δ_{17}	$R_2 = R_2 \times R_2$	$(4Y_t^4)$
	$R_3 = R_3 + R_3$	(Z_{t+1})		$R_7 = R_2 + R_2$	(T)
	$*$			$*$	
	$R_1 = R_1 + R_1$	$(2X_t)$		$R_5 = R_1 + R_5$	$(X_{t+1} - S)$
Δ_{14}	$R_2 = R_2 \times R_2$	(Y_t^2)	Δ_{18}	$R_4 = R_4 \times R_5$	$(M(X_{t+1} - S))$
	$R_2 = R_2 + R_2$	$(2Y_t^2)$		$R_2 = R_4 + R_7$	$(-Y_{t+1})$
	$*$			$R_2 = -R_2$	(Y_{t+1})
	$*$			$*$	

Table 11. The TPL$^{\mathcal{J}}$ algorithm in atomic blocks

TPL$^{\mathcal{J}}$
Input: $P = (X_1, Y_1, Z_1)$
Output: $3P = (X_3, Y_3, Z_3)$
Init: $R_1 = X_1,\ R_2 = Y_1,\ R_3 = Z_1$

Γ_1	$R_4 = R_3 \times R_3$	(Z_1^2)	Γ_9	$R_8 = R_6 \times R_7$	(T)
	*			$R_7 = R_7 + R_7$	$(8Y_1^2)$
	*			*	
	*			*	
Γ_2	$R_4 = R_4 \times R_4$	(Z_1^4)	Γ_{10}	$R_6 = R_4 \times R_5$	(ME)
	*			*	
	*			$R_6 = -R_6$	$(-ME)$
	*			$R_6 = R_8 + R_6$	$(T - ME)$
Γ_3	$R_5 = R_1 \times R_1$	(X_1^2)	Γ_{11}	$R_{10} = R_5 \times R_5$	(E^2)
	$R_6 = R_5 + R_5$	$(2X_1^2)$		*	
	*			*	
	$R_5 = R_5 + R_6$	$(3X_1^2)$		*	
Γ_4	$R_9 = a \times R_4$	(aZ_1^4)	Γ_{12}	$R_1 = R_1 \times R_{10}$	(X_1E^2)
	$R_4 = R_5 + R_9$	(M)		*	
	*			*	
	*			*	
Γ_5	$R_5 = R_2 \times R_2$	(Y_1^2)	Γ_{13}	$R_5 = R_{10} \times R_5$	(E^3)
	$R_6 = R_5 + R_5$	$(2Y_1^2)$		$R_8 = R_8 + R_6$	$(2T - ME)$
	*			$R_5 = -R_5$	$(-E^3)$
	$R_7 = R_6 + R_6$	$(4Y_1^2)$		*	
Γ_6	$R_5 = R_1 \times R_7$	$(4X_1Y_1^2)$	Γ_{14}	$R_4 = R_6 \times R_7$	$8Y_1^2(T - ME)$
	$R_8 = R_5 + R_5$	$(8X_1Y_1^2)$		$R_6 = R_6 + R_6$	$(2(T - ME))$
	*			$R_6 = -R_6$	$(2(ME - T))$
	$R_5 = R_5 + R_8$	$(12X_1Y_1^2)$		$R_1 = R_1 + R_4$	(X_3)
Γ_7	$R_8 = R_4 \times R_4$	(M^2)	Γ_{15}	$R_6 = R_6 \times R_8$	$(2(ME - T)(2T - ME))$
	*			$R_6 = R_6 + R_6$	$(4(ME - T)(2T - ME))$
	$R_8 = -R_8$	$(-M^2)$		*	
	$R_5 = R_5 + R_8$	(E)		$R_6 = R_6 + R_5$	$(4(ME - T)(2T - ME) - E^3)$
Γ_8	$R_3 = R_3 \times R_5$	(Z_3)	Γ_{16}	$R_2 = R_2 \times R_6$	(Y_3)
	*			*	
	*			*	
	*			*	

Table 12. The w-TPL$^{\mathcal{J}}$ algorithm in atomic blocks. The 16 blocks of TPL$^{\mathcal{J}}$ must be executed once, followed by the blocks Γ_{17} to Γ_{31} which have to be executed $w-1$ times. After the execution of TPL$^{\mathcal{J}}$, the point of coordinates (X_t, Y_t, Z_t) correspond to the point $3P$; at the end of the $w-1$ iterations, $3^w P = (X_3, Y_3, Z_3) = (X_t, Y_t, Z_t)$

w-**TPL**$^{\mathcal{J}}$
Input: $P = (X_1, Y_1, Z_1)$
Output: $3^w P = (X_3, Y_3, Z_3)$
Init: (X_t, Y_t, Z_t) is the result of TPL$^{\mathcal{J}}(P)$, $R_9 = aZ_1^4$, $R_{10} = E^2$

Γ_{17}	$R_4 = R_9 \times R_{10}$	$(aZ_t^4E^2)$	Γ_{25}	$R_6 = R_4 \times R_5$	(ME)
	*			*	
	*			$R_6 = -R_6$	$(-ME)$
	*			$R_6 = R_8 + R_6$	$(T - ME)$
Γ_{18}	$R_5 = R_1 \times R_1$	(X_t^2)	Γ_{26}	$R_{10} = R_5 \times R_5$	(E^2)
	$R_6 = R_5 + R_5$	$(2X_t^2)$		*	
	*			*	
	$R_5 = R_5 + R_6$	$(3X_t^2)$		*	
Γ_{19}	$R_9 = R_4 \times R_{10}$	(aZ_t^4)	Γ_{27}	$R_1 = R_1 \times R_{10}$	(X_tE^2)
	$R_4 = R_5 + R_9$	(M)		*	
	*			*	
	*			*	
Γ_{20}	$R_5 = R_2 \times R_2$	(Y_t^2)	Γ_{28}	$R_5 = R_{10} \times R_5$	(E^3)
	$R_6 = R_5 + R_5$	$(2Y_t^2)$		$R_8 = R_8 + R_6$	$(2T - ME)$
	*			$R_5 = -R_5$	$(-E^3)$
	$R_7 = R_6 + R_6$	$(4Y_t^2)$		*	
Γ_{21}	$R_5 = R_1 \times R_7$	$(4X_tY_t^2)$	Γ_{29}	$R_4 = R_6 \times R_7$	$(8Y_t^2(T - ME))$
	$R_8 = R_5 + R_5$	$(8X_tY_t^2)$		$R_6 = R_6 + R_6$	$(2(T - ME))$
	*			$R_6 = -R_6$	$(2(ME - T))$
	$R_5 = R_5 + R_8$	$(12X_tY_t^2)$		$R_1 = R_1 + R_4$	(X_{t+1})
Γ_{22}	$R_8 = R_4 \times R_4$	(M^2)	Γ_{30}	$R_6 = R_6 \times R_8$	$(2(ME - T)(2T - ME))$
	*			$R_6 = R_6 + R_6$	$(4(ME - T)(2T - ME))$
	$R_8 = -R_8$	$(-M^2)$		*	
	$R_5 = R_5 + R_8$	(E)		$R_6 = R_6 + R_5$	$(4(ME - T)(2T - ME) - E^3)$
Γ_{23}	$R_3 = R_3 \times R_5$	(Z_{t+1})	Γ_{31}	$R_2 = R_2 \times R_6$	(Y_{t+1})
	*			*	
	*			*	
	*			*	
Γ_{24}	$R_8 = R_6 \times R_7$	(T)			
	$R_7 = R_7 + R_7$	$(8Y_t^2)$			
	*				
	*				