New and Improved Methods to Analyze and Compute Double-Scalar Multiplications

Christophe Doche and Daniel Sutantyo

Abstract—We address several algorithms to perform a double-scalar multiplication on an elliptic curve. All the methods investigated are related to the Double-Base Number System (DBNS) and extend previous work of Doche, Kohel, and Sica [25]. We refine and rigorously prove the complexity analysis of the Joint Binary-Ternary (JBT) algorithm. Experiments are in line with the theory and show that the JBT requires approximately 6% less field multiplications than the standard Joint Sparse Form (JSF) method to compute $[n]P + [m]Q$. We also introduce a randomized version of the JBT, called JBT-Rand, that gives total control of the number of triplings in the expansion that is produced. So it becomes possible with the JBT-Rand to adapt and tune the number of triplings to the coordinate system and bit length that are used, in order to further decrease the cost of a double-scalar multiplication. Then, we focus on Koblitz curves. For extension degrees enjoying an optimal normal basis of type II, we discuss a Joint τ-DBNS approach that reduces the number of field multiplications by at least 35% over the traditional τ-JSF. For other extension degrees represented in polynomial basis, the Joint τ-DBNS is still relevant provided that appropriate bases conversion methods are used. In this situation, tests show that the speedup over the τ-JSF is then larger than 20%. Finally, when the use of the τ-DBNS becomes unrealistic, for instance because of the lack of an efficient normal basis or the lack of memory to allow an efficient conversion, we adapt the Joint Binary-Ternary algorithm to Koblitz curves giving rise to the Joint τ-$\bar{\tau}$ method whose complexity is analyzed and proved. The Joint τ-$\bar{\tau}$ induces a speedup of about 10% over the τ-JSF.

Index Terms—Elliptic Curve Cryptography, Double-Scalar Multiplication, Double-Base Number System, Koblitz Curves, Joint Sparse Form.

1 Introduction

We start by a quick presentation of all the objects and concepts used in the remainder of this paper.

1.1 Elliptic Curves and Cryptography

An *elliptic curve* is a plane nonsingular cubic with a rational point. The curve is said to be *defined over* K if the equation has coefficients in K. Concretely, elliptic curves discussed in this article are either defined over $\mathbb{F}_{2^d}$ by the equation

$$E_1 : y^2 + xy = x^3 + a_2x^2 + a_6, \quad a_2 \in \{0, 1\}, a_6 \in \mathbb{F}_{2^d} \quad (1)$$

with $a_6 \neq 0$, or over a large prime field $\mathbb{F}_p$ by

$$E_2 : y^2 = x^3 + a_4x + a_6, \quad a_4, a_6 \in \mathbb{F}_p \quad (2)$$

with $4a_4^3 + 27a_6^2 \not\equiv 0 \bmod p$. The conditions on the coefficients ensure the curves are nonsingular.

A point (x_1, y_1) satisfying (1) or (2) is an *affine point*. For a given extension L of K, the *set of L-rational points* of a curve E denoted by $E(L)$ is the set of all affine solutions with coefficients in L, together with a special point, called the *point at infinity*. Any set of rational points, for instance $E_1(\mathbb{F}_{2^d})$ or $E_2(\mathbb{F}_p)$, can be endowed with an abelian group structure. Indeed, there exists a notion of *addition* of points lying on the curve. The resulting point is again on the curve and remarkably this operation, denoted by $+$, is associative, which makes it a group law. The operation to add P to itself $n-1$ times, denoted by $[n]P = P + \cdots + P$, is called a *scalar multiplication*. The point $Q = [n]P$ can be computed very efficiently, however, for well selected curves, there is no efficient method that is known to solve the converse problem, called *discrete logarithm problem*, that is retrieve the integer n from the points Q and P.

Scalar multiplication is therefore at the heart of many elliptic curve cryptographic protocols whose security depends on the intractability of the discrete logarithm problem. This explains the abundant literature on the different techniques that have been developed to speedup this critical operation. For discussions on elliptic curves and how the underlying group structure can be used in cryptography, we refer readers to [36, 5, 28].

1.2 Scalar Multiplication and Double-Base Number System

The basic method in performing a scalar multiplication is the double-and-add method. It is a straightforward adaptation of the square-and-multiply used to compute an exponentiation. There is a close link between the method used to perform the scalar multiplication $[n]P$ and the representation of the integer n. For instance, the double-and-add method is linked to the binary representation of n. It relies on two basic operations: the addition $P + Q$ and the doubling $[2]P$. Recently, a new representation system using simultaneously bases 2 and

• *Authors are with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia.*
E-mail: {christophe.doche, daniel.sutantyo}@mq.edu.au.

3 and called *Double-Base Number System* (DBNS) has been introduced by Dimitrov et al. [17, 18]. With this system, the scalar n is represented as

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

A DBNS expansion can be easily obtained using a greedy-type algorithm whose principle is to find at each step the best approximation of a given integer among terms of the form $2^a 3^b$. The interest of this method lies in the proven low number of terms necessary to represent an integer. Indeed, for any integer n, the greedy method returns a DBNS expansion having $O(\frac{\log n}{\log\log n})$ terms [22].

To perform a scalar multiplication using DBNS, it is natural to introduce a new elementary operation on elliptic curves, called a *tripling*, whose aim is to return $[3]P$. This tripling is usually optimized so that it is faster than a doubling followed by an addition, as for instance in [19, 24]. However, despite an efficient tripling operation and a low number of terms, leading to a low number of additions, a direct use of the DBNS is inadequate for scalar multiplications in many practical applications. This is a consequence of the two dimensional nature of the DBNS that forces to repeat certain operations or store some intermediate computations while processing the expansion. Those requirements rule out the DBNS in many situations.

Another variant, introduced in [19] and called *Double-Base Chain* (DBC) is to produce an expansion of the form (3) satisfying the constraints $a_\ell \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant \cdots \geqslant b_1$. As a consequence, $[n]P$ can be obtained using a Horner-like scheme with exactly a_ℓ doublings, b_ℓ triplings, and only one register. The greedy method can easily be adapted to produce a DBC. Other methods exist as well, such as the *Binary-Ternary* method [13] and a generalization using a tree-based search [23]. For a fixed n, a DBC expansion is experimentally much longer than its DBNS counterpart returned by the greedy method. The tree-based approach, which gives shorter expansions in practice than the greedy method, returns a DBC with $O(\log n)$ terms on average [23]. Moreover, it has been shown recently that for any positive integer ℓ, there are ℓ-bit integers for which any DBC expansion requires $\Omega(\ell)$ terms [30].

1.3 Double-Scalar Multiplication

ECDSA signature generation requires the computation of a scalar multiplication $[n]P$. The verification mainly relies on the computation of a *double-scalar multiplication* of the form $[n]P + [m]Q$. Obviously, $[n]P + [m]Q$ can be computed as $[n]P$ and $[m]Q$ separately followed by an addition. If ℓ is the binary length of the maximum of m and n, the average cost is then 2ℓ doublings and ℓ additions.

Using so-called Shamir's trick, which was in fact introduced by Straus [35], it is possible to minimize the number of doublings and additions by jointly representing $\binom{n}{m}$ in binary, and then scanning the double bits from left-to-right. One performs a doubling followed by either 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, the average cost is ℓ doublings and $\frac{3}{4}\ell$ additions.

Solinas [34] extended this method by introducing the the *Joint Sparse Form* (JSF) representation of the form

$$\binom{n}{m} = \binom{n_{l-1} \ldots n_0}{m_{l-1} \ldots m_0}_{\text{JSF}}$$

where the digits n_i, m_i satisfy certain conditions granting uniqueness and optimality in the sense that JSF has the smallest joint Hamming weight among all joint signed representations of n and m. The average cost to compute $[n]P+[m]Q$ is now ℓ doublings and $\frac{\ell}{2}$ additions provided that $P+Q$ and $P-Q$ are precomputed and stored. The JSF is widely used in practice to compute double-scalar multiplications.

2 Double-Scalar Multiplication Using Joint Double-Base Chain

2.1 Joint Double-Base Chain

In [25], the DBC is generalised to perform double-scalar multiplications leading to the concept of *Joint Double-Base Chain* (JDBC). Given integers n and m, a JDBC expansion of n and m is of the form

$$\binom{n}{m} = \sum_{i=1}^{\ell} \binom{c_i}{d_i} 2^{a_i} 3^{b_i}, \quad \text{with } c_i, d_i \in \{-1, 0, 1\}$$

and with $a_\ell \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant \cdots \geqslant b_1$. Similarly to a single scalar multiplication based on a DBC, the computation of $[n]P + [m]Q$ can be done very efficiently with a JDBC, following a Horner-like scheme. Just like for the JSF, we assume that the points $P+Q$ and $P-Q$ are precomputed. The *density* of a JDBC expansion is defined as the number of terms in the expansion divided by the binary length of $\max(n, m)$. The density as well as a_ℓ, i.e. the largest power of 2, and b_ℓ, i.e. the largest power of 3, all determine the efficiency of a JDBC.

There are different techniques to compute a JDBC. One of them relies on a generalised greedy approach. However, the average of the density and of the values a_ℓ and b_ℓ seem difficult to analyze in this context. Instead, the work presented in [25], which generalizes the Binary-Ternary method [13], allows a precise analysis. The object of this section is to refine and extend the work in [25].

Before we describe an algorithm that returns a JDBC, let us first recall some useful notions. For a prime number p and a positive integer x, let $v_p(x)$ be the valuation of p at x, i.e. the largest exponent v such that $p^v \mid x$. For positive integers x and y, let $v_p(x, y) = \min(v_p(x), v_p(y))$ and let

$$\mathbb{X} = \big\{(x, y) \in \mathbb{Z}^2 \mid x, y > 0 \text{ and } v_2(x, y) = v_3(x, y) = 0\big\}.$$

For (x, y) different from $(1, 1)$, $\texttt{gain}(x, y)$ is defined as the largest factor of the form $2^{v_2(x-c,y-d)}3^{v_3(x-c,y-d)}$ among

all pairs (c,d) in $\{-1,0,1\}^2$. We also set $\texttt{gain}(1,1)=0$. So for instance, $\texttt{gain}(52,45)$ is equal to 2^2, corresponding to $c=0$ and $d=1$. We note that any nontrivial value of the `gain` is necessarily bigger than or equal to 3. Also, the `gain` function can be implemented very efficiently since in most cases the result only depends on the remainders of x and y modulo 6.

The algorithm returning a JDBC and described in [25] is called the *Joint Binary-Ternary* (JBT) method. See Algorithm 1.

2.2 Joint Binary-Ternary method

Algorithm 1. Joint Binary-Ternary method

INPUT: Two positive integers n and m with $(n,m)\neq(1,1)$.
OUTPUT: A JDBC of n and m.

1. $i \leftarrow 1$
2. $a_1 \leftarrow v_2(n,m)$ and $b_1 \leftarrow v_3(n,m)$
3. $x \leftarrow n/(2^{a_1}3^{b_1})$ and $y \leftarrow m/(2^{a_1}3^{b_1})$
4. **while** $x>1$ **or** $y>1$ **do**
5. $\quad g \leftarrow \texttt{gain}(x,y)$ with coefficients c_i, d_i
6. $\quad x \leftarrow (x-c_i)/g$
7. $\quad y \leftarrow (y-d_i)/g$
8. $\quad i \leftarrow i+1$
9. $\quad a_i \leftarrow a_i + v_2(g)$ and $b_i \leftarrow b_i + v_3(g)$
10. $c_i \leftarrow x$ and $d_i \leftarrow y$
11. **return** $\left(\binom{c_i}{d_i}2^{a_i}3^{b_i}\right)_{\text{JDBC}}$

After the execution of lines 3 and 7, (x,y) is certain to be in $\mathbb{X}$. Also, the algorithm terminates as $\texttt{gain}(x,y)>1$ at each step until $x=1$ and $y=1$.

Algorithm 1 returns the following expansion when applied to $n=542788$ and $m=462444$.

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

Thus $[542788]P+[462444]Q$ can be obtained with 11 doubling, 5 tripling, and 8 additions.

2.3 Analysis of the Joint Binary-Ternary Method

The Joint Binary-Ternary method is partially analyzed in [25] where approximations of the average density as well as the average values of a_ℓ and b_ℓ are provided. The main idea is as follows. The average density is equal to the average number of steps required by Algorithm 1 to terminate divided by $\log_2\big(\max(n,m)\big)$. In other words, the average density of a JDBC returned by Algorithm 1 is approximately equal to the inverse of the average of the `gain` computed in Line 5 of Algorithm 1. The average `gain` is then evaluated using probabilities. For integers $\alpha,\ \beta\geqslant 0$, let $p_{\alpha,\beta}$ denote the probability that a random pair of integers $(x,y)\in\mathbb{X}$ has `gain` $2^\alpha 3^\beta$. Then the average number of bits cleared at each step of Algorithm 1 is equal to

$$K=\sum_{\alpha=0}^{\infty}\sum_{\beta=0}^{\infty}(\alpha+\beta\log_2 3)p_{\alpha,\beta}. \tag{4}$$

In [25], a few probabilities $p_{\alpha,\beta}$ are explicitly determined through an exhaustive enumeration. This approach is limited as the execution time grows exponentially with α and β. We now give closed formulas for all the probabilities. This allows to compute the average density and the average values of a_ℓ and b_ℓ with arbitrary precision.

Theorem 1: Let $\alpha\geqslant 2$ and $\beta\geqslant 1$. Let $\gamma_1=\lceil\beta\log_2 3\rceil$, $\delta_1=\lceil\alpha\log_3 2\rceil$, and $\delta_2=\lceil(\alpha-1)\log_3 2\rceil$. Then we have $p_{0,0}=p_{1,0}=0$ and

- $p_{\alpha,\beta}=\dfrac{8}{2^{2\alpha}3^{2\beta}}$
- $p_{0,\beta}=\dfrac{2^{2\gamma_1}-10}{2^{2\gamma_1-4}3^{2\beta+1}}$
- $p_{1,\beta}=\dfrac{3\cdot 2^{2\gamma_1+2}-32}{2^{2\gamma_1+1}3^{2\beta+1}}$
- $p_{\alpha,0}=\dfrac{7\cdot 3^{2\delta_1-2}-2\cdot 3^{2\delta_1-2\delta_2}-5}{2^{2\alpha}3^{2\delta_1-2}}$.

The proof of Theorem 1 is given in Section 3.

Corollary 2: Let n and m two integers of the same binary length L. The density of the JDBC expansion representing $\binom{n}{m}$ and returned by Algorithm 1 is approximately equal to 0.39436. The largest power of 2 and the largest power of 3 in the expansion are on average respectively close to $0.55516L$ and $0.28066L$.

Proof: Since K is equal to the average number of bits discarded at each step, it is clear that the average length of an expansion returned by Algorithm 1 is equal to L/K. The density being the number of terms divided by L, it follows that the density is the inverse of K. We use (4) together with Theorem 1 to compute K with sufficient precision and invert it to find the claimed value. The average value of a_ℓ is equal to the average exponent of the power of 2 in the `gain` at each step, i.e.

$$\sum_{\alpha=0}^{\infty}\sum_{\beta=0}^{\infty}\alpha p_{\alpha,\beta}$$

multiplied by average length of the expansion, i.e. L/K. The average value of b_ℓ is computed in a similar way. □
We now discuss the proof of Theorem 1.

3 Proof of Theorem 1

3.1 Outline of the Proof

Our approach is to select a subspace $\mathbb{S}$ of $\mathbb{X}$ that is suitable, enumerate the number of integer pairs $(x,y)\in\mathbb{S}$ with $\texttt{gain}(x,y)=q$, with $q=2^\alpha 3^\beta$, and then dividing

this number by the total number of points in $\mathbb{S}$. In fact, we consider a sequence of subspaces $\mathbb{S}_i$ of increasing size and show that the proportion of pairs with a certain $\texttt{gain}$ equal to q remains the same, thus giving the probability $p_{\alpha,\beta}$.

Using modular arithmetic, it is easy to count the number of integer pairs (x, y) such that $q \mid (x + c)$ and $q \mid (y + d)$ with $c, d \in \{-1, 0, 1\}$. For example, in $[2, q + 1]^2$ there are nine such pairs, namely (x, y) satisfying $x \equiv -1$, 0, or 1 $(\bmod\ q)$ and $y \equiv -1$, 0, or 1 $(\bmod\ q)$. Furthermore, it is a simple exercise to show that the ratio of such pairs in $[2, rq+1]^2$ remains the same for any $r \geqslant 1$.

We need to refine this approach in two ways. First, since Algorithm 1 only deals with pairs of integers in $\mathbb{X}$, we need to disregard pairs with both x and y congruent to $0 \bmod 2$ or 3. Second, it is possible that $(x + c_1)$ and $(y+d_1)$ are both divisible by q, for some $c_1, d_1 \in \{-1, 0, 1\}$ even though $\texttt{gain}(x, y) > q$. We call such a pair an interference and apply the inclusion–exclusion principle to deal with them.

3.2 Notations and Supporting Lemmas

In the following, let $q = 2^\alpha 3^\beta$ and $s = 2^\gamma 3^\delta > q$, where $\alpha, \beta, \gamma, \delta$ are nonnegative integers. For k a positive integer, we introduce

$$\begin{aligned}\mathbb{X}_k &= \mathbb{X} \cap [2, k + 1]^2\\ \mathbb{G}_{q,k} &= \{(x, y) \in \mathbb{X}_k \mid \texttt{gain}(x, y) = q\}\\ \mathbb{H}_{q,k} &= \left\{(x, y) \in \mathbb{X}_k \mid (x \bmod q, y \bmod q) \in \{-1, 0, 1\}^2\right\}\end{aligned}$$

where $x \bmod q$ denotes the residue congruent to x modulo q with the smallest absolute value. Note also that we consider the interval $[2, k + 1]$ in $\mathbb{X}_k$ rather than $[1, k]$ to avoid dealing with the pair $(1, 1)$ whose $\texttt{gain}$ is somewhat special. We can now precisely define the notion of an interference.

Definition 3: Take a pair $(x, y) \in \mathbb{H}_{q,k}$. If there exists an $s > q$ such that $(x, y) \in \mathbb{H}_{s,k}$, then (x, y) is an *interference in $\mathbb{H}_{q,k}$ from $\mathbb{H}_{s,k}$*. The integer s is the *source of interferences*.

For example, take $(143, 17) \in \mathbb{H}_{16,288}$. Obviously, $\texttt{gain}(143, 17) = 18$ and this shows that $(143, 17)$ is an interference in $\mathbb{H}_{16,288}$ from $\mathbb{H}_{18,288}$.

Back to the probability $p_{\alpha,\beta}$, it is easy to see that $p_{\alpha,\beta}$ is equal to the limit of $|\mathbb{G}_{q,k}|/|\mathbb{X}_k|$ when k tends to infinity. The cardinality of $\mathbb{X}_k$ can be easily determined, whereas finding the cardinality of $\mathbb{G}_{q,k}$ requires a lot more effort. We start by enumerating all the pairs in $\mathbb{H}_{q,k}$ and then remove any interference in $\mathbb{H}_{q,k}$ using the inclusion–exclusion principle, see Section 3.3. Note that we only need to compute the cardinality of the sets $\mathbb{H}_{q,k}$, there is no need to actually identify the interferences.

3.2.1 Source of Interferences

We now investigate the possible source of interferences for a given q, i.e. the possible values for $s > q$ such that there exists a pair $(x, y) \in \mathbb{H}_{q,k} \cap \mathbb{H}_{s,k}$. The following result is a useful extension of the Chinese Remainder Theorem.

Lemma 4: Let n_1 and n_2 be two integers, non necessarily coprime. For integers a_1 and a_2, the system of equations

$$\begin{cases} x \equiv a_1 \pmod{n_1} \\ x \equiv a_2 \pmod{n_2} \end{cases} \tag{5}$$

has a unique solution modulo $\lambda = \operatorname{lcm}(n_1, n_2)$ if and only if $a_1 \equiv a_2 \pmod{\gcd(n_1, n_2)}$.

Proof: If $a_1 \not\equiv a_2 \pmod{\gcd(n_1, n_2)}$, it is clear that a solution cannot exist. If $a_1 \equiv a_2 \pmod{\gcd(n_1, n_2)}$, then write $\gcd(n_1, n_2)$ as $g_1 g_2$ such that n_1/g_1 is coprime with n_2/g_2. Then solve

$$\begin{cases} x \equiv a_1 \pmod{n_1/g_1} \\ x \equiv a_2 \pmod{n_2/g_2} \end{cases} \tag{6}$$

with the Chinese Remainder Theorem, finding the unique solution x_0 defined modulo $n_1 n_2 / \gcd(n_1, n_2)$, that is λ. Using $a_1 \equiv a_2 \pmod{\gcd(n_1, n_2)}$, it is easy to show that x_0 is also a solution to (5). Finally, any solution to (5) has to satisfy (6), and this ensures the uniqueness of the solution modulo λ. □

When they exist, the solutions to (5) form a residue class modulo λ, that we denote by $\left[[a_1]_{n_1}, [a_2]_{n_2}\right]_\lambda$.

Lemma 5: Assume that $\gcd(q, s) > 2$. Then if (x, y) is an interference in $\mathbb{H}_{q,k}$ from $\mathbb{H}_{s,k}$ then $(x, y) \in \mathbb{H}_{\lambda,k}$ where $\lambda = \operatorname{lcm}(q, s)$.

Proof: If (x, y) is an interference in $\mathbb{H}_{q,k}$ from $\mathbb{H}_{s,k}$, then

$$x \equiv c_1 \pmod{q} \quad \text{and} \quad x \equiv c_2 \pmod{s}$$

with $c_1, c_2 \in \{-1, 0, 1\}$. Lemma 4 ensures that there is a solution modulo λ if and only if

$$c_1 \equiv c_2 \pmod{\gcd(q, s)}.$$

Since $\gcd(q, s) > 2$, this is only possible when $c_1 = c_2$. Both q and s divides $x - c_1$, so does $\lambda = \operatorname{lcm}(q, s)$. Repeating the same argument for y, this implies that any interference (x, y) is an element of $\mathbb{H}_{\lambda,k}$. □

Lemma 6: If $q \mid s$ then $\mathbb{H}_{s,k} \subset \mathbb{H}_{q,k}$.

The proof is straightforward and left to the reader. Next we show that only a finite number of sources of interferences need to be considered.

Lemma 7: Assume that (x, y) is an interference in $\mathbb{H}_{q,k}$ from $\mathbb{H}_{s,k}$. Then $(x, y) \in \mathbb{H}_{2q,k} \cup \mathbb{H}_{3q,k}$ or if that is not the case, then (x, y) must be an element of

- $\mathbb{H}_{2^{\gamma_1},k}$ with $\gamma_1 = \lceil \beta \log_2 3 \rceil$ when $\alpha = 0$ and $\beta \geqslant 1$
- $\mathbb{H}_{2^{\gamma_1+1},k}$ when $\alpha = 1$ and $\beta \geqslant 1$
- $\mathbb{H}_{2 \cdot 3^{\delta_2 - 1},k} \cup \mathbb{H}_{3^{\delta_1},k}$ with $\delta_1 = \lceil \alpha \log_3 2 \rceil$ and $\delta_2 = \lceil (\alpha - 1) \log_3 2 \rceil$ when $\alpha \geqslant 2$ and $\beta = 0$.

Proof: For any value of q, $2q$ and $3q$ are a possible source of interferences. Moreover, due to Lemma 6, by removing interferences from $\mathbb{H}_{2q,k}$ and $\mathbb{H}_{3q,k}$ we also remove all interferences from all sets $\mathbb{H}_{s,k}$ such that $q \mid s$, since these are subsets of $\mathbb{H}_{2q,k}$ and $\mathbb{H}_{3q,k}$. By Lemma 5,

these two sets also include all interferences from all s with $\gcd(q,s) > 2$.

So what remains is to remove interferences from $\mathbb{H}_{s,k}$ such that $\gcd(q,s) = 2$. When $\alpha \geqslant 2$, there is no such s, so we are done. When $\alpha = 0$ and $\beta \geqslant 1$, we need to remove interferences from all s of the form 2^γ, but by Lemma 6 it is enough to remove interferences from the set $\mathbb{H}_{s,k}$ with $s = 2^{\gamma_1}$ where γ_1 is the smallest integer such that $2^{\gamma_1} > 3^\beta$, that is $\gamma_1 = \lceil \beta \log_2 3 \rceil$. The same arguments can be applied to derive the last two cases. □

3.2.2 The Cardinality of $\mathbb{H}_{q,k}$ and its Intersections

To conclude the proof of Theorem 1, we need to compute $|\mathbb{H}_{q,k}|$ and $|\mathbb{H}_{q,k} \cap \mathbb{H}_{s,k}|$ for each source of interferences s. Recall that $q = 2^\alpha 3^\beta$ and $s = 2^\gamma 3^\delta > q$.

Lemma 8: For a positive integer h, we have

(a) $|\mathbb{H}_{q,k}| = 8h^2$ when $\alpha \geqslant 1$, $\beta \geqslant 1$, and $k = hq$
(b) $|\mathbb{H}_{q,k}| = 24h^2$ when $\alpha = 0$, $\beta \geqslant 1$, and $k = 2hq$
(c) $|\mathbb{H}_{q,k}| = 64h^2$ when $\alpha \geqslant 2$, $\beta = 0$, and $k = 3hq$.

Proof: It is simple to count the number of pairs (x,y) in $[2, k+1]^2$ such that

$$(x \bmod q, y \bmod q) \in \{-1, 0, 1\}^2. \tag{7}$$

The main difficulty is to discount pairs that are not in $\mathbb{X}$ in order to obtain $|\mathbb{H}_{q,k}|$. For (a), There are 9 pairs in $[2, q+1]^2$ satisfying (7), of which one is not an element of $\mathbb{X}$, namely the pair with both x and y congruent to $0 \bmod q$. It follows that there are $8h^2$ pairs in $[2, hq+1]^2$. For (b), we also need to be mindful of pairs (x,y) such that $x \equiv y \equiv 0 \pmod 2$. Using Lemma 4, we form the six possible residue classes for x modulo $2q$:

$$\begin{array}{lll} \big[[-1]_q, [0]_2\big]_{2q} & \big[[0]_q, [0]_2\big]_{2q} & \big[[1]_q, [0]_2\big]_{2q} \\ \big[[-1]_q, [1]_2\big]_{2q} & \big[[0]_q, [1]_2\big]_{2q} & \big[[1]_q, [1]_2\big]_{2q} \end{array}$$

We do the same for y and out of the 36 total combinations for (x,y), we count 12 pairs such that x and y are jointly divisible by 2 or by 3. Thus, in $[2, 2q+1]^2$, there are 24 pairs satisfying (7) and that are in $\mathbb{X}$. The result follows. The approach is similar for (c), except that we work modulo $3q$. We form the nine possible residue classes for x modulo $3q$:

$$\begin{array}{lll} \big[[-1]_q, [-1]_3\big]_{3q} & \big[[0]_q, [-1]_3\big]_{3q} & \big[[1]_q, [-1]_3\big]_{3q} \\ \big[[-1]_q, [0]_3\big]_{3q} & \big[[0]_q, [0]_3\big]_{3q} & \big[[1]_q, [0]_3\big]_{3q} \\ \big[[-1]_q, [1]_3\big]_{3q} & \big[[0]_q, [1]_3\big]_{3q} & \big[[1]_q, [1]_3\big]_{3q} \end{array}$$

We do the same for y and out of the 81 total combinations for (x,y) satisfying (7), 17 pairs are not in $\mathbb{X}$. Thus, modulo $[2, 3q+1]^2$, there are 64 pairs satisfying (7) and that are in $\mathbb{X}$. The result follows. □

It easy to compute $|\mathbb{H}_{q,k} \cap \mathbb{H}_{s,k}|$ when $q \mid s$, as we know from Lemma 6 that $\mathbb{H}_{s,k} \subset \mathbb{H}_{q,k}$. Next, we address the case $q \nmid s$.

Lemma 9: Let q and s be such that $s > q > 2$ and $q \nmid s$. For a positive integer h, we set $k = \lambda h$, where $\lambda = \operatorname{lcm}(q,s)$. Then $|\mathbb{H}_{q,k} \cap \mathbb{H}_{s,k}|$ is equal to

(a) $8h^2$ when $\gcd(q,s) > 2$
(b) $24h^2$ when $\gcd(q,s) = 2$
(c) $64h^2$ when $\gcd(q,s) = 1$.

Proof: The proof is similar to that of Lemma 8. First, we form the possible residue classes for x and y in the intersection before removing pairs that are not in $\mathbb{X}$. For (a), we see that there are three possible residue classes for x

$$\big[[-1]_q, [-1]_s\big]_\lambda \quad \big[[0]_q, [0]_s\big]_\lambda \quad \big[[1]_q, [1]_s\big]_\lambda$$

The reason lies in $\gcd(q,s) > 2$ and Lemma 4, which ensures that the reduction of x modulo q and modulo s must be the same. The same possibilities hold for y. Removing, one pair that is not in $\mathbb{X}$, we find 8 pairs in $[2, \lambda+1]^2$ that belong to $\mathbb{H}_{s,k} \cap \mathbb{H}_{q,k}$. The result follows. For (b), since $\gcd(q,s) = 2$, it is not possible for an integer to be $[-1]_q$ and $[0]_s$ at the same time, as this would violate Bézout identity. Hence there are five possibilities for the residue class of x

$$\begin{array}{lll} \big[[-1]_q, [-1]_s\big] & \big[[-1]_q, [1]_s\big] & \big[[0]_q, [0]_s\big] \\ \big[[1]_q, [-1]_s\big] & \big[[1]_q, [1]_s\big] & \end{array}$$

The same choices hold for y and out of the 25 possible combinations, only the pair $\big(\big[[0]_q, [0]_s\big], \big[[0]_q, [0]_s\big]\big)$ is not in $\mathbb{X}$, giving (b). The last case can be obtained in the same way.

□

Finally, we are ready to put all the pieces together and conclude the proof of Theorem 1.

3.3 Inclusion–Exclusion Principle

We recall that the probability $p_{\alpha,\beta}$ is equal to the limit of $|\mathbb{G}_{q,k}|/|\mathbb{X}_k|$ when k tends to infinity. The cardinality of $\mathbb{G}_{q,k}$ and of $\mathbb{X}_k$ can be determined by the following result that is a straightforward consequence of the inclusion–exclusion principle. Let S and $T_1, T_2, \ldots, T_m$ be finite sets, then

$$\begin{aligned} \left| S \setminus \bigcup_{i=1}^{m} T_i \right| \;=\; & |S| \;-\; \sum_{i=1}^{m} |S \cap T_i| \\ & + \sum_{\substack{i,j \\ 1 \leqslant i < j \leqslant m}} |S \cap T_i \cap T_j| \\ & - \sum_{\substack{i,j,k \\ 1 \leqslant i < j < k \leqslant m}} |S \cap T_i \cap T_j \cap T_k| \\ & \;\;\vdots \\ & + \;(-1)^m |S \cap T_1 \cap \cdots \cap T_m|. \end{aligned} \tag{8}$$

We start with the computation of $|\mathbb{X}_k|$.

Lemma 10: Let k be a multiple of 6. Then $|\mathbb{X}_k| = 2k^2/3$.

Proof: In the interval $[2, k+1]$, there are $k/2$ integers divisible by 2, forming $k^2/4$ pairs (x,y) such that $v_2(x,y) \geqslant 1$. Similarly, there are $k^2/9$ pairs with $v_3(x,y) \geqslant 1$, and $k^2/36$ pairs with $v_6(x,y) \geqslant 1$, and so by the inclusion–exclusion principle

$$|\mathbb{X}_k| = k^2 - k^2/4 - k^2/9 + k^2/36 = 2k^2/3.$$

□

We are now ready to prove the Theorem.

Proof of Theorem 1: To address the first case, we compute $|\mathbb{G}_{q,k}|$ for $q = 2^\alpha 3^\beta$ and $\alpha \geqslant 2$, $\beta \geqslant 1$. Lemma 7 shows that the only interferences in $\mathbb{H}_{q,k}$ are from $\mathbb{H}_{2q,k}$ and $\mathbb{H}_{3q,k}$. Thus

$$\mathbb{G}_{q,k} = \mathbb{H}_{q,k} \setminus (\mathbb{H}_{2q,k} \cup \mathbb{H}_{3q,k}).$$

Applying equation (8) and Lemma 6, we see that

$$|\mathbb{G}_{q,k}| = |\mathbb{H}_{q,k}| - |\mathbb{H}_{2q,k}| - |\mathbb{H}_{3q,k}| + |\mathbb{H}_{2q,k} \cap \mathbb{H}_{3q,k}|.$$

We set $k = 6qh$ and apply Lemmas 8 and 9 to derive the cardinality of each component. We obtain

$$|\mathbb{G}_{q,k}| = 8 \times (36h^2 - 9h^2 - 4h^2 + h^2) = 192h^2.$$

Finally, Lemma 10 ensures that $|\mathbb{X}_k| = 24h^2q^2$ and we conclude that $|\mathbb{G}_{q,k}|/|\mathbb{X}_k| = 8/q^2$. As this result is independent of h, we conclude that $p_{\alpha,\beta} = 8/(2^{2\alpha}3^{2\beta})$ whenever $\alpha \geqslant 2$ and $\beta \geqslant 1$, as announced.

We follow the same steps for the remaining cases. The only difference is the added complexity of dealing with more sets to remove as per Lemma 7. □

4 Generalizations

4.1 Considering More Precomputed Points

One easy, yet useful, generalization is to allow nontrivial coefficients in the expansion. Regarding the multi-scalar multiplication, this translates into introducing more precomputed points. 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 $[5]P-[5]Q$. To find such an expansion, Algorithm 1 remains the same and only the function gain needs to be modified so that $\texttt{gain}(x, y)$ returns the largest factor $2^{v_2(x-c,y-d)}3^{v_3(x-c,y-d)}$ for $c, d \in \{-5, -1, 0, 1, 5\}$. It is clear that the average number of bits that is cleared at each step is now larger and it is possible to show that the average density of an expansion returned by this variant is approximately equal to 0.3120 [25]. This method compares favorably to other shorter multi-chain methods, for instance the hybrid method explained in [2], which uses 14 precomputed values for a density of 0.3209.

4.2 Randomized Joint Binary-Ternary Method

When computing a JDBC for n and m, one drawback of the joint binary-ternary method is that it does not provide any control on the number of doublings and of triplings in the chain that is returned. Indeed, the average values of a_ℓ and b_ℓ only depend on the probabilities $p_{\alpha,\beta}$ and the maximal binary length of the integers n and m. Also, algorithm 1 is deterministic.

It would be desirable to have more control on the chain that is produced, in particular, on the maximal number of triplings in the expansion, as this could allow us to find a JDBC with a lower cost. This is especially useful as different coordinate systems have different costs for additions, doublings, and triplings. Having the ability to play with different parameters would allow us to adjust them and get a better outcome for a given pair of integers.

It turns out that it is quite easy to address this problem by simply modifying the function gain. One possibility is to introduce some randomness in the determination of the result. Let $\texttt{gain}'(x, y)$ be a function similar to $\texttt{gain}(x, y)$ except that it returns the largest common divisor of $x+c$ and $y+d$, for $c, d \in \{-1, 0, 1\}$ but restricted only to the common divisors of $x + c$ and $y + d$ that are not a multiple of 3.

Now, let us fix a parameter ε between 0 and 1. For $\varepsilon = 1$, the new approach is completely similar to Algorithm 1. Things are different if $\varepsilon < 1$. First, when $\texttt{gain}(x, y) = 3^2$, we replace this gain by 2^3 when possible, i.e. whenever there are coefficients c', d' such that 2^3 divides both $x - c'$ and $y - d'$. Second and more importantly, we introduce a random variable ξ with uniformly distributed values in $[0, 1]$, that is evaluated every time we need to compute the gain. If $\xi < \varepsilon$ then $\texttt{gain}(x, y)$ is returned otherwise $\texttt{gain}'(x, y)$ is returned.

It is very easy to adjust the total number of triplings required by tuning the value of ε. The resulting method is called *JBT-Rand*. It is no longer deterministic as different expansions are generated for the same pair (n, m) even when the same ε is used. In fact, for the same pair, the costs of the expansions that are returned vary greatly from one execution to another. That is why in practice, this method is more suited when the value of the scalar multiple is fixed. In that case, the algorithm can be run several times in order to identify a chain with a suitable low complexity.

See Section 7 for experiments and Table 1 for concrete values of ε for different coordinate systems and sizes.

5 Double-Scalar Multiplication on Koblitz Curves

5.1 Koblitz Curves

The Joint Binary-Ternary algorithm gives rise to a JDBC that can be applied to perform a double-scalar multiplication on an elliptic curve defined over a finite field of any characteristic. However in characteristic 2, Koblitz curves [29] offer specific and very efficient scalar multiplication techniques. *Koblitz curves* are defined over $\mathbb{F}_2$ by

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

While points on E_{a_2} have coordinates in some extension $\mathbb{F}_{2^d}$, the coefficients of the curve are in $\mathbb{F}_2$ and therefore the *Frobenius endomorphism* ϕ defined by $\phi(x_1, y_1) = (x_1^2, y_1^2)$ acts on $E_{a_2}(\mathbb{F}_{2^d})$. It is well known that the Frobenius satisfies $\phi^2 - \mu\phi + [2] = [0]$, with $\mu = (-1)^{1-a_2}$. This equation can be used to compute $[n]P$ for any n, using only the Frobenius map and additions. To see how,

let us fix μ and let τ be a complex root of the polynomial $X^2-\mu X+2$. A complex number η such that $\eta = n_0+n_1\tau$ with $(n_0, n_1) \in \mathbb{Z}^2$ is called a *Kleinian integer* [20]. It is easy to see that the set of Kleinian integers denoted by $\mathbb{Z}[\tau]$ is a Euclidean ring for the norm

$$\mathrm{N}(\eta) = n_0^2 + \mu n_0 n_1 + 2n_1^2. \tag{9}$$

It follows that any Kleinian integer η has a *τ-adic representation* of the form

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

In particular, any integer n can be written as $n = T(\tau)$ for some polynomial T with all coefficients equal to one. It follows that the endomorphism $[n]$ is equal to $T(\phi)$. As a result, $[n]P$ can be computed with a *Frobenius-and-add* approach, that is a direct adaptation of the double-and-add method where a doubling is replaced by an application of the Frobenius endomorphism. This is of course very interesting since the cost of the Frobenius is at most 3 squarings as compared to the cost of a doubling, that is for instance, 4 multiplications and 5 squarings in López–Dahab coordinates [8].

In practice, computing $[n]P$ where $P \in E_{a_2}(\mathbb{F}_{2^d})$ and n is an integer of size 2^d is a little more involved. Indeed, the length ℓ of the τ-adic expansion of a Kleinian integer is linked to its norm. As a result, if we try to compute $[n]P$ with a Frobenius-and-add approach, we end up using a τ-adic expansion whose length is twice what we would expect, i.e. $2d$ instead of just d. Fortunately, in certain situations there is a much more efficient way to obtain $[n]P$. Simply compute η that is the remainder of n modulo $\frac{\tau^d-1}{\tau-1}$ and perform $[\eta]P$ using a τ-adic expansion that is of length d. Now, $[n]P = [\eta]P$ under the assumption that P is a point of prime order in $E_{a_2}(\mathbb{F}_{2^d})$, which is always the case in cryptographic protocols. From now on, we assume that this reduction has occurred, and we focus on the computation of $[\eta]P$ where η is a Kleinian integer of norm 2^d.

5.2 Alternative Representations of Kleinian Integers

5.2.1 Single-Base Methods

There are many other ways than a τ-adic expansion to represent Kleinian integers and some are attractive in the context of scalar multiplications. In particular, the *τ-Non-Adjacent Form*, τ-NAF for short, introduced in [33], is a generalization of the classical NAF for integers [31]. The τ-NAF has a minimal Hamming weight among all signed-digit representations with coefficients $\{-1, 0, 1\}$ [4] and its average density is only $\frac{1}{3}$. A generalization, called the τ-NAF$_w$ [33] uses $2^{w-2}-1$ precomputed points for an average density of $\frac{1}{w+1}$.

5.2.2 Double-Base Methods

In [7, 6, 20], Double-Base Number System ideas are applied to Kleinian integers. For a given $\eta \in \mathbb{Z}[\tau]$, there is an efficient algorithm described in [6] that returns a τ-DBNS expansion of the form

$$\eta = \sum_{i=1}^{\ell} \pm\tau^{a_i} z^{b_i}, \quad \text{where } z = 3 \text{ or } \bar{\tau}. \tag{10}$$

Given that the *Verschiebung* $\widehat{\phi}$, i.e. the conjugate endomorphism of ϕ can be computed with 2 multiplications and $2-a_2$ squarings [25], $\bar{\tau}$ is the preferred choice for the second base z in practice. Now, if $m = \sqrt{\mathrm{N}(\eta)}$, then the number of terms in the expansion returned by the algorithm described in [6] is shown to be less than $2\log m/\log\log m$ in the worst case and is less than $\big(1+o(1)\big)\log m/\log\log m$ on average. For experiments in Section 7, we use a different approach that tends to produce slightly more efficient τ-DBNS expansions in practice. It is a direct adaptation of the greedy method. Starting from $\rho = \eta$, we iterate $\rho \leftarrow \rho - \tau^\alpha\bar{\tau}^\beta$ where $\tau^\alpha\bar{\tau}^\beta$ is the closest approximation of ρ for the norm in $\mathbb{Z}[\tau]$. We stop when $\rho = 0$. Whereas the algorithm in [6] works from right to left, this greedy approach proceeds from left to right.

The blocking algorithm introduced in 2008 [21] relies on the precomputation of the optimal representations of all the Kleinian integers of a given τ-adic length w. By mean of an exhaustive search, those values are represented in double-base form using τ and $1-\tau$ with the constraint that the power of $1-\tau$ is bounded by a small fixed constant. This bound is tuned depending on the extension degree and of the length w for optimal results. The τ-adic representation of η is then sliced in blocks of length w and each block is replaced by the corresponding minimal double-base representation stored in a lookup table.

5.2.3 Scalar Multiplication with a τ-DBNS

Just like in the integer case, a τ-DBNS expansion has a sublinear number of terms. But unlike the integer case, even if the expansion does not satisfy $a_\ell \geqslant a_{\ell-1} \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant b_{\ell-1} \geqslant \cdots \geqslant b_1$, it can still be used to perform a scalar multiplication. Indeed, with a normal basis representation, it becomes possible to compute repeated applications of the Frobenius virtually for free. An efficient scalar multiplication relying on a τ-DBNS expansion can then be obtained as follows. First, order the terms in the expansion with respect to the b_i's, then apply a Horner-like scheme on those exponents and perform as many Frobenius as required at each step. For instance, if $\eta = \bar{\tau}^8\tau^5 - \bar{\tau}^5\tau^{12} + \bar{\tau}^3\tau^4 + \tau^7$, then

$$[\eta]P = \widehat{\phi}^3\big(\widehat{\phi}^2\big(\widehat{\phi}^3\phi^5(P) - \phi^{12}(P)\big) + \phi^4(P)\big) + \phi^7(P).$$

Of course, it is easy to rewrite the expression in order to save a few more Frobenius applications

$$[\eta]P = \phi^4\big(\widehat{\phi}^3\big(\widehat{\phi}^2\phi\big(\widehat{\phi}^3(P) - \phi^7(P)\big) + P\big) + \phi^3(P)\big).$$

Regarding the implementation itself, we have two options. Indeed, certain normal bases offer very good performance regarding field multiplications. In particular,

some type-II normal bases outperform traditional low-weight polynomial bases [11] and allow to carry out all the computations within the same basis. However, prime extension degrees for which such a basis exists are quite sparse, only 13 out of the 58 prime extension degrees between 160 and 500. This choice is further reduced when we consider Koblitz curves with a cardinality of the form cN, with $c = 2$ or 4 and N prime. Indeed, only three extension degrees in the range $[160, 500]$, namely 233, 239, and 359, enjoy a type-II normal basis and give rise to a Koblitz curve having a near prime number of points.

The other possibility is to perform Frobenius operations within a normal basis, to use a polynomial basis for other operations, and to change representation every time it is necessary. This time, conversion techniques exist for any extension degree d. The simplest technique is to store the change of base matrix and its inverse. The time and space complexity is $O(d^2)$, i.e. the same as a field multiplication. Coron et al. [16] use this method to convert an affine point from polynomial to normal basis. According to them, the conversion time is then approximately equivalent to one field multiplication. Since we are dealing with points in López–Dahab coordinates (X, Y, Z), we assume in the following that the total cost of the conversion to normal basis and then back to polynomial basis is 3 multiplications.

Obviously, if we used an expansion of the form (10) with simultaneously decreasing sequences of exponents, i.e. a τ-DBC, we would not have to consider normal bases or conversion techniques. However, a τ-DBC does not compete with the τ-NAF when it comes to computing a single scalar multiplication. We will see that the situation is slightly different when computing a double-scalar multiplication.

5.3 Double-Scalar Multiplication Algorithms

There is also a notion of *τ-adic Joint Sparse Form* (τ-JSF) [14], that is useful to perform a double-scalar multiplication $[\eta]P + [\kappa]Q$ on Koblitz curves. The τ-JSF and the JSF have very similar definitions and they have the same average joint density, that is $\frac{1}{2}\cdot$ One notable difference though is that the τ-JSF is not optimal in terms of joint Hamming weight among all joint signed τ-adic expansions.

Regarding double-base expansions, it is possible to produce Joint τ-DBNS expansions using a greedy approach and also joint τ-DBC expansions adapting the Joint Binary-Ternary method in the context of Kleinian integers.

5.3.1 Joint τ-Double-Base Number System

In this part, we assume that we can somehow repeatedly apply the Frobenius for free as explained in Section 5.2.3. In order to compute $[\eta]P + [\kappa]Q$, we can rely on a *τ-Joint Double-Base Number System*, τ-JDBNS for short, of the form

$$\binom{\eta}{\kappa} = \sum_{i=1}^{\ell} \binom{c_i}{d_i} \tau^{a_i} \overline{\tau}^{b_i}, \quad \text{with } c_i, d_i \in \{-1, 0, 1\}. \quad (11)$$

In order to derive such an expansion, it is possible to apply a greedy approach. Similarly to the approach described in Section 5.2.2, we start with $(\rho, \theta) = (\eta, \kappa)$ and then at each step we find the closest approximation of (ρ, θ) 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

$$d\big((\rho, \theta), (\rho', \theta')\big) = \sqrt{\mathrm{N}(\rho - \rho')^2 + \mathrm{N}(\theta - \theta')^2}$$

where $\mathrm{N}(\cdot)$ is the norm defined in (9). Then we do $(\rho, \theta) \leftarrow (\rho - c\tau^{\alpha}\overline{\tau}^{\beta}, \theta - d\tau^{\alpha}\overline{\tau}^{\beta})$ until we reach $(0, 0)$. Finding the closest approximation is time-consuming but the resulting chain allows a very efficient computation of $[\eta]P + [\kappa]Q$ provided the points $P + Q$ and $P - Q$ are precomputed and stored. Another way to obtain this double-scalar multiplication is to compute $[\eta]P$ and $[\kappa]Q$ separately with two independent τ-DBNS expansions and add them together.

Experiments, see Section 7, show that the τ-JDBNS requires approximately 20% less multiplications than the τ-JSF and that it offers limited advantage over the τ-DBNS used twice. Given the complexity to implement and run the τ-JDBNS method, its interest is limited. By contrast, computing two τ-DBNS expansions is quick and easy and it is only marginally less efficient than the τ-JDBNS method. Note that, to the best of our knowledge, an efficient computation of a double-scalar multiplication using two τ-DBNS expansions has never been proposed before nor investigated.

Finally, the blocking algorithm introduced for single scalar multiplications, see Section 5.2.2 and [21], has been generalized to double-scalar multiplications [1]. A table containing the optimal double-base representations of all pairs of a fixed τ-adic length w is populated by exhaustive search before the τ-adic representations of η and κ are sliced into blocks of length w. The paper claims a speedup of more than 12% against the τ-JSF for the extension degree 163 based on the execution times obtained with a hardware implementation. Since the breakdown of the different operations involved is not provided, it is difficult to compare this blocking method against our τ-JDBNS. Also, $[1 - \tau]P$ is obtained in [1] with a mixed addition whose cost is $8\mathrm{M} + 5\mathrm{S}$, whereas we use the formulas given in [25] to compute $\widehat{\phi}(P) = [(-1)^{1-a_2} - \tau]P$ with $2\mathrm{M} + (2 - a_2)\mathrm{S}$.

5.3.2 Joint τ-Double-Base Chain

For certain extension degrees, computing the Frobenius, although very cheap, is not free. Thus the τ-JDBNS is ruled out and we turn to the concept of *τ-Joint Double-Base Chain* (τ-JDBC) of the form (11) with the extra requirements that $a_\ell \geqslant a_{\ell-1} \geqslant \cdots \geqslant a_1$ and $b_\ell \geqslant b_{\ell-1} \geqslant \cdots \geqslant b_1$. We could obtain such an expansion with a modified greedy approach but we prefer adapting the

Joint Binary-Ternary method developed for integers. We call this approach the *Joint* τ-$\bar{\tau}$ method. The framework is exactly the same as in Section 2.2. The difference lies in the definition of the function `gain`. Certainly the notion of being largest does not translate well into the complex plane, so it is not as straightforward to determine the gain of a Kleinian integer pair. For example, given $x, y \in \mathbb{Z}[\tau]$ and some $c, d \in \{-1, 0, 1\}$, if both $(x + c)$ and $(y + d)$ are divisible by τ^2, $\tau\bar{\tau}$, and $\bar{\tau}^2$, which value should we choose for the gain? One possibility is to use the norm of these Kleinian integers, but unfortunately all three numbers have the same norm. It is therefore necessary to establish a way to choose the desired gain. To this end, we introduce the *weight function*

$$w(\tau^\alpha \bar{\tau}^\beta) = w_1^\alpha w_2^\beta$$

where w_1, w_2 are positive integers with $w_1 > w_2$ and α, and β are nonnegative integers. We then define

$$\texttt{cgain}_{w_1,w_2}(x, y)$$

to be the function that computes the Kleinian integer divisor $q = \tau^\alpha \bar{\tau}^\beta$ of both $(x+c)$ and $(y+d)$, $c, d \in \{-1, 0, 1\}$, such that $w(q)$ is maximized over all other candidates. The weight should be driven by the respective costs of ϕ and of $\widehat{\phi}$ and the total number of additions required. The weight is easy to adjust so that the function `cgain` delivers optimal values for the maximal powers of τ and $\bar{\tau}$. Provided that $\gcd(w_1, w_2) = 1$, the function `cgain` is injective over all Kleinian integers of the form $\tau^\alpha \bar{\tau}^\beta$ since w is injective, so effectively we are choosing the gain in the same manner as we did in the integer case. The algorithm to obtain a Joint τ-$\bar{\tau}$ expansion is a straightforward adaptation of Algorithm 1 where τ, $\bar{\tau}$, and `cgain` respectively replaces 2, 3, and `gain`.

Because $\mathbb{Z}[\tau]$ is Euclidean, every Kleinian integer has a unique decomposition into prime elements. Since τ and $\bar{\tau}$ have norm 2, we see that they are prime in $\mathbb{Z}[\tau]$. For any nonzero element $x \in \mathbb{Z}[\tau]$, we then define $v_\tau(x)$ and $v_{\bar{\tau}}(x)$ as the largest powers of τ and $\bar{\tau}$ dividing x. Let $v_\tau(x, y) = \min\big(v_\tau(x), v_\tau(y)\big)$ and $v_{\bar{\tau}}(x, y) = \min\big(v_{\bar{\tau}}(x), v_{\bar{\tau}}(y)\big)$ and let

$$\mathbb{U} = \big\{(x, y) \in \mathbb{Z}[\tau]^2 \mid x, y \neq 0 \text{ and } v_\tau(x, y) = v_{\bar{\tau}}(x, y) = 0\big\}.$$

Similar to Theorem 1 in the integer case, Theorem 11 gives the probability $\pi_{\alpha,\beta}$ that a pair of Kleinian integers $(x, y) \in \mathbb{U}$ satisfies $\texttt{cgain}_{w_1,w_2}(x, y) = \tau^\alpha \bar{\tau}^\beta$ for a given weight function w. Those probabilities are useful to determine the average density of a Joint τ-$\bar{\tau}$ expansion.

Theorem 11: For fixed weights $w_1 > w_2 > 0$ and exponents $\alpha \geqslant 2$, $\beta \geqslant 2$, let $\gamma_1 = \lceil \beta \log_{w_1} w_2 \rceil$, $\gamma_2 = \lceil 1 + (\beta - 1) \log_{w_1} w_2 \rceil$, and let $\delta_0 = \lceil \log_{w_2} w_1 \rceil$, $\delta_1 = \lceil \alpha \log_{w_2} w_1 \rceil$, and $\delta_2 = \lceil 1 + (\alpha - 1) \log_{w_2} w_1 \rceil$.
We have

- $\pi_{\alpha,\beta} = \dfrac{8}{2^{2\alpha+2\beta}}$
- $\pi_{0,\beta} = \begin{cases} 0 & \text{if } w_1 > w_2^\beta \\ \dfrac{1}{2^{2\beta-1}} & \text{if } w_2^\beta > w_1 > w_2^{\beta-1} \\ \dfrac{2^{2\gamma_1+4} - 160}{3(2^{2\beta+2\gamma_1})} & \text{if } w_2^{\beta-1} > w_1 \end{cases}$
- $\pi_{1,0} = \dfrac{2^{2\delta_0+1} - 24}{9 \cdot 2^{2\delta_0}}$
- $\pi_{1,1} = \dfrac{1}{18}$
- $\pi_{1,\beta} = \dfrac{3 \cdot 2^{2\gamma_2} - 32}{3 \cdot (2^{2\gamma_2+2\beta-1})}$
- $\pi_{\alpha,0} = \dfrac{2^{2\delta_1+4} - 160}{3(2^{2\alpha+2\delta_1})}$
- $\pi_{\alpha,1} = \dfrac{3 \cdot 2^{2\delta_2} - 32}{3 \cdot (2^{2\delta_2+2\alpha-1})}$.

To analyze the complexity of computing $[\eta]P + [\kappa]Q$ with the Joint τ-$\bar{\tau}$ method where both η and κ are of norm $O(2^d)$, we simply use Theorem 11. It is easy to compute the average density of the joint expansion, as the inverse of the average bit length of the norm of the gain at each step, i.e. the inverse of

$$\sum_{\alpha=0}^{\infty} \sum_{\beta=0}^{\infty} (\alpha + \beta) \pi_{\alpha,\beta}$$

as well as the average values of a_ℓ and b_ℓ. Depending on the coordinate systems used, it is straightforward to adjust the weights w_1 and w_2, in order to minimize the total cost of the double-scalar multiplication $[\eta]P + [\kappa]Q$. See Section 7 for details.

6 Proof of Theorem 11

6.1 Outline of the Proof

The general approach is the same as for the proof of Theorem 1. We subdivide the set of all Kleinian integer pairs using a complete set of representants modulo a certain Kleinian integer $s = \tau^\gamma \bar{\tau}^\delta$ and then compute the proportion of the pairs with $\texttt{cgain}_{(w_1,w_2)}$ against the total number of pairs modulo s. Let us further elaborate how we are going to do this. For the rest of the paper, let $(x, y) \in \mathbb{U}$. We also assume that w_1 and w_2 are fixed, and so we omit the subscript from the function $\texttt{cgain}(x, y)$.

6.2 Notations and Supporting Lemmas

In the integer case, there is an obvious choice for a set of representants modulo $2^\gamma 3^\delta$ that is $[2, 2^\gamma 3^\delta]$. Working with Kleinian integers, it is not so easy. First, we address the number of different residue classes modulo a Kleinian integer of the form $\tau^\alpha \bar{\tau}^\beta$.

Lemma 12: Let $k = \tau^\alpha \bar{\tau}^\beta$. There are $\mathrm{N}(k) = 2^{\alpha+\beta}$ residue classes modulo k.

Proof: The proof follows from the multiplicity of the norm of an ideal. Given an ideal $\mathfrak{a}$ in the ring

$\mathbb{Z}[\tau]$, then the *norm of* $\mathfrak{a}$ denoted by $\mathrm{N}(\mathfrak{a})$ is simply the number of classes in $\mathbb{Z}[\tau]/\mathfrak{a}$. Assume that $\mathfrak{a}$ has prime decomposition $\mathfrak{a} = \mathfrak{p}_1^{a_1}\mathfrak{p}_2^{a_2}\dots\mathfrak{p}_k^{a_k}$, it is well-known [15] that

$$\mathrm{N}(\mathfrak{a}) = \big(\mathrm{N}(\mathfrak{p}_1)\big)^{a_1}\big(\mathrm{N}(\mathfrak{p}_2)\big)^{a_2}\cdots\big(\mathrm{N}(\mathfrak{p}_k)\big)^{a_k}.$$

Since $\mathrm{N}(\tau) = \mathrm{N}(\bar{\tau}) = 2$, there can only be 2 residue classes modulo τ, i.e. $[0]_\tau$ and $[1]_\tau$ and 2 residue classes modulo $\bar{\tau}$, $[0]_{\bar{\tau}}$ and $[1]_{\bar{\tau}}$. Hence $\mathrm{N}\big((\tau)\big) = \mathrm{N}\big((\bar{\tau})\big) = 2$ and the result follows. □

In the following, let $q = \tau^\alpha\bar{\tau}^\beta$ and $k = \tau^\gamma\bar{\tau}^\delta$ where $\alpha,\beta,\gamma,\delta$ are nonnegative integers and such that $\mathrm{N}(k) > \mathrm{N}(q)$. Also, let $\mathbb{Z}_k[\tau]$ be a set of representants of $\mathbb{Z}[\tau]$ modulo k that does not contain 0, nor ±1. We introduce

$$\begin{aligned}\mathbb{U}_k &= \mathbb{U}\cap\mathbb{Z}_k[\tau]^2\\ \mathbb{D}_{q,k} &= \{(x,y)\in\mathbb{U}_k \mid \texttt{cgain}(x,y) = q\}\\ \mathbb{E}_{q,k} &= \big\{(x,y)\in\mathbb{U}_k \mid (x \bmod q, y \bmod q)\in\{-1,0,1\}^2\big\}.\end{aligned}$$

The definition of an interference is similar to Definition 3 with $\mathbb{H}_{t,k}$ being replaced by $\mathbb{D}_{t,k}$ and also using the weight function $w(s) > w(q)$ instead of comparing q and s directly. We now identify all the possible source of interferences. The following lemmas are adaptations of Lemmas 5-9, now tailored to the complex case.

Lemma 13: Let s,q such that $w(s) > w(q)$. If (x,y) is an interference in $\mathbb{E}_{q,k}$ from $\mathbb{E}_{s,k}$ and $\gcd(q,s)\nmid\tau\bar{\tau}$, then $(x,y)\in\mathbb{E}_{\lambda,k}$ where $\lambda = \mathrm{lcm}(q,s)$.

Proof: The proof is essentially the same with that of Lemma 5. Assume that (x,y) is an interference. We know that

$$x\equiv c_1 \pmod q \quad\text{and}\quad x\equiv c_2 \pmod s \tag{12}$$

with $c_1,c_2\in\{-1,0,1\}$. Since $\mathbb{Z}[\tau]$ is Euclidean, there is a notion of Chinese Remainder Theorem and Lemma 4 can be generalized for Kleinian integers. Together with (12), it follows that $c_1\equiv c_2 \pmod{\gcd(q,s)}$. This implies that $c_1 = c_2$ as $\mathrm{N}\big(\gcd(q,s)\big) > 4$. Both q and s divides $x - c_1$, and so does $\lambda = \mathrm{lcm}(q,s)$. We repeat the same argument for y in order to conclude. □

Lemma 14: If $q\mid s$, then $\mathbb{E}_{s,k}\subset\mathbb{E}_{q,k}$.

The proof is straightforward and together with Lemma 13, this shows that there is a finite number of source of interferences in this case as well.

6.2.1 Source of Interferences

Lemma 15: Let $\gamma_0 = \lceil\log_{w_1}w_2\rceil$ and similarly to Theorem 11, set $\gamma_1 = \lceil\beta\log_{w_1}w_2\rceil$, $\gamma_2 = \lceil 1+(\beta-1)\log_{w_1}w_2\rceil$, $\delta_0 = \lceil\log_{w_2}w_1\rceil$, $\delta_1 = \lceil\alpha\log_{w_2}w_1\rceil$, and $\delta_2 = \lceil 1+(\alpha-1)\log_{w_2}w_1\rceil$. Assume that (x,y) is an interference in $\mathbb{E}_{q,k}$ from $\mathbb{E}_{s,k}$. Then $(x,y)\in\mathbb{E}_{\tau q,k}\cup\mathbb{E}_{\bar{\tau}q,k}$ or if that is not the case, then (x,y) is an element of

- $\mathbb{E}_{\tau^{\gamma_1},k}k\cup\mathbb{E}_{\tau^{\gamma_2-1}\bar{\tau},k}k$ when $\alpha = 0$ and $\beta\geqslant 2$
- $\mathbb{E}_{\bar{\tau}^{\delta_0},k}k$ when $\alpha = 1$ and $\beta = 0$
- $\mathbb{E}_{\tau^{\gamma_0+1},k}k\cup\mathbb{E}_{\bar{\tau}^{\delta_0+1},k}k$ when $\alpha = 1$ and $\beta = 1$
- $\mathbb{E}_{\tau^{\gamma_1+1},k}k\cup\mathbb{E}_{\tau^{\gamma_2}\bar{\tau},k}k$ when $\alpha = 1$ and $\beta\geqslant 2$
- $\mathbb{E}_{\tau\bar{\tau}^{\delta_2-1},k}k\cup\mathbb{E}_{\bar{\tau}^{\delta_1},k}k$ when $\alpha\geqslant 2$ and $\beta = 0$
- $\mathbb{E}_{\tau\bar{\tau}^{\delta_2},k}k\cup\mathbb{E}_{\bar{\tau}^{\delta_1+1},k}k$ when $\alpha\geqslant 2$ and $\beta = 1$.

Proof: The proof is again the same with the one we saw earlier for Lemma 7. In each case, clearly τq and $\bar{\tau}q$ are a source of interferences. By Lemma 13, $\mathbb{E}_{\tau q,k}$ and $\mathbb{E}_{\bar{\tau}q,k}$ also contain all interferences from the sets $\mathbb{E}_{\tau s,k}$ such that $q\mid s$. For $\alpha,\beta\geqslant 2$, all possible source of interferences have $\gcd(q,s)\nmid\tau\bar{\tau}$, so we are done. For the rest of the cases, we also add all s such that $\gcd(q,s) = 1$, τ, $\bar{\tau}$, or $\tau\bar{\tau}$, with the exponents chosen to be the smallest ones such that $w(s) > w(q)$. Note that $\texttt{cgain}(x,y)$ is never equal to $\bar{\tau}$. This is because every Kleinian integer is 0 or 1 modulo τ and because of the choice $w_1 > w_2$. □

We now proceed to compute $|\mathbb{E}_{q,k}|$ and then remove all interferences in order to obtain $|\mathbb{D}_{q,k}|$.

6.2.2 The Cardinality of $\mathbb{E}_{q,k}$ and its Intersections

We need to compute $|\mathbb{E}_{q,k}|$ and $|\mathbb{E}_{q,k}\cap\mathbb{E}_{s,k}|$ for each source of interferences s. We recall that $q = 2^\alpha 3^\beta$ and $s = 2^\gamma 3^\delta$ such that $\mathrm{N}(s) > \mathrm{N}(q)$.

Lemma 16: For a nonzero Kleinian integer h, we have

(a) $|\mathbb{E}_{q,k}| = 8\,\mathrm{N}(h)^2$ when $\alpha\geqslant 2$, $\beta\geqslant 1$ or when $\alpha\geqslant 1$, $\beta\geqslant 2$, and $k = hq$
(b) $|\mathbb{E}_{q,k}| = 24\,\mathrm{N}(h)^2$ when $\alpha\geqslant 2$, $\beta = 0$ and $k = \bar{\tau}hq$
(c) $|\mathbb{E}_{q,k}| = 24\,\mathrm{N}(h)^2$ when $\alpha = 0$, $\beta\geqslant 2$ and $k = \tau hq$
(d) $|\mathbb{E}_{\tau\bar{\tau},k}| = 3\,\mathrm{N}(h)^2$ and $|\mathbb{E}_{\tau,k}| = 9\,\mathrm{N}(h)^2$ if $k = h\tau\bar{\tau}$.

Proof: We proceed as for Lemma 8. Among the $\mathrm{N}(q)^2$ pairs in $\mathbb{Z}_q[\tau]^2$ there are 9 pairs of Kleinian integers such that

$$(x \bmod q, y \bmod q)\in\{-1,0,1\}^2. \tag{13}$$

Discounting the pair where both x and y are a multiple of q, and then scaling by h, we obtain $|\mathbb{E}_{q,k}| = 8\,\mathrm{N}(h)^2$ for (a), as there are $\mathrm{N}(h)^2\,\mathrm{N}(q)^2$ pairs in $\mathbb{Z}_k[\tau]^2$. For (b), on top of considering Kleinian integers congruent to $-1,0$, or 1 modulo q, we also need to consider the congruence to 0 or 1 modulo $\bar{\tau}$, since we have to disregard pairs with $v_{\bar{\tau}}(x,y) > 0$. So we look at the following values modulo $\bar{\tau}q$

$$\begin{array}{ccc}\big[[-1]_q,[0]_{\bar{\tau}}\big]_{\bar{\tau}q} & \big[[0]_q,[0]_{\bar{\tau}}\big]_{\bar{\tau}q} & \big[[1]_q,[0]_{\bar{\tau}}\big]_{\bar{\tau}q}\\ \big[[-1]_q,[1]_{\bar{\tau}}\big]_{\bar{\tau}q} & \big[[0]_q,[1]_{\bar{\tau}}\big]_{\bar{\tau}q} & \big[[1]_q,[1]_{\bar{\tau}}\big]_{\bar{\tau}q}.\end{array}$$

Out of the 36 possible combinations, there are 12 combinations such that x and y are jointly divisible by τ or by $\bar{\tau}$. Thus, in $\mathbb{Z}_{\bar{\tau}q}[\tau]^2$ there are 24 pairs satisfying (13) and that are in $\mathbb{U}$. The result follows. The other two cases can be shown in the same manner. □

Lemma 17: Let $q = \tau^\alpha\bar{\tau}^\beta$ such that $q\neq\bar{\tau}$ and $q\neq\tau\bar{\tau}$ and let s such that $w(s) > w(q)$ and $q\nmid s$. For a nonzero Kleinian integer h, we set $k = \lambda h$, where $\lambda = \mathrm{lcm}(q,s)$. Then $|\mathbb{E}_{q,k}\cap\mathbb{E}_{s,k}|$ is equal to

(a) $64\,\mathrm{N}(h)^2$ when $\gcd(q,s) = 1$
(b) $24\,\mathrm{N}(h)^2$ when $\gcd(q,s) = \tau$, $\bar{\tau}$, or $\tau\bar{\tau}$
(c) $8\,\mathrm{N}(h)^2$ otherwise.

Furthermore, for α, $\beta\geqslant 2$, we have

(d) $\big|\mathbb{E}_{\tau^\alpha,k}\cap\mathbb{E}_{\tau\bar{\tau},k}\cap\mathbb{E}_{\bar{\tau}^\beta,k}\big| = 24\,\mathrm{N}(h)^2$.

Finally, for s different from τ and $\bar{\tau}$ and such that $\tau\bar{\tau} \nmid s$, we set $k = \mathrm{lcm}(\tau\bar{\tau}, s)h$ and we have

(e) $|\mathbb{E}_{\tau\bar{\tau},k} \cap \mathbb{E}_{s,k}| = 8\,\mathrm{N}(h)^2$.

Proof: The proof is the same as for Lemma 9. For example, for (a), we consider the nine residue classes modulo $\lambda = qs$,

$$\begin{array}{lll} \big[[-1]_q, [-1]_s\big]_\lambda & \big[[0]_q, [-1]_s\big]_\lambda & \big[[1]_q, [-1]_s\big]_\lambda \\ \big[[-1]_q, [0]_s\big]_\lambda & \big[[0]_q, [0]_s\big]_\lambda & \big[[1]_q, [0]_s\big]_\lambda \\ \big[[-1]_q, [1]_s\big]_\lambda & \big[[0]_q, [1]_s\big]_\lambda & \big[[1]_q, [1]_s\big]_\lambda \end{array}$$

and out of 81 pairs (x, y), we count 64 pairs (x, y) with $v_\tau(x, y) = v_{\bar{\tau}}(x, y) = 0$. The other results can be established in a similar way. □

6.2.3 Inclusion–Exclusion Principle

The probability $\pi_{\alpha,\beta}$ is equal to the limit of $|\mathbb{D}_{q,k}|/|\mathbb{U}_k|$ when k tends to infinity. As in the integer case, this ratio is constant for any k of the form $\tau^\gamma\bar{\tau}^\delta$ whose norm is large enough.

The cardinalities of $\mathbb{D}_{q,k}$ and of $\mathbb{U}_k$ can be determined by the inclusion–exclusion principle given by equation (8). Let us start with $|\mathbb{U}_k|$.

Lemma 18: Let $k = \tau^\alpha\bar{\tau}^\beta$ for $\alpha, \beta \geqslant 1$ then $|\mathbb{U}_k| = 9\,\mathrm{N}(k)^2/16$.

Proof: We know that there are only two residue classes modulo τ. Because $\tau \mid k$, this implies that half the representants in $\mathbb{Z}_k[\tau]$ are divisible by τ. By Lemma 12 there are $\mathrm{N}(k)$ residue classes modulo k, so it follows that there are $\mathrm{N}(k)/2$ representants that are divisible by τ, forming $\mathrm{N}(k)^2/4$ pairs with $v_\tau(x, y) > 0$. Similarly there are $\mathrm{N}(k)^2/4$ pairs with $v_{\bar{\tau}}(x, y) > 0$. Finally, there are a further $\mathrm{N}(k)/4$ representants modulo k that are divisible by $\tau\bar{\tau}$, forming $\mathrm{N}(k)^2/16$ pairs with $v_\tau(x, y) > 0$ and $v_{\bar{\tau}}(x, y) > 0$. Hence, using inclusion–exclusion, we obtain

$$\begin{aligned} |\mathbb{U}_k| &= \mathrm{N}(k)^2 - \mathrm{N}(k)^2/4 - \mathrm{N}(k)^2/4 + \mathrm{N}(k)^2/16 \\ &= 9\,\mathrm{N}(k)^2/16. \end{aligned}$$

□

We can now prove the main result.

Proof of Theorem 11: Based on Lemmas 15, 16, and 17, it is straightforward to derive $|\mathbb{D}_{q,k}|$ for a given q, using the inclusion–exclusion formula, exactly like how it was done in the proof of Theorem 1. It remains to select an appropriate k and compute the ratio $|\mathbb{D}_{q,k}|/|\mathbb{U}_k|$ with Lemma 18. For instance, to compute $|\mathbb{D}_{q,k}|$ for $q = \tau^\alpha\bar{\tau}^\beta$ with $\alpha, \beta \geqslant 2$, we remove interferences in $\mathbb{E}_{q,k}$ from $\mathbb{E}_{s,k}$ where $s = \tau q$ or $s = \bar{\tau} q$. These two sets contain all possible interferences for q as indicated by Lemma 15. Using the inclusion–exclusion principle, we have

$$|\mathbb{D}_{q,k}| = |\mathbb{E}_{q,k}| - |\mathbb{E}_{\tau q,k}| - |\mathbb{E}_{\bar{\tau} q,k}| + |\mathbb{E}_{\tau q,k} \cap \mathbb{E}_{\bar{\tau} q,k}|.$$

Fixing $k = \tau\bar{\tau}qh$, and using Lemma 16 (a), we find that $|\mathbb{E}_{q,k}| = 8 \times 16\,\mathrm{N}(h)^2$, $|\mathbb{E}_{\tau q,k}| = |\mathbb{E}_{\bar{\tau} q,k}| = 8 \times 4\,\mathrm{N}(h)^2$. Lemma 17 (c) gives $|\mathbb{E}_{\tau q,k} \cap \mathbb{E}_{\bar{\tau} q,k}| = 8\,\mathrm{N}(h)^2$. Finally, it is easy to obtain $|\mathbb{U}_k| = 9\,\mathrm{N}(q)^2\,\mathrm{N}(h)^2$, and we conclude that

$$\frac{|\mathbb{D}_{q,k}|}{|\mathbb{U}_k|} = \frac{8(16 - 4 - 4 + 1)}{9\,\mathrm{N}(q)^2} = \frac{8}{2^{2\alpha+2\beta}}$$

giving $\pi_{\alpha,\beta}$ in that case. The other cases are obtained after similar, but somewhat more tedious, computations. □

7 Experiments

We now present experimental results for the methods presented above in large characteristic p and for Koblitz curves.

Our experiments are implemented in C++ using the libraries NTL [32] and GMP [27]. The program is executed on an AMD Phenom II X6 1055T running at 2.80Ghz.

In each case, we generated 10,000 random pairs of scalars, computed their corresponding double-base expansions, and compared the complexity of each method in various situations against state of the art techniques, i.e. the JSF and τ-JSF. In the following, M, S, and I represent respectively a multiplication, a squaring, and an inversion in $\mathbb{F}_q$. The cost of any precomputation is included in our results. Indeed, most double-scalar multiplication techniques make use of the precomputed points $P + Q$ and $P - Q$. When required, we always compute those two points simultaneously in affine coordinates at a cost of $\mathrm{I} + 4\mathrm{M} + 2\mathrm{S}$.

The results are in line with Corollary 2 and Theorem 11. See Tables 2 to 5 for the actual results.

7.1 Conversion Overhead

Before we report on the actual double-scalar multiplications, let us address the complexity to convert two integers to a suitable joint extension. Our implementation of the JSF follows [5, Algorithm 9.27]. For the Joint DBC obtained by the JBT method, it is critical to carefully implement the gain as most of the time is spent evaluating this function. In fact, a very simple observation allows to greatly speed up its evaluation. Indeed, although the arguments of the gain are multiprecision integers x and y, it is enough in practice to work on their reduction modulo an integer of the form 2^a3^b that is large enough. In our implementation, we reduce x and y modulo the single precision integer $2^{12}3^{10}$. With those settings, our implementation may return a value that is strictly smaller than the actual gain, but this occurs with very small probability, in fact less than 3.22×10^{-6}. More importantly, for sizes larger than 192 bits, the result is obtained at least six times faster than with a multiprecision implementation. Our experiments indicate that computing a JSF expansion for scalars of size 192 to 320 bits takes from 264μs to 452μs on average. For the same range of sizes, the JBT method relying on a gain determined as explained above returns a JDBC expansion in only 88μs to 153μs on average, that is almost three times faster than the JSF. We believe that an implementation in hardware will perform just as well given the great simplicity of the operations involved. As a comparison, a scalar multiplication on an elliptic curve defined over a prime field of size 192 to 320 bits

represented in Jacobian coordinates takes from 1648μs to 3873μs when computed with the JSF.

Regarding Koblitz curves, it is well-known that hardware implementations perform extremely well. Finding efficient hardware implementations for conversion techniques is a research topic in its own right [12, 3]. Since we have not implemented any conversion technique in hardware in this work, we do not give a detailed comparison between Joint τ-$\bar{\tau}$ and τ-JSF methods. However, based on the simplicity of the operations involved and the analogy with the integer case, we believe that the former is faster to execute than the latter. Indeed, the same observation as for the JBT applies. Namely, instead of performing multiprecision divisions in the ring $\mathbb{Z}[\tau]$ to compute the function gain, it is much faster to reduce the arguments first modulo a suitable Kleinian integer of the form $\tau^a\bar{\tau}^b$. The subsequent operations are then simple divisions by τ and $\bar{\tau}$ of Kleinian integers of very small norm.

7.2 Joint Binary-Ternary methods in char p

In prime characteristic, we investigate two different scenarios: short Weierstrass form together with Jacobian coordinates, and Edwards curves using Inverted Edwards coordinates. Edwards curves [26, 9] allow very fast arithmetic, especially when the constants c and d are suitably chosen, but unlike short Weierstrass form, certain elliptic curves defined over $\mathbb{F}_p$ cannot be represented in Edwards form. Indeed, Edwards curves cover only roughly 40% of all isomorphism classes.

For both curves, we compared the average running costs between JSF, JBT, and JBT-Rand. To make comparisons easier, we set 1S = 0.8M, as per the usual practice. Regarding the JBT-Rand, for each size and coordinate system, we initially did some testing to estimate the value of ε minimizing the overall complexity of the scalar multiplication. Figures are given in Table 1.

TABLE 1
Values of ε used for the JBT-Rand method

	192 bits	256 bits	320 bits
Jacobian	0.84	0.88	0.90
Inverted Edwards	0.78	0.82	0.82

We then ran 10,000 experiments and for each randomly chosen pair, we repeated the computation a 1,000 times and kept the chain with minimal complexity. Regarding the computation of $P+Q$ and $P-Q$, we assume that an inversion is about 10M, giving a total complexity of 15.6M. Results are displayed in Tables 2 and 3. The average length ℓ of the representation corresponds to the number of mixed additions required plus one, whereas the average number of doublings and triplings necessary are denoted by a_ℓ and b_ℓ in the Tables.

7.2.1 Weierstrass form

The short Weierstrass form is as follows

$$y^2 = x^3 + a_4x + a_6, \quad a_4, a_6 \in \mathbb{F}_p$$

In Jacobian coordinates, a point P is represented by (X, Y, Z) and corresponds to $(X/Z^2, Y/Z^3)$ in affine whenever $Z \neq 0$.

To perform a scalar multiplication, we rely on the following fundamental operations:

- mixed addition [7M + 4S]
- doubling [2M + 8S]
- tripling [6M + 10S]

The complexity of each operation [8] is given in brackets. Doubling and tripling formulas include a multiplication by a_4 but no particular assumption is made on the value of this constant. In other words a multiplication by a_4 counts as a full multiplication.

With those figures, the Joint Binary-Ternary method gives an improvement over the JSF of about 6%, whereas the randomized version JBT-Rand shows a speedup bigger than 7.5%. See Table 2 for details.

7.2.2 Edwards Curves

Edwards curves have been introduced in [26] and generalized in [9]. In inverted Edwards coordinates [10], a point P is represented by (X, Y, Z) and corresponds to $(Z/X, Z/Y)$ in affine whenever $X, Y \neq 0$. For this exercise, we assume that multiplication by the constants c and c^2d can be neglected, for instance, because the corresponding values are small or with a very low Hamming weight.

Again, we rely on the same operations but with different complexities [8]:

- mixed addition [8M + S]
- doubling [3M + 4S]
- tripling [9M + 4S]

Not surprisingly, the improvement is more modest, around 2%, with Inverted Edwards coordinates as a doubling has a very low cost compared to a tripling. The results are shown on Table 3.

7.3 Methods for Koblitz curves

Moving on to characteristic 2, we now report our findings on Koblitz curves of the form

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

See Section 5 for a quick presentation of Koblitz curves and standard scalar multiplication techniques.

We work in López–Dahab coordinates, where a point P is represented by (X, Y, Z) and corresponds to the affine point $(X/Z, Y/Z^2)$ whenever $Z \neq 0$. We use the following operations to compute $[\eta]P + [\kappa]Q$:

- mixed addition [8M + 5S]
- Frobenius [3S]
- Verschiebung [2M + $(2 - a_2)$S]

The standard technique to compute a double-scalar multiplication is the τ-JSF. It relies on the precomputed affine points $P + Q$ and $P - Q$ and uses mixed additions and Frobenius operations. Double-base techniques also use the dual of the Frobenius, i.e. the Verschiebung. In all

our experiments, we considered the curve E_1 so the cost of $\widehat{\phi}$ is $2\mathrm{M}+\mathrm{S}$.

In the following, we investigate two different scenarios. In the first case, we assume that the cost of a Frobenius operation is negligible, either because all computations are performed in normal basis or because in polynomial basis, we first convert a field element into normal basis before performing any Frobenius operation. As a result, we assume that the Frobenius can be evaluated as many times as necessary for free. In the second case, the Frobenius although very cheap is not free. Thus we need to restrict its usage and turn to the concept of τ-JDBC and the Joint τ-$\bar{\tau}$ method.

7.3.1 Greedy τ-JDBNS and τ-DBNS

The greedy τ-JDBNS relies on the representation

$$\binom{\eta}{\kappa} = \sum_{i=1}^{\ell} \binom{c_i}{d_i} \tau^{a_i} \bar{\tau}^{b_i}, \quad \text{with } c_i, d_i \in \{-1, 0, 1\}.$$

The algorithm that returns such an expansion is presented in Section 5.3.1.

The greedy τ-DBNS produces two independent double-base expansions for η and κ

$$\eta = \sum_{i=1}^{\ell_1} c_{1,i} \tau^{a_{1,i}} \bar{\tau}^{b_{1,i}} \quad \text{and} \quad \kappa = \sum_{i=1}^{\ell_2} c_{2,i} \tau^{a_{2,i}} \bar{\tau}^{b_{2,i}}$$

with $c_{1,i}$ and $c_{2,i}$ in $\{-1, 0, 1\}$. The two expansions are then evaluated separately and added together in order to obtain $[\eta]P + [\kappa]Q$. Note that, for this method, we set $\ell = \ell_1 + \ell_2$, $a_\ell = a_{1,\ell} + a_{2,\ell}$, and $b_\ell = b_{1,\ell} + b_{2,\ell}$. To compute $[\eta]P$, we first sort the terms of the τ-DBNS expansion of η with respect to the exponents of $\bar{\tau}$ in descending order. We then apply a Horner-like scheme to obtain the successive powers of the Verschiebung. The Frobenius is applied as many times as necessary and essentially for free. See Section 5.2.3 for a concrete example. This approach can be easily generalized to the τ-JDBNS. One difference between τ-JDBNS and τ-DBNS, is that the latter does not require to precompute $P+Q$ and $P-Q$ as $[\eta]P$ and $[\kappa]Q$ are computed separately.

For both methods, it is easy to control the largest power b_ℓ of $\bar{\tau}$ in the expansions and it turns out that this parameter has a great impact on the overall complexity of $[\eta]P + [\kappa]Q$. For each size, the results displayed on Tables 4 and 5 use values of b_ℓ optimized experimentally.

Extensions degrees offering a normal basis with an efficient multiplication and that are compatible with cryptographic applications are quite rare. In the range $[160, 500]$, only 233, 239, and 359 enjoy this feat. For those extensions, a mixed addition costs $8\mathrm{M}$, a Frobenius is for free, and a Verschiebung needs $2\mathrm{M}$. Also, regarding the computations of $P+Q$ and $P-Q$, we assume that $1I = 15M$, as an inversion is usually more expensive in normal basis, giving an overall complexity of $19\mathrm{M}$. With proper parameters, tests show that both τ-JDBNS and τ-DBNS achieve a speedup of more than 35% over τ-JSF, as illustrated in Table 4.

For extension degrees other than 233, 239, and 359, we mainly work in polynomial basis. There exist normal bases, however they do not offer an efficient multiplication. So their use is limited to the evaluation of the Frobenius that is again for free. In practice, during the execution of the Horner-like scheme, the three coordinates of the point are converted to normal basis just before applying the Frobenius and then converted back to polynomial basis once the result is determined. We assume that the conversion cost is $3\mathrm{M}$ in total. We also make the standard assumption that a squaring is worth $0.1\mathrm{M}$. So a mixed additions costs $8.5\mathrm{M}$, a Frobenius is for free, a Verschiebung needs $2.1\mathrm{M}$, and we need as many conversions, worth $3\mathrm{M}$ each, as we have terms in the expansion. Finally, assuming that an inversion is worth about $10\mathrm{M}$, computing $P+Q$ and $P-Q$ requires about $14\mathrm{M}$. Results show that a speedup close to 20% is realistic in those conditions. See Table 5 for details.

7.3.2 Joint τ-$\bar{\tau}$ method

When the cost of the Frobenius cannot be neglected, we rely on the Joint τ-$\bar{\tau}$ method discussed in Section 5.3.2. Theorem 11 can be used to determine the optimal weights and we found that the choice $w_1 = 8$ and $w_2 = 3$ give reasonably good results across the board. All the operations are performed in polynomial basis. A mixed addition needs $8.5\mathrm{M}$, Frobenius requires $0.3\mathrm{M}$, and a Verschiebung $2.1\mathrm{M}$. Again, precomputing $P+Q$ and $P-Q$ requires approximately $14\mathrm{M}$. On average, the speedup over the JSF is close to 10% as shown in Table 5.

TABLE 2
JSF compared to JBT and JBT-Rand in Jacobian coordinates

	192 bits					256 bits					320 bits				
	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%
JSF	96	192	0	2598	–	128	256	0	3462	–	160	320	0	4326	–
JBT	77	104	55	2438	-6.1	102	139	74	3248	-6.1	128	173	92	4056	-6.2
JBT-Rand	73	118	46	2395	-7.8	98	155	64	3462	-7.7	123	191	81	3997	-7.6

TABLE 3
JSF compared to JBT and JBT-Rand in Inverted Edwards coordinates

	192 bits					256 bits					320 bits				
	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%
JSF	96	192	0	2042	–	128	256	0	2721	–	160	320	0	3399	–
JBT	77	104	55	2003	-1.9	102	139	74	2668	-1.9	128	173	92	3331	-2.0
JBT-Rand	74	128	40	1945	-4.7	99	166	57	2598	-4.5	124	206	72	3252	-4.3

TABLE 4
τ-JSF compared to τ-JDBNS and τ-DBNS in normal basis

	233 bits					239 bits					359 bits				
	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%
τ-JSF	117	233	0	947	–	120	239	0	971	–	180	359	0	1451	–
τ-JDBNS	70	224	12	595	-37.1	72	230	12	608	-37.3	106	348	18	895	-38.3
τ-DBNS	71	445	24	610	-35.5	73	457	24	625	-35.6	110	697	24	917	-36.8

ACKNOWLEDGMENTS

The authors would like to thank Igor Shparlinski with whom we had many discussions related to this paper.

TABLE 5

τ-JSF compared to τ-JDBNS, τ-DBNS, and Joint τ-$\bar{\tau}$ in polynomial basis

	163 bits					233 bits					347 bits				
	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%	ℓ	a_ℓ	b_ℓ	Cost	%
τ-JSF	82	163	0	752	–	117	233	0	1070	–	174	347	0	1589	–
τ-JDBNS	49	154	12	596	-20.7	70	224	12	837	-21.8	101	335	22	1221	-23.2
τ-DBNS	50	305	24	611	-18.8	71	445	24	856	-20.0	106	673	26	1255	-21.0
Joint τ-$\bar{\tau}$	60	100	58	671	-10.8	87	145	84	962	-10.1	129	217	126	1434	-9.8

REFERENCES

[1] J. Adikari, V. S. Dimitrov, and R. J. Cintra. A new algorithm for double scalar multiplication over Koblitz curves. In *International Symposium on Circuits and Systems (ISCAS 2011)*, pages 709–712. IEEE, 2011. {8}

[2] J. Adikari, V. S. Dimitrov, and L. Imbert. Hybrid binary-ternary number system for elliptic curve cryptosystems. *IEEE Trans. Computers*, 60(2):254–265, 2011. {6}

[3] J. Adikari, V. S. Dimitrov, and K. U. Järvinen. A Fast Hardware Architecture for Integer to τNAF Conversion for Koblitz Curves. *IEEE Trans. Comput.*, 61(5):732–737, May 2012. {12}

[4] R. Avanzi, C. Heuberger, and H. Prodinger. Minimality of the Hamming Weight of the τ-NAF for Koblitz Curves and Improved Combination with Point Halving. In *Proceedings of SAC 2005*, volume 3897 of *Lecture Notes in Comput. Sci.*, pages 332–344. Springer, 2006. {7}

[5] 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. {1, 11}

[6] 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. {7}

[7] 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/`. {7}

[8] D. J. Bernstein and T. Lange. Explicit-formulas database. see `http://www.hyperelliptic.org/EFD/`. {7, 12}

[9] 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. {12}

[10] 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-Verlag. {12}

[11] D. J. Bernstein and T. Lange. Type-II Optimal Polynomial Bases. In *Third International Conference on Arithmetic of Finite Fields – WAIFI 2010*, pages 41–61. Springer-Verlag, 2010. {8}

[12] B. B. Brumley and K. U. Järvinen. Conversion Algorithms and Implementations for Koblitz Curve Cryptography. *IEEE Trans. Comput.*, 59(1):81–92, January 2010. {12}

[13] 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. {2}

[14] 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. {8}

[15] H. Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer, 1996. {10}

[16] J.-S. Coron, D. M'Raïhi, and C. Tymen. Fast generation of pairs $(k, [k]P)$ for Koblitz elliptic curves. In *Proceedings of SAC 2001*, volume 2259 of *Lecture Notes in Computer Science*, pages 151–164. Springer, 2001. {8}

[17] 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}

[18] V. S. Dimitrov and T. Cooklev. Two Algorithms for Modular Exponentiation Using Nonstandard Arithmetics. *IEICE Trans. Fundamentals*, E78-A(1):82–87, 1995. {2}

[19] 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}

[20] 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. {7}

[21] V. S. Dimitrov, K. U. Järvinen, M. J. Jacobson Jr., W. F. Chan, and Z. Huang. Provably Sublinear Point Multiplication on Koblitz Curves and Its Hardware Implementation. *IEEE Trans. Comput.*, 57(11):1469–1481, November 2008. {7, 8}

[22] V. S. Dimitrov, G. A. Jullien, and W. C. Miller. An Algorithm for Modular Exponentiation. *Information Processing Letters*, 66(3):155–159, 1998. {2}

[23] 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. {2}

[24] C. Doche, T. Icart, and D. R. Kohel. Efficient Scalar Multiplication by Isogeny Decompositions. In *Public Key Cryptography – PKC 2006*, volume 3958 of *Lecture Notes in Comput. Sci.*, pages 191–206. Springer, 2006. {2}

[25] 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-Verlag, 2009. {1, 2, 3, 6, 7, 8}

[26] H. M. Edwards. A normal form for elliptic curves. *Bull. Amer. Math. Soc. (N.S.)*, 44(3):393–422 (electronic), 2007. {12}

[27] Free Software Foundation. GNU Multiple Precision library, version 5.0.4, 2012. available at `http://gmplib.org/`. {11}

[28] D. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag, Berlin, 2003. {1}

[29] N. Koblitz. CM-curves with good cryptographic properties. In Joan Feigenbaum, editor, *Advances in Cryptology - Proceedings of CRYPTO 1991*, volume 576 of *Lecture Notes in Computer Science*, pages 279–287, Berlin, 1991. Springer. {6}

[30] T. Lou, X. Sun, and C. Tartary. Bounds and Trade-offs for Double-Base Number Systems. *Information Processing Letters*, 111(10):488–493, 2011. {2}

[31] F. Morain and J. Olivos. Speeding up the Computations on an Elliptic Curve using Addition-Subtraction Chains. *Inform. Theor. Appl.*, 24:531–543, 1990. {7}

[32] V. Shoup. NTL: A Library for doing Number Theory, version 5.5.2, 2009. available at `http://www.shoup.net/ntl`. {11}

[33] J. A. Solinas. Efficient arithmetic on Koblitz curves. *Designs, Codes and Cryptography*, 19:195–249, 2000. {7}

[34] J. A. Solinas. Low-weight binary representations for pairs of integers. Combinatorics and Optimization Research Report CORR 2001-41, University of Waterloo, 2001. {2}

[35] E. G. Straus. Addition chains of vectors (problem 5125). *Amer. Math. Monthly*, 70:806–808, 1964. {2}

[36] L. C. Washington. *Elliptic Curves*. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton, FL, 2003. Number theory and cryptography. {1}

Christophe Doche received his PhD in algorithmic number theory from the Université Bordeaux I, France in 2000. He is currently senior lecturer in the Department of Computing in Macquarie University of Sydney, Australia. His research interests include number theory and elliptic curve cryptography.

Daniel Sutantyo received his MSc degree in computing from Macquarie University of Sydney, Australia in 2008. He is currently completing his PhD in the Department of Computing in Macquarie University. His research interests include cryptography, particularly elliptic curve cryptography, and number theory.