# Double-Base Number System for Multi-Scalar Multiplications

Christophe Doche[1], David R. Kohel[2], and Francesco Sica[3]

[1] Department of Computing
Macquarie University, Australia
doche@ics.mq.edu.au – http://www.comp.mq.edu.au/~doche
[2] Université de la Mediterranée
Aix-Marseille II, France
kohel@maths.usyd.edu.au – http://echidna.maths.usyd.edu.au/~kohel/index.html
[3] Department of Mathematics and Computer Science – Acecrypt
Mount Allison University, Sackville, Canada
fsica@mta.ca – http://www.acecrypt.com

9th September 2008

**Abstract.** The Joint Sparse Form is currently the standard representation system to perform multi-scalar multiplications of the form $[n]P + m[Q]$. We introduce the concept of Joint Double-Base Chain, a generalization of the Double-Base Number System to represent simultaneously $n$ and $m$. This concept is relevant because of the high redundancy of Double-Base systems, which ensures that we can find a chain of reasonable length that uses exactly the same terms to compute both $n$ and $m$. Furthermore, we discuss an algorithm to produce such a Joint Double-Base Chain. Because of its simplicity, this algorithm is straightforward to implement, efficient, and also quite easy to analyze. Namely, in our main result we show that the average number of terms in the expansion is less than $0.3945 \log_2 n$. With respect to the Joint Sparse Form, this induces a reduction by more than 20% of the number of additions. As a consequence, the total number of multiplications required for a scalar multiplications is minimal for our method, across all the methods using two precomputations, $P + Q$ and $P - Q$. This is the case even with coordinate systems offering very cheap doublings, in contrast with recent results on scalar multiplications. Several variants are discussed, including methods using more precomputed points and a generalization relevant for Koblitz curves. Our second contribution is a new way to evaluate $\overline{\phi}$, the dual endomorphism of the Frobenius. Namely, we propose formulae to compute $\pm\overline{\phi}(P)$ with at most 2 multiplications and 2 squarings in $\mathbb{F}_{2^d}$. This represents a speed-up of about 50% with respect to the fastest techniques known. This has very concrete consequences on scalar and multi-scalar multiplications on Koblitz curves.

**Keywords.** Elliptic curve cryptography, scalar multiplication, Double-Base Number System, Koblitz curves.

## 1 Introduction

### 1.1 Elliptic Curve Cryptography

An *elliptic curve* defined over a field $K$ can be seen as the set of points with coordinates in $\overline{K}$ lying on a cubic with coefficients in $K$. Additionally, the curve must be smooth, and if this is realized, the set of points lying on the curve can be endowed with an abelian group structure. This remarkable property, known for many centuries, has been exploited about twenty years ago to implement fundamental public-key cryptographic primitives [18, 17]. We refer to [23] for a thorough, yet accessible, presentation of elliptic curves and to [2, 16, 6, 7] for a discussion focused on cryptographic applications. In this context, two classes of elliptic curves are particularly relevant:

- curves defined over a large prime field $\mathbb{F}_p$ represented by a Weierstraß equation

$$y^2 = x^3 + a_4x + a_6, \quad a_4, a_6 \in \mathbb{F}_p \text{ such that } 4a_4^3 + 27a_6^2 \neq 0$$

  or an Edwards form [5]

$$x^2 + y^2 = 1 + dx^2y^2, \quad d \in \mathbb{F}_p \setminus \{0, 1\}.$$

- *Koblitz curves* defined over $\mathbb{F}_2$

$$y^2 + xy = x^3 + a_2x^2 + 1, \quad a_2 \in \{0, 1\}$$

  where points belong to some extension of finite degree $\mathbb{F}_{2^d}$.

The core operation in many elliptic curve cryptographic protocols is a *scalar multiplication*, which consists in computing the $[n]P$, given a point $P$ on the curve and some integer $n$. The standard method to perform a scalar multiplication is the double-and-add, which needs $\ell$ doublings and on average $\frac{\ell}{2}$ additions for integers of length $\ell$.

### 1.2 Double-Base Number System

The Double-Base Number System (DBNS) was initially introduced by Dimitrov and Cooklev [10] and later used in the context of elliptic curve cryptography [11]. With this system, an integer $n$ is represented as

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

To find an expansion representing $n$, we can use a greedy-type algorithm whose 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 $2^a3^b$. Then compute the difference and reapply the process.

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

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

In [13], 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 multiplication. Indeed, if the sequences of the powers of 2 and 3 are not simultaneously decreasing, as it is the case in Example 1, it becomes a challenge to obtain $[n]P$ without using too many doublings or triplings and without extra temporary variables.
This observation leads to the concept of *Double-Base Chain* (DBC), introduced in [11], where we explicitly look for expansions such that $a_\ell \geqslant a_{\ell-1} \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant b_{\ell-1} \geqslant \cdots \geqslant b_1$. This guarantees that exactly $a_\ell$ doublings and $b_\ell$ triplings are needed to compute $[n]P$. Note that it is easy to modify the greedy algorithm to return a DBC. Also, a tree-based algorithm has been recently developed with the same purpose [14].

**Example 2.** *A modified version of the greedy algorithm returns the following DBC*

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

A DBC expansion is always longer than a DBNS one, but computing a scalar multiplication with it is now straightforward. The most natural method is probably to proceed from right-to-left. With this approach 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 $[2^{a_\ell - a_{\ell-1}}3^{b_\ell - b_{\ell-1}}]P$, then add $c_{\ell-1}P$ and multiply the result by $[2^{a_{\ell-1}-a_{\ell-2}}3^{b_{\ell-1}-b_{\ell-2}}]$. Repeating this eventually gives $[n]P$, as illustrated with the chain of Example 2

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

## 1.3 Multi-Scalar Multiplication

A signature verification mainly requires a double-scalar multiplication of the form computing $[n]P + [m]Q$. Obviously, $[n]P$, $[m]Q$ can be computed separately and added together at the cost of $2\ell$ doublings and $\ell$ additions, on average, assuming that $n$ and $m$ are both of length $\ell$. More interestingly, $[n]P + [m]Q$ can also be obtained as the result of a combined operation called a *multi-scalar multiplication*. So called Shamir's trick, in fact a special case of an idea of Straus [22], allows to minimize the number of doublings and additions by jointly representing $\binom{n}{m}$ in binary. Scanning the bits from left to right, we perform a doubling at each step, followed by an addition of $P$, $Q$ or $P+Q$ if the current bits of $n$ and $m$ are respectively $\binom{1}{0}$, $\binom{0}{1}$, or $\binom{1}{1}$. If $P + Q$ is precomputed, we see that $[n]P + [m]Q$ can be obtained with $\ell$ doublings and $\frac{3\ell}{4}$ additions, on average.

It is possible to do better, as shown by Solinas [21], using the redundancy and flexibility of signed-binary expansions. Indeed, the Joint Sparse Form (JSF) is a representation of the form

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} n_{\ell-1} & \ldots & n_0 \\ m_{\ell-1} & \ldots & m_0 \end{pmatrix}_{\text{JSF}}$$

such that the digits $n_i, m_i$ fulfill certain conditions. Given two integers $n$ and $m$, there is an efficient algorithm computing the JSF of $n$ and $m$. If $\max(n, m)$ is of length $\ell$, then the number of terms is at most $\ell + 1$ and the number of nonzero columns is $\frac{\ell}{2}$ on average. Also, the JSF is proven to be optimal, that is for any given pair $(n, m)$, the JSF has the smallest density among all joint signed-binary representations of $n$ and $m$.

**Example 3.** *The joint sparse form of* $n = 542788$ *and* $m = 462444$ *is equal to*

$$\begin{pmatrix} n \\ m \end{pmatrix} = \begin{pmatrix} 100\bar{1}000100\bar{1}0100\bar{1}0\bar{1}00 \\ 10000100100001000100 \end{pmatrix}_{\text{JSF}}$$

*where* $\bar{1}$ *stands for* $-1$. *The computation of* $[n]P + [m]Q$ *requires 9 additions and 20 doublings, given that* $P + Q$ *and* $P - Q$ *are precomputed and stored.*

## 2 Joint Double-Base Number System

In the present article, we introduce the *Joint Double-Base Number System* (JDBNS) that allows to represent two integers $n$ and $m$ as

$$\begin{pmatrix} n \\ m \end{pmatrix} = \sum_{i=1}^{\ell} \begin{pmatrix} c_i \\ d_i \end{pmatrix} 2^{a_i} 3^{b_i}$$

with $c_i, d_i \in \{-1, 0, 1\}$.
To compare with other representation systems, we define the *density* of a JDBNS expansion as the number of terms in the expansion divided by the binary length of $\max(n, m)$.

**Example 4.** *We have*

$$\begin{pmatrix} 542788 \\ 462444 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} 2^{11}3^5 + \begin{pmatrix} 1 \\ \bar{1} \end{pmatrix} 2^9 3^4 + \begin{pmatrix} 1 \\ 1 \end{pmatrix} 2^6 3^4 + \begin{pmatrix} \bar{1} \\ 1 \end{pmatrix} 2^4 3^4$$
$$- \begin{pmatrix} 1 \\ 1 \end{pmatrix} 3^5 + \begin{pmatrix} 1 \\ 1 \end{pmatrix} 2^2 3 + \begin{pmatrix} \bar{1} \\ 1 \end{pmatrix} 2^2 - \begin{pmatrix} 1 \\ 1 \end{pmatrix}.$$

*This expansion has only 8 terms for a density equal to* 0.4. *However, we see that it belongs to this class of expansions that cannot be used directly with a Horner-like scheme, just like in the one-dimension case, cf. Section 1.2. As a consequence, we need more than 11 doublings and 5 triplings to compute the corresponding multi-scalar multiplication without storing any intermediate computation.*

That is why we also introduce the concept of *Joint Double-Base Chain* (JDBC) where the sequences of exponents satisfy $a_\ell \geqslant a_{\ell-1} \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant b_{\ell-1} \geqslant \cdots \geqslant b_1$. With this additional constraint, the computation of $[n]P + [m]Q$ can be done very efficiently provided that the points $P + Q$ and $P - Q$ are precomputed.

**Example 5.** *A JDBC for $n$ and $m$ is as follows*

$$\begin{pmatrix} 542788 \\ 462444 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} 2^{14}3^3 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} 2^{12} 3^3 + \begin{pmatrix} \bar{1} \\ 1 \end{pmatrix} 2^9 3^3 + \begin{pmatrix} 1 \\ 1 \end{pmatrix} 2^9 3^2 + \begin{pmatrix} \bar{1} \\ 1 \end{pmatrix} 2^7 3^2$$
$$+ \begin{pmatrix} 0 \\ 1 \end{pmatrix} 2^6 3^2 + \begin{pmatrix} 1 \\ \bar{1} \end{pmatrix} 2^4 3^2 + \begin{pmatrix} 1 \\ 1 \end{pmatrix} 2^4 3 + \begin{pmatrix} 0 \\ 1 \end{pmatrix} 2^2 3 + \begin{pmatrix} 1 \\ 0 \end{pmatrix} 2^2.$$

*This expansion has 2 more terms than the JDBNS expansion of Example 4, but setting $T_1 = P + Q$ and $T_2 = P - Q$, it is now trivial to compute*

*$[n]P + [m]Q$ as*

$$[2^2]\big([3]\big([2^2]\big([3]\big([2^2]\big([2]\big([2^2]\big([3]\big([2^3]([2^2]T_1+P)-T_2\big)+T_1\big)-T_2\big)+Q\big)+T_2\big)+T_1\big)+Q\big)+P\big)$$

*using exactly 14 doublings and 3 triplings. Note that the right-to-left method that works for a single scalar multiplication, cf. Section 1.2, cannot be adapted in this context. Indeed, we see that $[2^2]P$ is of no use to get the next term in the chain, namely $[2^2 3]Q$. As a consequence, we need 4 doublings and a tripling to obtain $[2^2]P + [2^2 3]Q$.*

Again, the greedy algorithm can be modified to return a JDBC, however, the resulting algorithm suffers from a certain lack of efficiency and it is difficult to analyze. The method we discuss next is efficient, in the sense that it produces quickly very short chains, and simple allowing a detailed complexity analysis.

## 3 Joint Binary-Ternary Algorithm and Generalizations

In [8], Ciet *et al.* propose a *binary/ternary method* to perform a scalar multiplication by means of doublings, triplings, and additions. Let $v_p(x)$ denote the *p-adic valuation* of $x$, 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 $x$ that is coprime to 6. This implies that $x$ mod 6 is equal to 1 or 5. Setting $x = x-1$ or $x = x+1$ allows to repeat the process at the cost of a subtraction or an addition.
We propose to generalize this method in order to compute a JDBC. First, let us introduce some notation. For two integers $x$ and $y$, we denote $\min\big(v_p(x), v_p(y)\big)$ by $v_p(x,y)$. It is the largest power of $p$ that divides $x$ and $y$ *simultaneously*.

**Definition 6.** *We denote by $\mathcal{C}$ the set of all the pairs of positive integers $(x,y)$ such that $v_2(x,y) = v_3(x,y) = 0$.*

Also, we introduce the function $\mathrm{gain}(x,y)$, which computes the largest factor $2^{v_2(x-c,y-d)}3^{v_3(x-c,y-d)}$ for $c,d \in \{-1,0,1\}$. If several pairs of coefficients achieve the maximum, then any pair can be returned. We set $\mathrm{gain}(1,1) = 0$.

**Example 7.** *The function* gain *applied to $(52,45)$ returns $2^2$, corresponding to $c = 0$ and $d = 1$. Indeed, it is easy to check that out of all the* *possibilities,* $(52, 44)$ *can be divided by the biggest term of the form* $2^\alpha 3^\beta$*, that is* $2^2$.

Note that this function gain can be implemented very efficiently by looking at the remainders of $x$ and $y$ modulo 6.

## 3.1 Algorithm

The principle of the algorithm is straightforward. Take two positive integers $n$ and $m$. Divide by $2^{v_2(n,m)}3^{v_3(n,m)}$ in order to obtain $(x, y) \in \mathcal{C}$. The idea is then to call the function $\text{gain}(x, y)$ and clear the common powers of 2 and 3 in $x - c$, $y - d$, where $c$ and $d$ are the coefficients maximizing this factor. (In case several pairs of coefficients achieve the same gain, any pair can be chosen.) The result is therefore a new pair in $\mathcal{C}$ so that we can iterate the process, namely compute the corresponding gain, divide by the common factor, and so on. Since $x$ and $y$ remain positive and decrease at each step, we will have at some point $|x| \leqslant 1$ and $|y| \leqslant 1$, which causes the algorithm to terminate.

**Algorithm 1.** Joint Binary-Ternary representation

INPUT: Two integers $n$ and $m$ such that $n > 1$ or $m > 1$.
OUTPUT: A joint DB-Chain computing $n$ and $m$ simultaneously.

```
1.  $i \leftarrow 1$                                                  [current index]
2.  $a_1 \leftarrow v_2(n,m)$ and $b_1 \leftarrow v_3(n,m)$          [common powers of 2 and 3]
3.  $x \leftarrow n/(2^{a_1}3^{b_1})$ and $y \leftarrow m/(2^{a_1}3^{b_1})$      [$(x,y) \in \mathcal{C}$]
4.  while $x > 1$ or $y > 1$ do
5.      $g \leftarrow \text{gain}(x,y)$
6.      $c_i \leftarrow c$ and $d_i \leftarrow d$                    [coefficients for that gain]
7.      $i \leftarrow i + 1$
8.      $x \leftarrow (x - c_i)/g$ and $y \leftarrow (y - d_i)/g$         [$(x,y) \in \mathcal{C}$]
9.      $a_i \leftarrow a_i + v_2(g)$ and $b_i \leftarrow b_i + v_3(g)$
10. $c_i \leftarrow x$ and $d_i \leftarrow y$                            [$c_i, d_i \in \{0,1\}$]
11. return $\left(\binom{c_i}{d_i} 2^{a_i} 3^{b_i}\right)_{\text{JDBC}}$
```

**Example 8.** *Algorithm 1 executed on* $n = 542788$ *and* $m = 462444$ *gives the following intermediate values, displayed just before the execution of*

*Line 7*

| $i$ | $g$ | $x$ | $y$ | $c_i$ | $d_i$ | $a_i$ | $b_i$ |
|---|---|---|---|---|---|---|---|
| 1 | $2^2$ | 135697 | 115611 | 1 | $\bar{1}$ | 2 | 0 |
| 2 | $2^1 3^1$ | 33924 | 28903 | 0 | 1 | 4 | 0 |
| 3 | $3^1$ | 5654 | 4817 | $\bar{1}$ | $\bar{1}$ | 5 | 1 |
| 4 | $3^1$ | 1885 | 1606 | 1 | 1 | 5 | 2 |
| 5 | $2^2$ | 628 | 535 | 0 | $\bar{1}$ | 5 | 3 |
| 6 | $3^1$ | 157 | 134 | 1 | $\bar{1}$ | 7 | 3 |
| 7 | $2^2$ | 52 | 45 | 0 | 1 | 7 | 4 |
| 8 | $2^2 3^1$ | 13 | 11 | 1 | $\bar{1}$ | 9 | 4 |
| 9 | — | 1 | 1 | 1 | 1 | 11 | 5 |

*We deduce that*

$$\begin{pmatrix}542788\\462444\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix}2^{11}3^5 + \begin{pmatrix}1\\\bar{1}\end{pmatrix}2^9 3^4 + \begin{pmatrix}0\\1\end{pmatrix}2^7 3^4 + \begin{pmatrix}1\\\bar{1}\end{pmatrix}2^7 3^3 + \begin{pmatrix}0\\\bar{1}\end{pmatrix}2^5 3^3$$
$$+ \begin{pmatrix}1\\1\end{pmatrix}2^5 3^2 - \begin{pmatrix}1\\1\end{pmatrix}2^5 3 + \begin{pmatrix}0\\1\end{pmatrix}2^4 + \begin{pmatrix}1\\\bar{1}\end{pmatrix}2^2.$$

Since doublings and triplings have different costs in different coordinate systems, it would be desirable to have some control of the largest powers of 2 and 3 in the expansion, which is not the case with Algorithm 1. To have more control on these values, we can modify the function gain. For instance, instead of returning the largest factor of the form $2^\alpha 3^\beta$, we can instead implement the function gain to return the factor having the largest power of 2 or 3.

## 3.2 Complexity Analysis

In the following, given integers $n$ and $m$ of a certain size, we compute the average density of a JDBC obtained with Algorithm 1, as well as the average values of the maximal powers of 2 and 3 in the joint expansion. This will in turn provide the average number of additions, doublings and triplings that are necessary to compute $[n]P + [m]Q$.
But let us start with the density of an expansion, which depends directly on the average number of bits cleared at each step of Algorithm 1. Given a pair $(x, y) \in \mathcal{C}$ and fixed values $\alpha$ and $\beta$, we determine the probability $p_{\alpha,\beta}$ that $\text{gain}(x, y) = 2^\alpha 3^\beta$ by enumerating the number of pairs having the desired gain in a certain square $S$ and dividing by the total number of pairs in $\mathcal{C} \cap S$.
Let $S_{\gamma,\delta}$ denote the set $[1, 2^\gamma 3^\delta]^2$. The total number of pairs we investigate is given by the following Lemma.

**Lemma 1.** *Given two integers $\gamma$ and $\delta$, the cardinality of $\mathcal{C} \cap S_{\gamma,\delta}$, is equal to $2^{2\gamma+1}3^{2\delta-1}$.*

The proof is straightforward and is left to the reader. Next, let us choose $\gamma, \delta$ to actually compute $p_{\alpha,\beta}$. At first glance, it seems that the square $S_{\alpha+1,\beta+1}$ is a good candidate for that. In fact, we can use it provided that when we consider a larger square, say $S_{\alpha+\kappa+1,\beta+\eta+1}$, the number of pairs having a gain equal to $2^\alpha 3^\beta$ and the total number of pairs in $\mathcal{C}$ are both scaled by the same factor: $2^{2\kappa}3^{2\eta}$. Indeed, we expect that if $(x, y)$ has a gain equal to $2^\alpha 3^\beta$, then all the pairs of the form $(x+i2^{\alpha+1}3^{\beta+1}, y+j2^{\alpha+1}3^{\beta+1})$ with $(i, j) \in [0, 2^\kappa 3^\eta - 1]^2$ will have the same gain. However, this is not the case. For instance, $\mathrm{gain}(26, 35) = 3^2$ whereas $\mathrm{gain}(26 + 2 \times 3^3, 35 + 5 \times 2 \times 3^3) = 4^2$. These interferences are inevitable, but intuitively, they will become less and less frequent and will eventually disappear for a set large enough. The following result makes this observation more precise.

**Lemma 2.** *Let $\alpha$ and $\beta$ be two nonnegative integers. Take $\gamma$ such that $2^\gamma > 2^\alpha 3^\beta$ and $\delta$ such that $3^\delta > 2^\alpha 3^\beta$. Then, for any $(x, y) \in \mathcal{C}$ whose gain is equal to $2^\alpha 3^\beta$, we have*

$$\mathrm{gain}(x + i2^\gamma 3^\delta, y + j2^\gamma 3^\delta) = \mathrm{gain}(x, y)$$

*for all $(i, j) \in \mathbb{Z}^2$.*

*Proof.* Clearly, we have $\mathrm{gain}(x, y) \leqslant \mathrm{gain}(x + i2^\gamma 3^\delta, y + j2^\gamma 3^\delta)$. To show the converse inequality, assume that there are $i$ and $j$ such that $\mathrm{gain}(x + i2^\gamma 3^\delta, y + j2^\gamma 3^\delta) = 2^{\alpha_1}3^{\beta_1} > \mathrm{gain}(x, y)$. If the coefficients corresponding to this gain are $c$ and $d$, we see that

$$\begin{aligned} v_2(x - c + i2^\gamma 3^\delta) \geqslant \alpha_1 \quad &\text{and} \quad v_2(y - d + j2^\gamma 3^\delta) \geqslant \alpha_1, \\ v_3(x - c + i2^\gamma 3^\delta) \geqslant \beta_1 \quad &\text{and} \quad v_3(y - d + j2^\gamma 3^\delta) \geqslant \beta_1. \end{aligned}$$

Then using that $v_p(r + s) = \min\big(v_p(r), v_p(s)\big)$ whenever $v_p(r) \neq v_p(s)$, we deduce that $v_2(x - c) \geqslant \alpha_1$, when $v_2(x - c) \neq v_2(i2^\gamma 3^\delta)$. If $v_2(x - c) = v_2(i2^\gamma 3^\delta)$, then we see that $v_2(x - c) \geqslant \gamma$. The same is true for $v_2(y - d)$ and we obtain that $v_2(x - c)$ and $v_2(y - d)$ are larger than $\min(\alpha_1, \gamma)$. In the same way, we can show that $v_3(x - c)$ and $v_3(y - d)$ are larger than $\min(\beta_1, \delta)$. Because $\gamma$ and $\beta$ have been chosen such that $2^\gamma$ and $3^\delta$ are strictly bigger than $2^\alpha 3^\beta$ and since also $2^{\alpha_1}3^{\beta_1} > 2^\alpha 3^\beta$, this proves that in any case $\mathrm{gain}(x, y) > 2^\alpha 3^\beta$, which is contrary to the hypothesis. □

Lemma 2 gives a lower bound for $p_{\alpha,\beta}$. Indeed, let us consider a larger set $S_{\gamma+\kappa,\delta+\eta}$. Then to any pair $(x,y) \in \mathcal{C} \cap S_{\gamma,\delta}$ whose gain is $2^\alpha 3^\beta$, we can associate the elements $(x + i2^\gamma 3^\delta, y + j2^\gamma 3^\delta)$ with $(i,j) \in [0, 2^\kappa 3^\eta - 1]^2$ that are in $\mathcal{C} \cap S_{\gamma+\kappa,\delta+\eta}$ and that have the same gain as $(x,y)$.
Conversely, if $(x_1,y_1) \in \mathcal{C} \cap S_{\gamma+\kappa,\delta+\eta}$ and $\operatorname{gain}(x_1,y_1) = 2^\alpha 3^\beta$, then $(x_1,y_1)$ can be written $(x + i2^\gamma 3^\delta, y + j2^\gamma 3^\delta)$ with $(x,y) \in S_{\gamma,\delta}$ and $\operatorname{gain}(x,y) = \operatorname{gain}(x_1,y_1)$.
Overall, this ensures that scanning $S_{\gamma,\delta}$ gives the exact probability for a pair to have a gain equal to $2^\alpha 3^\beta$ and allows to compute the first few probabilities.
The following lemma will help us deal with the remaining cases.

**Lemma 3.** *The probability $p_{\alpha,\beta}$ is bounded above by $\frac{1}{2^{2\alpha+1}3^{2\beta-3}}$ for any nonnegative $\alpha, \beta$.*

*Proof.* There are 6 integers in the interval $[1, 2^{\alpha+1}3^{\beta+1}]$ that are divisible by $2^\alpha 3^\beta$. In total, we have 18 elements $x_0$ such that $x_0 - 1, x_0$, or $x_0 + 1$ is divisible by $2^\alpha 3^\beta$. In the square $S_{\gamma,\delta}$, the pairs having a gain equal to $2^\alpha 3^\beta$ must be of the form $(x_0 + i2^{\alpha+1}3^{\beta+1}, y_0 + j2^{\alpha+1}3^{\beta+1})$ where $x_0$ and $y_0$ are one of the 18 elements above and with $(i,j) \in [0, 2^{\gamma-\alpha-1}3^{\delta-\beta-1} - 1]^2$. So, there are at most $2^{2(\gamma-\alpha)}3^{2(\delta+1-\beta)}$ pairs with a gain equal to $2^\alpha 3^\beta$. We conclude by dividing by the number of elements in $\mathcal{C} \cap S_{\gamma,\delta}$. □

We can now prove our main result.

**Theorem 1.** *Let $n \geqslant m$ be two integers such that $\gcd(n,m)$ is coprime with 6. The average density of the JDBC computing $\binom{n}{m}$ and returned by Algorithm 1 belongs to the interval $[0.3942, 0.3945]$. The average values of the biggest powers of 2 and 3 in the corresponding chain are approximately equal to $0.55\log_2 n$ and $0.28\log_2 n$.*

*Proof of Theorem 1.* We determine the first probabilities $p_{\alpha,\beta}$ using Lemmas 1 and 2. Namely, we enumerate pairs having a gain equal to $2^\alpha 3^\beta$ in the square $S_{\gamma,\delta}$, with $\gamma$ and $\delta$ as in Lemma 2. With an appropriate implementation, we need to investigate only $2^{2(\gamma-\alpha)}3^{2(\delta+1-\beta)}$ pairs and a quick computation gives us $p_{\alpha,\beta}$. We have performed the computations for $0 \leqslant \alpha \leqslant 8$ and $0 \leqslant \beta \leqslant 5$, and results show that these parameters cover more than 99.99% of the cases. We found that the probability $p_{i,j}$ is equal to $2^{-2i+3}3^{-2j}$ for $i \geqslant 2$ and $j \geqslant 1$. For $i \leqslant 1$ or $j = 0$, the probabilities do not seem to follow any pattern:

$$p_{0,0} = 0 \quad p_{0,1} = \tfrac{2}{3^2} \quad p_{0,2} = \tfrac{41}{2^3 3^4} \quad p_{0,3} = \tfrac{169}{2^5 3^6} \quad p_{0,4} = \tfrac{2729}{2^9 3^8} \quad p_{0,5} = \tfrac{10921}{2^{11}3^{10}}$$

$$p_{1,0} = 0 \; p_{1,1} = \tfrac{5}{3^3} \; p_{1,2} = \tfrac{95}{2^4 3^5} \; p_{1,3} = \tfrac{383}{2^6 3^7} \; p_{1,4} = \tfrac{6143}{2^{10} 3^9} \; p_{1,5} = \tfrac{24575}{2^{12} 3^{11}}$$

$$p_{2,0} = \tfrac{5}{2.3^2} \; p_{3,0} = \tfrac{7}{2^3 3^2} \; p_{4,0} = \tfrac{17}{2^3 3^4} \; p_{5,0} = \tfrac{635}{2^7 3^6} \; p_{6,0} = \tfrac{637}{2^6 3^6} \; p_{7,0} = \tfrac{2869}{2^{10} 3^8} \; p_{8,0} = \tfrac{51665}{2^{13} 3^{10}}.$$

Now, if the gain of $(x, y)$ is equal to $2^\alpha 3^\beta$ in Line 5 of Algorithm 1, then the sizes of $x$ and $y$ both decrease by $(\alpha + \beta \log_2 3)$ bits in Line 8. Therefore, if $K$ denotes the average number of bits eliminated at each step of Algorithm 1, we have

$$K = \sum_{\alpha=0}^{\infty} \sum_{\beta=0}^{\infty} p_{\alpha,\beta}(\alpha + \beta \log_2 3).$$

Thus

$$K \geqslant \sum_{\alpha=0}^{8} \sum_{\beta=0}^{5} p_{\alpha,\beta}(\alpha + \beta \log_2 3)$$

and thanks to the values above, we obtain $K \geqslant 2.53519$. Using Lemma 3, we bound the remaining terms in the double sum by

$$\begin{aligned}\sum_{\alpha=0}^{\infty} \sum_{\beta=0}^{\infty} \frac{\alpha + \beta \log_3 2}{2^{2\alpha+1} 3^{2\beta-3}} - \sum_{\alpha=0}^{8} \sum_{\beta=0}^{5} \frac{\alpha + \beta \log_3 2}{2^{2\alpha+1} 3^{2\beta-3}} &= \frac{2921}{3981312} + \frac{7741 \log_3 2}{31850496} \\ &= 0.001118\ldots,\end{aligned}$$

showing that $K \leqslant 2.53632$. The density being the inverse of $K$, we deduce the bounds of the theorem.
Similarly, we deduce that on average we divide by $2^{\overline{\alpha}} 3^{\overline{\beta}}$ at each step for some $\overline{\alpha} \in [1.40735, 1.40810]$ and $\overline{\beta} \in [0.71158, 0.71183]$. Also, the average of the largest power of 2 (respectively 3) in an expansion is equal to $\overline{\alpha}$ (respectively $\overline{\beta}$) multiplied by the average length of the expansion. We deduce the approximations claimed in the theorem from the computations above. $\square$

Since the JSF has a joint density of $\frac{1}{2}$, we see that a JDBC returned by Algorithm 1 has on average 21% less terms than a JSF expansion, whereas both representation systems require exactly 2 precomputations. See Table 1 to appreciate the overall impact of the Joint Binary-Ternary algorithm on multi-scalar multiplications.

### 3.3 Variants of the Joint Binary-Ternary Method

One simple generalization is to allow nontrivial coefficients in the expansion. This corresponds to use more precomputed points when computing a multi-scalar multiplication. For instance, if we allow the coefficients in the expansion to be $0, \pm 1, \pm 5$, then 10 points must be stored to compute $[n]P + [m]Q$ efficiently. Namely, $P + Q$, $P - Q$, $[5]P$, $[5]Q$, $[5]P + Q$, $[5]P - Q$, $P + [5]Q$, $P - [5]Q$, $[5]P + [5]Q$, and $[5]P - [5]Q$. The only difference with Algorithm 1 lies in the function $\text{gain}(x, y)$, which now computes the largest factor $2^{v_2(x-c,y-d)}3^{v_3(x-c,y-d)}$ for $c, d \in \{-5, -1, 0, 1, 5\}$. Clearly, the average number of bits that is gained at each step is larger than in Algorithm 1, and indeed, following the ideas behind Theorem 1, it is possible to show that the average density of an expansion returned by this variant is approximately equal to 0.3120. Note that this approach gives shorter multi-chains on average than the hybrid method explained in [1] that uses 14 precomputations for a density of 0.3209.

If we want to add a new value, e.g. 7, to the set of coefficients, we have to use 22 precomputed points, which does not seem realistic. Note that if the computations are performed on a device with limited memory, storing 10 points is already too much. A possibility is to precompute only $P + Q$, $P - Q$, $[5]P$, and $[5]Q$ and use only coefficients of the form $\binom{1}{0}$, $\binom{0}{1}$, $\binom{1}{1}$, $\binom{1}{\bar{1}}$, $\binom{5}{0}$, $\binom{0}{5}$ and their opposite in the JDBC. In this scenario, adding a new coefficient has a moderate impact on the total number of precomputations. Again, the only difference lies in the function gain. It is easy to perform an analysis of this method following the steps that lead to Theorem 1. This is left to the interested reader.

Another variant, we call the *Tree-Based Joint Binary-Ternary method*, is a generalization of the tree-based approach to compute single DB-Chains described in [14]. Namely, instead of selecting the coefficients $c, d$ that give the maximal gain in order to derive the next pair of integers, the idea is to build a tree containing nodes $(x, y)$ corresponding to all the possible choices of coefficients. The rationale behind this strategy is that taking a maximal gain at each step is not necessarily the best choice overall. Giving a certain flexibility can allow to find shorter expansions. The downside is that the number of nodes grows exponentially so that the algorithm becomes quickly out of control. A practical way to deal with this issue is to trim the tree at each step, by keeping only a fixed number $B$ of nodes, for instance the $B$ smallest ones (e.g. with respect to the Euclidean norm). Tests show that the value $B$ does not have to be very large in order to introduce a significant gain. In practice, we use $B = 4$, which achieves a good balance between the computation time and the quality of the chain obtained.

**Algorithm 2.** Tree-Based Joint Binary-Ternary method

INPUT: Two integers $n$ and $m$ such that $n > 1$ or $m > 1$ and a bound $B$.
OUTPUT: A tree containing a joint DB-chain computing $n$ and $m$.

```
1.  Initialize a tree $\mathcal{T}$ with root node $(n, m)$
2.  if $v_2(n, m) > 0$ or $v_3(n, m) > 0$ then
3.      $g \leftarrow 2^{v_2(n,m)} 3^{v_3(n,m)}$
4.      Insert the child $\left(\frac{n}{g}, \frac{m}{g}\right)$ under the node $(n, m)$
5.  repeat
6.      for each leaf node $\mathcal{L} = (x, y)$ in $\mathcal{T}$ do        [insert 8 children]
7.          for each pair $(c, d) \in \{-1, 0, 1\}^2 \setminus \{(0, 0)\}$ do
8.              $g_{c,d} \leftarrow 2^{v_2(x-c, y-d)} 3^{v_3(x-c, y-d)}$
9.              $\mathcal{L}_{c,d} \leftarrow \left(\frac{x-c}{g_{c,d}}, \frac{y-d}{g_{c,d}}\right)$ and insert $\mathcal{L}_{c,d}$ under $\mathcal{L}$
10.     Discard any redundant leaf node
11.     Discard all but the $B$ smallest leaf nodes
12. until a leaf node is equal to $(1, 1)$
13. return $\mathcal{T}$
```

**Remarks 9.**

(i) The choice $B = 1$ corresponds to the Joint Binary-Ternary method. It is clear that on average, the larger $B$ is, the shorter will be the expansion. However, a precise complexity analysis of Algorithm 2 seems rather difficult.

(ii) To find an actual JDBC computing $n$ and $m$, go through the intermediate nodes of any branch having a leaf node equal to $(1, 1)$.

(iii) To select the nodes that we keep in Line 11, we use a weight function that is in our case simply the size of the gain, of the form $2^\alpha 3^\beta$. To have more control on the largest powers of 2 and 3 in the expansion, we can use another weight function, e.g. depending on $\alpha$ or $\beta$.

(iv) It is straightforward to mix the tree-based approach with the use of nontrivial coefficients. Simply, modify Line 7 to handle different sets of coefficients.

(v) This algorithm is relevant for scalar multiplications as well. Namely, if we want to obtain $[n]P$, one possibility is to split $[n]$ as $n_0 + 2^A 3^B n_1$, where $A$ and $B$ are fixed constants adjusted so that $n_0$ and $n_1$ have approximately the same size and also to control the number of doublings and triplings. Then run Algorithm 1 to find a chain computing $n_0$ and $n_1$ simultaneously.

**Example 10.** *To compute* $[542788]P + [462444]Q$*, the JSF needs 9 additions and 20 doublings, whereas the joint Binary-Ternary method only requires 8 additions, 11 doublings, and 5 triplings, cf. Examples 1 and 8. Applying Algorithm 2 with* $B = 4$*, we find that*

$$\begin{pmatrix}542788\\462444\end{pmatrix} = \begin{pmatrix}1\\1\end{pmatrix}2^{11}3^5 + \begin{pmatrix}1\\\bar{1}\end{pmatrix}2^9 3^4 + \begin{pmatrix}1\\1\end{pmatrix}2^6 3^4 + \begin{pmatrix}\bar{1}\\1\end{pmatrix}2^4 3^4 \\ - \begin{pmatrix}1\\1\end{pmatrix}2^3 3^3 + \begin{pmatrix}\bar{1}\\0\end{pmatrix}2^2 3^2 + \begin{pmatrix}1\\\bar{1}\end{pmatrix}2^2 3 + \begin{pmatrix}1\\0\end{pmatrix}2^2.$$

*This last expansion still requires 11 doublings and 5 triplings but only 7 additions.*
*Adding* $\pm 5$ *to the set of coefficients allows the tree-based approach to find the even shorter expansion*

$$\begin{pmatrix}542788\\462444\end{pmatrix} = \begin{pmatrix}1\\0\end{pmatrix}2^7 3^6 + \begin{pmatrix}5\\5\end{pmatrix}2^7 3^6 - \begin{pmatrix}5\\1\end{pmatrix}2^7 3^3 + \begin{pmatrix}1\\\bar{5}\end{pmatrix}2^6 3 + \begin{pmatrix}0\\5\end{pmatrix}2^6 + \begin{pmatrix}1\\\bar{5}\end{pmatrix}2^2.$$

We have run some experiments to compare these methods in different situations. The outcome of these tests is gathered in Table 2.

## 3.4 Experiments

We have run some tests to compare the different methods discussed so far for different sizes ranging from 192 to 512 bits. More precisely, we have investigated the Joint Sparse Form (JSF), the Joint Binary-Ternary (JBT), and its Tree-Based variant with parameter $B$ adjusted to 4 (Tree-JBT). All these methods require only 2 precomputations. Also, for the same set of integers, we have looked at methods relying on more precomputed values. The variant of the Tree-Based explained above that needs only $5P$ and $5Q$ on top of $P + Q$ and $P - Q$ is denoted Tree-JBT$_5$. In this spirit Tree-JBT$_7$ needs also $7P$ and $7Q$, whereas Tree-JBT$_{5^2}$ needs all the possible combinations, that is 10 precomputations.
Table 1 displays the different parameters for each method, in particular the length of the expansion, corresponding to the number of additions and the number of doublings and triplings. The values obtained are inline with those announced in Theorem 1. The notation $\#\mathcal{P}$ stands for the number of precomputed points required by each method.

| | Size | 192 bits | | | 256 bits | | | 320 bits | | | 384 bits | | | 448 bits | | | 512 bits | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | $\#\mathcal{P}$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell\ell$ | $a_\ell$ | $b_\ell$ | |
| JSF | 2 | 96 | 192 | 0 | 128 | 256 | 0 | 160 | 320 | 0 | 190 | 384 | 0 | 224 | 448 | 0 | 256 | 512 | 0 |
| JBT | 2 | 77 | 104 | 55 | 102 | 138 | 74 | 128 | 174 | 92 | 153 | 208 | 110 | 179 | 241 | 130 | 204 | 279 | 146 |
| Tree-JBT | 2 | 72 | 107 | 53 | 96 | 141 | 72 | 119 | 178 | 89 | 143 | 214 | 107 | 167 | 248 | 126 | 190 | 281 | 145 |
| Tree-JBT$_5$ | 4 | 64 | 105 | 54 | 85 | 141 | 71 | 106 | 176 | 90 | 126 | 211 | 108 | 147 | 246 | 126 | 169 | 281 | 145 |
| Tree-JBT$_7$ | 6 | 60 | 102 | 55 | 80 | 137 | 74 | 99 | 171 | 93 | 119 | 204 | 112 | 139 | 238 | 131 | 158 | 273 | 150 |
| Tree-JBT$_{5^2}$ | 10 | 54 | 105 | 54 | 72 | 140 | 72 | 89 | 176 | 90 | 107 | 210 | 109 | 125 | 245 | 127 | 142 | 283 | 144 |
| Hybrid | 14 | 61 | 83 | 69 | 82 | 110 | 92 | 102 | 138 | 115 | 123 | 165 | 138 | 143 | 193 | 161 | 164 | 220 | 184 |

**Table 1.** Parameters of JDBC obtained by various methods

To demonstrate the validity of our approach, we have chosen to compare the scalar multiplication methods within the coordinate systems with the fastest doublings available, namely inverted Edwards coordinates [15]. With this choice a doubling can be obtained with 3M + 4S, a mixed addition with 8M + S, and a tripling with 9M + 4S.
Table 2 gives the overall number of multiplications needed for a scalar multiplications with a particular method, using inverted Edwards coordinates. To ease the comparisons, we make the usual assumption that 1S $\approx$ 0.8M. These tests show that the Joint Binary-Ternary, introduced in this article, is faster than the Joint Sparse Form. The Tree-Based variant is even faster, but the time necessary to derive the expansion is considerably higher than the simple Joint Binary-Ternary. Beyond the speed-up, that is close to 5%, it is interesting to notice that even with very cheap doublings, Double-Base like methods are faster. Regarding methods requiring more precomputed values, it is to be noted that all the variants introduced in this paper use significantly less precomputed points than the Hybrid method [1] and are all faster, even without counting the cost of precomputations. Note however that the figures in Table 2 include those costs.

## 4 Koblitz curves

The results above can be applied to compute a scalar multiplication on any elliptic curve. However, in practice, these techniques concern mainly curves defined over a prime field of large characteristic.
For Koblitz curves,

$$E_{a_2} : y^2 + xy = x^3 + a_2x^2 + 1, \text{ with } a_2 \in \{0, 1\}$$

| Method | Size $\#\mathcal{P}$ | 192 bits $N_{\mathrm{M}}$ | 256 bits $N_{\mathrm{M}}$ | 320 bits $N_{\mathrm{M}}$ | 384 bits $N_{\mathrm{M}}$ | 448 bits $N_{\mathrm{M}}$ | 512 bits $N_{\mathrm{M}}$ |
|---|---|---|---|---|---|---|---|
| JSF | 2 | 2044 | 2722 | 3401 | 4104 | 4818 | 5531 |
| JBT | 2 | 2004 | 2668 | 3331 | 4037 | 4724 | 5417 |
| Tree-JBT | 2 | 1953 | 2602 | 3248 | 3938 | 4605 | 5292 |
| Tree-JBT$_5$ | 4 | 1878 | 2501 | 3125 | 3792 | 4432 | 5095 |
| Tree-JBT$_7$ | 6 | 1847 | 2461 | 3077 | 3735 | 4365 | 5015 |
| Tree-JBT$_{5^2}$ | 10 | 1795 | 2390 | 2984 | 3624 | 4234 | 4862 |
| Hybrid | 14 | 1905 | 2537 | 3168 | 3843 | 4492 | 5159 |

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

there exists a nontrivial endomorphism, the *Frobenius* denoted by $\phi$ and defined by $\phi(x,y) = (x^2,y^2)$. Let $\mu = (-1)^{1-a_2}$, then it is well-known that the Frobenius satisfies $\phi^2 - \mu\phi + [2] = [0]$. So in some sense, the complex number $\tau$ such that $\tau^2 - \mu\tau + 2 = 0$ represents $\phi$. If an integer $n$ is equal to some polynomial in $\tau$, then the endomorphism $[n]$ will be equal to the same polynomial in $\phi$. The elements of the ring $\mathbb{Z}[\tau]$, called *Kleinian integers cf.* [12], thus play a key role in scalar multiplications on Koblitz curves.

## 4.1 Representation of Kleinian Integers

Let us denote the norm in $\mathbb{Z}[\tau]$ by $\mathrm{N}(\cdot)$. It is easy to show that $\mathbb{Z}[\tau]$ is an Euclidean ring and thus any element $\eta \in \mathbb{Z}[\tau]$ has a *$\tau$-adic representation* of the form

$$\eta = \sum_{i=0}^{\ell-1} c_i\tau^i, \quad \text{with} \quad c_i \in \{0,1\}.$$

There are also signed-digit representations and among them, the $\tau$-NAF has a distinguished status, achieving an optimal density of $\frac{1}{3}\cdot$ Its generalization, the $\tau$-NAF$_w$, has an average density of $\frac{1}{w+1}$ for $2^{w-2}-1$ precomputed points.
In [4, 3, 12], the concept of Double-Base is extended to Kleinian integers. In particular, for a given $\eta \in \mathbb{Z}[\tau]$, there is an efficient algorithm described in [3] that returns a $\tau$-DBNS expansion of the form

$$\eta = \sum_{i=1}^{\ell} \tau^{a_i} z^{b_i},$$

where $z = 3$ or $\bar{\tau}$.

This method produces in general an expansion whose terms cannot be ordered such that $a_\ell \geqslant a_{\ell-1} \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant b_{\ell-1} \geqslant \cdots \geqslant b_1$.
Unlike what we have seen in Section 1.2, such an expansion can still be used to compute a scalar multiplication in certain situations. The price to pay is to incorporate conversion routines between polynomial and normal bases [20] to compute repeated applications of the Frobenius for free. This approach is described in [19].
Since implementing these conversion techniques can be tricky, especially on devices with limited capabilities, we will not follow this path and introduce instead the concept of *$\tau$-DB-Chains* ($\tau$-DBC) where, as in the integer case, we ask that $a_\ell \geqslant a_{\ell-1} \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant b_{\ell-1} \geqslant \cdots \geqslant b_1$ in the expansion above.
The algorithm described in [3] could be adapted to return a $\tau$-DBC, however the implementation would certainly be tricky and the analysis quite involved. Instead, we can generalize the greedy algorithm or the binary-ternary method to produce such a chain.

## 4.2 Scalar Multiplication

The $\tau$-adic representation of $\eta$ implies that

$$[\eta]P = \sum_{i=0}^{\ell-1} c_i \phi^i(P).$$

Now, if we work in the extension $\mathbb{F}_{2^d}$, and if $\eta \in \mathbb{Z}$ is of size $2^d$, the length $\ell$ of the $\tau$-adic expansion of $\eta$ is twice longer as what we expect, that is $2d$ instead of $d$. That is why in practice, we first compute $\delta = \eta \bmod \frac{\tau^d - 1}{\tau - 1}$. Under appropriate conditions, we have $[\delta]P = [\eta]P$ with $\delta$ of length half then length of $\eta$. From now on, we assume that this reduction has been done and that the length of $\eta$ is approximately $d$.
Computing $[\eta]P$ with a Frobenius-and-add approach involves $\frac{d}{2}$ additions on average and $d$ Frobenius, that is $2d$ squarings. With the $\tau$-NAF$_w$, we need $2^{w-2} - 1 + \frac{d}{w+1}$ additions and some memory to store $2^{w-2} - 1$ precomputed points. The complexity of the $\tau$-DBNS is well understood, however as mentioned earlier, it requires change of basis techniques that are not available in our scenario.
The complexity of the $\tau$-DBC is much more difficult to analyze. Only some experiments give an indication of its performance, and tests show that the $\tau$-DBC cannot compete with for instance the $\tau$-NAF. The problem comes from the cost of the second endomorphism that is too expensive to balance

the saving in terms of additions. To make use of the $\tau$-DBC, it is crucial to reduce this cost. There is little hope to reduce significantly the cost of a tripling, that is why we focus our efforts on $\overline{\phi}$.
Obviously, we can implement $\overline{\phi}(P) = \mu P - \phi(P)$ with a subtraction and, in López–Dahab coordinates, this corresponds to the cost of a mixed addition, *i.e.* $8\mathrm{M} + 5\mathrm{S}$, where M and S are respectively the cost of a multiplication and a squaring in $\mathbb{F}_{2^d}$. But it is possible to do better. Indeed, we can replace $\overline{\phi}$ by the halving map using the equation $\phi\overline{\phi}(P) = [2]P$. A halving works on the point $P = (x_1, y_1)$ represented as $(x_1, \lambda_1)$ with $\lambda_1 = x_1 + y_1/x_1$. It involves solving a quadratic equation, computing a square root and a trace, and performing at least one multiplication [2]. It is thus difficult to accurately analyze the cost of a halving, but half the cost of a mixed López–Dahab addition, that is $4\mathrm{M} + 4\mathrm{S}$, is a reasonable estimate. This is still too expensive to justify the use of the $\tau$-DBC to compute scalar multiplication. We show next how to compute $\pm\overline{\phi}(P)$ in a much more efficient way.

## 4.3 Fast Evaluation of $\overline{\phi}$

In this part, we show how to compute $\pm\overline{\phi}(P)$ in López–Dahab coordinates with $2\mathrm{M} + \mathrm{S}$ when $a_2 = 1$ and $2\mathrm{M} + 2\mathrm{S}$ when $a_2 = 0$.

**Lemma 4.** *Let $P_1 = (X_1 : Y_1 : Z_1)$ be a point in López–Dahab coordinates on the curve $E_{a_2}$. Let $P_2 = \overline{\phi}(P_1)$. Then the López–Dahab coordinates of $P_2$, namely $(X_2 : Y_2 : Z_2)$ satisfy*

$$\begin{aligned} X_2 &= (X_1 + Z_1)^2 \\ Z_2 &= X_1 Z_1 \\ Y_2 &= \big(Y_1 + (1 - a_2)X_2\big)\big(Y_1 + a_2X_2 + Z_2\big) + (1 - a_2)Z_2^2 \end{aligned}$$

*The coordinates of the negative of $P_2$ are equal to $(X_2 : Y_2' : Z_2)$ with $Y_2' = \big(Y_1 + a_2X_2\big)\big(Y_1 + (1 - a_2)X_2 + Z_2\big) + (1 - a_2)Z_2^2$.*

*Proof of Lemma 4.* The negative of $P_2 = (X_2 : Y_2 : Z_2)$ in López–Dahab coordinates is $(X_2 : Y_2 + X_2Z_2 : Z_2)$. So, the formulae for $P_2$ and $-P_2$ given above are equivalent. Now let us show that $-P_2 = -\overline{\phi}(P_1) = \phi(P_1) - P_1$ when $a_2 = 1$. First, assume that $P_1 = \pm\phi(P_1)$. In this case, we must have $x_1^2 = x_1$. If $x_1 = 0$ then $P_1 = (0, 1)$ is a point of 2-torsion. This implies that $\phi(P_1) - P_1 = P_\infty$. The formulae above show that $(X_2 : Y_2 : Z_2) = (1 : 0 : 0)$ that is also $P_\infty$. If $x_1 = 1$, then we must have $y_1^2 + y_1 = 1$, which

is not compatible with the hypothesis $P_1 = \pm\phi(P_1)$.
Now, if $P_1 \neq \pm\phi(P_1)$, set $x_1 = X_1/Z_1$ and $y_1 = Y_1/Z_1^2$. The affine coordinates $(x_2, y_2)$ of $\phi(P_1) - P_1$ are given by

$$\begin{aligned} x_2 &= \lambda^2 + \lambda + x_1 + x_1^2 + 1, \\ y_2 &= \lambda(x_1 + x_2) + x_2 + x_1 + y_1, \end{aligned}$$

where $\lambda = \dfrac{y_1 + x_1 + y_1^2}{x_1 + x_1^2}\cdot$ We deduce that

$$x_2 = \frac{y_1^4 + x_1y_1^2 + x_1^2y_1^2 + y_1^2 + x_1^2y_1 + x_1y_1 + x_1^6 + x_1^5 + x_1^2}{x_1^4 + x_1^2}\cdot$$

Let $T = y_1^2 + x_1y_1 + x_1^3 + x_1^2 + 1$. Since $P_1$ is on $E_1$, we have $T = 0$ and it is easy to check that the numerator of $x_2$ is equal to $x_1^5 + x_1 + T^2 + T(x_1 + 1)$ so that

$$x_2 = \frac{x_1^5 + x_1}{x_1^4 + x_1^2} = \frac{x_1^5 + x_1^3}{x_1^4 + x_1^2} + \frac{x_1^3 + x_1}{x_1^4 + x_1^2} = x_1 + \frac{1}{x_1}\cdot$$

Also,

$$y_2 = \frac{y_1^2 + x_1^3y_1 + x_1^2y_1 + y_1 + x_1^2}{(1 + x_1)x_1^2}$$

and we notice that

$$y_1^2 + x_1^3y_1 + x_1^2y_1 + y_1 + x_1^2 + T = (1 + x_1)(x_1^2y_1 + y_1 + x_1^2 + x_1 + 1).$$

Substituting $x_1$ by $X_1/Z_1$ and $y_1$ by $Y_1/Z_1^2$ in these formulas, we deduce that

$$\begin{aligned} X_2 &= (X_1 + Z_1)^2 \\ Z_2 &= X_1Z_1 \\ Y_2 &= (Y_1 + X_2)(Y_1 + Z_2). \end{aligned}$$

Only the last equation is not totally obvious. The initial expression of $Y_2$ is $X_1^2Y_1 + X_1^2Z_1^2 + Z_1^4 + Y_1Z_1^2 + X_1Z_1^3$. It is easy to check that it can be factorized into $(Y_1 + X_1^2 + Z_1^2)(Y_1 + X_1Z_1)$ when we add $Y_1^2 + X_1Y_1Z_1 + X_1^3Z_1 + X_1^2Z_1^2 + Z_1^4 = 0$ to it. This gives the result when $a_2 = 1$.
The proof is similar when $a_2 = 0$. □

Note that this new way to compute $\overline{\phi}$ is also beneficial to the $\tau$-DBNS, especially regarding the algorithm described in [3]. A direct application of the formulae above induce a speed-up on the overall scalar multiplication ranging from 15% to 20%.

### 4.4 Multi-Scalar Multiplication Algorithms

To perform $[\eta]P+[\kappa]Q$ at once, there is also a notion of $\tau$-adic Joint Sparse Form, $\tau$-JSF [9]. The $\tau$-JSF and the JSF have very similar definitions, they have the same average joint density, that is $\frac{1}{2}$, however the optimality of the JSF does not carry over to the $\tau$-JSF. Namely, for certain pairs in $\mathbb{Z}[\tau]$, the joint density of the $\tau$-JSF expansion is not minimal across all the signed $\tau$-adic expansions computing this pair.
Now, let us explain how we can produce joint $\tau$-DBNS expansions and more importantly joint $\tau$-DBC.
The generalization of the greedy-type method is straightforward. At each step, find the closest approximation of $(\eta, \kappa)$ of the form $(c\tau^{\alpha}\overline{\tau}^{\beta}, d\tau^{\alpha}\overline{\tau}^{\beta})$ with $c, d \in \{-1, 0, 1\}$ with respect to the distance $d\big((\eta, \kappa), (\eta', \kappa')\big) = \sqrt{\mathrm{N}(\eta-\eta')^2 + \mathrm{N}(\kappa-\kappa')^2}$. Then subtract the closest approximation and repeat the process until we reach $(0, 0)$. To find a joint $\tau$-DBC, do the same except that this search must be done under constraint, just like in the integer case.
Another possibility is to adapt the method developed in Section 3. We call this approach the Joint-$\tau\overline{\tau}$ method. The framework is exactly the same, the only difference lies in the function gain. This time $\mathrm{gain}(\eta, \kappa)$ computes a suitable common factor $\tau^{\alpha}\overline{\tau}^{\beta}$ of the elements $(\eta - c, \kappa - d)$ for $c, d \in \{-1, 0, 1\}$. We are not interested in the factor having the largest norm, instead we prefer to control the largest power of $\overline{\tau}$, as this has a crucial impact on the overall complexity. This can be done quite easily by adjusting certain parameters. For each choice of the function gain, there is a corresponding algorithm, that we believe could be analyzed quite easily, following the integer case. However, it is still not totally clear what is the optimal choice for the gain at the moment, and so we defer such an analysis. Instead, we have run some experiments, detailed next.

## 5 Experiments

We have run some tests to compare the $\tau$-JSF with the Joint-$\tau\overline{\tau}$ for popular sizes used with Koblitz curves, ranging from 163 to 571 bits. Table 3 displays the different parameters for each method, in particular the length of the expansion, the values $a_{\ell}$ and $b_{\ell}$ corresponding respectively to the number of additions, the number of applications of $\phi$ and of $\overline{\phi}$, as well as the total number of multiplications $N_{\mathrm{M}}$ in $\mathbb{F}_{2^d}$ needed to perform a multi-scalar multiplication for the corresponding size. Note that both methods require only 2 precomputations and the figures include those costs. Also to ease comparisons we have made the usual assumption that $1\mathrm{S} \approx 0.1\mathrm{M}$.

Results show that our approach introduces overall improvements of 8 to 9% over the $\tau$-JSF regarding scalar multiplications.

| | 163 bits | | | 233 bits | | | 283 bits | | | 347 bits | | | 4409 bits | | | 571 bits | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell$ | $\ell$ | $a_\ell$ | $b_\ell \ell$ | $a_\ell$ | $b_\ell$ | |
| $\tau$-JSF | 82 | 163 | 0 | 117 | 233 | 0 | 142 | 283 | 0 | 174 | 347 | 0 | 205 | 409 | 0 | 286 | 571 | 0 |
| $N_{\mathrm{M}}$ | | 738 | | | 1050 | | | 1272 | | | 1558 | | | 1834 | | | 2555 | |
| Joint-$\tau\overline{\tau}$ | 65 | 116 | 44 | 92 | 167 | 62 | 112 | 204 | 76 | 137 | 251 | 93 | 161 | 295 | 110 | 224 | 412 | 155 |
| $N_{\mathrm{M}}$ | | 671 | | | 955 | | | 1154 | | | 1410 | | | 1665 | | | 2318 | |

**Table 3.** Comparison between the $\tau$-JSF and the Joint-$\tau\overline{\tau}$

## References

1. J. Adikari, V. Dimitrov, and L. Imbert. Hybrid Binary-Ternary Joint Sparse Form and its Application in Elliptic Curve Cryptography. Preprint, Available at: `http://eprint.iacr.org/2008/`.
2. R. M. Avanzi, H. Cohen, C. Doche, G. Frey, K. Nguyen, T. Lange, and F. Vercauteren. *Handbook of Elliptic and Hyperelliptic Curve Cryptography.* Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2005.
3. R. M. Avanzi, V. S. Dimitrov, C. Doche, and F. Sica. Extending Scalar Multiplication using Double Bases. In *Advances in Cryptology – Asiacrypt 2006*, volume 4284 of *Lecture Notes in Comput. Sci.*, pages 130–144. Springer, 2006.
4. R. M. Avanzi and F. Sica. Scalar Multiplication on Koblitz Curves using Double Bases. In *Proceedings of Vietcrypt 2006*, volume 4341 of *Lecture Notes in Comput. Sci.*, pages 131–146, Berlin, 2006. Springer-Verlag. See also Cryptology ePrint Archive, Report 2006/067, `http://eprint.iacr.org/`.
5. 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-Verlag.
6. I. F. Blake, G. Seroussi, and N. P. Smart. *Elliptic Curves in Cryptography*, volume 265 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, Cambridge, 1999.
7. I. F. Blake, G. Seroussi, and N. P. Smart. *Advances in Elliptic Curve Cryptography*, volume 317 of *London Mathematical Society Lecture Note Series.* Cambridge University Press, Cambridge, 2005.
8. M. Ciet, M. Joye, K. Lauter, and P. L. Montgomery. Trading Inversions for Multiplications in Elliptic Curve Cryptography. *Des. Codes Cryptogr.*, 39(2):189–206, 2006.
9. M. Ciet, T. Lange, F. Sica, and J.-J. Quisquater. Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms. In *Advances in Cryptology – Eurocrypt 2003*, volume 2656 of *Lecture Notes in Comput. Sci.*, pages 388–400. Springer-Verlag, 2003.

10. 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.
11. 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.
12. V. S. Dimitrov, K. Jarvine, M. J. Jacobson Jr, W. F. Chan, and Z. Huang. FPGA Implementation of Point Multiplication on Koblitz Curves Using Kleinian Integers. In *to appear in Proceedings of CHES 2006*, Lecture Notes in Comput. Sci., 2006.
13. V. S. Dimitrov, G. A. Jullien, and W. C. Miller. An Algorithm for Modular Exponentiation. *Information Processing Letters*, 66(3):155–159, 1998.
14. C. Doche and L. Habsieger. A Tree-Based Approach for Computing Double-Base Chains. In *ACISP 2008*, volume 5107 of *Lecture Notes in Comput. Sci.*, pages 433–446. Springer, 2008.
15. D. J. Bernstein and T. Lange, *Explicit-formulas database.* See `http://www.hyperelliptic.org/EFD/`
16. D. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag, Berlin, 2003.
17. N. Koblitz. Elliptic curve cryptosystems. *Math. Comp.*, 48(177):203–209, 1987.
18. V. S. Miller. Use of Elliptic Curves in Cryptography. In *Advances in Cryptology – Crypto 1985*, volume 218 of *Lecture Notes in Comput. Sci.*, pages 417–426. Springer-Verlag, Berlin, 1986.
19. K. Okeya, T. Takagi, and C. Vuillaume. Short Memory Scalar Multiplication on Koblitz Curves. In *Proceedings of CHES 2005*, volume 3659 of *Lecture Notes in Comput. Sci.*, pages 91–105. Springer, 2005.
20. D. J. Park, S. G. Sim, and P. J. Lee. Fast scalar multiplication method using change-of-basis matrix to prevent power analysis attacks on Koblitz curves. In Springer Verlag, editor, *Proceedings of WISA 2003*, volume 2908 of *Lecture Notes in Computer Science*, pages 474–488, 2003.
21. J. A. Solinas. Low-weight binary representations for pairs of integers. Combinatorics and Optimization Research Report CORR 2001-41, University of Waterloo, 2001.
22. E. G. Straus. Addition chains of vectors (problem 5125). *Amer. Math. Monthly*, 70:806–808, 1964.
23. L. C. Washington. *Elliptic Curves.* Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2003. Number theory and cryptography.