The Double-Base Number System and its Application to Elliptic Curve Cryptography

Vassil Dimitrov, Laurent Imbert, Pradeep Mishra

MATHEMATICS OF COMPUTATION
S 0025-5718(07)02048-0
Article electronically published on December 11, 2007

THE DOUBLE-BASE NUMBER SYSTEM AND ITS APPLICATION TO ELLIPTIC CURVE CRYPTOGRAPHY

VASSIL DIMITROV, LAURENT IMBERT, AND PRADEEP K. MISHRA

Abstract. We describe an algorithm for point multiplication on generic elliptic curves, based on a representation of the scalar as a sum of mixed powers of 2 and 3. The sparseness of this so-called double-base number system, combined with some efficient point tripling formulae, lead to efficient point multiplication algorithms for curves defined over both prime and binary fields. Side-channel resistance is provided thanks to side-channel atomicity.

1. Introduction

Since its discovery by Miller [38] and Koblitz [33] in 1985, Elliptic Curve Cryptography (ECC) has been the subject of a vast amount of publications. Of particularly interest is the quest for fast and side-channel resistant implementations. ECC bases its theoretical robustness on the Elliptic Curve Discrete Logarithm Problem (ECDLP), for which no subexponential algorithm is known. The main operation of any ECC protocol is to compute the point $[n]P = P + \cdots + P$ (n times), for $n \in \mathbb{Z}$ and P a point on the curve. This operation, called the point or scalar multiplication, is the most time consuming and must be carefully implemented. Adaptations of fast exponentiation algorithms [23] have been proposed. The double-and-add algorithm, an adaptation of the square-and-multiply exponentiation method, can be used to compute $[n]P$ in $\log n$ point doublings and $(\log n)/2$ point additions on average. Since the opposite of a point $(-P)$ is easily computed, signed digit representations allow one to reduce the number of point additions: the Non Adjacent Form (NAF), also known as the modified Booth recoding, requires $(\log n)/3$ point additions on average. Window methods (w-NAF) can be used to further reduce the number of additions to $(\log n)/(w+1)$, at the extra cost of a small amount of precomputations (one needs to precompute the points jP for $j = 1, 3, \ldots, 2^{w-1} - 1$; the points $\pm jP$ are used in the point multiplication algorithm). Methods based on efficiently computable endomorphisms on special curves, such as Koblitz curves, are also very attractive. Since the original submission of this manuscript, several interesting papers have been published, which merge the properties of the double-base number system and the efficiently computable endomorphisms on these curves [17, 3].

Received by the editor June 5, 2006 and, in revised form, February 26, 2007.

2000 *Mathematics Subject Classification.* Primary 14H52; Secondary 14G50, 68R99.

Key words and phrases. ECC, point multiplication, double-base number system, side-channel atomicity.

This work was funded by the NSERC Strategic Grant: Novel Implementation of Cryptographic Algorithms on Custom Hardware Platforms.

This work was done during the second author's leave of absence from the CNRS at the University of Calgary.

In this paper, we propose a scalar multiplication algorithm based on a representation of the scalar n as a sum of mixed powers of two coprime integers $p.q$, 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, let $p = 2, q = 3$ and n be 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. Although this sparseness does not immediately lead to algorithmic improvements, it outlines one of the main features of this number system and serves as a good starting point for potential applications in cryptography. Double-base representations have recently attracted curiosity in the cryptographic community: Avanzi, Ciet and Sica have investigated double-bases in the case of Koblitz curves, by letting one of the bases be an algebraic number [13, 4]. In [19], Doche et al. proposed a very efficient tripling algorithm[1] for a particular family of curves by using isogeny decompositions; in this context, finding short double-base expansions is also of primary importance. Very recently, Doche and Imbert proposed an extension of the idea which lead to significant speedups in double-base point multiplications for generic curves [20]. This is achieved by considering double-base expansions with digit sets larger than $\{-1, 0, 1\}$.

This paper is an extension of the author's paper at Asiacrypt 2005 [16]. The present version contains a more detailed presentation of the double-base number system, including a theorem on the number of double-base representations for a given positive integer, and some numerical results that illustrate the properties of this encoding scheme, particularly its redundancy and sparseness. It also gives more details on the most important step of the greedy approach used for the conversion from binary, i.e., finding the best approximation of a given integer of the form $p^a q^b$. An efficient alternative solution, which requires some precomputed values to be stored in lookup tables, is presented in [20].

In order to best exploit the sparse and ternary nature of this representation scheme, we also propose new formulae for some useful point operations (tripling, quadrupling, etc.) for generic elliptic curves. We consider curves defined over $\mathbb{F}_p$ with Jacobian coordinates, and curves over $\mathbb{F}_{2^m}$ with both affine and Jacobian coordinates. Some of these formulae are already present in [16]. The derivations are given with more details in the present paper.

Since their discovery by Kocher [35, 34], side-channel attacks (SCA) have become the most serious threat for cryptographic devices. Therefore, protection against various kinds of SCA (power analysis, electromagnetic attacks, fault attacks, etc.) has become a major issue and an interesting area of research. Several countermeasures have been proposed in the literature. We refer interested readers to [8, 2] for details. In this work we consider a solution proposed by Chavalier-Mames et al. called side-channel atomicity [10]. The field operations used in the ADD and DBL curve operations are rearranged and divided into small identical groups, called atomic blocks. These blocks all contain the same operations, in the very same order, to become indistinguishable from the side-channel information leaked to the adversary. Therefore, the trace of a computation composed of a series of ADD and DBL looks like a series of atomic blocks; the adversary cannot distinguish which block belongs to which operation from the side-channel information. Thus the sequence

[1]Note that it is possible to trade one multiplication for a squaring in their formula.

of execution of the curve operations is blinded. This effectively resists simple power attacks.

The sequel of the paper is organized as follows: In Section 2, we introduce the double-base number system, its main properties, and some related problems in number theory and combinatorics. We briefly recall the basics of elliptic curve cryptography and the costs of the classical curve operations in Section 2.2. In Section 3, we present several new curve formulae for the operations that arise in the DBNS point multiplication algorithm presented in Section 4. Finally, we compare our algorithm with several other methods in Section 5.

2. Background

2.1. **The double-base number system.** In this section, we present the main properties of the double-base number system, along with some numerical results in order to provide the reader with some intuitive ideas about this representation scheme and the difficulty of some underlying open problems. We have intentionally omitted the proofs of previously published results. The reader is encouraged to check the references for more details.

We will need the following definitions.

Definition 1 (***S***-integer)**.** Given a set of primes S, an S-integer is a positive integer whose prime factors all belong to S.

Definition 2 (double-base number system)**.** Given p, q, two relatively prime positive integers, the double-base number system (DBNS) is a representation scheme into which every positive integer n is represented as the sum or difference of $\{p, q\}$-integers, i.e., numbers of the form $p^a q^b$:

$$n = \sum_{i=1}^{l} s_i \, p^{a_i} q^{b_i}, \quad \text{with } s_i \in \{-1, 1\} \text{ and } a_i, b_i \geq 0. \tag{1}$$

The size, or length, of a DBNS expansion is equal to the number of terms l in (1). In the following, we will only consider expansions of n as sums of $\{2, 3\}$-integers; i.e., DBNS with $p = 2, q = 3$.

Whether one considers signed ($s_i = \pm 1$) or unsigned ($s_i = 1$) expansions, this representation scheme is highly redundant. For instance, if we assume unsigned double-base representations only, we can prove that 10 has exactly 5 different DBNS representations, 100 has exactly 402 different DBNS representations, $1,000$ has exactly $1,295,579$ different DBNS representations, etc. The following theorem holds.

Theorem 1. *Let n be a positive integer. The number of unsigned DBNS representations of n is given by the following recursing function. $f(1) = 1$ and for $n \geq 1$,*

$$f(n) = \begin{cases} f(n-1) + f(n/3) & \text{if} \quad n \equiv 0 \pmod 3, \\ f(n-1) & \text{otherwise.} \end{cases} \tag{2}$$

Proof. Let us consider the diophantine equation

$$n = h_0 + 3h_1 + 9h_2 + \cdots + 3^k h_k, \tag{3}$$

where $k = \lfloor \log_3(n) \rfloor$ and $h_i \geq 0$ for $i = 0, \ldots, k$. Let $h^{(m)} = (h_0^{(m)}, \ldots, h_k^{(m)})$ be the m-th solution of (3). By substituting each $h_i^{(m)}$ into (3) with its (unique) binary representation, we obtain a specific partition of n as the sum of numbers of the form

$2^a 3^b$. Our problem thus reduces to counting the number of solutions $g(n)$ of (3). This is a very classical integer partition problem, which is known to be associated with the following generating function (see [45] for example):

$$G(z) = \frac{1}{(1-z)(1-z^3)\dots(1-z^{3^k})}. \tag{4}$$

We will prove that (2) admits the same generating function; i.e., that $f(n) = g(n) = [z^n]G(z)$, where the symbol $[z^n]G(z)$ denotes the coefficient of degree n in the series $G(z)$.

Let $F(z) = \sum_{n=1}^{\infty} z^n f(n)$ be the generating function associated with (2). We find that

$$\begin{aligned} F(z) &= \sum_{n=1}^{\infty} z^{3n} f(3n) + \sum_{\substack{n=1 \\ 3\nmid n}}^{\infty} z^n f(n) \\ &= \sum_{n=1}^{\infty} z^{3n} \left(f(3n-1) + f(n)\right) + \sum_{\substack{n=2 \\ 3\nmid n}}^{\infty} z^n f(n-1) + z f(1) \\ &= z + \sum_{n=2}^{\infty} z^n f(n-1) + \sum_{n=1}^{\infty} z^{3n} f(n) \\ &= z + zF(z) + F(z^3). \end{aligned}$$

Thus

$$F(z) = \frac{z}{1-z} + \frac{z^3}{(1-z)(1-z^3)} + \frac{z^9}{(1-z)(1-z^3)(1-z^9)} + \cdots. \tag{5}$$

By noticing that, for $n \geq 1$, the coefficient of z^n in the series $z/(1-z)$ is equal to the coefficient of z^n in the series $1/(1-z)$ and by expressing all terms in (5) with denominator $\prod_i (1-z^{3^i})$, we obtain that

$$[z^n]\, F(z) = [z^n]\, \frac{1}{(1-z)(1-z^3)\dots(1-z^{3^k})} = [z^n]\, G(z), \tag{6}$$

which concludes the proof. □

It is quite clear that the above theorem also applies to numbers of the form $2^a s^b$, where s is an odd integer greater than 1. In this case, the number of solutions of the corresponding partition problem is given by a function similar to (2), where 3 is replaced by s. $\hat{f}(1) = 1$ and for $n \geq 1$,

$$\hat{f}(n) = \begin{cases} \hat{f}(n-1) + \hat{f}(n/s) & \text{if} \quad n \equiv 0 \pmod{s}, \\ \hat{f}(n-1) & \text{otherwise.} \end{cases}$$

Apparently, Mahler was the first to consider the problem of finding good approximations of $\hat{f}(n)$ in his work from 1940 on the Mordel's functional equation [37]. He proved that $\log \hat{f}(n) \approx \dfrac{(\log n)^2}{2 \log s}$. In 1953, Pennington [40] obtained a very good approximation of $\log \hat{f}(sn)$, which gives us an extremely accurate estimation of the number of partitions of n as the sum of $\{2, s\}$-integers:

$$\hat{f}(sn) = e^{O(1)} \left(\frac{n^{C_1 \log n} n^{C_2} \log n^{C_1 \log \log n}}{n^{2C_1 \log \log n} \log n^{C_3 \log n}} \right),$$

where C_1, C_2, C_3 are explicitconstants depending only on s

Theorem 1 tells us that there exist very many ways to represent a given integer in DBNS. Some of these representations are of special interest, most notably the ones that require the minimal number of $\{2,3\}$-integers; that is, an integer can be represented as the sum of l terms, but cannot be represented with $(l-1)$ or fewer. These so-called *canonic* representations are extremely sparse. For example, 127 has 783 different unsigned representations, among which 6 are canonic requiring only three $\{2,3\}$-integers. An easy way to visualize DBNS numbers is to use a two-dimensional array (the columns represent the powers of 2 and the rows represent the powers of 3) into which each non-zero cell contains the sign of the corresponding term. For example, the six canonic representations of 127 are given in Table 1.

TABLE 1. The six canonic unsigned DBNS representations of 127

$2^2 3^3 + 2^1 3^2 + 2^0 3^0 = 108 + 18 + 1$

	1	2	4
1	1		
3			
9		1	
27			1

$2^2 3^3 + 2^4 3^0 + 2^0 3^1 = 108 + 16 + 3$

	1	2	4	8	16
1					1
3	1				
9					
27			1		

$2^5 3^1 + 2^0 3^3 + 2^2 3^0 = 96 + 27 + 4$

	1	2	4	8	16	32
1			1			
3						1
9						
27	1					

$2^3 3^2 + 2^1 3^3 + 2^0 3^0 = 72 + 54 + 1$

	1	2	4	8
1	1			
3				
9				1
27		1		

$2^6 3^0 + 2^1 3^3 + 2^0 3^2 = 64 + 54 + 9$

	1	2	4	8	16	32	64
1							1
3							
9	1						
27		1					

$2^6 3^0 + 2^2 3^2 + 2^0 3^3 = 64 + 36 + 27$

	1	2	4	8	16	32	64
1							1
3							
9			1				
27	1						

Some numerical facts provide a good impression about the sparseness of the DBNS. The smallest integer requiring three $\{2,3\}$-integers in its unsigned canonic DBNS representation is 23; the smallest integer requiring four $\{2,3\}$-integers in its unsigned canonic DBNS representation is 431. Similarly, the next smallest integers requiring five, six and seven $\{2,3\}$-integers are $18,431$, $3,448,733$, and $1,441,896,119$, respectively. The next record-setter would be most probably bigger than one trillion.

If one considers signed representations, 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 most probably 4985, but to prove it rigorously, one has to show that none of the following exponential diophantine equations has a solution.

Conjecture 1. *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.

One way to tackle this problem would be to extend the results from Skinner [41] on the diophantine equation $ap^x + bq^y = c + dp^z q^w$, to the case where a, b, c, d are not necessarily positive integers. Deriving similar results for a four-term equation (that is, proving that a given number does not admit a signed DBNS representation with 4 terms) seems, however, to be a much more difficult problem.

Finding one of the canonic DBNS representations in a reasonable amount of time, especially for large integers, seems to be a very difficult task. Fortunately, one can use a greedy approach to find a fairly sparse representation very quickly. Given $n > 0$, Algorithm 1 below returns a signed DBNS representation for n. Although it sometimes fails in finding a canonic representation,[2] it is very easy to implement and, more importantly, it guarantees an expansion of sublinear length. Indeed, one of the most important theoretical results about the double-base number system is the following theorem from [18]. It gives us an estimate for the number of terms that one can expect to represent a positive integer.

Algorithm 1 Greedy algorithm

Input A positive integer n
Output The sequence of triples $(s_i, a_i, b_i)_{i\geq 0}$ such that $n = \sum_i s_i 2^{a_i} 3^{b_i}$ with $s_i \in \{-1, 1\}$ and $a_i, b_i \geq 0$

```
s ← 1
while n ≠ 0 do
    Find the best approximation of n of the form z = 2^a 3^b
    print (s, a, b)
    if n < z then
        s ← −s
    n ← |n − z|
```

Theorem 2. *Algorithm* 1 *terminates after* $k \in O(\log n / \log\log n)$ *steps.*

Sketch of proof. (See [18] for a complete proof). Clearly, we have $k \in O(\log n)$ by taking the 2-adic or 3-adic expansions of n. A result by Tijdeman [44] states that there exists an absolute constant C such that there is always a number of the form $2^a 3^b$ between $n - n/(\log n)^C$ and n. Let $n = n_0 > n_1 > n_2 > \cdots > n_l > n_{l+1}$ be the sequence of integers obtained via Algorithm 1. Clearly, for all $i = 0, \ldots, l$, it satisfies $n_i = 2^{a_i} 3^{b_i} + n_{i+1}$ with $n_{i+1} < n_i/(\log n_i)^C$ By defining

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

$l = l(n)$ such that $n_l > f(n) \geq n_{l+1}$, we obtain that $k = l(n) + O(\log f(n))$. The proof is completed by showing that the function $f(n) = \exp(\log n/\log\log n)$ gives $l(n) \in O(\log n/\log\log n)$. □

The complexity of the greedy algorithm mainly depends on the complexity of step 3: finding the $\{2,3\}$-integer which best approximates n. The problem can be reformulated in terms of linear forms of logarithms. For the best default approximation, one has to find two integers $a, b \geq 0$ such that

$$a\log 2 + b\log 3 \leq \log n, \tag{7}$$

and such that no other integers $a', b' \geq 0$ give a better left approximation to $\log n$.

In [7], Berthè and Imbert proposed an algorithm based on Ostrowski's number system [1] for real numbers [6]; a number system associated with the series $(q_i\alpha - p_i)_{i\geq 0}$, where $(p_i/q_i)_{i\geq 0}$ is the series of the convergents of the continued fraction expansion of an irrational number $\alpha \in]0,1[$. In this system, every real number $-\alpha \leq \beta < 1-\alpha$ can be uniquely written as

$$\beta = \sum_{i=1}^{+\infty} b_i(q_{i-1}\alpha - p_{i-1}), \tag{8}$$

where

$$\begin{cases} 0 \leq b_1 \leq a_1 - 1, \text{ and } 0 \leq b_i \leq a_i \text{ for } i > 1, \\ b_i = 0 \text{ if } b_{i+1} = a_{i+1}, \\ b_i \neq a_i \text{ for infinitely many even and odd integers.} \end{cases}$$

The algorithm presented in [7] uses the fact that β can be approximated modulo 1 by numbers of the form $A\alpha$; the best successive approximations being given by the series

$$A_j = \sum_{i=1}^{j} b_i q_{i-1}. \tag{9}$$

By setting $\alpha = \log 2/\log 3$, and $\beta = \{\log n/\log 3\}$ (where $\{\}$ denotes the fractional part), the solution of our problem is given by $a = \sum_{i=1}^{m} b_i q_{i-1}$ and $b = \lfloor\beta\rfloor - \sum_{i=1}^{m} b_i p_{i-1}$, where m is the largest integer such that $b \geq 0$.

One can prove that the algorithm proposed in [7] to find the best approximation of n of the form $2^a 3^b$ has complexity $O(\log\log n)$. Since the greedy algorithm finishes in $O(\log n/\log\log n)$ iterations, its complexity is thus in $O(\log n)$. It is important to remark that for a recoding algorithm between two additive number systems, we cannot do better.

2.2. **Elliptic curve cryptography.**

Definition 3. An elliptic curve E over a field K, denoted by E/K, is defined by its Weierstraß equation

$$E/K : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \tag{10}$$

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

In practice, the general equation (10) can be greatly simplified by applying admissible changes of variables. If the characteristic of K is not equal to 2 and 3, one can rewrite it as

$$y^2 = x^3 + a_4x + a_6, \tag{11}$$

where $a_4, a_6 \in K$, and $\Delta = 4a_4^3 + 27a_6^2 \neq 0$.

When the characteristic of K is equal to 2, the *ordinary* or *non-supersingular* form[3] of an elliptic curve is given by

$$y^2 + xy = x^3 + a_2x^2 + a_6, \tag{12}$$

where $a_2, a_6 \in K$ and $\Delta = a_6 \neq 0$.

The set $E(K)$ of K-rational points on an elliptic curve E/K consists of the affine points (x, y) satisfying (10) along with the special point $\mathcal{O}$ called the *point at infinity*. It forms an abelian group, where the operation (denoted additively) is defined by the well-known law of chord and tangent (see [2] for details). Given P, Q on the curve, the group law slightly differs as to whether one considers the computation of $P + Q$ with $P \neq \pm Q$ or the computation of $P + P = [2]P$. We talk about point addition (ADD) and point doubling (DBL).

There exist many ways to represent the points of $E(K)$. In affine coordinates ($\mathcal{A}$), both the ADD and DBL operations involve expensive field inversions (to compute the slope of the chord/tangent). In order to avoid these inversions, several inversion-free systems of coordinates have been proposed. The choice of such a system has to be made according to several parameters including memory constraints and the relative cost between one field inversion and one field multiplication, often called the $[i]/[m]$ ratio. For binary fields, several works report a ratio $[i]/[m]$ between 3 and 10 depending on the implementation options (see [15, 26]). For prime fields, however, this ratio is more difficult to estimate precisely. In [22], Fong et al. consider that $[i]/[m] > 40$ on general-purpose processors. In fact, different experiments can provide very different results. If, for example, one uses GMP [24] to compare the cost of these two operations over large prime fields, one does not necessarily notice a huge difference in terms of computational time. This is due to the fact that the multiplication and reduction (modulo p) algorithms implemented in GMP are generic; i.e. they do not take into account the possible special form of the modulus. When implementing ECC/HECC algorithms, it is a good idea to use primes that allow fast modular arithmetic, such as those recommended by the NIST [39], the SEC Group [43], or more generally the primes belonging to what Bajard et al. called the Mersenne family [42, 11, 5]. In these cases, the multiplication becomes much more efficient than the inversion. In hardware implementations using inversion-free systems, the space for the inverter is often saved and the single final inversion is done using Fermat's little theorem. Although the overhead due to inversions in less dramatic for curves defined over $\mathbb{F}_{2^m}$, affine coordinates are not necessarily the best choice in practice, especially for software implementations [15]. In this paper we consider projective coordinates for curves defined over $\mathbb{F}_p$ and both affine and projective for curves defined over $\mathbb{F}_{2^m}$. More exactly, we use Jacobian coordinates ($\mathcal{J}$), a special class of projective coordinates, where the point $(X : Y : Z)$ corresponds to the affine point $(X/Z^2, Y/Z^3)$ when $Z \neq 0$. The point at infinity is represented as $(1 : 1 : 0)$. The opposite of $(X : Y : Z)$ is $(X : -Y : Z)$. Clearly there exist infinitely many points in the projective space which correspond to the same affine point. We use the common abusive notation $(X : Y : Z)$ to represent any representative of the equivalence class given by the relation of projection.

As we shall see, our DBNS-based point multiplication algorithm uses several basic operations (addition, doubling, tripling, etc.). In Sections 2.2.1 and 2.2.2, we recall the complexity of some of these curve operations, expressed in terms of the

[3]By opposition to supersingular curves of the form $y^2 + a_3y = x^3 + a_4x + a_6$ with $a_3 \neq 0$.

number of elementary operations in the field K. The interested reader is encouraged to check the literature [27, 2] for detailed descriptions of these algorithms. We use $[i]$, $[s]$ and $[m]$ to denote the cost of one inversion, one squaring and one multiplication, respectively. We always leave out the cost of field additions. For curves defined over $\mathbb{F}_p$ it is widely assumed that $[s] = 0.8[m]$. It is therefore a good idea to trade multiplications in favor of squarings whenever possible. However, as we shall see, our algorithms can be protected against SCA using side-channel atomicity [10]. In such cases, because squarings and multiplications must be performed using the same multiplier in order to be indistinguishable, we have to consider that $[s] = [m]$. For curves defined over binary fields, however, squarings are free (when normal bases are used to represent the elements of $\mathbb{F}_{2^m}$) or of negligible cost (squaring is a linear operation in polynomial basis); the complexity is thus mainly driven by the numbers of inversions and multiplications.

2.2.1. *Elliptic curves defined over $\mathbb{F}_p$.* When Jacobian coordinates are used the addition ($\text{ADD}^{\mathcal{J}}$) and doubling ($\text{DBL}^{\mathcal{J}}$) operations 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 $a_4 = -3$ (Brier and Joye [9] proved that most randomly chosen curves can be mapped to an isogeneous curve with $a_4 = -3$). Also, if one of the points is given in affine coordinates ($Z = 1$), then the cost of the so-called *mixed addition* ($\mathcal{J} + \mathcal{A} \to \mathcal{J}$) reduces to $8[m]+3[s]$. Several algorithms have been proposed for repeated doublings (the computation of $[2^w]P$) in the case of binary fields [25, 36]. For prime fields, an algorithm was proposed by Itoh et al. in [28], which is more efficient than w invocations of $\text{DBL}^{\mathcal{J}}$. In the general case ($a_4 \neq -3$) it requires $4w[m]+(4w+2)[s]$. When $a_4 = -3$ the cost of w-$\text{DBL}^{\mathcal{J}}$ is exactly the same as the cost of w doublings; i.e. $4w[m] + 4w[s]$. In Table 2, we summarize the complexity of these different elliptic curve operations. In the third column, we give the minimum number of registers required to achieve the corresponding complexities. See [2, chapter 13] for a complete description.

TABLE 2. Elliptic curve operations in Jacobian coordinates for curves defined over $\mathbb{F}_p$

Curve operation	Complexity	# Registers
$\text{DBL}^{\mathcal{J}}$	$4[m] + 6[s]$	6
$\text{DBL}^{\mathcal{J},\, a_4=-3}$	$4[m] + 4[s]$	5
$\text{ADD}^{\mathcal{J}}$	$12[m] + 4[s]$	7
$\text{ADD}^{\mathcal{J}+\mathcal{A}}$	$8[m] + 3[s]$	7
w-$\text{DBL}^{\mathcal{J}}$	$4w[m] + (4w + 2)[s]$	7

2.2.2. *Elliptic curves defined over $\mathbb{F}_{2^m}$.* For curves defined over $\mathbb{F}_{2^m}$ we give the cost of the most common operations in affine and Jacobian coordinates,[4] as they both have practical interest. Note that we only consider ordinary curves (for details concerning *supersingular* curves, see [2]).

[4]We do not consider Lopez-Dahab coordinates, which are also very attractive for curves defined over binary fields, simply because we did not find any good tripling formula in this system of coordinate.

Affine coordinates: The addition ($\text{ADD}^{\mathcal{A}}$) and doubling ($\text{DBL}^{\mathcal{A}}$) operations can be computed with the same number of field operations in $1[i] + 1[s] + 2[m]$. In [21], Eisenträger et al. proposed efficient formulae for tripling ($\text{TPL}^{\mathcal{A}}$), double-and-add ($\text{DA}^{\mathcal{A}}$) and triple-and-add ($\text{TA}^{\mathcal{A}}$). By trading some inversions for a small number of multiplications, their results have been further improved by Ciet et al. [12] when $[i]/[m] > 6$. We summarize the complexities of each of these operations in Table 4, with the break-even points between the two options for $\text{DA}^{\mathcal{A}}$, $\text{TPL}^{\mathcal{A}}$ and $\text{TA}^{\mathcal{A}}$.

TABLE 3. Elliptic curve operations in affine coordinates for curves defined over $\mathbb{F}_{2^m}$

Curve operation	[21]	[12]	break-even point
$\text{DBL}^{\mathcal{A}}$	$1[i] + 1[s] + 2[m]$		–
$\text{ADD}^{\mathcal{A}}$	$1[i] + 1[s] + 2[m]$		–
$\text{DA}^{\mathcal{A}}$	$2[i] + 2[s] + 3[m]$	$1[i] + 2[s] + 9[m]$	$[i]/[m] = 6$
$\text{TPL}^{\mathcal{A}}$	$2[i] + 2[s] + 3[m]$	$1[i] + 4[s] + 7[m]$	$[i]/[m] = 4$
$\text{TA}^{\mathcal{A}}$	$3[i] + 3[s] + 4[m]$	$2[i] + 3[s] + 9[m]$	$[i]/[m] = 5$

Jacobian coordinates: The use of Jacobian coordinates for curves defined over $\mathbb{F}_{2^m}$ was proposed by Hankerson et al. in [26], after noticing that their software implementation[5] using affine coordinates was leading to a ratio $[i]/[m] \simeq 10$. In the general case, an addition requires $16[m] + 3[s]$; it reduces to $11[m] + 3[s]$ if one of the points is given in affine coordinates. The doubling operation can be computed in $5[m] + 5[s]$, including one multiplication by a_6 (see [2, chapter 13] for detailed algorithms).

TABLE 4. Elliptic curve operations in Jacobian coordinates for curves defined over $\mathbb{F}_{2^m}$

Curve operation	Complexity	# Registers
$\text{DBL}^{\mathcal{J}}$	$5[m] + 5[s]$	5
$\text{ADD}^{\mathcal{J}}$	$16[m] + 3[s]$	10
$\text{ADD}^{\mathcal{J}+\mathcal{A}}$	$11[m] + 3[s]$	–

We note that, although the doubling algorithm can be computed using 5 multiplications and 5 squarings, it requires 6 atomic blocks if one considers (s, m, a)-blocks (i.e., blocks composed of 1 squaring, 1 multiplication and 1 addition in that order). Since squarings are almost free over $\mathbb{F}_{2^m}$, it is much better to consider (s, s, m, a)-blocks, as it indeed allows one to perform a doubling in 5 blocks; i.e. in 5 multiplications. We express both the doubling and the mixed addition with (s, s, m, a)-blocks in Tables 15 and 16 of Appendix B.

3. New curve arithmetic formulae

This section is devoted to new, efficient curve operations, which have been defined in order to best exploit the sparseness and the ternary nature of the DBNS representation. Namely, we give formulae for:

[5]For hardware implementation, however, affine coordinates seem to be the best choice.

- point tripling, consecutive triplings, as well a very specific consecutive doublings following (consecutive) tripling(s) in Jacobian coordinates for curves defined over $\mathbb{F}_p$,
- point tripling, point quadrupling (QPL$^{\mathcal{A}}$) and combined quadruple-and-add (QA$^{\mathcal{A}}$) in affine coordinates for curves defined over $\mathbb{F}_{2^m}$,
- point tripling (TPL$^{\mathcal{J}}$) in Jacobian coordinates for curves defined over $\mathbb{F}_{2^m}$.

3.1. **Curves defined over $\mathbb{F}_p$ using Jacobian coordinates.** In this section, we derive equations to obtain an efficient point tripling formula (TPL$^{\mathcal{J}}$) in Jacobian coordinates for curves defined over $\mathbb{F}_p$. (This formula was already present in [16].) Then, we explain how some field operations can be saved when several triplings (w-TPL$^{\mathcal{J}}$) have to be computed, or when several doublings have to be computed right after one or more triplings (w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$). As we shall see in Section 4, this very specific operation occurs quite often in our scalar multiplication algorithm.

To simplify, we start with affine coordinates. Let $P = (x_1, y_1) \in E(K)$ be a point on an elliptic curve E defined by (11). By definition, we have $[2]P = (x_2, y_2)$, where

$$\lambda_1 = \frac{3x_1^2 + a_4}{2y_1}, \quad x_2 = \lambda_1^2 - 2x_1, \quad y_2 = \lambda_1(x_1 - x_2) - y_1 \ . \tag{13}$$

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

$$\begin{aligned} \lambda_2 &= \frac{y_2 - y_1}{x_2 - x_1} \\ &= -\lambda_1 - \frac{2y_1}{x_2 - x_1} \\ &= -\frac{3x_1^2 + a_4}{2y_1} - \frac{8y_1^3}{(3x_1^2 + a_4)^2 - 12x_1y_1^2} \ . \end{aligned} \tag{14}$$

We further remark that

$$\begin{aligned} x_3 &= \lambda_2^2 - x_1 - x_2 \\ &= \lambda_2^2 - x_1 - \lambda_1^2 + 2x_1 \\ &= (\lambda_2 - \lambda_1)(\lambda_2 + \lambda_1) + x_1 \end{aligned} \tag{15}$$

and

$$\begin{aligned} y_3 &= \lambda_2(x_1 - x_3) - y_1 \\ &= -\lambda_2(\lambda_2 - \lambda_1)(\lambda_2 + \lambda_1) - y_1 \ . \end{aligned} \tag{16}$$

Thus $[3]P = (x_3, y_3)$ can be computed directly from 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 formula in Jacobian coordinates. Let $P = (X_1 : Y_1 : Z_1)$ be a point on the curve $\neq \mathcal{O}$. Then the point $[3]P = (X_3 : Y_3 : Z_3)$ is given by

$$\begin{aligned} X_3 &= 8Y_1^2(T - ME) + X_1E^2, \\ Y_3 &= Y_1\left(4(ME - T)(2T - ME) - E^3\right), \\ Z_3 &= Z_1E, \end{aligned} \tag{17}$$

where $M = 3X_1^2 + a_4Z_1^4$, $E = 12X_1Y_1^2 - M^2$ and $T = 8Y_1^4$.

The cost of (17) is $10[m] + 6[s]$. If one uses side-channel atomicity to resist SCA, this is equivalent to $16[m]$. We express the TPL$^{\mathcal{J}}$ algorithm in terms of atomic blocks in Table 13 of Appendix A. In comparison, computing $[3]P$ using the doubling and addition algorithms from [10], expressed as a repetition of atomic blocks, costs $10[m] + 16[m] = 26[m]$.

As we shall see, consecutive triplings; i.e., expressions of the form $[3^w]P$, occur quite often in our point multiplication algorithm. From (17), we remark that the computation of the intermediate value $M = 3X_1^2 + a_4Z_1^4$ requires $1[m] + 3[s]$ (we omit the multiplication by 3). If we need to compute $[9]P$, we then have to evaluate $M' = 3X_3^2 + a_4Z_3^4$. Since $Z_3 = Z_1E$, we have $a_4Z_3^4 = a_4Z_1^4E^4$ (where $E = 12X_1Y_1^2 - M^2$), and $a_4Z_1^4$ and E^2 have already been computed. Hence, using these precomputed subexpressions, we can compute $M' = 3X_3^2 + (a_4Z_1^4)(E^2)^2$, 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 always better than w invocations of the tripling algorithm. The atomic blocks version of w-TPL$^{\mathcal{J}}$ is given in Table 14 of Appendix A. Note that the idea of reusing a_4Z^4 for multiple doublings was first proposed by Cohen et al. in [14], where modified Jacobian coordinates are proposed.

From Table 2, DBL$^{\mathcal{J}}$ normally requires $4[m] + 6[s]$, or equivalently 10 blocks of computation if side-channel atomicity is used. The scalar multiplication algorithm presented in the next section very often[6] requires a w'-DBL$^{\mathcal{J}}$ to be computed right after a w-TPL$^{\mathcal{J}}$. This is due to the fact that we impose some conditions on the double-base expansion of the scalar k (see details in Section 4). When this occurs, it is possible to save $1[s]$ for the first DBL$^{\mathcal{J}}$ using subexpressions computed for the last tripling. The next $(w'-1)$-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 5, together with the number of registers required in each case.

TABLE 5. Tripling algorithms in Jacobian coordinates for curves defined over $\mathbb{F}_p$

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

3.2. **Curves defined over $\mathbb{F}_{2^m}$ using affine coordinates.** In this section, we propose new affine formulae for the computation of $[3]P$, $[4]P$ and $[4]P \pm Q$ for curves defined over $\mathbb{F}_{2^m}$.

Let us first recall the equations for the doubling operation. Given $P = (x_1, y_1)$, $P \neq -P$, we have $[2]P = (x_2, y_2)$, where

$$\lambda_1 = x_1 + \frac{y_1}{x_1}, \quad x_2 = \lambda_1^2 + \lambda_1 + a_2, \quad y_2 = \lambda_1(x_1 + x_2) + x_2 + y_1. \tag{18}$$

[6]The only exceptions occur when the expansion of k contains a series of consecutive $\{2, 3\}$-integers with identical binary exponents.

We shall compute $[3]P = (x_3, y_3)$ as $[3]P = P + [2]P$. From (18), we obtain after simplifications

$$x_3 = (\lambda_1 + \lambda_2 + 1)^2 + \lambda_2 + \lambda_1 + 1 + x_1, \quad y_3 = \lambda_2(x_1 + x_3) + x_3 + y_1, \tag{19}$$

where $\lambda_1 = x_1 + y_1/x_1$ and $\lambda_2 = (y_1 + y_2)/(x_1 + x_2)$. In order to reduce the number of field operations for the computations of x_3 and y_3, we want to get a convenient expression for $(\lambda_1 + \lambda_2 + 1)$. We start by expressing $(x_1 + x_2)$ in terms of x_1 only. We have:

$$x_1 + x_2 = x_1 + \lambda_1^2 + \lambda_1 + a_2 = \frac{x_1^4 + (y_1^2 + x_1 y_1 + a_2 x_1^2)}{x_1^2}.$$

From (12), since P is on the curve, we define $\alpha = x_1^4 + x_1^3 + a_6$ such that

$$x_1 + x_2 = \frac{\alpha}{x_1^2}. \tag{20}$$

Now, going back to the expression for λ_2, we have:

$$\begin{aligned}
\lambda_2 &= \lambda_1 + \frac{x_2}{x_1 + x_2} \\
&= \lambda_1 + \frac{x_1}{x_1 + x_2} + 1 \\
&= \lambda_1 + \frac{x_1^3}{\alpha} + 1 && (21) \\
&= \lambda_1 + \frac{x_1^4 + a_6}{\alpha} && (22) \\
&= \frac{\alpha(x_1^2 + y_1) + x_1^4 + a_6}{x_1 \alpha},
\end{aligned}$$

$$\lambda_2 = \frac{\beta}{x_1 \alpha}, \tag{23}$$

where $\beta = \alpha(x_1^2 + y_1) + x_1^4 + a_6$. From (21), we remark that

$$\lambda_1 + \lambda_2 + 1 = \frac{x_1^3}{\alpha}. \tag{24}$$

Replacing (23) and (24) in (19), we finally get

$$x_3 = \left(\frac{x_1^4}{x_1 \alpha}\right)^2 + \frac{x_1^4}{x_1 \alpha} + x_1, \tag{25}$$

$$y_3 = \frac{\beta}{x_1 \alpha}(x_1 + x_3) + x_3 + y_1. \tag{26}$$

It is easy to see that computing α requires $1[m] + 2[s]$; 1 extra $[m]$ gives β and $1/(x_1\alpha)$ is computed with another $1[m] + 1[i]$. The total cost for this new point tripling is thus $1[i] + 3[s] + 6[m]$. This is always better than the formula proposed in [12] (it saves $1[m] + 1[s]$) and becomes faster than the equation from [21] as soon as $[i] \geq 3[m]$ (see Table 3 for the exact costs of these previous methods).

For the quadrupling operations, the trick used in [21] by Eisenträger et al., which consists in evaluating only the x-coordinate of $[2]P$ when computing $[2]P \pm Q$, can also be applied to speed-up the quadrupling ($\text{QPL}^{\mathcal{A}}$) operation. From (19), we compute $[4]P = [2]([2]P) = (x_4, y_4)$ as

$$\lambda_2 = x_2 + \frac{y_2}{x_2}, \quad x_4 = \lambda_2^2 + \lambda_2 + a_2, \quad y_4 = \lambda_2(x_1 + x_4) + x_4 + y_1. \tag{27}$$

We observe that the computation of y_2 can be avoided by evaluating λ_2 as

$$\lambda_2 = \frac{x_1^2}{x_2} + \lambda_1 + x_2 + 1. \tag{28}$$

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

For the $\mathrm{QA}^{\mathcal{A}}$ operation, we evaluate $[4]P \pm Q$ as $[2]([2]P) \pm Q$ using one doubling ($\mathrm{DBL}^{\mathcal{A}}$) and one double-and-add ($\mathrm{DA}^{\mathcal{A}}$), 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 $[3]P + (P + Q)$ which requires $4[i] + 4[s] + 6[m]$.

In [12], Ciet et al. have improved an algorithm by Guajardo and Paar [25] for the computation of $[4]P$; their new method requires $1[i] + 5[s] + 8[m]$. Based on their costs, $\mathrm{QA}^{\mathcal{A}}$ is best evaluated as $([4]P) \pm Q$ using one quadrupling ($\mathrm{QPL}^{\mathcal{A}}$) followed by one addition ($\mathrm{ADD}^{\mathcal{A}}$) in $2[i] + 6[s] + 10[m]$. In Table 6 below, we summarize the costs and break-even points between our new formulae and the algorithms proposed in [12]. With such small break-even points, however, it remains unclear which formulae will give the best overall performance in practical situations.

TABLE 6. Tripling and quadrupling algorithms in affine coordinates for curves defined over $\mathbb{F}_{2^m}$

Operation	present work	[12]	break-even point
$\mathrm{TPL}^{\mathcal{A}}$	$1[i] + 3[s] + 6[m]$	$1[i] + 4[s] + 7[m]$	–
$\mathrm{QPL}^{\mathcal{A}}$	$2[i] + 3[s] + 3[m]$	$1[i] + 5[s] + 8[m]$	$[i]/[m] = 5$
$\mathrm{QA}^{\mathcal{A}}$	$3[i] + 3[s] + 5[m]$	$2[i] + 6[s] + 10[m]$	$[i]/[m] = 5$

3.3. **Curves defined over $\mathbb{F}_{2^m}$ using Jacobian coordinates.** As pointed out by Hankerson et al. in [26], affine coordinates might not be the best option for software implementations. In this section, we propose a tripling algorithm for curves defined over $\mathbb{F}_{2^m}$ using Jacobian coordinates. (We did not find an efficient tripling formula using Lopez-Dahad coordinates.)

Let us first recall the doubling formula. If $P = (X_1 : Y_1 : Z_1)$, we compute $[2]P = (X_2 : Y_2 : Z_2)$ as

$$\begin{aligned} X_2 &= B + a_6 C^4, \\ Z_2 &= X_1 C^2, \\ Y_2 &= BZ_2 + (A + D + Z_2)X_2, \end{aligned}$$

where $A = X_1^2$, $B = A^2$, $C = Z_1^2$ and $D = Y_1 Z_1$.

For the point tripling operation, we compute $[3]P = P + [2]P = (X_3 : Y_3 : Z_3)$ by deriving, for example, the addition formula from [2]. We easily obtain

$$Z_3 = (X_1 Z_2^2 + X_2 Z_1^2) Z_1 Z_2 = (X_1^3 Z_1^2 + X_2) Z_2 Z_1^3 = F Z_1^3,$$

with $E = (X_1 Z_1^2 + X_2)$ and $F = E Z_2$. We then compute X_3 as

$$\begin{aligned} X_3 &= a_2 Z_3^2 + (Y_1 Z_2^3 + Y_2 Z_1^3)(Y_1 Z_2^3 + Y_2 Z_1^3 + Z_3) + (X_1 Z_2^2 + X_2 Z_1^2)^3 \\ &= a_2 F^2 Z_1^6 + (Y_1 X_1^3 Z_1^6 + Y_2 Z_1^3)(Y_1 X_1^3 Z_1^6 + Y_2 Z_1^3 + F Z_1^3) + (X_1^3 Z_1^4 + X_2 Z_1^2)^3 \\ &= H Z_1^6, \end{aligned}$$

where $H = (a_2 F^2 + G(G+F) + E^3)$ and $G = (Y_1 X_1^3 Z_1^3 + Y_2)$. Finally, Y_3 can be computed as

$$\begin{aligned} Y_3 &= (Y_1 Z_2^3 + Y_2 Z_1^3 + Z_3) X_3 + ((X_1 Z_2^2 + X_2 Z_1^2) Z_1)^2 ((Y_1 Z_2^3 + Y_2 Z_1^3) X_2 \\ &\quad + (X_1 Z_1^2 + X_2 Z_1^2) Z_1 Y_2) \\ &= (Y_1 X_1^3 Z_1^3 + Y_2 + F) H Z_1^9 + (X_1 Z_1^2 + X_2)^2 ((X_1^3 Y_1 Z_1^3 + Y_2) X_2 \\ &\quad + (X_1^3 Z_1^2 + X_2) Y_2) Z_1^9 \\ &= I Z_1^9, \end{aligned}$$

where $I = (G+F)H + E^2(GX_2 + EY_2)$.

Using the fact that $[3]P = (X_3 : Y_3 : Z_3) = (HZ_1^6 : IZ_1^9 : FZ_1^3) = (H : I : F)$, the operation count is $15[m] + 7[s]$, including two multiplications by a_2 and a_6. In terms of memory, it requires 12 registers. We note that computing $[3]P$ as $[2]P + P$; i.e. using one doubling followed by one addition, would cost $21[m] + 8[s]$. We give an atomic version of this algorithm in Table 17 of Appendix B.

4. Scalar multiplication and double-base chains

In this section, we present a generic scalar multiplication algorithm which takes advantage of the properties of the double-base number and the efficient curve formulae presented in the previous sections. This generic algorithm can be easily adapted to different cases; we give the complexities for curves defined over $\mathbb{F}_p$ using Jacobian coordinates and for curves defined over $\mathbb{F}_{2^m}$ using both affine and Jacobian coordinates.

Everything would be easy and we would have nothing else to say if it was possible to use the greedy algorithm presented in Section 2.1 for the conversion. Unfortunately, in order to reduce the number of doublings and/or triplings, our algorithm requires the scalar n to be represented in a particular double-base form. More precisely, we need to express $n > 0$ as $n = \sum_{i=1}^{l} s_i 2^{a_i} 3^{b_i}$, with $s_i \in \{-1, 1\}$, where the exponents form two decreasing sequences; i.e., $a_1 \geq a_2 \geq \cdots \geq a_l \geq 0$ and $b_1 \geq b_2 \geq \cdots \geq b_l \geq 0$. More formally, we endow the set of $\{2,3\}$-integers with the order $\precsim$ induced by the product order on $\mathbb{N}^2$:

$$2^a 3^b \precsim 2^{a'} 3^{b'} \Leftrightarrow a \leq a', b \leq b'. \tag{29}$$

These particular DBNS representations allow us to expand n in a Horner-like fashion such that all partial results can be reused during the computation of $[n]P$. In fact, such a double-base expansion for n defines a double-base chain computing n.

Definition 4 (Double-base chain)**.** Given $n > 0$, a sequence $(C_i)_{i>0}$ of positive integers satisfying:

$$C_1 = 1, \quad C_{i+1} = 2^{u_i} 3^{v_i} C_i + s, \text{ with } s \in \{-1, 1\} \tag{30}$$

for some $u_i, v_i \geq 0$, and such that $C_l = n$ for some $l > 0$, is called a double-base chain computing n. The length l of a double-base chain is equal to the number of $\{2,3\}$-integers in (1) used to represent the integer n.

Note that it is always possible to find a double-base chain computing n; 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 per se a very intriguing problem. Let $g(n)$ denote the number of (unsigned) double-base chains computing n. Clearly, since the binary expansion is a trivial case, one has $g(3n) \geq 1 + g(u)$, and thus $g(3^n) \geq n + 1$. If $\overline{g}(n) = \frac{1}{n}\sum_{t=0}^{u} g(t)$, we conjecture that, for large n, one has $\log n < \overline{g}(n)$; and maybe $\lim_{n\to\infty} \frac{\log n}{\overline{g}(n)} = 0$. Moreover, it is possible to prove that $g(n) = 1$, if and only if, either $n \in \{0, 1, 2\}$ or $n = 2^a 3 - 1$, for $a \geq 1$.

If necessary, such a specific DBNS representation of any w-bit positive integer n can be computed using Algorithm 2 below; a modified version of the greedy algorithm which takes into account the order $\precsim$ on the exponents.

Algorithm 2 Greedy algorithm with restricted exponents

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

Output The sequence $(s_i, a_i, b_i)_{i>0}$ such that $n = \sum_{i=1}^{l} s_i 2^{a_i} 3^{b_i}$, with $2^{a_i} 3^{b_i} \precsim 2^{a_{i-1}} 3^{b_{i-1}}$ for $i \geq 1$

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

Two important parameters of this algorithm are the upper bounds for the binary and ternary exponents in the expansion of n, called a_{max} and b_{max}, respectively. Clearly, we have $a_{max} < \log_2(n) < w$ and $b_{max} < \log_3(n) \approx 0.63w$. Our experiments showed that using these utmost values for a_{max} and b_{max} does not result in short expansions. Indeed, when the best approximation of a given integer of the form $2^a 3^b$ is either close to a power of 2 (i.e. b is small) or close to a power of 3 (i.e. a is small), the resulting double-base chains are likely to be the binary or the balanced ternary expansions. We want to avoid this phenomenon by selecting a_{max}, b_{max} such that $2^{a_{max}} 3^{b_{max}}$ is slightly greater than n, and a_{max} is not too large/small compared to b_{max}. The optimal values for a_{max}, b_{max} seem difficult to determine in the general case as they clearly depend (but not only) on the relative cost between the doubling and the tripling operations. Instead, we consider the following heuristic which leads to good results in practice: if $n = (n_{w-1} \dots n_1 n_0)_2$ is a randomly chosen w-bit integer (i.e. $n_{w-1} \neq 0$), we initially set $a_{max} = x$ and $b_{max} = y$, where $2^x 3^y$ is a very good, non-trivial (i.e. $y \neq 0$) approximation of 2^w. Then, in order to get sequences of exponents satisfying the conditions $2^{a_i} 3^{b_i} \precsim 2^{a_{i-1}} 3^{b_{i-1}}$ for $i \geq 1$, the new largest exponents are updated according to the values of a and b obtained in step 3.

We can now present a generic point multiplication algorithm which can be easily adapted to various cases depending on the field over which the curve is defined and the curve operations we have at our disposal.

Algorithm 3 Generic DBNS Scalar Multiplication

Input An integer $n = \sum_{i=1}^{l} s_i\, 2^{a_i} 3^{b_i}$, with $s_i \in \{-1, 1\}$, and such that $2^{a_i} 3^{b_i} \precsim 2^{a_{i-1}} 3^{b_{i-1}}$ for $i \geq 1$; and a point $P \in E(K)$
Output the point $[n]P \in E(K)$
1: $Q \leftarrow [s_1]P$
2: **for** $i = 1, \ldots, l-1$ **do**
3: $\quad u_i \leftarrow a_i - a_{i+1}, \quad v_i \leftarrow b_i - b_{i+1}$
4: $\quad Q \leftarrow [3^{v_i}]Q$
5: $\quad Q \leftarrow [2^{u_i}]Q$
6: $\quad Q \leftarrow Q + [s_{i+1}]P$
7: **Return** Q

The complexity of Algorithm 3 depends on the number of doublings, triplings and mixed additions that have to be performed: the total number of additions is equal to the length l of the double-base expansion of n, and the number of doublings and triplings are equal to $a_1 \leq a_{max}$ and $b_1 \leq b_{max}$, respectively. However, the complexity can be more precisely evaluated if one considers the exact cost of each iteration, by counting the exact number of field operations (inversions, multiplications and squarings) required in steps 4 to 6.

In fact, given $n > 0$, Algorithm 3 immediately gives us a double-base chain for n. Let W_i be the exact number of curve operations required to compute $[C_i]P$ from $[C_{i-1}]P$. We clearly have $C_1 = 1$ and $W_1 = 0$ (we set Z to P or $-P$ at no cost in step 1). Hence, the total cost for computing $[n]P$ from input point P is given by

$$W_l = \sum_{i=1}^{l} W_i. \tag{31}$$

In the next three sections, we consider three cases: curves defined over $\mathbb{F}_p$ using Jacobian coordinates, and curves defined over $\mathbb{F}_{2^m}$ using both affine and Jacobian coordinates. Extensions to other cases, for example for curves defined over fields of characteristic three, can be easily derived.

4.1. **Curves defined over $\mathbb{F}_p$ with Jacobian coordinates.** In this case, steps 4 and 5 can be implemented using the w-TPL$^{\mathcal{J}}$/w'-DBL$^{\mathcal{J}}$ operation presented in Section 4.1. When $u_i = 0$ or $v_i = 0$ in Step 3, w-TPL$^{\mathcal{J}}$ or w-DBL$^{\mathcal{J}}$ are called instead. The addition is always a mixed addition (ADD$^{\mathcal{J}+\mathcal{A}}$). We have:

$$W_i = v_i\text{-TPL}^{\mathcal{J}}/u_i\text{-DBL}^{\mathcal{J}} + \text{ADD}^{\mathcal{J}+\mathcal{A}}. \tag{32}$$

The total cost is given by (31).

4.2. **Curves defined over $\mathbb{F}_{2^m}$ with Jacobian coordinates.** In this case we did not find any way to save some operations for consecutive doublings and/or triplings. The addition is always a mixed addition. We have:

$$W_i = v_i \times \text{TPL}^{\mathcal{J}} + u_i \times \text{DBL}^{\mathcal{J}} + \text{ADD}^{\mathcal{J}+\mathcal{A}}. \tag{33}$$

Again, the total cost is given by (31).

4.3. **Curves defined over $\mathbb{F}_{2^m}$ with affine coordinates.** In this case, the algorithm can be further optimized in order to take advantage of the quadrupling and combined quadruple-and-add algorithms presented in Section 3.2. Algorithm 4 below is an adaptation of our generic algorithm.

Algorithm 4 DBNS scalar multiplication for curves over $\mathbb{F}_{2^m}$ using affine coordinates

Input An integer $n = \sum_{i=1}^{l} s_i\, 2^{a_i} 3^{b_i}$, with $s_i \in \{-1, 1\}$, and such that $2^{a_i} 3^{b_i} \precsim 2^{a_{i-1}} 3^{b_{i-1}}$ for $i \geq 1$; and a point $P \in E(K)$
Output the point $[n]P \in E(K)$
1: $Q \leftarrow [s_1]P$
2: **for** $i = 1, \ldots, l-1$ **do**
3: $\quad u_i \leftarrow a_i - a_{i+1}, \quad v_i \leftarrow b_i - b_{i+1}$
4: $\quad$ **if** $u_i = 0$ **then**
5: $\quad\quad Q \leftarrow [3]([3^{v_i - 1}]Q) + [s_{i+1}]P$
6: $\quad$ **else**
7: $\quad\quad Q \leftarrow [3^{v_i}]Q$
8: $\quad\quad Q \leftarrow [4^{\lfloor (u_i - 1)/2 \rfloor}]Q$
9: $\quad\quad$ **if** $u_i \equiv 0 \pmod 2$ **then**
10: $\quad\quad\quad Q \leftarrow [4]Q + [s_{i+1}]P$
11: $\quad\quad$ **else**
12: $\quad\quad\quad Q \leftarrow [2]Q + [s_{i+1}]P$
13: **Return** Q

We remark that although $l - 1$ additions are required to compute $[n]P$, we never actually use the addition operation (ADD$^{\mathcal{A}}$); simply because we combine each addition with either a doubling (step 13), a tripling (step 6) or a quadrupling (step 11), using the DA$^{\mathcal{A}}$, TA$^{\mathcal{A}}$ and QA$^{\mathcal{A}}$ operations. Note also that the TA$^{\mathcal{A}}$ operation for computing $[3]P \pm Q$ is only used in step 6, when $u_i = 0$. Another approach of similar cost is to start with all the quadruplings plus one possible doubling when u_i is odd, and then perform $v_i - 1$ triplings followed by one final triple-and-add.

The expression for W_i is a little more complicated; we have:

$$W_i = \delta_{u_i,0}\left((v_i - 1)\,T + TA\right) + (1 - \delta_{u_i,0})\left(v_i\,T + \left\lfloor \frac{u_i - 1}{2} \right\rfloor Q + \delta_{|u_i|_2,0}\,QA + \delta_{|u_i|_2,1}\,DA\right), \tag{34}$$

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 $|u_i|_2$ denotes $u_i \bmod 2$.

5. Comparisons and experimental results

In this section, we illustrate the efficiency of the proposed variants of the generic algorithm by providing experimental results and comparisons with classical methods (double-and-add, NAF, w-NAF) and some recently proposed algorithms: a ternary/binary approach from [12] for curves defined over binary fields using affine coordinates; and two algorithms from Izu et al. published in [29] and [31] for curves

defined over prime fields. In the latter, we consider the protected version of our algorithm, combined with Joye and Tymen's randomization technique to counteract differential attacks [32].

If we assume that n is a randomly chosen integer, it is well known that the double-and-add algorithm requires $\log n$ doublings and $\log n/2$ 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 $\log n/(w+1)$. Unfortunately, it seems very difficult to give such a theoretical estimate for double-base chains. When the exponents do not have to satisfy any other conditions than being positive integers, it can be proved [18] that the greedy algorithm returns expansions of length $O(\log n/\log\log n)$. However, for double-base chains, 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. Therefore, in order to estimate the average number of $\{2,3\}$-integers required to represent n, and to precisely evaluate the complexity of our point multiplication algorithms, we have performed several numerical experiments, over 10000 randomly chosen 160-bit integers. The results are presented below.

5.1. **Curves defined over $\mathbb{F}_p$ with Jacobian coordinates.** 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 $159\times 10+80\times 11=2470[m]$; similarly, the NAF and 4-NAF methods require, on average, $2173[m]$ and $1942[m]$, respectively. The results of our numerical experiments in the case of double-base chains are presented in Table 7.

TABLE 7. Average complexity of our scalar multiplication algorithm obtained using 10000 randomly chosen 160-bit integers for different values a_{max}, b_{max} and curves defined over $\mathbb{F}_p$ using Jacobian coordinates

a_{max}	b_{max}	l	Complexity	$[s]=[m]$	$[s]=0.8[m]$
57	65	44.09	$758.25[s]+1236.60[m]$	1994.86	1843.20
76	53	37.23	$770.23[s]+1132.45[m]$	1902.69	1748.64
95	41	36.63	$812.24[s]+1072.48[m]$	1884.73	1722.28
103	36	38.39	$840.44[s]+1061.33[m]$	1901.78	1733.69

In order to compare our algorithm with the side-channel resistant algorithms presented in [29, 31, 30], we also give the uniform costs in terms of the equivalent number of field multiplications in the last two columns. Note that, if side-channel atomicity is used to prevent simple analysis, squarings cannot be optimized and must be computed using a general multiplier; one must therefore consider $[s]=[m]$. In the last column of Table 7, we also give the complexity in terms of the equivalent number of multiplications assuming $[s]=0.8[m]$.

In Table 8, we summarize the costs of several scalar multiplication algorithms. In order to present fair comparisons, we add the extra cost of Joye and Tymen's randomization technique ($41[m]$ assuming $[i]=30[m]$) which can be used to resist differential analysis. The figures for the algorithms from Izu, Möller and Takagi are taken from [29] and [31] 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 $a_{max} = 95$ and $b_{max} = 41$, as these values lead to the best operation count.

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 [29]	2449
Izu, Takagi 2005 [31]	2629
DBNS	1926

We remark that the DBNS algorithm requires fewer operations than the other methods. It represents a gain of 23.29% over the double-and-add, 13% over the NAF, 2.8% over 4-NAF, 21.35% over [29] and 26.7% over [31]. Moreover, it does not require precomputations like the 4-NAF and the algorithms from Izu et al.

5.2. **Curves defined over $\mathbb{F}_{2^m}$ with Jacobian coordinates.** As noticed in Section 3.3, the cost of our tripling is almost equivalent to that of one doubling followed by a mixed addition. From our numerical experiments, we remark that the average length l of the double-base chains obtained with the greedy algorithm for different values a_{max}, b_{max} lies between $2\log n/9$ and $\log n/3$. Unfortunately, with such an efficient doubling formula, even the shortest double-base chains result in too many additions (about 40 for 160-bit scalar) and too many triplings to defeat algorithms based on doublings only. Therefore, an optimized algorithm is very likely to behave like the NAF algorithm which only uses doublings and mixed additions; on average the NAF requires $5[m] \times 159 + 11[m] \times 53 = 1378[m]$. This is confirmed by our numerical results presented in Table 9. The best results are not obtained for shortest chains but for the expansions which minimize the number of triplings (and maximize the number of doublings).

TABLE 9. Average complexity of our DBNS point multiplication algorithm obtained using 10000 randomly chosen 160-bit integers for different values a_{max}, b_{max} and curves defined over $\mathbb{F}_{2^m}$ using Jacobian coordinates

a_{max}	b_{max}	l	Complexity (#[m] only)
57	65	44.09	$1708[m]$
76	53	37.23	$1566[m]$
95	41	36.63	$1478[m]$
103	36	38.39	$1459[m]$
156	3	52.41	$1374[m]$
159	1	53.10	$1370[m]$

For curves over $\mathbb{F}_{2^m}$ and Jacobian coordinates, the double-base approach will thus only become a serious alternative if one can find a better tripling formula, or an algorithm leading to shorter double-base chains.

5.3. **Curves defined over $\mathbb{F}_{2^m}$ with affine coordinates.** We summarize our experimental results and the comparisons with other classical methods in Table 10. These results have been obtained with the curve operations that have the best complexity when the ratio $[i]/[m]$ is small. Indeed, when the relative cost of an inversion increases, inversion-free coordinates become rapidly more interesting (see [26]).

For completeness, we also give the operation counts for a recent ternary/binary algorithm presented in [12], which is based on the following recursive decomposition: if $n \equiv 0$ or $3 \pmod 6$, return $[3]([n/3]P)$; if $n \equiv 2$ or $4 \pmod 6$, return $[2]([k/2]P)$; if $n \equiv 1 \pmod 6$, i.e., $n = 6m + 1$, return $[2]([3m]P) + P$; if $n \equiv 5 \pmod 6$, i.e., $n = 6m - 1$, return $[2]([3m]P) - P$. The recursion stops whenever $n = 1$ and returns P. Applying this recursive decomposition to any positive scalar n leads, of course, to a double-base chain which 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 better representations having the same properties on the exponents and, at the same time, leading to shorter chains and reduced complexity.

TABLE 10. Average complexity and comparisons with other methods of our DBNS point multiplication algorithm obtained using 10000 randomly chosen 160-bit integers and different values a_{max}, b_{max} for curves defined over $\mathbb{F}_{2^m}$ using affine coordinates

a_{max}	b_{max}	l	Complexity	$[i] = 3[m]$	$[i] = 10[m]$
57	65	44.09	$228.69[i] + 242.32[s] + 335.66[m]$	1022	2066
76	53	37.23	$216.17[i] + 235.09[s] + 324.84[m]$	973	1932
95	41	36.63	$211.74[i] + 235.86[s] + 322.38[m]$	958	1898
103	36	38.39	$211.15[i] + 237.50[s] + 322.46[m]$	956	1906
Double-and-add			$244.31[i] + 244.82[s] + 407.09[m]$	1139	2847
NAF			$217.22[i] + 217.91[s] + 380.63[m]$	1031	2550
Ternary/Binary			$222.11[i] + 222.84[s] + 353.04[m]$	1019	2573

We remark that our algorithm requires fewer inversions and multiplications than the other methods. In order to clarify the comparison, we report, in the last two columns of Table 10, the cost in terms of the equivalent number of multiplications assuming $[i] = 3[m]$ and $[i] = 10[m]$. In the first case, our algorithm represents a speed-up of about 16% over the double-and-add, 7% over the NAF and 6% over the ternary/binary approach proposed in [12]. In the more realistic case (for software implementations) $[i] = 10[m]$, the speed-ups are even more important; 33% over the double-and-add, 25% over the NAF and 26% over the ternary/binary approach.

6. CONCLUSIONS

In this paper, we proposed several variants of a generic point multiplication algorithm based on the representation of n as $\sum_i \pm 2^{a_i} 3^{b_i}$. Among many nice properties, this representation scheme, called the double-base number system, offers the advantage of being very sparse. In the context of our scalar multiplication algorithm, the extra condition that the sequences of exponents must decrease does not allow us to claim sublinearity for the length of the double-base chains. However,

we provided extensive numerical evidence that demonstrates the efficiency of this approach compared to existing methods of similar nature.

Acknowledgments

The authors are very thankful to the anonymous reviewers for their very useful comments and for suggesting to us the improved tripling formula in affine coordinates for curves over binary fields. They also thank Nicolas Meloni for the improved tripling formula in Jacobian coordinates for curves over binary fields.

Appendix A. Curves defined over $\mathbb{F}_p$ using Jacobian coordinates

In this appendix, we give the algorithms for $\text{DBL}^{\mathcal{J}}$ (including the case when a doubling is performed right after a tripling), w-$\text{DBL}^{\mathcal{J}}$, $\text{TPL}^{\mathcal{J}}$ and w-$\text{TPL}^{\mathcal{J}}$, expressed in atomic blocks, for curves defined over $\mathbb{F}_p$, with Jacobian coordinates.

Table 11. The $\text{DBL}^{\mathcal{J}}$ algorithm in atomic blocks. When $\text{DBL}^{\mathcal{J}}$ is called right after w-$\text{TPL}^{\mathcal{J}}$, the blocks Δ_2, Δ_3 and Δ_4 can be replaced by the blocks Δ_2' and Δ_3' to save one multiplication

$\text{DBL}^{\mathcal{J}}$ / $\mathbb{F}_p$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $[2]P = (X_3 : Y_3 : Z_3) = (R_1 : R_2 : R_3)$
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 12. 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 11) 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 $[2]P$. After $w-1$ iterations $[2^w]P = (X_3 : Y_3 : Z_3) = (X_t : Y_t : Z_t)$

w-DBL$^{\mathcal{J}}$ / $\mathbb{F}_p$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $[2^w]P = (R_1 : R_2 : R_3)$
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 13. The TPL$^{\mathcal{J}}$ algorithm in atomic blocks

TPL$^{\mathcal{J}}$ / $\mathbb{F}_p$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $[3]P = (X_3 : Y_3 : Z_3) = (R_1 : R_2 : R_3)$
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}	$R_6 = R_6 \times R_8$	
	$*$			$R_6 = R_6 + R_6$	
	$R_8 = -R_8$	$(-M^2)$		$*$	
	$R_5 = R_5 + R_8$	(E)		$R_6 = R_6 + R_5$	
Γ_8	$R_3 = R_3 \times R_5$	(Z_3)	Γ_{16}	$R_2 = R_2 \times R_6$	(Y_3)
	$*$			$*$	
	$*$			$*$	
	$*$			$*$	

TABLE 14. 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 $[3]P$; 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$^{\mathcal{J}}$ / $\mathbb{F}_p$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $[3^w]P = (R_1 : R_2 : R_3)$
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}	$R_6 = R_6 \times R_8$	
	$*$			$R_6 = R_6 + R_6$	
	$R_8 = -R_8$	$(-M^2)$		$*$	
	$R_5 = R_5 + R_8$	(E)		$R_6 = R_6 + R_5$	
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)			
	$R_7 = R_7 + R_7$	$(8Y_t^2)$			
	$*$				
	$*$				

APPENDIX B. CURVES DEFINED OVER $\mathbb{F}_{2^m}$ USING JACOBIAN COORDINATES

In this appendix, we give the algorithms for DBL$^{\mathcal{J}}$, ADD$^{\mathcal{J}+\mathcal{A}}$ and TPL$^{\mathcal{J}}$, expressed in atomic blocks, for curves defined over $\mathbb{F}_{2^m}$, with Jacobian coordinates. We consider (s, s, m, a)-blocks to avoid the use of 6 blocks for the doubling. Note that the mixed addition and the tripling can be expressed in 11 and 16 (s, m, a)-blocks respectively.

TABLE 15. The $\mathrm{DBL}^{\mathcal{J}}$ algorithm for curves over $\mathbb{F}_{2^m}$ using Jacobian coordinates

$\mathrm{DBL}^{\mathcal{J}}$ / $\mathbb{F}_{2^m}$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $[2]P = (X_3 : Y_3 : Z_3) = (R_1 : R_2 : R_3)$
Init: $R_1 = X_1,\ R_2 = Y_1,\ R_3 = Z_1$

Δ_1	$R_4 = R_1^2$	(X_1^2)	Δ_4	$*$	
	$R_5 = R_4^2$	$(X1^4)$		$*$	
	$R_2 = R_2 \times R_3$	(Y_1Z_1)		$R_3 = R_5 \times R_3$	$(X_1^4Z_3)$
	$R_2 = R_4 + R_2$	$(X_1^2 + Y_1Z_1)$		$*$	
Δ_2	$R_3 = R_3^2$	(Z_1^2)	Δ_5	$*$	
	$R_4 = R_3^2$	(Z_1^4)		$*$	
	$R_3 = R_1 \times R_3$	(Z_3)		$R_2 = R_2 \times R_1$	
	$R_2 = R_2 + R_3$			$R_2 = R_3 + R_2$	(Y_3)
Δ_3	$R_1 = R_4^2$	(Z_1^8)			
	$*$				
	$R_1 = a_6 \times R_1$	$(a_6Z_1^8)$			
	$R_1 = R_5 + R_1$	(X_3)			

TABLE 16. The $\mathrm{ADD}^{\mathcal{J}+\mathcal{A}}$ algorithm for curves over $\mathbb{F}_{2^m}$ using Jacobian coordinates

$\mathrm{ADD}^{\mathcal{J}+\mathcal{A}}$ / $\mathbb{F}_{2^m}$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1),\ Q = (X_2 : Y_2 : 1)$
Output: $P + Q = (X_3 : Y_3 : Z_3) = (R_4 : R_5 : R_3)$
Init: $R_1 = X_1,\ R_2 = Y_1,\ R_3 = Z_1,\ R_4 = X_2,\ R_5 = Y_2$

Δ_1	$R_6 = R_3^2$	(Z_1^2)	Δ_7	$*$	
	$*$			$*$	
	$R_7 = R_4 \times R_6$	(B)		$R_4 = a_2 \times R_8$	$(a_2Z_3^2)$
	$R_1 = R_1 + R_7$	(E)		$*$	
Δ_2	$R_7 = R_1^2$	(E^2)	Δ_8	$*$	
	$*$			$*$	
	$R_6 = R_6 \times R_3$	(Z_1^3)		$R_2 = R_2 \times R_6$	(FI)
	$*$			$R_4 = R_4 + R_2$	$(a_2Z_3^2 + FI)$
Δ_3	$*$		Δ_9	$*$	
	$*$			$*$	
	$R_6 = R_6 \times R_5$	$(Y_2Z_1^3)$		$R_2 = R_7 \times R_1$	(E^3)
	$R_2 = R_2 + R_6$	(F)		$R_4 = R_4 + R_2$	(X_3)
Δ_4	$*$		Δ_{10}	$*$	
	$*$			$*$	
	$R_3 = R_1 \times R_3$	(Z_3)		$R_4 = R_6 \times R_4$	(IX_3)
	$R_6 = R_2 + R_3$	(I)		$*$	
Δ_5	$R_8 = R_3^2$	(Z_3^2)	Δ_{11}	$*$	
	$*$			$*$	
	$R_4 = R_2 \times R_4$	(FX_2)		$R_5 = R_8 \times R_5$	(Z_3^2H)
	$*$			$R_5 = R_4 + R_5$	(Y_3)
Δ_6	$*$				
	$*$				
	$R_5 = R_3 \times R_5$	(Z_3Y_2)			
	$R_5 = R_4 + R_5$	(H)			

TABLE 17. The $\text{TPL}^{\mathcal{J}}$ algorithm for curves defined over $\mathbb{F}_{2^m}$ using Jacobian coordinates

$\text{TPL}^{\mathcal{J}}$ / $\mathbb{F}_{2^m}$ / Jacobian
Input: $P = (X_1 : Y_1 : Z_1)$
Output: $[3]P = (X_3 : Y_3 : Z_3) = (R_6 : R_4 : R_{11})$
Init: $R_1 = X_1,\ R_2 = Y_1,\ R_3 = Z_1$

Γ_1	$R_4 = R_1^2$	$(A = X_1^2)$	Γ_9	$R_6 = R_1 \times R_2$	(F^2)
	$R_5 = R_1^2$	$(B = X_1^4)$		$*$	
	$R_6 = R_2 \times R_3$	(Y_1Z_1)		$R_6 = a_2 \times R_6$	(a_2F^2)
	$*$			$R_7 = R_4 + R_{11}$	$(G + F)$
Γ_2	$R_7 = R_3^2$	$(C = Z_1^2)$	Γ_{10}	$R_{11} = R_{10}^2$	(E^2)
	$R_8 = R_7^2$	(Z_1^4)		$*$	
	$R_7 = R_1 \times R_7$	(Z_2)		$R_{12} = R_4 \times R_7$	$(G(G + F))$
	$R_9 = R_4 + R_6$	$(A + D)$		$R_6 = R_6 + R_{12}$	
Γ_3	$R_8 = R_8^2$	(Z_1^8)	Γ_{11}	$*$	
	$*$			$*$	
	$R_8 = a_6 \times R_8$	$(a_6Z_1^8)$		$R_{12} = R_{10} \times R_{11}$	(E^3)
	$R_8 = R_5 + R_8$	(X_2)		$R_6 = R_6 + R_{12}$	$(H = X_3)$
Γ_4	$*$		Γ_{12}	$*$	
	$*$			$*$	
	$R_5 = R_5 \times R_7$	(BZ_2)		$R_6 = R_6 \times R_7$	$(H(G + F))$
	$R_9 = R_9 + R_7$	$(A + D + Z_2)$		$*$	
Γ_5	$*$		Γ_{13}	$*$	
	$*$			$*$	
	$R_9 = R_9 \times R_8$			$R_4 = R_4 \times R_8$	(GX_2)
	$R_{10} = R_7 + R_8$	(E)		$*$	
Γ_6	$*$		Γ_{14}	$*$	
	$*$			$*$	
	$R_{11} = R_{10} \times R_7$	$(F = Z_3)$		$R_5 = R_{10} \times R_5$	(EY_2)
	$R_5 = R_5 + R_9$	(Y_2)		$R_4 = R_4 + R_5$	$(GX_2 + EY_2)$
Γ_7	$*$		Γ_{15}	$*$	
	$*$			$*$	
	$R_4 = R_4 \times R_7$	(AZ_2)		$R_4 = R_{11} \times R_4$	
	$*$			$R_4 = R_4 + R_6$	$(I = Y_3)$
Γ_8	$*$				
	$*$				
	$R_4 = R_4 \times R_6$	(AZ_2D)			
	$R_4 = R_4 + R_5$	(G)			

References

[1] J.-P. Allouche and J. Shallit, *Automatic sequences*, Cambridge University Press, 2003. MR1997038 (2004k:11028)

[2] R. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of elliptic and hyperelliptic curve cryptography*, CRC Press, 2005. MR2162716 (2007f:14020)

[3] R. Avanzi, V. Dimitrov, C. Doche, and F. Sica, *Extending scalar multiplication using double bases*, Advances in Cryptology, ASIACRYPT'06, Lecture Notes in Computer Science, vol. 4284, Springer, 2006, pp. 130–144.

[4] R. Avanzi and F. Sica, *Scalar multiplication on Koblitz curves using double bases*, Cryptology ePrint Archive, Report 2006/067, 2006, http://eprint.iacr.org/2006/067.

[5] J.-C. Bajard, L. Imbert, and T. Plantard, *Modular number systems: Beyond the Mersenne family*, Proceedings of the 11th International Workshop on Selected Areas in Cryptography, SAC'04, Lecture Notes in Computer Science, vol. 3357, Springer, 2005, pp. 159–169. MR2181315 (2006h:94071)

[6] V. Berthé, *Autour du système de numération d'Ostrowski*, Bulletin of the Belgian Mathematical Society **8** (2001), 209–239. MR1838931 (2002k:68147)

[7] V. Berthé and L. Imbert, *On converting numbers to the double-base number system*, Advanced Signal Processing Algorithms, Architecture and Implementations XIV, Proceedings of SPIE, vol. 5559, SPIE, 2004, pp. 70–78.

[8] I. F. Blake, G. Seroussi, and N. P. Smart, *Advances in elliptic curve cryptography*, London Mathematical Society Lecture Note Series, no. 317, Cambridge University Press, 2005. MR2166105

[9] É. Brier and M. Joye, *Fast point multiplication on elliptic curves through isogenies*, Applied Algebra, Algebraic Algorithms and Error-Correcting Codes, AAECC 2003, Lecture Notes in Computer Science, vol. 2643, Springer, 2003, pp. 43–50. MR2042411 (2005a:14029)

[10] 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** (2004), no. 6, 760–768.

[11] J. Chung and A. Hasan, *More generalized Mersenne numbers*, Selected Areas in Cryptography, SAC'03, Lecture Notes in Computer Science, vol. 3006, Springer, 2004, pp. 335–347. MR2094740 (2005f:94089)

[12] M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery, *Trading inversions for multiplications in elliptic curve cryptography*, Designs, Codes and Cryptography **39** (2006), no. 2, 189–206. MR2209936 (2006j:94057)

[13] M. Ciet and F. Sica, *An analysis of double base number systems and a sublinear scalar multiplication algorithm*, Progress of Cryptology, Mycrypt 2005, Lecture Notes in Computer Science, vol. 3715, Springer, 2005, pp. 171–182.

[14] H. Cohen, A. Miyaji, and T. Ono, *Efficient elliptic curve exponentiation using mixed coordinates*, Advances in Cryptology, ASIACRYPT'98, Lecture Notes in Computer Science, vol. 1514, Springer, 1998, pp. 51–65. MR1726152

[15] E. De Win, S. Bosselaers, S. Vandenberghe, P. De Gersem, and J. Vandewalle, *A fast software implementation for arithmetic operations in* GF(2^n), Advances in Cryptology, ASIACRYPT'96, Lecture Notes in Computer Science, vol. 1163, Springer, 1996, pp. 65–76. MR1486049

[16] V. Dimitrov, L. Imbert, and P. K. Mishra, *Efficient and secure elliptic curve point multiplication using double-base chains*, Advances in Cryptology, ASIACRYPT'05, Lecture Notes in Computer Science, vol. 3788, Springer, 2005, pp. 59–78. MR2236727

[17] V. S Dimitrov, K. Järvinen, M. J. Jacobson, Jr., W. F. Chan, and Z. Huang, *FPGA implementation of point multiplication on Koblitz curves using Kleinian integers*, Cryptographic Hardware and Embedded Systems, CHES'06, Lecture Notes in Computer Science, vol. 4249, Springer, 2006, pp. 445–459.

[18] V. S. Dimitrov, G. A. Jullien, and W. C. Miller, *An algorithm for modular exponentiation*, Information Processing Letters **66** (1998), no. 3, 155–159. MR1627991 (99d:94023)

[19] C. Doche, T. Icart, and D. R. Kohel, *Efficient scalar multiplication by isogeny decompositions*, Public Key Cryptography, PKC'06, Lecture Notes in Computer Science, vol. 3958, Springer, 2006, pp. 191–206.

[20] C. Doche and L. Imbert, *Extended double-base number system with applications to elliptic curve cryptography*, Progress in Cryptology, INDOCRYPT'06, Lecture Notes in Computer Science, vol. 4329, Springer, 2006, pp. 335–348.

[21] K. Eisenträger, K. Lauter, and P. L. Montgomery, *Fast elliptic curve arithmetic and improved Weil pairing evaluation*, Topics in Cryptology – CT-RSA 2003, Lecture Notes in Computer Science, vol. 2612, Springer, 2003, pp. 343–354. MR2080147

[22] K. Fong, D. Hankerson, J. Lòpez, and A. Menezes, *Field inversion and point halving revisited*, IEEE Transactions on Computers **53** (2004), no. 8, 1047–1059.

[23] D. M. Gordon, *A survey of fast exponentiation methods*, Journal of Algorithms **27** (1998), no. 1, 129–146. MR1613189 (99g:94014)

[24] T. Granlund, *GMP, the GNU multiple precision arithmetic library*, see: http://www.swox.com/gmp/.

[25] J. Guajardo and C. Paar, *Efficient algorithms for elliptic curve cryptosystems*, Advances in Cryptology, CRYPTO'97, Lecture Notes in Computer Science, vol. 1294, Springer, 1997, pp. 342–356. MR1630403 (99b:94033)

[26] D. Hankerson, J. Lòpez Hernandez, and A. Menezes, *Software implementation of elliptic curve cryptography over binary fields*, Cryptographic Hardware and Embedded Systems, CHES'00, Lecture Notes in Computer Science, vol. 1965, Springer, 2000, pp. 1–24.

[27] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to elliptic curve cryptography*, Springer, 2004. MR2054891 (2005c:94049)

[28] K. Itoh, M. Takenaka, N. Torii, S. Temma, and Y. Kurihara, *Fast implementation of public-key cryptography on a DSP TMS320C6201*, Cryptographic Hardware and Embedded Systems, CHES'99, Lecture Notes in Computer Science, vol. 1717, Springer, 1999, pp. 61 – 72.

[29] T. Izu, B. Möller, and T. Takagi, *Improved elliptic curve multiplication methods resistant against side channel attacks*, Progress in Cryptology, INDOCRYPT'02, Lecture Notes in Computer Science, vol. 2551, Springer, 2002, pp. 269–313.

[30] T. Izu and T. Takagi, *A fast parallel elliptic curve multiplication resistant against side channel attacks*, Public Key Cryptography, PKC'02, Lecture Notes in Computer Science, vol. 2274, Springer, 2002, pp. 280–296.

[31] ———, *Fast elliptic curve multiplications resistant against side channel attacks*, IEICE Transactions Fundamentals **E88-A** (2005), no. 1, 161–171.

[32] M. Joye and C. Tymen, *Protections against differential analysis for elliptic curve cryptography – an algebraic approach*, Cryptographic Hardware and Embedded Systems, CHES'01, Lecture Notes in Computer Science, vol. 2162, Springer, 2001, pp. 377 – 390. MR1946618 (2003k:94031)

[33] N. Koblitz, *Elliptic curve cryptosystems*, Mathematics of Computation **48** (1987), no. 177, 203–209. MR866109 (88b:94017)

[34] P. Kocher, J. Jaffe, and B. Jun, *Differential power analysis*, Advances in Cryptology, CRYPTO'99, Lecture Notes in Computer Science, vol. 1666, Springer, 1999, pp. 388–397.

[35] P. C. Kocher, *Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems*, Advances in Cryptology, CRYPTO'96, Lecture Notes in Computer Science, vol. 1109, Springer, 1996, pp. 104–113.

[36] J. Lopez and R. Dahab, *An improvement of the Guajardo-Paar method for multiplication on non-supersingular elliptic curves*, Proceedings of the XVIII International Conference of the Chilean Society of Computer Science, SCCC'98, 1998, pp. 91–95.

[37] K. Mahler, *On a special functional equation*, Journal of the London Mathematical Society **s1-15** (1940), no. 2, 115–123. MR0002921 (2:133e)

[38] V. S. Miller, *Uses of elliptic curves in cryptography*, Advances in Cryptology, CRYPTO'85, Lecture Notes in Computer Science, vol. 218, Springer, 1986, pp. 417–428. MR851432 (88b:68040)

[39] National Institute of Standards and Technology, *FIPS PUB 186-2: Digital signature standard (DSS)*, National Institute of Standards and Technology, January 2000.

[40] W. B. Pennington, *On Mahler's partition problem*, Annals of Mathematics **57** (1953), no. 3, 531–546. MR0053959 (14:846m)

[41] C. M. Skinner, *On the diophantine equation* $ap^x + bq^y = c + dp^z q^w$, Journal of Number Theory **35** (1990), 194–207. MR1057322 (91h:11021)

[42] J. Solinas, *Generalized mersenne numbers*, Research Report CORR-99-39, Center for Applied Cryptographic Research, University of Waterloo, Waterloo, ON, Canada, 1999.
[43] Certicom Research The SECG group, *SEC 2: Recommended elliptic curve domain parameters*, Standard for Efficient Cryptography, September 2000, http://www.secg.org/.
[44] R. Tijdeman, *On the maximal distance between integers composed of small primes*, Compositio Mathematica **28** (1974), 159–162. MR0345917 (49:10646)
[45] H. S. Wilf, *Generatingfunctionology*, 2nd ed., Academic Press Inc., 1994. MR1277813 (95a:05002)

Department of Mathematics, Centre for Information Security and Cryptography, University of Calgary, 2500 University drive N.W., Calgary, AB, T2N 1N4, Canada
E-mail address: dimitrov@vlsi.enel.ucalgary.ca

Department of Mathematics, Centre for Information Security and Cryptography, University of Calgary, 2500 University drive N.W., Calgary, AB, T2N 1N4, Canada
Current address: LIRMM, University Montpellier 2, CNRS, 161 rue Ada, 34392 Montpellier, France
E-mail address: Laurent.Imbert@lirmm.fr

Department of Mathematics, Centre for Information Security and Cryptography, University of Calgary, 2500 University drive N.W., Calgary, AB, T2N 1N4, Canada
E-mail address: pradeep@math.ucalgary.ca