On the Enumeration of Double-Base Chains with Applications to Elliptic Curve Cryptography

Christophe Doche

Department of Computing
Macquarie University, Australia
christophe.doche@mq.edu.au.

Abstract. The Double-Base Number System (DBNS) uses two bases, 2 and 3, in order to represent any integer n. A Double-Base Chain (DBC) is a special case of a DBNS expansion. DBCs have been introduced to speed up the scalar multiplication $[n]P$ on certain families of elliptic curves used in cryptography. In this context, our contributions are twofold. First, given integers n, a, and b, we outline a recursive algorithm to compute the number of different DBCs with a leading factor dividing 2^a3^b and representing n. A simple modification of the algorithm allows to determine the number of DBCs with a specified length as well as the actual expansions. In turn, this gives rise to a method to compute an optimal DBC representing n, i.e. an expansion with minimal length. Our implementation is able to return an optimal expansion for most integers up to 2^{60} bits in a few minutes. Second, we introduce an original and potentially more efficient approach to compute a random scalar multiplication $[n]P$, based on the concept of controlled DBC. Instead of generating a random integer n and then trying to find an optimal, or at least a short DBC to represent it, we propose to directly generate n as a random DBC with a chosen leading factor 2^a3^b and length ℓ. To inform the selection of those parameters, in particular ℓ, which drives the trade-off between the efficiency and the security of the underlying cryptosystem, we enumerate the total number of DBCs having a given leading factor 2^a3^b and a certain length ℓ. The comparison between this total number of DBCs and the total number of integers that we wish to represent a priori provides some guidance regarding the selection of suitable parameters. Experiments indicate that our new Near Optimal Controlled DBC approach provides a speedup of at least 10% with respect to the NAF for sizes from 192 to 512 bits. Computations involve elliptic curves defined over $\mathbb{F}_p$, using the Inverted Edwards coordinate system and state of the art scalar multiplication techniques.

Keywords. Double-base number system, elliptic curve cryptography.

1 Introduction

1.1 Elliptic Curve Cryptography

An *elliptic curve* E defined over a field K is a nonsingular projective plane cubic together with a point with coordinates in K. For cryptographic applications, the field K is always finite. In practice, it is a large prime field $\mathbb{F}_p$ or a binary field $\mathbb{F}_{2^a}$. We refer to [23] for a mathematical presentation of elliptic curves and to [1, 16] for a discussion focused on cryptographic applications.
There are different ways to represent the curve E, in particular with a Weierstraß equation or in Edwards form [13, 3]. Irrespective of the representation, the set of points lying on the curve E can be endowed with an abelian group structure. This property has been exploited for about twenty five years to implement public-key cryptographic primitives.
The core operation in elliptic curve cryptography is the *scalar multiplication*, which consists in computing $[n]P$ given a point P on the curve E and some integer n. Several methods exist relying on different representations of n. One of the simplest approach relies on the *non-adjacent form* (NAF) [20, 19], which allows to compute $[n]P$ with t doublings and $t/3$ additions on average, where t is the binary length of n. The approach discussed next is more sophisticated and has recently received increasing attention.

1.2 Double-Base Number System

The *Double-Base Number System* (DBNS) was introduced by Dimitrov and Cooklev [5] and later used in the context of elliptic curve cryptography [6]. With this system, an integer n is represented as

$$n = \sum_{i=1}^{\ell} c_i 2^{a_i} 3^{b_i}, \text{ with } c_i \in \{-1, 1\}. \tag{1}$$

This representation is highly redundant and an expansion can easily be found with a greedy-type approach. The principle is to find at each step the best approximation of a given integer in terms of a $\{2,3\}$-*integer*, *i.e.* an integer of the form $2^a 3^b$. Then compute the difference and reapply the process until we reach zero.

Example 1 *Applying this approach to* $n = 542788$*, we find that*

$$542788 = 2^8 3^7 - 2^3 3^7 + 2^4 3^3 - 2.3^2 - 2.$$

In [7], Dimitrov *et al.* show that for any integer n, this greedy approach returns a DBNS expansion of n having at most $O\big(\frac{\log n}{\log\log n}\big)$ terms. However, in general this system is not well suited for scalar multiplications. For instance, in order to compute $[542788]P$ from the DBNS expansion given in Example 1, it seems that we need more than 8 doublings and 7 triplings unless we can use extra storage to keep certain intermediate results. But, if we are lucky enough that the terms in the expansion can be ordered in such a way that their powers of 2 and 3 are both decreasing, then it becomes trivial to obtain $[n]P$.

1.3 Double-Base Chain

The concept of *Double-Base Chain* (DBC), introduced in [6], corresponds to an expansion of the form

$$\sum_{i=1}^{\ell} c_i 2^{a_i} 3^{b_i}, \text{ with } c_i \in \{-1, 1\} \tag{2}$$

$$\text{such that } \; a_1 \geqslant a_2 \geqslant \cdots \geqslant a_\ell \; \text{ and } \; b_1 \geqslant b_2 \geqslant \cdots \geqslant b_\ell. \tag{3}$$

Equivalently, (3) means that $2^{a_\ell} 3^{b_\ell} \mid \cdots \mid 2^{a_2} 3^{b_2} \mid 2^{a_1} 3^{b_1}$. It guarantees that exactly a_ℓ doublings, b_ℓ triplings, $\ell - 1$ additions, and at most two variables are sufficient to compute $[n]P$. It is straightforward to adapt the greedy algorithm to return a DBC.

Example 2 *A modified greedy algorithm returns the following DBC*

$$542788 = 2^{14}3^3 + 2^{12}3^3 - 2^{10}3^2 - 2^{10} + 2^6 + 2^2.$$

The DBC expansion returned by the greedy approach is always at least as long than its DBNS counterpart. Furthermore, it has been shown in [18] that for any size t, there exists a t-bit integer n such that any DBC representing n needs at least $\Omega(t)$ terms. But the DBC has the advantage to offer a much more direct and easy way to compute a scalar multiplication. The most natural approach is probably to proceed from right-to-left. With this method, each term $2^{a_i}3^{b_i}$ is computed individually and all the terms are added together. This can be implemented using two variables. The left-to-right method, which can be seen as a Horner-like scheme, needs only one variable. Simply initialize it with $[c_1 2^{a_1 - a_2} 3^{b_1 - b_2}]P$, then add $c_2 P$ and apply $[2^{a_2 - a_3} 3^{b_2 - b_3}]$ to the result. Repeating this process eventually gives $[n]P$, as illustrated with the chain of Example 2

$$[542788]P = [2^2]\big([2^4]\big([2^4]\big([3^2]\big([2^2 3]([2^2]P + P) - P\big) - P\big) + P\big) + P\big).$$

Note that there are other methods to compute a DBC, see for instance a tree-based algorithm developed in [9]. There exist also several variants and generalizations of the DBC. For instance, the extended DBC [10] relies on nontrivial coefficients and precomputed points in order to obtain shorter chains. There is also a notion of joint DBC [11, 12] for double scalar multiplications of the form $[n]P + [m]Q$. Next we are interested to find the best possible chains for a given integer. To this end, we introduce the following.

Definition 1 *We call the largest $\{2,3\}$-integer of a DBC chain in absolute value, i.e. $2^{a_1}3^{b_1}$ in (2), the* leading factor *of the chain. It encapsulates the total number of doublings and of triplings necessary to compute $[n]P$.*

Among all the different DBCs with a leading factor dividing $2^a 3^b$ and representing n, the DBCs with minimal length play a special role as they minimize the number of additions required to compute $[n]P$. This observation gives rise to the following definition.

Definition 2 *Given integers a, b, and n, a DBC with a leading factor dividing 2^a3^b and representing n is said to be* optimal for n, *if its length ℓ is minimal across all the DBCs with leading factor dividing 2^a3^b and representing n.*

Remark 1 *For the purpose of this study, we slightly modify the definitions of a Double-Base expansion and of a DBC so that we can precisely and meaningfully enumerate them. Concretely, we require that each term $2^{a_i}3^{b_i}$ appears at most once in any expansion or chain. In practice, expansions always fulfill this property. Also, this requirement is not a real constraint since $2^{a_i}3^{b_i} + 2^{a_i}3^{b_i} = 2^{a_i+1}3^{b_i}$. From now on, when we use the terms double-base expansion or DBC, this restriction is implied.*

Definition 3 *An* unsigned Double-Base Chain *is a DBC of the form (2) such that all the coefficients c_i's are equal to* 1 *and satisfying (3).*

Some properties of the set containing all the unsigned DBCs of a given integer n, in particular its structure and cardinality, are studied in [17]. Next, we investigate the number of signed DBCs representing a given integer.

2 Enumerating DBCs Representing a Given Integer

2.1 Partition Problem

Given an integer n, the number $p(n)$ of partitions of n of the form

$$n = d_k + \cdots + d_2 + d_1 \quad \text{with} \quad d_1 \mid d_2 \mid \cdots \mid d_k$$

is studied by Erdős and Loxton in [14]. The authors also introduce $p_1(n)$ as the number of partitions of n of the form $n = d_k + \cdots + d_2 + 1$ with $d_1 \mid d_2 \mid \cdots \mid d_k$. They observe that $p(n) = p_1(n) + p_1(n+1)$ and that

$$p_1(n) = \sum_{d \mid n-1, d>1} p_1\left(\frac{n-1}{d}\right).$$

2.2 Enumerating DBCs

Mimicking their approach, we introduce $q(a, b, n)$, the number of signed partitions of n of the form

$$n = d_k \pm d_{k-1} \pm \cdots \pm d_2 \pm d_1 \quad \text{with} \quad d_1 \mid d_2 \mid \cdots \mid d_k \mid 2^a3^b.$$

Clearly, $q(a, b, n)$ corresponds to the number of DBCs with a leading factor dividing 2^a3^b and representing n. Note that in the signed version, it is necessary to take into account a and b, the largest powers of 2 and 3. Indeed, we observe that $1 = 2^k - \sum_{i=0}^{k-1} 2^i$ for any $k > 0$. This shows that the number of signed representations of any integer is infinite. Obviously, the problem disappears when

we bound the leading factor of an expansion by $2^a 3^b$. Similarly, we introduce $q_1(a,b,n)$ as the number of partitions of n of the form

$$n = d_k \pm d_{k-1} \pm \cdots \pm d_2 + 1 \quad \text{with} \quad d_2 \mid \cdots \mid d_k \mid 2^a 3^b$$

and $q_{\bar{1}}(a,b,n)$ as the number of partitions of n of the form

$$n = d_k \pm d_{k-1} \pm \cdots \pm d_2 - 1 \quad \text{with} \quad d_2 \mid \cdots \mid d_k \mid 2^a 3^b.$$

In the following, we denote the valuation of u at 2 and 3 by $\mathrm{val}_2(u)$ and $\mathrm{val}_3(u)$, respectively.

Proposition 1 *We have*

1. $q(a,b,n) = q_1(a,b,n) + q_{\bar{1}}(a,b,n) + q_{\bar{1}}(a,b,n-1)$.
2. $$q_1(a,b,n) = \sum_{\substack{d \mid \gcd(n-1, 2^a 3^b) \\ d>1}} q_1\left(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \frac{n-1}{d}\right) + \sum_{\substack{d \mid \gcd(n-1, 2^a 3^b) \\ d>1}} q_{\bar{1}}\left(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \frac{n-1}{d}\right)$$
3. $$q_{\bar{1}}(a,b,n) = \sum_{\substack{d \mid \gcd(n+1, 2^a 3^b) \\ d>1}} q_1\left(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \frac{n+1}{d}\right) + \sum_{\substack{d \mid \gcd(n+1, 2^a 3^b) \\ d>1}} q_{\bar{1}}\left(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \frac{n+1}{d}\right).$$
4. $q_1(a,b,1) = 1$, *if* $a \geqslant 0$ *and* $b \geqslant 0$, *and* $q_1(a,b,1) = 0$ *otherwise.*
5. $q_{\bar{1}}(a,b,1) = a$, *if* $a \geqslant 0$ *and* $b \geqslant 0$, *and* $q_{\bar{1}}(a,b,1) = 0$ *otherwise.*

Proof.

1. We observe that any DBC representing n must end by 1, -1, or a term that is a nontrivial divisor of the leading factor. These three sets form a partition of all the DBCs representing n. By definition, the cardinality of the first two sets is $q_1(a,b,n)$ and $q_{\bar{1}}(a,b,n)$. There exists a bijection between this last set and the set of DBCs representing $n-1$ ending with -1. Note that we could also compute $q(a,b,n)$ as $q_1(a,b,n) + q_{\bar{1}}(a,b,n) + q_1(a,b,n+1)$.
2. Let us consider a DBC with a leading factor dividing $2^a 3^b$, ending with 1, and representing n. Then this DBC can be written $\sum_i c_i 2^{a_i} 3^{b_i} \pm d + 1$ where $d > 1$ and $d \mid 2^{a_i} 3^{b_i}$ for all i. If we denote $\alpha = \mathrm{val}_2(d)$ and $\beta = \mathrm{val}_3(d)$, we see that

the chain $\sum_i c_i 2^{a_i-\alpha}3^{b_i-\beta} \pm 1$ represents $(n-1)/d$. We note that its leading factor must divide $2^{a-\alpha}3^{b-\beta}$ and it ends by 1 or -1. Also, by construction, the factor d is a divisor of $n-1$ and of 2^a3^b. Reciprocally, take $d = 2^\alpha 3^\beta$ a common divisor of $n-1$ and 2^a3^b. Then for any DBC with a leading factor dividing $2^{a-\alpha}3^{b-\beta}$ and representing $(n-1)/d$, it corresponds a unique DBC with a leading factor dividing 2^a3^b, finishing with 1 and representing n.

3. The proof is similar to 2., except that we need to consider DBCs of the form $\sum_i c_i 2^{a_i}3^{b_i} \pm d - 1$.
4. We assume that each term $2^{a_i}3^{b_i}$ appears at most once, cf Remark 1. With this constraint in mind, it is easy to check that there is a unique DBC ending with 1 and representing 1, namely the chain 1.
5. Regarding the DBCs representing 1 and ending with -1, we note that for any $k > 0$, we have $2^k - \sum_{i=0}^{k-1} 2^i = 1$. In particular, the previous formula for $k = 1$ up to a gives rise to a total number of a different DBCs with a leading factor dividing 2^a3^b, ending with -1, and representing 1. It is easy to see that there is no other solution. This shows that $q_{\bar{1}}(a, b, 1) = a$, when $a \geqslant 0$ and $b \geqslant 0$. □

Using Proposition 1, it is possible to compute $q(a, b, n)$ recursively, for any tuple (a, b, n).

Example 3 *We have* $q(14, 5, 542788) = 2092690$. *In other words, there are* 2092690 *different DBCs with a leading factor dividing* $2^{14}3^{10}$ *and representing* 542788.

Remark 2 *The approach is highly recursive but precomputing small values can greatly speed up computations. For instance, precomputing* $q_1(a, b, n)$ *and* $q_{\bar{1}}(a, b, n)$ *for all* $(a, b, n) \in [0, 30] \times [0, 20] \times [1, 1000]$ *allows to deal with numbers of size up to* 30 *bits in a few seconds.*

2.3 Enumerating DBCs of Bounded Length

A simple modification of the algorithm outlined above allows to determine the total number of different DBCs of length less or equal to ℓ with a leading factor dividing 2^a3^b and representing an integer n. Namely, we introduce a new parameter ℓ to keep track of the length of the DBC. It is straightforward to check that

$$q(a, b, \ell, n) = q_1(a, b, \ell, n) + q_{\bar{1}}(a, b, \ell, n) + q_{\bar{1}}(a, b, \ell + 1, n - 1).$$

Additionally, $q_1(a, b, \ell, n)$ and $q_{\bar{1}}(a, b, \ell, n)$ satisfy relations similar to the ones expressed in Proposition 1. For instance,

$$\begin{aligned} q_1(a, b, \ell, n) = &\sum_{\substack{d \mid \gcd(n-1, 2^a3^b) \\ d>1}} q_1\left(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \ell - 1, \frac{n-1}{d}\right) \\ &+ \sum_{\substack{d \mid \gcd(n-1, 2^a3^b) \\ d>1}} q_{\bar{1}}\left(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \ell - 1, \frac{n-1}{d}\right). \end{aligned}$$

Finally, it is easy to see that $q_1(a, b, \ell, 1) = \min(1, \max(0, \ell))$ and $q_{\bar{1}}(a, b, \ell, 1) = \min(a, \max(0, \ell - 1))$. This gives rise to Algorithms 1 and 2.

Algorithm 1. $q_1(a, b, \ell, n)$

INPUT: An integer n and parameters a, b, and ℓ.
OUTPUT: The number of DBCs representing n, ending with 1, having a leading factor dividing $2^a 3^b$, and a length less than or equal to ℓ.

1. **if** $n \leqslant 0$ **or** $a < 0$ **or** $b < 0$ **or** $\ell \leqslant 0$ **then return** 0
2. **else if** $n = 1$ **then**
3. **if** $a \geqslant 0$ **and** $b \geqslant 0$ **then return** $\min(1, \max(0, \ell))$
4. **else return** 0
5. **else if** $n > 1$ **then**
6. $D \leftarrow \gcd(n - 1, 2^a 3^b)$
7. $s \leftarrow 0$
8. **for** each divisor $d > 1$ of D **do**
9. $s \leftarrow s + q_1\big(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \ell - 1, \frac{n-1}{d}\big)$
10. $s \leftarrow s + q_{\bar{1}}\big(a - \mathrm{val}_2(d), b - \mathrm{val}_3(d), \ell - 1, \frac{n-1}{d}\big)$
11. **return** s

Algorithm 2. $q_{\bar{1}}(a, b, \ell, n)$

INPUT: An integer n and parameters a, b, and ℓ.
OUTPUT: The number of DBCs representing n, ending with -1, having a leading factor dividing $2^a 3^b$, and a length less than or equal to ℓ.

1. **if** $n \leqslant 0$ **or** $a < 0$ **or** $b < 0$ **or** $\ell \leqslant 0$ **then return** 0
2. **else if** $n = 1$ **then**
3. **if** $a \geqslant 0$ **and** $b \geqslant 0$ **then return** $\min\big(a, \max(0, \ell - 1)\big)$
4. **else return** 0
5. **else if** $n > 1$ **then**
6. $D \leftarrow \gcd(n + 1, 2^a 3^b)$
7. $s \leftarrow 0$
8. **for** each divisor $d > 1$ of D **do**
9. $s \leftarrow s + q_1\big(a - \mathrm{val}_2(d, 2), b - \mathrm{val}_3(d, 3), \ell - 1, \frac{n+1}{d}\big)$
10. $s \leftarrow s + q_{\bar{1}}\big(a - \mathrm{val}_2(d, 2), b - \mathrm{val}_3(d, 3), \ell - 1, \frac{n+1}{d}\big)$
11. **return** s

Example 4 *Using Algorithms 1 and 2, we see that among the* 2092690 *different DBCs with a leading factor dividing* $2^{14}3^{10}$ *and representing* 542788*, there are three optimal chains of length* 5*,* 81 *chains of length* 6*,* 843 *of length* 7*,* 5005 *of length* 8*,* 19715 *of length* 9*,* 56148 *of length* 10*, and so on. The total number is bounded as for instance, there cannot be a DBC of length greater or equal to* 26 *since the leading factor is at most* $2^{14}3^{10}$*.*

2.4 Optimal DBCs

Using the algorithms described in the previous part, it is simple to determine the optimal length of a DBC representing an integer n with a leading factor dividing 2^a3^b. Simply compute $q(a, b, \ell, n)$ for increasing values of $\ell \geqslant 1$ until a positive cardinality is returned. Also, along with the total number of DBCs, it is possible to return the list of all the actual DBCs representing an integer, by introducing a few simple modifications in the Algorithms 1 and 2. We note that we can further modify Algorithms 1 and 2 so that we compute only the DBCs having a specified length. Also, in case we are only interested in finding an optimal chain for a given integer n, we can implement a simple early abort technique to terminate the search once a DBC of a certain given size has been found. This is possible because these algorithms perform a depth-first search.

Example 5 *Among the three optimal DBCs of length* 5 *with leading factor dividing* $2^{14}3^{10}$ *and representing* 542788*, one is*

$$2^8 3^7 - 2^6 3^5 - 2^6 3^3 + 2^6 3 + 2^2.$$

The running time of this approach is largely driven by the length of the optimal chain that is returned. Typically, it takes a few seconds for chains of length 12 up to a few hours for length 15. In general, it is practical to determine an optimal DBC for integers of size around 60 to 70 bits. See Section 5.1 and Table 1 for details including actual experiments and timings of our `C++` implementation that is available from our homepage, see [8].
So it is clear that computing an optimal DBC for a scalar of size around 200 bits, i.e. the kind of size typically used in elliptic curve cryptography, is completely out of reach with this approach. Instead, we consider another approach to efficiently perform a random scalar multiplication $[n]P$.

3 Enumerating DBCs with Given Parameters

Instead of computing the number of DBCs representing a given integer n, this time we want to count the number of different DBCs with a given leading factor 2^a3^b and a given length ℓ.

Remark 3 *The same problem is straightforward for DBNS expansions. Indeed, we see from (1) that there are* $2^\ell \binom{(a+1)(b+1)}{\ell}$ *different expansions of length* ℓ *and such that* $\max a_i = a$ *and* $\max b_i = b$*. Note that all the expansions are different in this count, but the integers they represent are not necessarily all different.*

It is more involved to determine the number of unsigned DBCs (see Definition 3) and of DBCs with a given leading factor $2^a 3^b$ and a given length ℓ.

3.1 First Properties

Definition 4 *Let $S_\ell(a,b)$ denote the number of unsigned DBCs of length ℓ with a leading factor equal to $2^a 3^b$. Let $T_\ell(a,b)$ denote the number of unsigned DBCs of length ℓ with a leading factor dividing $2^a 3^b$.*

Proposition 2 *Let $\ell \geqslant 1$. We have:*

1. $S_{\ell+1}(a,b) = T_\ell(a,b) - S_\ell(a,b)$.
2. $$T_{\ell+1}(a,b) = \sum_{i=0}^{a}\sum_{j=0}^{b}\big[(a-i+1)(b-j+1)-1\big]S_\ell(i,j).$$
3. *$S_\ell(a,b)$ and $T_\ell(a,b)$ are both symmetrical polynomials.*
4. *The leading terms of $S_\ell(a,b)$ and of $T_\ell(a,b)$ are respectively $\frac{(ab)^{\ell-1}}{(\ell-1)!^2}$ and $\frac{(ab)^\ell}{\ell!^2}$.*

Proof. The first three relations are a simple consequence of the definitions of $S_\ell(a,b)$ and $T_\ell(a,b)$. To prove 4. we first note that $S_\ell(a,b)$ is of degree $2\ell-2$ and $T_\ell(a,b)$ is of degree 2ℓ. This can be shown by induction based on $S_1(a,b)=1$, $T_1(a,b)=(a+1)(b+1)$, and using 1. and 2. We can now prove 4. by induction. The property is true for $S_1(a,b)$ and $T_1(a,b)$. Also, by 1. and given that $S_{\ell+1}(a,b)$ and $T_\ell(a,b)$ are of the same degree, it is clear that their leading terms are equal. So by the induction hypothesis, it is clear that the property holds for $S_\ell(a,b)$, for all $\ell \geqslant 1$. Now assuming it holds for $T_{\ell-1}(a,b)$, let us show that it holds for $T_\ell(a,b)$. Using the induction hypothesis, we observe that the leading term of

$$T_\ell(a,b) = \sum_{i=0}^{a}\sum_{j=0}^{b}\big[(a-i+1)(b-j+1)-1\big]S_{\ell-1}(i,j)$$

is equal to the leading term of

$$\frac{1}{(\ell-2)!^2}\sum_{i=0}^{a}\sum_{j=0}^{b}(a-i)(b-j)(ij)^{\ell-2}.$$

Next, we note that the leading term of $\displaystyle\sum_{i=0}^{a} i^k$ is $\frac{1}{(k+1)}a^{k+1}$. We deduce that the leading term of

$$\sum_{i=0}^{a}\sum_{j=0}^{b}(a-i)(b-j)(ij)^{\ell-2}$$

is

$$(ab)^\ell\left(\frac{1}{\ell^2}+\frac{1}{(\ell-1)^2}-\frac{2}{\ell(\ell-1)}\right)=\frac{(ab)^\ell}{((\ell-1)\ell)^2}.$$

It follows that the leading term of $T_\ell(a,b)$ is $\frac{(ab)^\ell}{\ell!^2}$, as expected. □

Remark 4 *The number of signed DBCs of length ℓ with a leading factor equal to 2^a3^b and dividing 2^a3^b can be easily deduced from $S_\ell(a,b)$ and $T_\ell(a,b)$, respectively. Namely, it is only necessary to multiply by a factor 2^ℓ. Note that all those DBCs represent positive and negative integers. But it is easy to see that the sign of the integer represented by a chain corresponds to the sign of the largest term of the chain. See Lemma 1 in Section 4. So if we are only interested in DBCs representing positive values, the multiplication factor between unsigned and signed DBCs should be $2^{\ell-1}$.*

3.2 Explicit Computations

Recall that $S_1(a,b) = 1$ and $T_1(a,b) = (a+1)(b+1)$. Proposition 2 can then be used to explicitly determine the polynomials S_ℓ and T_ℓ of rank $\ell \geqslant 2$ recursively. For instance, we have $S_2(a,b) = ab + a + b$ from 1. and $T_2(a,b) = \frac{1}{4}(ab + 2a + 2b)T_1(a,b)$ using 2. We can then compute $S_3(a,b)$, then $T_3(a,b)$, and so on.
In practice, however, the complexity of those polynomials rapidly grows with ℓ and it becomes quickly impossible to compute them formally. Fortunately, we are only interested by the value of these polynomials at a specific pair (a_0, b_0). This can be done very efficiently using some precomputations and Lagrange interpolation. Since S_ℓ is a polynomial of degree $\ell - 1$ in a and $\ell - 1$ in b, it is enough to know the value of S_ℓ at ℓ^2 pairs (a_i, b_j), for $(i,j) \in [1,\ell]^2$ in order to compute $S_\ell(a_0, b_0)$. First, for each $i \in [1,\ell]$, we interpolate with respect to the second coordinate based on the values $S_\ell(a_i, b_j)$, for $j \in [1,\ell]$. We obtain ℓ polynomials in variable b. Specializing those polynomials at b_0, we obtain ℓ values and a second Lagrange interpolation, followed by a specialization at a_0 gives $S_\ell(a_0, b_0)$. Note that in order to find the Lagrange polynomial $P(x)$ interpolating the points $(x_k, f(x_k))$, it is faster, in our case, to use the following formulas

$$P(x) = w(x) \sum_{k=1}^{\ell} \frac{f(x_k)}{w'(x_k)(x - x_k)} \quad \text{with} \quad w(x) = \prod_{j=1}^{\ell} (x - x_j)$$

rather than a more classical approach such as Aitken method. For each length ℓ, the ℓ^2 precomputed values can be obtained with Proposition 2. There is a similar approach for evaluating T_ℓ at (a_0, b_0).
Our PARI/GP implementation allows to deal efficiently with length ℓ up to 150. For most pairs (a,b), it takes less than 50ms to evaluate $S_\ell(a,b)$ or $T_\ell(a,b)$. In any case, at most a few seconds are necessary. The corresponding precomputations require about 45 MB. Only 10 MB are necessary to handle lengths ℓ up to 100. See [8] to access the actual implementation.

3.3 Generalization to Multi-Base Chains

It is easy to generalize the previous results to Multi-Base Chains. Let $p_1, \ldots, p_k$ be k pairwise coprime bases. A *Multi-Base Chain* (MBC) allows to represent a

positive integer n as

$$n = \sum_{i=1}^{\ell} c_i p_1^{a_{1,i}} \dots p_k^{a_{k,i}}, \quad \text{with} \quad c_1 = 1 \quad \text{and} \quad c_i = \pm 1, \quad \text{for} \quad i > 1$$

and $a_{j,1} \geqslant a_{j,2} \geqslant \cdots \geqslant a_{j,\ell}$, for all $j \in [1, k]$. An *unsigned Multi-Base Chain* is similar to a Multi-Base Chain except that all the c_i's are equal to 1. In any case, we assume that the term $p_1^{a_{1,i}} \dots p_k^{a_{k,i}}$ appears at most once in any expansion.

Definition 5 *Let $\underline{a}$ denote the vector $(a_1, \dots, a_k)$ and let $S_\ell(\underline{a})$ be the number of unsigned Multi-Base Chains of length ℓ satisfying $a_{j,1} = a_j$, for all j. Also, let $T_\ell(\underline{a})$ be the number of unsigned Multi-Base Chains of length ℓ satisfying $a_{j,1} \leqslant a_j$, for all j.*

The following Proposition is a simple generalization of Proposition 2. The proof is also similar.

Proposition 3 *Let $\ell \geqslant 1$. We have*

1. $S_{\ell+1}(\underline{a}) = T_\ell(\underline{a}) - S_\ell(\underline{a})$.
2. $$T_{\ell+1}(\underline{a}) = \sum_{i_1=0}^{a_1} \cdots \sum_{i_k=0}^{a_k} \left[\prod_{j=1}^{k} (a_j - i_j + 1) - 1 \right] S_\ell(\underline{a}).$$
3. $S_\ell(\underline{a})$ *and* $T_\ell(\underline{a})$ *are both symmetrical polynomials.*
4. *The leading terms of* $S_\ell(\underline{a})$ *and of* $T_\ell(\underline{a})$ *are respectively* $\frac{(a_1 \dots a_k)^{\ell-1}}{(\ell-1)!^k}$ *and* $\frac{(a_1 \dots a_k)^{\ell}}{\ell!^k}$.

Remark 5 *Again the number of MBCs of length ℓ with a leading factor equal or dividing $p_1^{a_1,1} \dots p_k^{a_k,1}$ can be easily deduced from $S_\ell(\underline{a})$ or $T_\ell(\underline{a})$. Namely, it is only necessary to multiply by a factor $2^{\ell-1}$.*

Example 6 *For $k = 3$, we have*

$$\begin{aligned}
S_1(\underline{a}) &= (a_1 + 1)(a_2 + 1)(a_3 + 1), \\
T_1(\underline{a}) &= 1, \\
S_2(\underline{a}) &= \frac{1}{8}(a_1 a_2 a_3 + 2a_1 a_2 + 2a_1 a_3 + 2a_2 a_3 + 4a_1 + 4a_2 + 4a_3) S_1(\underline{a}), \\
T_2(\underline{a}) &= a_1 a_2 a_3 + a_1 a_2 + a_1 a_3 + a_2 a_3 + a_1 + a_2 + a_3.
\end{aligned}$$

4 Controlled DBC for Scalar Multiplication

For cryptographic applications, we propose a new way to perform a random scalar multiplication based on the concept of *controlled DBC*. The idea is to directly generate a random DBC expansion instead of choosing a random integer n and then finding a corresponding DBC to represent it.

Definition 6 *Given a leading factor 2^a3^b and a given length ℓ, the* controlled DBC approach *refers to the generation of a DBC expansion*

$$\sum_{i=1}^{\ell} c_i 2^{a_i} 3^{b_i}, \text{ with } c_i \in \{-1, 1\}$$

such that $c_1 = 1$, $a_1 = a$, $b_1 = b$, and whose $\ell - 1$ remaining terms $c_i 2^{a_i} 3^{b_i}$ are selected to satisfy $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_\ell$ and $b_1 \geqslant b_2 \geqslant \cdots \geqslant b_\ell$.

This has two main advantages. Although very efficient, the greedy approach still requires some time to return a DBC. No conversion is necessary with this approach. Furthermore, there is no guarantee that the DBC expansion returned by the greedy approach is optimal. In fact, we have evidence that the greedy method returns a DBC that is far from optimal in general, especially for large integers. See Section 5.2 and Figure 2. By choosing the DBC expansion first, in particular its leading factor as well as its length, we can get closer to the average optimal length. As a result, we can perform a scalar multiplication faster than with the DBC obtained with the greedy approach by saving many additions. This approach raises a few questions, in particular, regarding a suitable selection of the length. For a given size and a given leading factor, it is possible to estimate the length which corresponds heuristically to the average optimal length of a DBC representing integers of that size with that leading factor. See Definition 7 for the notion of Near Optimal Length.
First, let us address the range of the integers that can be represented a priori with a DBC having a leading term equal to 2^a3^b.

4.1 Integer Range

The following result provides an answer.

Lemma 1 *Any DBC with leading factor 2^a3^b belongs to the interval*

$$\left[\frac{3^b+1}{2},\ 2^{a+1}3^b - \frac{3^b+1}{2}\right].$$

It follows that the sign of the integer represented by a DBC with leading factor equal to 2^a3^b is driven by the sign of the coefficient of the leading factor in the DBC.

Proof. It is not difficult to see that the largest integer represented with a DBC having a leading factor equal to 2^a3^b can be constructed with a greedy-type approach. In other words, it is enough to pick the largest available term at each step to end up with the largest possible integer. Starting from 2^a3^b, the next term in the DBC is of the form 2^i3^j with $i \leqslant a$, $j \leqslant b$, and $(i, j) \neq (a, b)$. Assuming that $a \geqslant 1$, clearly, $2^{a-1}3^b$ is the largest possible integer we can pick.

If $a = 0$, then there is no choice but to pick 3^{b-1}. Repeating this argument, we deduce that the largest integer that can be represented is

$$2^a 3^b + \sum_{j=0}^{a-1} 2^j 3^b + \sum_{j=0}^{b-1} 3^j = 2^{a+1} 3^b + \frac{3^b - 1}{2}.$$

Similarly, the smallest integer corresponds to $\frac{3^b+1}{2}$. Finally, it is obvious that if a DBC starts with $-2^a 3^b$, then the integers that can be represented with this DBC belong to the interval

$$\left[-2^{a+1} 3^b + \frac{3^b + 1}{2},\ -\frac{3^b + 1}{2} \right].$$

So integers represented by a DBC starting with $2^a 3^b$ are always positive and those represented by a DBC starting with $-2^a 3^b$ are always negative. □

The work in Section 3 gives the exact cardinality of the set containing all the DBCs with selected parameters. It is then tempting to select a length ℓ giving rise to as many DBCs as there are integers in the interval given in Lemma 1. However, this is ignoring that in general an integer has many different DBCs representations.

4.2 Redundancy and Near Optimal Length

In the controlled DBC approach, we need to be careful in selecting the length ℓ, as generating DBCs that are not long enough could compromise the security of the cryptosystem by severely restricting the number of scalars that can be represented with those chains. What length is then long enough? See Definition 7 for the notion Near Optimal Length addressing this question.

For various leading factors up to $2^{30}3^{10}$ and length between 1 and 12, we have computed the number of different optimal representations of integers having an optimal DBC with this particular leading factor and length. For every selection of parameters, we consider between $10,000$ and 100 such integers. We then compute the average number of optimal DBCs for each length between 1 and 12, taking into account all the possible leading factors. This search was carried out with the algorithms developed in Section 2. The data fit an exponential regression of the form $y = \exp(0.4717x - 1.1683)$ with $R^2 = 0.9975$, see Figure 1.

To double-check the relevance of this estimate, we investigate DBCs having a leading factor of the form $2^{3\ell}$. We know that in this case the optimal length is ℓ, which corresponds to the NAF. We then compute the number of DBCs with a leading factor equal to 3ℓ and a length equal to ℓ using what we have done in Section 3. Dividing this quantity by $2^{3\ell+1}$, which corresponds approximately to the number of integers that can be represented a priori, we should obtain an estimate of the average number of optimal DBCs representing an integer, i.e. something close to $\exp(0.4717\ell - 1.1683)$. For all $\ell \in [10, 100]$, the ratio between these two quantities lies in the interval $[0.0974, 3.384]$. This tends to confirm the

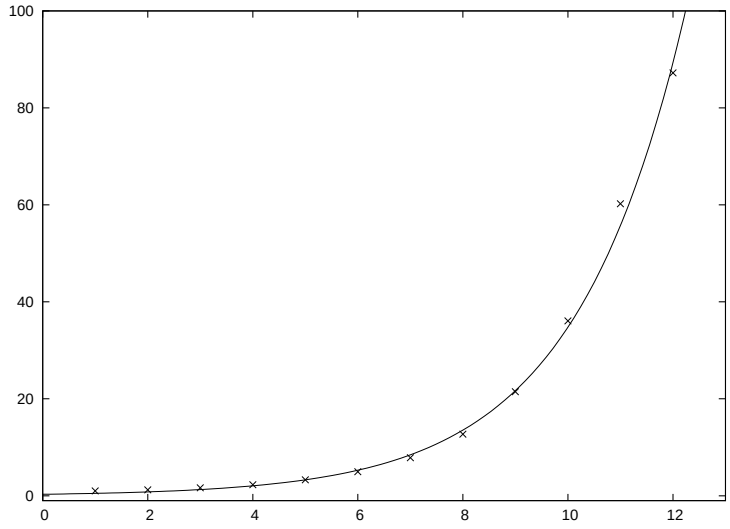

Fig. 1. Curve $\exp(0.4717x - 1.1683)$ fitting the experimental data

relevance of our estimate, at least for relatively small values of b (0 in this case).

Definition 7 *For a leading factor equal to* $2^a 3^b \simeq 2^t$*, the* Near Optimal Length *corresponds to the integer value* ℓ *minimizing*

$$\left| 2^{\ell-1} S_\ell(a,b) - 2^t \lceil \exp(0.4717\ell - 1.1683) \rceil \right|.$$

Indeed, we expect that the average number of different DBC expansions of length ℓ representing the same integer is close to $\lceil \exp(0.4717\ell - 1.1683) \rceil$. Heuristically, we also expect that this redundancy factor multiplied by 2^t is equal to $2^{\ell-1} S_\ell(a,b)$ for the average optimal length ℓ.

4.3 Applications to Elliptic Curve Cryptography

For a chosen coordinate system representing a point on an elliptic curve and the corresponding complexities of a doubling, a tripling, and a mixed addition, it is possible to determine the optimal parameters, i.e. leading factor $2^a 3^b$ and length ℓ, which minimize the overall cost of a scalar multiplication with that particular coordinate system, without compromising the security of the system.

Definition 8 *For a given coordinate system and a bit size* t*, the* Near Optimal Controlled (NOC) DBC *method refers to the generation of a Controlled DBC with Near Optimal Length, which minimizes the costs of a scalar multiplication.*

In practice, we first select the bit size t, then consider all the possible pairs (a, b) such that $2^a 3^b \simeq 2^t$. For each pair (a, b), we work out the corresponding Near Optimal Length ℓ. Then we can compute the overall complexity to perform a scalar multiplication based on a controlled DBC with leading factor $2^a 3^b$ and length ℓ. It is then a matter of selecting the pair (a, b) corresponding to the lowest complexity overall. See Figure 2 and Tables 2 and 3.

5 Experiments

We have implemented the work described in Section 3 in `C++` using `NTL 6.0.0` [21] built on top of `GMP 5.1.2` [15]. The approach described in Section 4 is implemented in `PARI/GP 2.7.1` [22]. See [8] to access the actual `C++` and `PARI/GP` implementations. All the programs are executed on a `quad core i7-2620` at `2.70Ghz`.

5.1 Optimal DBC Search

Given an integer n, the running time of Algorithms 1 and 2 to find the optimal length of a DBC representing n with a leading factor dividing $2^a 3^b$ is largely driven by the length ℓ of this optimal expansion. It usually takes several minutes for DBCs of length 14. See Table 1.

Length ℓ	9	10	11	12	13	14
Time in s	1.08	5.21	28.52	66.38	214.80	757.91

Table 1. Average running times to find an optimal DBC of length ℓ

Considering integers related to π, the longest optimal DBC that we have been able to compute corresponds to the 69-bit integer 314159265358979323846 with a leading factor equal to $2^{38}3^{19}$ and length 18. It takes about 22 hours to show that there is no expansion of length less than or equal to 17 and it takes a bit less than six hours to return an optimal expansion of length 18 with the early abort technique mentioned in Section 2.4. Interestingly, the greedy approach returns a DBC of length 18 so that we can obtain an optimal DBC in no time, in that particular case.

5.2 Comparison Between Greedy and Near Optimal Length

We have run some tests for sizes 192, 256, 320, 384, 448, and 512 bits. For each size t, we have considered various leading factors of the form $2^a 3^b \simeq 2^t$. More precisely, we fix a between $t/2$ and t, compute the corresponding b, and then compute the average length of the DBCs returned by the greedy method

for $5,000$ random integers. We also compute the Near Optimal Length of a DBC with leading factor equal to 2^a3^b, see Definition 7 in Section 4.2. Our computations indicate that considering controlled DBCs that are 20 to 30% shorter than those returned by the greedy algorithm should not significantly reduce the set of integers that can be represented. See Figure 2, which shows a comparison for size $t = 320$. The x-coordinate axis corresponds to a between 160 and 315. The y-coordinate axis corresponds to the average length of the DBCs.

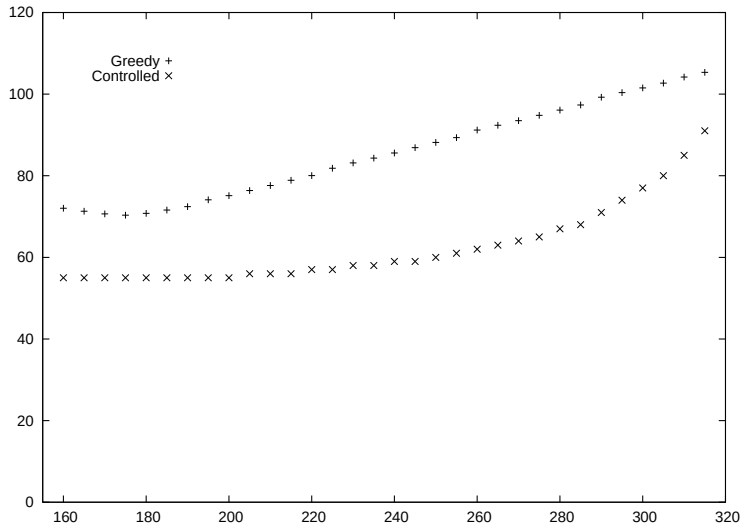

Fig. 2. Comparison between the average length of the DBCs returned by the greedy method and the Near Optimal Length for size 320 bits

5.3 Scalar Multiplication

In this part, we are interested in the potential savings introduced by our new scalar multiplication framework described in Section 4, in particular using the notion of Near Optimal Controlled DBC, see Definition 8.
In the following, we select the Inverted Edwards coordinate system [4] for a curve defined over a large prime field $\mathbb{F}_p$. This system offers a very fast doubling and a reasonably cheap mixed addition and tripling [2]. More precisely, the respective costs of a doubling, mixed addition, and tripling are 3M + 4S, 8M + S, and 9M + 4S, where M and S stand respectively for a multiplication and a squaring in $\mathbb{F}_p$. To allow easy comparisons and as customary, we assume that S = 0.8M.
Until now, computing $[n]P$ for a random n, in Inverted Edwards coordinates with a DBC was not really worth it. Indeed, only the greedy method was fast

enough to return a DBC in a reasonable time and the overall savings obtained were marginal with respect to the NAF, whose recoding can be achieved much faster. With the NAF, we perform t doublings and approximately $t/3$ mixed additions in order to compute $[n]P$ where n is of size t bits.
In Table 2, we display the parameters, costs, and speedups corresponding to different methods, for various sizes between 192 and 512. First, we consider the Near Optimal Controlled DBC approach, then the greedy method, and finally the NAF. LF stands for the leading factor and ℓ is the length of the corresponding expansion. The costs are expressed in terms of the number of multiplications needed to compute $[n]P$ but do not take into account the effort to produce each expansion. Regarding the NOC DBC, we determine for each size the optimal leading factor 2^a3^b and corresponding Near Optimal Length ℓ minimizing the costs of the scalar multiplication, as explained in Section 5.2. Similarly, for the greedy approach we rely on the computations of Section 5.2.

	NOC			Greedy			NAF			Speedups	
Size	LF	ℓ	Cost	LF_1	ℓ_1	Cost_1	LF_2	ℓ_2	Cost_2	S_1	S_2
192	$2^{151}3^{26}$	37	1570.20	$2^{116}3^{48}$	44.63	1688.74	2^{192}	64.00	1744.80	7.02%	10.01%
256	$2^{198}3^{37}$	48	2092.60	$2^{153}3^{65}$	58.73	2249.62	2^{256}	85.33	2329.33	6.98%	10.16%
320	$2^{260}3^{38}$	62	2612.40	$2^{180}3^{89}$	70.80	2816.04	2^{320}	106.67	2913.87	7.23%	10.35%
384	$2^{297}3^{55}$	71	3128.40	$2^{217}3^{106}$	84.74	3375.51	2^{384}	128.00	3498.40	7.32%	10.58%
448	$2^{369}3^{50}$	86	3645.80	$2^{254}3^{123}$	98.73	3935.42	2^{448}	149.33	4082.93	7.36%	10.71%
512	$2^{406}3^{67}$	95	4161.80	$2^{286}3^{143}$	112.07	4495.22	2^{512}	170.67	4667.47	7.42%	10.83%

Table 2. Theoretical comparison between NOC, greedy, and NAF methods

To validate these theoretical results, we have developed an implementation in `C++` using `NTL 6.0.0` [21] built on top of `GMP 5.1.2` [15]. The program is compiled and executed on a `quad core i7-2620` at `2.70Ghz`. For $t = 192$, 256, 320, 384, 448, and 512, we generate a random prime number p_t having bit size t. For each p_t, we then create a total of 100 curves of the form

$$E : x^2 + y^2 = c^2(1 + dx^2y^2)$$

defined over $\mathbb{F}_{p_t}$, where c and d are small random values. For each curve E, we determine a random point P on E. Next, we select 100 random scalars in the interval $[0, p_t - 1]$. The corresponding NAF and greedy DBC expansions with a leading factor as in Table 2 are then computed for each scalar. For each t, we also directly create 100 random DBC expansions of length ℓ returned by the controlled DBC approach. Since we only want to assess the efficiency of the scalar multiplication, our only constraint is to generate a DBC with the specified

length ℓ and leading factor as in Table 2. In practice, the method used to generate the expansions should be thoroughly designed and analyzed to ensure that the integers that are produced are uniformly distributed. This will be the object of some future work.
The experiments confirm the theoretical complexity analysis provided in Table 2, especially regarding S_2. The discrepancy between the theoretical and the experimental values of S_1 can be explained by a ratio M/S that is closer to 0.95 in NTL rather than 0.8 as initially assumed.
See Table 3 for actual timings. Note that the respective times necessary to compute the expansions for each method are not counted.

	NOC	Greedy	NAF	Speedups	
Size	Time in ms	Time in ms	Time in ms	S_1	S_2
192	0.822	0.861	0.939	4.58%	12.49%
256	1.444	1.531	1.642	5.73%	12.08%
320	2.446	2.584	2.766	5.35%	11.58%
384	3.511	3.703	3.960	5.17%	11.33%
448	5.088	5.392	5.729	5.65%	11.20%
512	6.569	6.982	7.408	5.91%	11.32%

Table 3. Comparison of running times of NOC, greedy, and NAF methods

6 Conclusion and Future Work

In this article, we have introduced new techniques to compute an optimal DBC representing a given integer. The algorithms that we have developed allow to tackle sizes of around 60 to 70 bits in a reasonable time.
We have also developed a new way to produce DBCs, namely the controlled DBC approach, which allows to directly create a DBC expansion instead of selecting an integer and converting it to DBC format. This idea raises a few issues regarding the choice of parameters, in particular the length of the expansion.
We use heuristics to estimate the average length of an optimal DBC expansion representing an integer of a certain bit size with a given leading factor. This estimate is based on the enumeration of the DBCs with given parameters and the expected number of different optimal DBCs representing the same integer.
For a given size and coordinate system, these heuristics allow to determine the optimal parameters, i.e. leading factor and length, which minimize the overall costs of a scalar multiplication of that size. This gives rise to the concept of Near Optimal Controlled DBC. Our experiments show speedups for this approach in

excess of 10% over the NAF and of about 5% over the greedy method. Those computations do not take into account the time necessary to produce the expansions. So the interest of this new method is even greater as the expansions do not have to be computed unlike for the greedy and NAF methods.
In future, we aim at studying the redundancy of DBCs more accurately in order to find an upper bound on the number of DBCs of a certain length, representing an integer of a certain size.
Also, given a leading factor, once we have an estimate of the length of the expansion, the problem remains to actually create random controlled DBC expansions, such that the corresponding integers are uniformly distributed.
This question is not addressed in the present paper and will be the object of some future work.

7 Acknowledgments

The author would like to thank the GAATI group and the Department of Mathematics of the University of French Polynesia for hosting him while this research was carried out.

References

Numbers in curly brackets at the end specify the pages where the citations occur.

1. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, T. Lange, K. Nguyen, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005. {2}
2. D. J. Bernstein and T. Lange. Explicit-formulas database. See `http://www.hyperelliptic.org/EFD/`. {16}
3. D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *Advances in Cryptology – Asiacrypt 2007*, volume 4833 of *Lecture Notes in Comput. Sci.*, pages 29–50, Berlin, 2007. Springer. {2}
4. D. J. Bernstein and T. Lange. Inverted Edwards Coordinates. In *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes – AAECC 2007*, volume 4851 of *Lecture Notes in Comput. Sci.*, pages 20–27, Berlin, 2007. Springer. {16}
5. V. S. Dimitrov and T. Cooklev. Hybrid Algorithm for the Computation of the Matrix Polynomial $I + A + \cdots + A^{N-1}$. *IEEE Trans. on Circuits and Systems*, 42(7):377–380, 1995. {2}
6. V. S. Dimitrov, L. Imbert, and P. K. Mishra. Efficient and Secure Elliptic Curve Point Multiplication Using Double-Base Chains. In *Advances in Cryptology – Asiacrypt 2005*, volume 3788 of *Lecture Notes in Comput. Sci.*, pages 59–78. Springer, 2005. {2, 3}
7. V. S. Dimitrov, G. A. Jullien, and W. C. Miller. An Algorithm for Modular Exponentiation. *Information Processing Letters*, 66(3):155–159, 1998. {2}
8. C. Doche. C++ and PARI/GP implementations to compute optimal and enumerate Double-Base Chains. See `http://www.comp.mq.edu.au/~doche`. {8, 10, 15}

9. C. Doche and L. Habsieger. A Tree-Based Approach for Computing Double-Base Chains. In *Information Security and Privacy, 13th Australasian Conference, ACISP 2008*, volume 5107 of *Lecture Notes in Comput. Sci.*, pages 433–446. Springer, 2008. {3}
10. C. Doche and L. Imbert. Extended Double-Base Number System with applications to Elliptic Curve Cryptography. In *Advances in Cryptology - Proceedings of Indocrypt 2006*, volume 4329 of *Lecture Notes in Comput. Sci.*, pages 335–348, Berlin, 2006. Springer. {3}
11. C. Doche, D. R. Kohel, and F. Sica. Double-Base Number System for Multi-scalar Multiplications. In *Advances in Cryptology – Eurocrypt 2009*, volume 5479 of *Lecture Notes in Comput. Sci.*, pages 502–517. Springer, 2009. {3}
12. C. Doche and D. Sutantyo. New and Improved Methods to Analyze and Compute Double-Scalar Multiplications. *IEEE Trans. Comput.*, 63(1):230–242, 2014. {3}
13. H. M. Edwards. A normal form for elliptic curves. *Bull. Amer. Math. Soc. (N.S.)*, 44(3):393–422 (electronic), 2007. {2}
14. P. Erdős and J. H. Loxton. Some problems in partitio numerorum. *J. Austral. Math. Soc. Ser. A*, 27(3):319–331, 1979. {4}
15. Free Software Foundation. GNU Multiple Precision Library. {15, 17}
16. D. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Berlin, 2003. {2}
17. L. Imbert and F. Philippe. Strictly chained (p, q)-ary partitions. *Contrib. Discrete Math.*, 5(2):119–136, 2010. {4}
18. T. Lou, X. Sun, and C. Tartary. Bounds and Trade-offs for Double-Base Number Systems. *Information Processing Letters*, 111(10):488–493, 2011. {3}
19. F. Morain and J. Olivos. Speeding up the Computations on an Elliptic Curve using Addition-Subtraction Chains. *Inform. Theor. Appl.*, 24:531–543, 1990. {2}
20. G. Reitwiesner. Binary arithmetic. *Adv. Comput.*, 1:231–308, 1962. {2}
21. V. Shoup. NTL: A Library for doing Number Theory. {15, 17}
22. The PARI Group, Bordeaux. *PARI/GP, version 2.7.1*, 2014. {15}
23. L. C. Washington. *Elliptic Curves*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2003. Number theory and cryptography. {2}