A Tree-Based Approach for Computing Double-Base Chains

Christophe Doche[1,⋆] and Laurent Habsieger[2]

[1] Department of Computing
Macquarie University, Australia
doche@ics.mq.edu.au
[2] Institut Camille Jordan, CNRS UMR 5208
Université Lyon 1, 69622 Villeurbanne Cedex, France
Laurent.Habsieger@math.univ-lyon1.fr

Abstract. We introduce a tree-based method to find short Double-Base chains. As compared to the classical greedy approach, this new method is not only simpler to implement and faster, experimentally it also returns shorter chains on average. The complexity analysis shows that the average length of a chain returned by this tree-based approach is $\frac{\log_2 n}{4.6419}$. This tends to suggest that the average length of DB-chains generated by the greedy approach is not $O(\log n/\log\log n)$. We also discuss generalizations of this method, namely to compute Step Multi-Base Representation chains involving more than 2 bases and extended DB-chains having non-trivial coefficients.

Keywords: Double-base number system, scalar multiplication, elliptic curve cryptography.

1 Introduction

In the context of public-key cryptography, elliptic curves have attracted more and more attention since their introduction about twenty years ago by Miller and Koblitz [19, 21]. The main reason is that the only known algorithms to solve the discrete logarithm problem on a well-chosen elliptic curve all have an exponential-time complexity. This is in contrast with the existence of sub-exponential time algorithms to factor integers or to solve discrete logarithm problems in finite fields.

For a general presentation of elliptic curves, we recommend [22]. We refer to the following books [1, 7, 8, 18] for a discussion of elliptic curves in the context of cryptography.

Given a point P on a curve E and a integer n, the operation to compute the point $[n]P$ is called a *scalar multiplication.* It is the most time-consuming operation in many curve-based cryptographic protocols. Not surprisingly, this operation has been the subject of intense research, as indicated by the abundant

⋆ This work was partially supported by ARC Discovery grant DP0881473.

literature on this particular topic. The standard method to efficiently compute such a scalar multiplication is the *double and add*, also known as the *left-to-right* since it scans the bits of n from the left to the right and performs a doubling for each bit, followed by an addition in case the current bit of n is 1. The number of doublings is therefore equal to $\log_2 n$ whereas the number of additions depends on the *density* of the binary representation of n, which is equal to $\frac{1}{2}\log_2 n$, on average. Scalar multiplications being similar to exponentiations, all the techniques used to speed up the computation of x^n can be used to obtain $[n]P$. See [17] for an exhaustive presentation of exponentiation techniques. For instance, we can consider *windowing methods*, which rely on the representation of the scalar in a larger basis. The expansion is then shorter and it includes nontrivial coefficients. As a result, less additions are necessary to compute $[n]P$ but precomputations must be used to take advantage of this approach.

Also, since computing the negative of a point P can be done virtually at no cost, a further gain can be obtained by considering *signed digit representations* of the scalar n, involving negative coefficients and giving a smaller density. This is the principle of the NAF whose density is $\frac{1}{3}\cdot$ Using signed coefficients greater than one leads to window NAF methods having an even smaller density.

Another possibility to accelerate the computation of $[n]P$ is to make use of special *endomorphisms*. An endomorphism of E is a rational map which sends the point at infinity onto itself. Examples of such endomorphisms include $[k]$, the multiplication by k map, for any integer k. We have seen that the double and add method relies on doublings and additions to compute $[n]P$. Other endomorphisms, potentially faster than doublings, could be used to compute scalar multiplications more efficiently. For instance, on Koblitz curves, *cf.* [16], the Frobenius endomorphism ϕ defined by $\phi(x, y) = (x^2, y^2)$ is trivial to compute and for any integer n, the map $[n]$ can always be expressed as $\sum_i d_i\phi^i$, for some d_i's in $\{-1, 0, 1\}$. From this observation, it is possible to devise a very efficient *Frobenius and add* scalar multiplication algorithm which does not require any doubling.

Tripling that sends P on $[3]P$, is also a very natural endomorphism to consider. Using triplings to compute scalar multiplications can be traced back to 2003, *cf.* [9]. For elliptic curves defined over a finite field of large characteristic, Ciet *et al.*, managed to eliminate an inversion to obtain a tripling in affine coordinates with 1 field inversion, 7 multiplications, and 4 squarings, which we abbreviate by $1\text{I} + 7\text{M} + 4\text{S}$. Note that the naïve computation of $[3]P$ as $[2]P + P$ requires $2\text{I} + 4\text{M} + 3\text{S}$. Later Dimitrov *et al.* [14], showed how to compute $[3]P$ with $10\text{M} + 6\text{S}$. Then Doche *et al.* considered a special family of curves for which a tripling can be obtained with as little as $6\text{M} + 6\text{S}$, *cf.* [5, 13]. Recently, Bernstein *et al.* [6] described a tripling on well chosen Edwards curves that needs only $9\text{M} + 4\text{S}$.

New representations are needed to fully take advantage of these efficient tripling maps.

2 Double-Base Number System

In [9], Ciet *et al.* propose a *binary/ternary method* to perform a scalar multiplication by means of doublings, triplings, and additions. Let $v_p(m)$ denotes the *p-adic valuation* of the integer m, then the principle of this method is as follows. Starting from some integer n and a point P, divide n by $2^{v_2(n)}$ and perform $v_2(n)$ doublings, then divide the result by $3^{v_3(n)}$ and perform $v_3(n)$ triplings. At this point, we have some integer m that is coprime to 6. This implies that $m \bmod 6$ must be equal to 1 or 5. Setting $m = m - 1$ or $m = m + 1$ allows to repeat the process at the cost of a subtraction or an addition.

In fact, the binary/ternary method computes an expansion that is a particular case of a more general type of representation, called the *Double-Base Number System*, *DBNS* for short. It was initially introduced by Dimitrov and Cooklev [11] and later used in the context of elliptic curve cryptography [14]. With this system, an integer n is represented as

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

It is not hard to see that there is no unique DBNS representation of n. In fact, this system is highly redundant and among all the possibilities, it seems always possible to find an expansion involving very few terms. Let us be more precise. A greedy approach is used to find such an expansion. Its principle is to find at each step the best approximation of a certain integer (n initially) 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.

Example 1. *Take the integer* $n = 841232$. *We have the sequence of approximations*

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424, \\ 1424 &= 2^1 3^6 - 34, \\ 34 &= 2^2 3^2 - 2. \end{aligned}$$

As a consequence, $841232 = 2^7 3^8 + 2^1 3^6 - 2^2 3^2 + 2^1$.

In [15], Dimitrov *et al.* showed that for any integer n, this greedy approach returns a DBNS expansion of n involving at most $O\big(\frac{\log n}{\log \log n}\big)$ signed $\{2,3\}$-integers.

Even if this class of DBNS is very sparse, it is in general not suitable to compute scalar multiplications. Indeed, we need *at least* $\max a_i$ doublings and $\max b_i$ triplings to compute $[n]P$ using (1). It is easy to perform only $\max a_i$ doublings *or* $\max b_i$ triplings. For that simply order the terms with respect to the powers of 2 *or* to the powers of 3. However, the challenge with this type of DBNS is to attain these two lower bounds *simultaneously.* Now, if by chance the sequences of exponents are simultaneously decreasing, *i.e.* $a_1 \geqslant a_2 \geqslant \cdots \geqslant a_\ell$ and $b_1 \geqslant b_2 \geqslant \cdots \geqslant b_\ell$, it becomes trivial to compute $[n]P$ with $\max a_i$ doublings *and* $\max b_i$ triplings.

This remark leads to the concept of *Double-Base chain*, *DB-chain* for short, introduced in [14], where we explicitly look for expansions having this property. In fact, the binary/ternary method discussed in [9] implicitly produces a DB-chain. Another way to obtain a DB-chain is to modify the greedy algorithm. At each step, simply restrain the search of the best exponents (a_{j+1}, b_{j+1}) to the interval $[0, a_j] \times [0, b_j]$.

Example 2. *A DB-chain for n can be derived from the following sequence of equalities*

$$\begin{aligned} 841232 &= 2^7 3^8 + 1424, \\ 1424 &= 2^1 3^6 - 34, \\ 34 &= 3^3 + 7, \\ 7 &= 3^2 - 2, \\ 2 &= 3^1 - 1. \end{aligned}$$

As a consequence, $841232 = 2^7 3^8 + 2^1 3^6 - 3^3 - 3^2 + 3^1 - 1$ *and* $[841232]P$ *can be obtained by the following Horner-like computation*

$$[841232]P = [3]\big([3]\big([3]([2^1 3^3]([2^6 3^2]P + P) - P\big) - P\big) + P\big) - P.$$

We can see that this DB-chain is strictly longer than the DBNS expansion given in Example 1. Experiments show that this is true in general as well but it is not known whether the bound $O\big(\frac{\log n}{\log \log n}\big)$ on the number of terms is still valid for DB-chains. Another concern for DB-chains, which also affects DBNS expansions, is the time required to actually find such an expansion. In particular, the search of the closest approximation of an integer in terms of a $\{2, 3\}$-integer can be relatively long. See [4, 12] for a review of different methods to find such an approximation.

Remark 3. *The DBNS has still some merit in cryptography, for instance on supersingular curves defined over $\mathbb{F}_3$. In this case, a tripling is virtually free, so that $[n]P$ can be computed with* $\max a_i$ *doublings,* $O\big(\frac{\log n}{\log \log n}\big)$ *additions, and the necessary number of triplings [10]. The same holds for a generalization of the DBNS in the context of Koblitz curves [2].*

Recent developments regarding DB-chains include the use of new endomorphisms, such as optimized quintupling [20]. This leads to a new type of representation, called *Step Multi-Base Representation*, SMBR for short, where an integer n is represented as

$$n = \sum_{i=1}^{\ell} d_i 2^{a_i} 3^{b_i} 5^{c_i} \text{ with } d_i \in \{-1, 1\} \tag{2}$$

and the exponents $(a_i), (b_i), (c_i)$ form three separate monotonic decreasing sequences. Again, a variant of the greedy algorithm is used to derive such an expansion. However, finding the best approximation of an integer in terms of a

$\{2,3,5\}$-integer is not easy and thus finding a short SMBR of an integer on the fly can be a problem, at least for certain devices.

Also, the concept of *extended DBNS* has been proposed in [12]. The idea is to introduce nontrivial coefficients in DBNS expansions, namely represent an integer by

$$n = \sum_{i=1}^{\ell} d_i 2^{a_i} 3^{b_i} \text{ with } |d_i| \in \mathcal{S}, \tag{3}$$

where $\mathcal{S}$ is a set of predefined coefficients. The computation of $[n]P$ then relies on the set of precomputed values $[d]P$ for all d in $\mathcal{S}$.

3 A New Approach

The proposed method to compute DB-chains can be seen as a generalization of the binary/ternary approach. First, let us assume that n is coprime to 6. We can start building a tree by considering two leaves corresponding to $n-1$ and $n+1$. After removing the powers of 2 and 3 from $n-1$ and $n+1$, we can reapply the process and for each node, add and subtract 1. Repeating this will create a binary tree. Eventually, one of its branch will reach 1 leading to a DB-chain expansion. Obviously, this approach is too costly for integers in the cryptographic range, say those of length 160 bits and above. However, we can eliminate certain branches and hope that the overall length of the DB-chain will not be affected too much. Fixing a bound B at the beginning, we can keep only the B smallest nodes before creating the next level of the tree. Note that it is very important that the nodes that are kept are all different. The algorithm is as follows.

Algorithm 1. Tree-based DB-chain search

```
INPUT: An integer $n$ and a bound $B$.
OUTPUT: A binary tree containing a DB-chain computing $n$.

1.  Set $t \leftarrow f(n)$                    $[f(n) = n/(2^{v_2(n)}3^{v_3(n)})]$
2.  Initialize a binary tree $\mathcal{T}$ with root node $t$
3.  **repeat**
4.      **for** each leaf node $m$ in $\mathcal{T}$ insert 2 children
5.          Left child $\leftarrow f(m-1)$
6.          Right child $\leftarrow f(m+1)$
7.      Discard any redundant leaf node
8.      Discard all but the $B$ smallest leaf nodes
9.  **until** a leaf node is equal to 1
10. **return** $\mathcal{T}$
```

Example 4. *Let us compute a DB-chain for $n = 841232$ using Algorithm 1. First, set $B = 2$. We obtain the chain*

$$841232 = 2^{18}3^1 + 2^{14}3^1 - 2^{11}3^1 + 2^9 + 2^4.$$

Then setting $B = 4$, we obtain the even shorter chain

$$841232 = 2^7 3^8 + 2^6 3^3 - 2^5 3^2 - 2^4.$$

See the appendix for details, including the trees returned by Algorithm 1 in each case.

Remark 5. *As sketched in the previous example and as we will see in Section 6, the size of the bound B has a great impact on the length of the DB-chain. Experimentally, $B = 4$ is a good compromise between the size of the tree and the quality of the expansion found. With these settings, the computation of the DB-chain is very fast since every operation in Algorithm 1 is totally elementary.*

4 Complexity Analysis

First, let us analyze the binary/ternary method. We show that the average length of the DB-chain of an integer n returned by the binary/ternary method has the upper bound $\frac{\log_2 n}{4.3774}\cdot$ For that, we investigate the average number of bits gained at each step. Take an integer $m > 1$ coprime to 6. Setting $m' = m-1$ or $m' = m+1$ so that 6 is a divisor of m', it is not hard to see that the probability for $\alpha \geqslant 1$ to be $v_2(m')$ and for $\beta \geqslant 1$ to be $v_3(m')$ is

$$\frac{1}{2^{\alpha-1}}\left(1-\frac{1}{2}\right)\frac{1}{3^{\beta-1}}\left(1-\frac{1}{3}\right)\cdot$$

The corresponding gain in that case is $\alpha + \beta\log_2 3$. So, the average number of bits gained is

$$\sum_{\alpha=1}^{\infty}\sum_{\beta=1}^{\infty}\frac{\alpha+\beta\log_2 3}{2^{\alpha-1}3^{\beta}} = 2 + \frac{3}{2}\log_2 3 = 4.3774\ldots$$

Since each step is independent of the other ones, we deduce that the average number of iterations, and therefore the average length of the DB-chain returned by the binary/ternary method, is bounded above by $\frac{\log_2 n}{4.3774}\cdot$ Also, the average values for α and β at each step are respectively 2 and $\frac{3}{2}$, which implies that on average

$$a_1 = \frac{\log_2 n}{1+\frac{3}{4}\log_2 3} \approx 0.4569\log_2 n \quad\text{and}\quad b_1 = \frac{\log_2 n}{\frac{4}{3}+\log_2 3} \approx 0.3427\log_2 n$$

in the DB-chain computed by the binary/ternary method. Since $a_1 = \max a_i$, $b_1 = \max b_i$ and the sequences of exponents are simultaneously decreasing, we need a_1 doublings and b_1 triplings, on top of $\frac{\log_2 n}{4.3774}$ additions to compute $[n]P$ on average.

We use a similar probabilistic argument to analyze Algorithm 1 and obtain the following result.

Theorem 1. *The average number of bits gained at each step in Algorithm 1 is* $4.6419\ldots$. *It follows that the average length of a DB-chain returned by Algorithm 1 is approximately equal to* $\frac{\log_2 n}{4.6419}\cdot$ *Also the average values for* a_1 *and* b_1 *are approximately equal to* $0.5569 \log_2 n$ *and* $0.2795 \log_2 n$*, respectively.*

Proof. Let us fix $B = 1$. This means that of the two leaf nodes created at each step, only the smallest is kept before reapplying the process. Now imagine that $m > 1$ is an integer coprime to 6. Let $\alpha_1 = v_2(m-1)$ and $\beta_1 = v_3(m-1)$. Similarly, let $\alpha_2 = v_2(m+1)$ and $\beta_2 = v_3(m+1)$. Set $\alpha = \alpha_1 + \alpha_2$ and $\beta = \beta_1 + \beta_2$. Then it is easy to show that $\alpha \geqslant 3$ and $\beta \geqslant 1$. Furthermore, we can show that there are four possibilities for (α_1, β_1) and (α_2, β_2), namely

$$\begin{array}{ll}
(\alpha-1,\beta), & (1,0) \\
(1,0), & (\alpha-1,\beta) \\
(\alpha-1,0), & (1,\beta) \\
(1,\beta), & (\alpha-1,0).
\end{array}$$

Considering residues modulo $2^{\alpha+1}3^{\beta+1}$, we see that all the cases occur with the same probability $\frac{1}{2^{\alpha-1}3^{\beta}}\cdot$

The maximal gain for the first two cases is $\alpha - 1 + \beta \log_2 3$, whereas it is $\max(\alpha-1, 1+\beta\log_2 3)$, for the last two. It follows that on average, the gain at each step is

$$\sum_{\alpha=3}^{\infty}\sum_{\beta=1}^{\infty} \frac{2(\alpha-1) + 2\beta\log_2 3 + 2\max(\alpha-1, 1+\beta\log_2 3)}{2^{\alpha-1}3^{\beta}}. \tag{4}$$

Computing only the first few terms in this sum, we find that the gain is equal to $4.6419\ldots$ This shows that Algorithm 1 performs better than the binary/ternary approach whose average gain is $4.3774\ldots$

When $B > 1$, we cannot precisely compute the average gain at each step for we do not know which branch will be selected in the end to compute a DB-chain for n. However, it is clear that on average the gain at each step will be larger than or equal to the gain when $B = 1$. In fact, tests show that even for very small $B > 1$, Algorithm 1 finds strictly shorter chains than when $B = 1$. For instance, the average gain at each step when $B = 4$ is close to 4.90, *cf.* Section 6, especially Table 1. Note that for very particular integers, a larger B can increase the length of the chain. The smallest example of this unexpected phenomenon is 31363, for which Algorithm 1 returns a DB-chain of length 5 when $B = 1$ and of length 6 when $B = 2$.

Concerning the average value of a_1 and b_1, a similar computation to (4) shows that we divide on average by $2^{2.5851}$ and $3^{1.2976}$ at each step. Multiplying by the average length of the chain, we deduce the quantities claimed in Theorem 1. □

To close this section, let us investigate the worst case for Algorithm 1 when $B = 1$. It corresponds to a minimal gain at each step. Based on the proof of Theorem 1, we see that this occurs when $(\alpha_1, \beta_1) = (2, 0)$ and $(\alpha_2, \beta_2) = (1, 1)$ or the converse. Given an integer ℓ, let us consider the DB-chain of length ℓ

$$2^{\ell-1}3^{\ell-1} - \sum_{i=0}^{\ell-2} 2^i 3^i$$

and let us denote by m_ℓ the actual integer corresponding to this chain. Then it is easy to see that m_ℓ is the smallest integer for which Algorithm 1 returns a DB-chain having at least ℓ terms. Indeed, it is clear that $m_\ell - 1$ is congruent to 4 mod 8 and to 1 mod 3. In the same way, $m_\ell + 1$ is congruent to 6 mod 36. Applying Algorithm 1 to m_ℓ, it follows that $(\alpha_1, \beta_1) = (2, 0)$ and $(\alpha_2, \beta_2) = (1, 1)$. So, the next nodes in the tree are $(m_\ell - 1)/4$ and $(m_\ell + 1)/6$. The smallest that is kept, *i.e.* $(m_\ell + 1)/6$ is in fact $m_{\ell-1}$.

The case $(\alpha_1, \beta_1) = (1, 1)$ and $(\alpha_2, \beta_2) = (2, 0)$, corresponds to the slightly larger integer $\sum_{i=0}^{\ell-1} 2^i 3^i$.

5 Generalizations

We can generalize the tree-based search in order to obtain other kinds of DB-chains. The first variant is to produce extended DB-chains, *i.e.* including non-trivial coefficients as in (3). This approach has been successfully exploited in [12] to compute very short DB-chains and to perform scalar multiplications using precomputations. In our case, given a set of coefficients $\mathcal{S}$, it is enough to slightly modify Algorithm 1, namely Lines 4, 5, and 6, to return extended DB-chains. Given a leave node in the tree, the only change is to insert $2|\mathcal{S}|$ children instead of just 2. More precisely, for each integer $d \in \mathcal{S}$, create 2 leave nodes corresponding to $f(m - d)$ and $f(m + d)$. The rest of the algorithm remains unchanged.

Another DB-chain variant is the SMBR. Again a simple change in Algorithm 1 allows to easily find short SMBRs. For instance, to find expansions as in (2), set the function $f(n)$ to return $n/(2^{v_2(n)}3^{v_3(n)}5^{v_5(n)})$. The number of nodes created at each step is the same as for regular DB-chains so that the computation of SMBRs is very fast. Note that it is also possible to mix the two ideas, *i.e.* return a SMBR with coefficients. For the bases 2, 3, and 5, the set of coefficients $\mathcal{S} = \{1, 7, 11, 13\}$ is a particularly effective choice. Indeed, after applying f to a certain node, we obtain an integer congruent to $\pm 1, \pm 7, \pm 11$, or ± 13 modulo 30. This implies that one of the eight children created for this node is divisible by 30. So, the gain at each step is larger than $\log_2 30$, giving rise to a very short chain.

6 Experiments

In this part, we analyze results of computations performed on random integers of various sizes. The first observation is that even when $B = 1$, the length of the DB-chain returned by Algorithm 1 is in general less than what is obtained from the greedy approach. The impact of B on the average length of the chains has also been particularly tested. For each size, we ran Algorithm 1 on a thousand random integers with different choices of B, namely $B = 1, 2, 4, 8, 16, 32, 64$, and 128. As expected, the length of the DB-chains tends to decrease when we increase B. However, experiments show that this decrease is quite slow, *cf.* Figure 1. As a result, there is little benefit in choosing a large B, especially since the time complexity to compute a chain linearly depends on B. That is why for subsequent DB-chain computations, B was set to 4, as it is a good compromise between the length of the chain and the time necessary to find it.

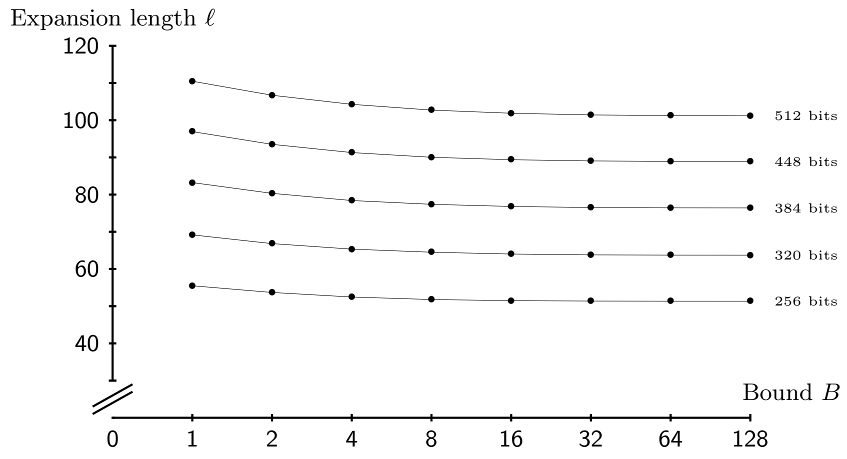


Fig. 1. Impact of B on the average length of DB-chains returned by Algorithm 1

Next, we compare the lengths of DB-chains obtained with various methods: binary/ternary, more importantly the greedy algorithm, and our new tree-based approach. We include the NAF for the record. In each case, ten thousand integers were tested. The average length of tree-based DB-chains is approximately 10% shorter than the length of chains returned by the greedy method, *cf.* Table 1. Note also that Algorithm 1 is somewhat easier to implement and faster in practice than the greedy approach.

Finally, we investigate the cost of a scalar multiplication on an elliptic curve in Edwards form

$$x^2 + y^2 = (1 + dx^2y^2)$$

using inverted Edwards coordinates [6]. Furthermore, we assume that multiplications by d can be neglected, so that the complexities of addition, doubling,

Table 1. Parameters of DB-chains obtained by various methods

Size	256 bits			320 bits			384 bits			448 bits			512 bits		
Method	ℓ	a_1	b_1	ℓ	a_1	b_1	ℓ	a_1	b_1	ℓ	a_1	b_1	ℓ	a_1	b_1
NAF	85.3	256	0	106.7	320	0	128.0	384	0	149.3	448	0	170.7	512	0
Bin./tern.	58.0	116.5	87.1	72.5	145.7	109.0	87.2	175.1	130.9	101.8	204.3	152.9	116.5	233.5	174.9
Greedy	58.0	150.7	65.5	72.5	189.1	81.6	87.0	228.5	97.2	101.5	266.7	113.4	116.2	305.1	129.6
Tree	52.5	145.7	68.6	65.5	182.0	86.2	78.4	218.4	103.5	91.4	255.2	120.7	104.3	291.3	138.3

Table 2. Complexity of various scalar multiplication methods for different sizes

Size	256 bits		320 bits		384 bits		448 bits		512 bits	
Method	N_{M}	Gain	N_{M}	Gain	N_{M}	Gain	N_{M}	Gain	N_{M}	Gain
NAF	1817.60	—	2423.47	—	3029.33	—	3635.20	—	4241.06	—
Bin./tern.	1761.19	3.10%	2353.59	2.88%	2944.94	2.79%	3537.21	2.70%	4129.57	2.63%
Greedy	1725.45	5.07%	2301.96	5.01%	2879.12	4.96%	3455.23	4.95%	4032.41	4.92%
Tree	1691.31	6.95%	2255.80	6.92%	2820.99	6.88%	3385.97	6.86%	3950.26	6.86%

and tripling are respectively 9M + S, 3M + 4S, and 9M + 4S, *cf.* [5]. To simplify comparisons between the different methods used to produce DB-chains, we make the usual assumption S = 0.8M. We these settings, N_M is the number of field multiplications necessary to compute $[n]P$ with each method, *cf.* Table 2. Note that the speed-up is expressed with respect to the NAF representation.

In case the device performing the computations allows some precomputations, it is possible to build DB-chains having nontrivial coefficients, *cf.* Section 5. In this case as well, tests show that the tree-based approach returns chains 10% shorter than the greedy algorithm. However, in this situation it is probably best to express the scalar multiple in some window-NAF representation and to take advantage of the very fast doublings provided by inverted Edwards coordinates, as pointed out in [3].

7 Conclusion

In this work, we describe a new method to find short DB-chains. So far DB-chains have been exclusively obtained with a greedy approach, relying on the search of the closest $\{2, 3\}$-integer to a given number. However to accommodate the main proprerty of DB-chains, that is simultaneously decreasing sequences of exponents, this search is done under constraints that tends to increase the length of the chain. Our new method, called tree-based search, only produces DB-chains. Given a parameter B, it consists in building a binary tree and eliminating all but the smallest B nodes at each step. Quite surprisingly, even for very small B, the algorithm performs extremely well. In fact, the tree-based approach outclasses the greedy algorithm for every choice of B we tested. As a side effect of its simplicity, this method is very easy to implement, *cf.* Algorithm 1. It is also straightforward to analyze. As shown in Section 4, the average length of a DB-chain returned by the tree-based method is approximately equal to $\frac{\log_2 n}{4.6419}$ when

$B = 1$. This is interesting because the complexity of the greedy algorithm is not well understood, when it comes to compute DB-chains. For instance, it is an open question to decide if the average length of DB-chains returned the greedy approach is $O\left(\frac{\log n}{\log \log n}\right)$ or not. This work suggests that it is not the case. Note however that for scalar multiples n routinely used in elliptic curve cryptography, typically in the range 192 to 320 bits, $\log \log n$ is between 4.8910 and 5.4018, so not too far away from 4.6419. Regarding scalar multiplications, our method suits devices where the use of precomputations is limited, if not impossible. In this situation, the tree-based approach induces an overall speed-up close to 7% over the NAF that is still widely used in practice.

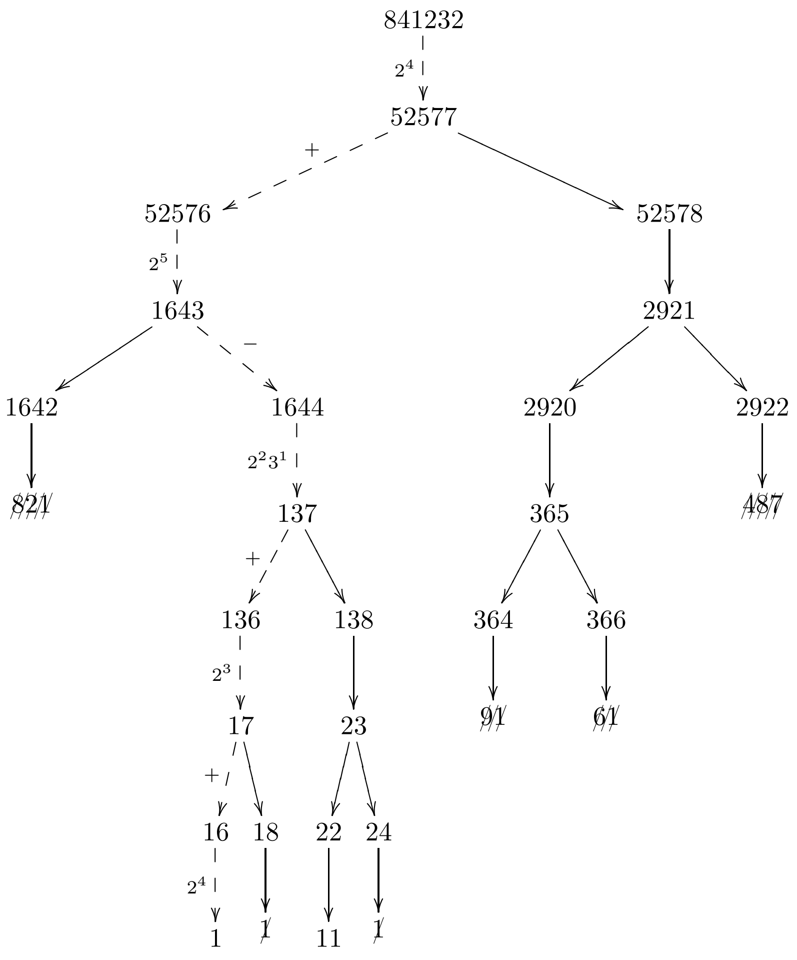
References

1. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Nguyen, K., Lange, T., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. In: Discrete Mathematics and its Applications, Chapman & Hall/CRC, Boca Raton (2005)
2. Avanzi, R.M., Dimitrov, V.S., Doche, C., Sica, F.: Extending Scalar Multiplication Using Double Bases. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 130–144. Springer, Heidelberg (2006)
3. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: Optimizing double-base elliptic-curve single-scalar multiplication. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 167–182. Springer, Heidelberg (2007)
4. Berthé, V., Imbert, L.: On Converting Numbers to the Double-Base Number System. In: Luk, F.T. (ed.) Advanced Signal Processing Algorithms, Architecture and Implementations XIV. Proceedings of SPIE, vol. 5559, pp. 70–78 (2004)
5. Bernstein, D.J., Lange, T.: Explicit-formulas database, `http://www.hyperelliptic.org/EFD/`
6. Bernstein, D.J., Lange, T.: Inverted Edwards Coordinates. In: Boztaş, S., Lu, H.-F(F.) (eds.) AAECC 2007. LNCS, vol. 4851, pp. 20–27. Springer, Heidelberg (2007)
7. Blake, I.F., Seroussi, G., Smart, N.P.: Elliptic Curves in Cryptography. London Mathematical Society Lecture Note Series, vol. 265. Cambridge University Press, Cambridge (1999)
8. Blake, I.F., Seroussi, G., Smart, N.P.: Advances in Elliptic Curve Cryptography. London Mathematical Society Lecture Note Series, vol. 317. Cambridge University Press, Cambridge (2005)
9. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading Inversions for Multiplications in Elliptic Curve Cryptography. Des. Codes Cryptogr. 39(2), 189–206 (2006)
10. Ciet, M., Sica, F.: An Analysis of Double Base Number Systems and a Sublinear Scalar Multiplication Algorithm. In: Dawson, E., Vaudenay, S. (eds.) Mycrypt 2005. LNCS, vol. 3715, pp. 171–182. Springer, Heidelberg (2005)
11. Dimitrov, V.S., Cooklev, T.: 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)
12. Imbert, L., Doche, C.: Extended Double-Base Number System with Applications to Elliptic Curve Cryptography. In: Barua, R., Lange, T. (eds.) INDOCRYPT 2006. LNCS, vol. 4329, pp. 335–348. Springer, Heidelberg (2006)

13. Doche, C., Icart, T., Kohel, D.R.: Efficient Scalar Multiplication by Isogeny Decompositions. In: Yung, M., Dodis, Y., Kiayias, A., Malkin, T.G. (eds.) PKC 2006. LNCS, vol. 3958, pp. 191–206. Springer, Heidelberg (2006)
14. Dimitrov, V.S., Imbert, L., Mishra, P.K.: Efficient and Secure Elliptic Curve Point Multiplication Using Double-Base Chains. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 59–78. Springer, Heidelberg (2005)
15. Dimitrov, V.S., Jullien, G.A., Miller, W.C.: An Algorithm for Modular Exponentiation. Information Processing Letters 66(3), 155–159 (1998)
16. Doche, C., Lange, T.: Arithmetic of Special Curves. In: [1], pp. 355–388
17. Doche, C.: Exponentiation. In: [1], pp. 145–168
18. Hankerson, D., Menezes, A.J., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2003)
19. Koblitz, N.: Elliptic Curve Cryptosystems. Math. Comp. 48(177), 203–209 (1987)
20. Mishra, P.K., Dimitrov, V.S.: Efficient Quintuple Formulas for Elliptic Curves and Efficient Scalar Multiplication Using Multibase Number Representation. In: Garay, J.A., Lenstra, A.K., Mambo, M., Peralta, R. (eds.) ISC 2007. LNCS, vol. 4779, pp. 390–406. Springer, Heidelberg (2007)
21. Miller, V.S.: Use of Elliptic Curves in Cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
22. Washington, L.C.: Elliptic Curves. In: Discrete Mathematics and its Applications. Number theory and cryptography, Chapman and Hall, Boca Raton (2003)

Appendix: Detailed Examples

Let us build a binary tree and find a DB-chain for $n = 841232$. First, set $B = 2$. Some extra information has been added to the tree returned by Algorithm 1 in order to facilitate the computation of the DB-chain. The branch that is actually used to compute the DB-chain appears in dashed line.



From this tree we deduce that

$$841232 = 2^4\big(2^5\big(2^2 3^1\big(2^3(2^4+1)+1\big)-1\big)+1\big)$$

which implies that

$$841232 = 2^{18}3^1 + 2^{14}3^1 - 2^{11}3^1 + 2^9 + 2^4.$$

That is one term less than for the chain obtained with the greedy algorithm, *cf.* Example 2.

If we execute the algorithm again with $B = 4$, we obtain the following tree

$$\begin{tikzcd}
&&&&&&& 841232 \arrow[d, dashed, "2^4"'] \\
&&&&&&& 52577 \arrow[dllll] \arrow[drrrr, dashed, "-"] \\
&&& 52576 \arrow[d] &&&&&&&& 52578 \arrow[d, dashed, "2^1 3^2"'] \\
&&& 1643 \arrow[dll] \arrow[drr] &&&&&&&& 2921 \arrow[dll] \arrow[drr, dashed, "-"] \\
& 1642 \arrow[d] &&&& 1644 \arrow[d] &&&& 2920 \arrow[d] &&&& 2922 \arrow[d, dashed, "2^1 3^1"'] \\
& 821 \arrow[dl] \arrow[dr] &&&& 137 \arrow[dl] \arrow[dr] &&&& 365 \arrow[dl] \arrow[dr] &&&& 487 \arrow[dl, dashed, "+"'] \arrow[dr] \\
820 \arrow[d] && 822 \arrow[d] && 136 \arrow[d] && 138 \arrow[d] && 364 \arrow[d] && 366 \arrow[d] && 486 \arrow[d, dashed, "2^1 3^5"'] && 488 \arrow[d] \\
\cancel{205} && \cancel{137} && 17 && 23 && \cancel{91} && 61 && 1 && \cancel{61}
\end{tikzcd}$$

from which we derive the expansion

$$841232 = 2^4\big(2^1 3^2\big(2^1 3^1(2^1 3^5 + 1) - 1\big) - 1\big)$$

that leads to the even shorter DB-chain

$$841232 = 2^7 3^8 + 2^6 3^3 - 2^5 3^2 - 2^4.$$