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
[2] Université de la Mediterranée, Aix-Marseille II, France
kohel@maths.usyd.edu.au
[3] Department of Mathematics and Computer Science – AceCrypt
Mount Allison University, Sackville, Canada
fsica@mta.ca

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 $\widehat{\phi}$, the dual endomorphism of the Frobenius. Namely, we propose formulae to compute $\pm\widehat{\phi}(P)$ with at most 2 multiplications and 2 squarings in the base field $\mathbb{F}_{2^d}$. This represents a speed-up of about 50% with respect to the fastest known techniques. 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.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-642-01001-9_35

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

A. Joux (Ed.): EUROCRYPT 2009, LNCS 5479, pp. 502–517, 2009.

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 on the curve can be endowed with an abelian group structure. This remarkable property has been exploited for about twenty years to implement fundamental public-key cryptographic primitives. We refer to [21] for a thorough, yet accessible, presentation of elliptic curves and to [2,16] for a discussion focused on cryptographic applications. In this context, two classes of elliptic curves are particularly relevant: those defined over a large prime field $\mathbb{F}_p$, represented either by a Weierstraß equation or an Edwards form [5] and Koblitz curves defined over $\mathbb{F}_2$.

The core operation in elliptic curve cryptography is a *scalar multiplication*, which consists in computing $[n]P$ given a point P on the curve and some integer n. Several methods exist relying on different representations of n. Among them, the *non-adjacent form* (NAF) allows to compute $[n]P$ with ℓ doublings and $\frac{\ell}{3}$ additions on average, where ℓ is the binary length of n.

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 $2^a 3^b$. 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 multiplications. For instance, in order to compute $[542788]P$ from the DBNS expansion given in Example 1, it seems we need more than 8 doublings and 7 triplings unless we can use extra storage to keep certain intermediate results. But, if we are lucky enough that the terms in the expansion can be ordered in such a way that their powers of 2 and 3 are both decreasing, then it becomes trivial to obtain $[n]P$.

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_ℓ doublings, b_ℓ triplings, $\ell - 1$ additions, and at most two registers are sufficient to compute $[n]P$. It is easy to modify the greedy algorithm to return a DBC. A tree-based algorithm has also 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 registers.

The left-to-right method, which can be seen as a Horner-like scheme, needs only one register. Simply initialize it with $[2^{a_\ell - a_{\ell-1}}3^{b_\ell - b_{\ell-1}}]P$, then add $c_{\ell-1}P$ and apply $[2^{a_{\ell-1} - a_{\ell-2}}3^{b_{\ell-1} - b_{\ell-2}}]$ to the result. Repeating this eventually gives $[n]P$, as illustrated with the chain of Example 2

$$[542788]P = [2^2]\big([2^4]\big([2^4]\big([3^2]\big([2^23]([2^2]P + P) - P\big) - P\big) + P\big) + P\big).$$

1.3 Multi-scalar Multiplication

A signature verification mainly requires a computation of the form $[n]P + [m]Q$. Obviously, $[n]P$, $[m]Q$ can be computed separately and added together at the cost of 2ℓ doublings and $\frac{2\ell}{3}$ additions on average, using the NAF and assuming that n and m are both of length ℓ. 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, a special case of an idea of Straus [20], 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 ℓ doublings and $\frac{3\ell}{4}$ additions, on average.

It is possible to do better, as shown by Solinas [19], using the redundancy and flexibility of signed-binary expansions. Indeed, the *Joint Sparse Form* (JSF) is a representation of the form

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

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 and if $\max(n, m)$ is of length ℓ, 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*

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

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

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

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 $\max(n, m)$. It is easy to find an expansion with a very low density, however, just like in the one-dimension case, *cf.* Section 1.2, it cannot be used directly together with a Horner-like scheme. 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 4. *A JDBC for* n *and* m *is as follows*

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

Based on this representation, it is now trivial to compute $[n]P + [m]Q$ *with a Horner-like scheme. Note however that the right-to-left method mentioned in Section 1.2 cannot be adapted in this context.*

Again, the greedy algorithm can be modified to return a JDBC, however, the resulting algorithm suffers from a certain lack of efficiency and is difficult to analyze. The method we discuss next is efficient, in the sense that it quickly produces 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. Thus setting $x = x - 1$ or $x = x + 1$ allows to repeat the process at the cost of an addition or a subtraction.

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 corresponds to the largest power of p that divides x and y *simultaneously*. Next, we denote by $\mathbb{X}$ the set of all pairs of positive integers (x, y) such that $v_2(x, y) = v_3(x, y) = 0$. Finally, for positive x and y, we introduce the function $\mathrm{gain}(x, y)$. We set $\mathrm{gain}(1, 1) = 0$, whereas for pairs $(x, y) \neq (1, 1)$, we define $\mathrm{gain}(x, y)$ as the largest factor $2^{v_2(x-c,y-d)}3^{v_3(x-c,y-d)}$ among all $c, d \in \{-1, 0, 1\}$.

For instance, $\mathrm{gain}(52, 45)$ is equal to 2^2, corresponding to $c = 0$ and $d = 1$. Note that this function can be implemented very efficiently, since in most cases it depends only on the remainders of x and y modulo 6.

3.1 Algorithm

Let us explain our generalization. Take two positive integers n and m. Divide by $2^{v_2(n,m)}3^{v_3(n,m)}$ in order to obtain $(x, y) \in \mathbb{X}$. The idea is then to call the function $\mathrm{gain}(x, y)$ 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 a new pair in $\mathbb{X}$ so that we can iterate the process, namely compute the corresponding

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 $y \leftarrow m/(2^{a_1}3^{b_1})$ $[(x, y) \in \mathbb{X}]$
4. **while** $x > 1$ **or** $y > 1$ **do**
5. $\quad g \leftarrow \mathrm{gain}(x, y)$ [with coefficients c_i, d_i]
6. $\quad x \leftarrow (x - c_i)/g$ and $y \leftarrow (y - d_i)/g$ $[(x, y) \in \mathbb{X}]$
7. $\quad i \leftarrow i + 1$, $a_i \leftarrow a_i + v_2(g)$, and $b_i \leftarrow b_i + v_3(g)$
8. $c_i \leftarrow x$ and $d_i \leftarrow y$ $[c_i, d_i \in \{0, 1\}]$
9. **return** $\left(\binom{c_i}{d_i}2^{a_i}3^{b_i}\right)_{\mathrm{JDBC}}$

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 $y \leqslant 1$, causing the algorithm to terminate.

Example 5. *Algorithm 1 with input* $n = 542788$ *and* $m = 462444$ *returns the following expansion*

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

Note that we have no control on the largest powers of 2 and 3 in the expansion returned by Algorithm 1. This can be a drawback since doublings and triplings have different costs in different coordinate systems. To have more control, simply modify the function gain. One possibility is to adjust the returned factor and coefficients depending on the remainders of x and y modulo a chosen constant. For instance, we can decide that when $x \equiv 4 \pmod{12}$ and $y \equiv 3 \pmod{12}$, then the gain should be 3 rather than 2^2. Doing that in each case, we can thus favor doublings or triplings. A complete analysis of the corresponding method is totally obvious. This is not the case for the general method that we address now.

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 $[n]P + [m]Q$.

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 $(x, y) \in \mathbb{X}$ and fixed values α and β, 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 $\mathbb{S}$ and dividing by the total number of pairs in $\mathbb{X} \cap \mathbb{S}$. Let $\mathbb{S}_{\gamma,\delta}$ denote the square $[1, 2^\gamma 3^\delta]^2$. The total number of pairs we investigate is given by the following Lemma.

Lemma 1. *Given two integers* γ *and* δ*, the cardinality of* $\mathbb{X} \cap \mathbb{S}_{\gamma,\delta}$*, 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 γ, δ to actually compute $p_{\alpha,\beta}$. At first glance, it seems that the square $\mathbb{S}_{\alpha+1,\beta+1}$ is a good candidate for that. In fact, we can use it provided that when we consider a larger square, say $\mathbb{S}_{\alpha+\kappa+1,\beta+\eta+1}$, the number of pairs having a gain equal to $2^\alpha 3^\beta$ and the total number of pairs in $\mathbb{X}$ 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, $\text{gain}(26, 35) = 3^2$ whereas $\text{gain}(26 + 2 \times 3^3, 35 + 5 \times 2 \times 3^3) = 2^4$. These interferences are inevitable, but intuitively, they will become less and less frequent and will eventually disappear if we scan a set large enough. The following result makes this observation more precise.

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

$$\text{gain}(x + i2^\gamma 3^\delta, y + j2^\gamma 3^\delta) = \text{gain}(x, y), \text{ for all } (i, j) \in \mathbb{Z}^2.$$

Lemma 2 gives a lower bound for $p_{\alpha,\beta}$. Indeed, let us consider a larger set $\mathbb{S}_{\gamma+\kappa,\delta+\eta}$. Then to any pair $(x, y) \in \mathbb{X} \cap \mathbb{S}_{\gamma,\delta}$ 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 $\mathbb{X} \cap \mathbb{S}_{\gamma+\kappa,\delta+\eta}$ and that have the same gain as (x, y). Conversely, if $(x_1, y_1) \in \mathbb{X} \cap \mathbb{S}_{\gamma+\kappa,\delta+\eta}$ and $\text{gain}(x_1, y_1) = 2^\alpha 3^\beta$, then (x_1, y_1) can be written $(x+i2^\gamma 3^\delta, y+j2^\gamma 3^\delta)$ with $(x, y) \in \mathbb{S}_{\gamma,\delta}$ and $\text{gain}(x, y) = \text{gain}(x_1, y_1)$. Overall, this ensures that scanning $\mathbb{S}_{\gamma,\delta}$ 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 deals 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 α, β.*

The proofs of Lemmas 2 and 3 can be found in the extended version of the article available online [15]. 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. 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 $\mathbb{S}_{\gamma,\delta}$, with γ and δ 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 $p_{\alpha,\beta}$. 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:

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

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

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

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 6. Therefore, if K denotes the average number of bits eliminated at each step of Algorithm 1, we have

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

and thanks to the values above, we obtain $K \geqslant 2.53519$. Using Lemma 3, we bound the remaining terms in the double sum to get $K \leqslant 2.53632$. 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 $\overline{\beta} \in [0.71158, 0.71183]$. To obtain the average of the largest power of 2 (respectively 3) in an expansion, we simply note that it is equal to $\overline{\alpha}$ (respectively $\overline{\beta}$) multiplied by the average length of the expansion.□

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 2 to appreciate the impact of the Joint Binary-Ternary algorithm overall 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 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. 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 [14]. Namely, instead of selecting the coefficients c, d that give the maximal gain in order to derive the next pair of integers, simply build a tree containing nodes (x, y) corresponding to all the possible choices of coefficients. The idea 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. $\quad g \leftarrow 2^{v_2(n,m)}3^{v_3(n,m)}$
4. $\quad$ Insert the child $\left(\frac{n}{g}, \frac{m}{g}\right)$ under the node (n, m)
5. **repeat**
6. $\quad$ **for** each leaf node $\mathcal{L} = (x, y)$ in $\mathcal{T}$ **do** [insert 8 children]
7. $\quad\quad$ **for** each pair $(c, d) \in \{-1, 0, 1\}^2 \setminus \{(0, 0)\}$ **do**
8. $\quad\quad\quad g_{c,d} \leftarrow 2^{v_2(x-c,y-d)}3^{v_3(x-c,y-d)}$
9. $\quad\quad\quad \mathcal{L}_{c,d} \leftarrow \left(\frac{x-c}{g_{c,d}}, \frac{y-d}{g_{c,d}}\right)$ and insert $\mathcal{L}_{c,d}$ under $\mathcal{L}$
10. $\quad$ Discard any redundant leaf node
11. $\quad$ Discard all but the B smallest leaf nodes
12. **until** a leaf node is equal to $(1, 1)$
13. **return** $\mathcal{T}$

Remarks 6

(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 α or β.
(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.

Example 7. *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 5. Applying Algorithm 2 with* $B = 4$*, we find that*

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

This last expansion still requires 11 doublings and 5 triplings but saves one addition.

Next, we describe some experiments aimed at comparing all these methods in different situations.

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 $[5]P$ and $[5]Q$ on top of $P+Q$ and $P-Q$ is denoted Tree-JBT$_5$. In this spirit Tree-JBT$_7$ needs extra points $[7]P$ and $[7]Q$, whereas Tree-JBT$_{5^2}$ needs all the possible combinations, such as $[5]P+[5]Q$, that is 10 precomputations in total. 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.

To demonstrate the validity of our approach, we compare it against the Joint Sparse Form and the hybrid method [1]. These methods have the best known density when using respectively two and 14 precomputed points. Furthermore, we performed computations using *inverted Edwards coordinates* [7]. This system offers so efficient doublings that it makes a Double-Base approach irrelevant for single scalar multiplications [6]. Indeed, with this system a doubling can be

Table 1. Parameters of JDBC obtained by various methods

	Size	192 bits			256 bits			320 bits			384 bits			448 bits			512 bits		
Method	$\#\mathcal{P}$	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	$b_\ell\ell$	a_ℓ	b_ℓ	
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$_{52}$	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

obtained with 3M + 4S, a mixed addition with 8M + S, and a tripling with 9M + 4S, where M and S represent respectively a multiplication and a squaring in $\mathbb{F}_p$. To ease the comparisons, we make the usual assumption that 1S $\approx$ 0.8M.

Table 2 gives the overall number of multiplications needed for a scalar multiplication with a particular method, using inverted Edwards coordinates. These tests show that the Joint Binary-Ternary 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. Nevertheless, they are all faster when counting the costs of precomputations, as shown in Table 2. One can check that this is still the case even when those costs are ignored.

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

	Size	192 bits	256 bits	320 bits	384 bits	448 bits	512 bits
Method	$\#\mathcal{P}$	N_{M}	N_{M}	N_{M}	N_{M}	N_{M}	N_{M}
JSF	2	2044	2722	3401	4062	4758	5436
JBT	2	2004	2668	3331	3995	4664	5322
Tree-JBT	2	1953	2602	3248	3896	4545	5197
Tree-JBT$_5$	4	1920	2543	3168	3792	4414	5042
Tree-JBT$_7$	6	1907	2521	3137	3753	4365	4980
Tree-JBT$_{52}$	10	1890	2485	3079	3677	4270	4862
Hybrid	14	2047	2679	3311	3943	4575	5207

To conclude, note that JDBC expansions are relevant for scalar multiplications as well. Namely, if we want to compute $[n]P$, one possibility is to split n as $n_0 + 2^A 3^B n_1$, where A and B are fixed constants chosen so that n_0 and n_1 have approximately the same size and also to adjust the number of doublings and triplings. Then run Algorithm 1 or 2 to find a chain computing n_0 and n_1 simultaneously and derive $[n]P$ as $[n_0]P + [n_1]Q$, where $Q = [2^A 3^B]P$.

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,

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

there exists a nontrivial endomorphism, the *Frobenius* denoted by ϕ 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 τ such that $\tau^2 - \mu\tau + 2 = 0$ represents ϕ. If an integer n is equal to some polynomial in τ, then the endomorphism $[n]$ will be equal to the same polynomial in ϕ. The elements of the ring $\mathbb{Z}[\tau]$, called *Kleinian integers* [12], thus play a key role in scalar multiplications on Koblitz curves.

4.1 Representation of Kleinian Integers

It is easy to show that $\mathbb{Z}[\tau]$ is an Euclidean ring and thus any element $\eta \in \mathbb{Z}[\tau]$ has a τ*-adic representation* of the form

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

There are also signed-digit representations and among them, the τ-NAF has a distinguished status, achieving an optimal density of $\frac{1}{3}\cdot$ Its generalization, the τ-NAF$_w$, has an average density of $\frac{1}{w+1}$ for $2^{w-2} - 1$ precomputed points.

In [3,4,12], 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 τ-DBNS expansion of the form

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

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 [18] to compute repeated applications of the Frobenius for free. This approach is described in [17].

Since implementing these conversion techniques can be challenging, especially on devices with limited capabilities, we will not follow this path and introduce instead the concept of τ*-Double-Base Chains* (τ-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 τ-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 τ-adic representation of η 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 ℓ of the τ-adic expansion of η is twice as long 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 δ of length half then length of η. From now on, we assume that this reduction has been done and that the length of η is approximately d.

Computing $[\eta]P$ with the τ-NAF involves $\frac{d}{3}$ additions on average and d Frobenius that need at most $3d$ squarings in $\mathbb{F}_{2^d}$. With the τ-NAF$_w$, we need $2^{w-2} - 1 + \frac{d}{w+1}$ additions, the same amount of Frobenius, and some memory to store $2^{w-2} - 1$ precomputed points. The complexity of the τ-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 τ-DBC is much more difficult to analyze. Only some experiments give an indication of its performance, and tests show that the τ-DBC cannot compete, for instance with the τ-NAF. The problem comes from the cost of the second endomorphism that is too expensive to balance the saving induced on the number of additions. To make use of the τ-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 $\widehat{\phi}$.

Obviously, we can implement $\widehat{\phi}(P) = \mu P - \phi(P)$ with a subtraction and, in López–Dahab coordinates, this corresponds to the cost of a mixed addition, *i.e.* 8M + 5S, 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 $\widehat{\phi}$ by the halving map using the equation $\phi\widehat{\phi}(P) = [2]P$. A halving works on the point $P = (x_1, y_1)$ represented as (x_1, λ_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, *cf.* [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 4M + 4S, is a reasonable estimate. This is still too expensive to justify the use of the τ-DBC to compute a scalar multiplication. We show next how to compute $\pm\widehat{\phi}(P)$ in a much more efficient way.

4.3 Fast Evaluation of $\widehat{\phi}$

In this part, we show how to compute $\pm\widehat{\phi}(P)$ 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} and let $P_2=\widehat{\phi}(P_1)$. Then the López–Dahab coordinates of P_2, namely $(X_2:Y_2:Z_2)$ satisfy*

$$X_2=(X_1+Z_1)^2,\quad Z_2=X_1Z_1,$$
$$Y_2=\big(Y_1+(1-a_2)X_2\big)\big(Y_1+a_2X_2+Z_2\big)+(1-a_2)Z_2^2.$$

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

The proof can be found in the extended version of the article [15].

This new way to compute $\widehat{\phi}$ is also beneficial to the τ-DBNS, especially regarding the algorithm described in [3]. A direct application of the formulae above induces 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 τ-adic Joint Sparse Form, τ-JSF [9]. The τ-JSF and the JSF have very similar definitions. For instance, they have the same average joint density, that is $\frac{1}{2}\cdot$ However the optimality of the JSF does not carry over to the τ-JSF. Namely, for certain pairs in $\mathbb{Z}[\tau]$, the joint density of the τ-JSF expansion is not minimal across all the signed τ-adic expansions computing this pair.

Now, let us explain how we can produce joint τ-DBNS expansions and more importantly joint τ-DBC. The generalization of the greedy-type method is straightforward. At each step, find the closest approximation of (η,κ) 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 $d\big((\eta,\kappa),(\eta',\kappa')\big)=\sqrt{\mathrm{N}(\eta-\eta')^2+\mathrm{N}(\kappa-\kappa')^2}$, where $\mathrm{N}(\cdot)$ is the norm in $\mathbb{Z}[\tau]$. Then subtract the closest approximation and repeat the process until we reach $(0,0)$. To find a joint τ-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 gain(η,κ) 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 as it is suggested at the end of Section 3.1. For each choice of the function gain, there is a corresponding algorithm, that should be analyzed quite easily, following the integer case. We decided to run some experiments first to inform us on the optimal choices for the function gain. A summary is detailed next.

4.5 Experiments

We have run some tests to compare the τ-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_ℓ and b_ℓ corresponding respectively to the number of additions, the number of applications of ϕ and of $\widehat{\phi}$, as well as the total number of multiplications N_{M} in $\mathbb{F}_{2^d}$ needed to perform a multi-scalar multiplication for the corresponding size. 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 improvements regarding scalar multiplications of 8 to 9% in total over the τ-JSF.

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

	163 bits			233 bits			283 bits			347 bits			4409 bits			571 bits		
Method	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	b_ℓ	ℓ	a_ℓ	$b_\ell\ell$	a_ℓ	b_ℓ	
τ-JSF	82	163	0	117	233	0	142	283	0	174	347	0	205	409	0	286	571	0
N_{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_{M}		671			955			1154			1410			1665			2318	

References

1. Adikari, J., Dimitrov, V., Imbert, L.: Hybrid Binary-Ternary Joint Sparse Form and its Application in Elliptic Curve Cryptography, `http://eprint.iacr.org/2008/`
2. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Nguyen, K., Lange, T., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton (2005)
3. Avanzi, R.M., Dimitrov, V.S., Doche, C., Sica, F.: Extending scalar multiplication using double bases. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 130–144. Springer, Heidelberg (2006)
4. Avanzi, R.M., Sica, F.: Scalar multiplication on koblitz curves using double bases. In: Nguyên, P.Q. (ed.) VIETCRYPT 2006. LNCS, vol. 4341, pp. 131–146. Springer, Heidelberg (2006)
5. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer, Heidelberg (2007)
6. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: Optimizing double-base elliptic-curve single-scalar multiplication. In: Srinathan, K., Rangan, C.P., Yung, M. (eds.) INDOCRYPT 2007. LNCS, vol. 4859, pp. 167–182. Springer, Heidelberg (2007)
7. Bernstein, D.J., Lange, T.: Explicit-formulas database, `http://www.hyperelliptic.org/EFD/`
8. Ciet, M., Joye, M., Lauter, K., Montgomery, P.L.: Trading Inversions for Multiplications in Elliptic Curve Cryptography. Des. Codes Cryptogr. 39(2), 189–206 (2006)

9. Ciet, M., Lange, T., Sica, F., Quisquater, J.-J.: Improved algorithms for efficient arithmetic on elliptic curves using fast endomorphisms. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 388–400. Springer, Heidelberg (2003)
10. Dimitrov, V.S., Cooklev, T.: Hybrid Algorithm for the Computation of the Matrix Polynomial $I+A+\cdots+A^{N-1}$. IEEE Trans. on Circuits and Systems 42(7), 377–380 (1995)
11. Dimitrov, V.S., Imbert, L., Mishra, P.K.: Efficient and secure elliptic curve point multiplication using double-base chains. In: Roy, B. (ed.) ASIACRYPT 2005. LNCS, vol. 3788, pp. 59–78. Springer, Heidelberg (2005)
12. Dimitrov, V.S., Järvinen, K.U., Jacobson Jr., M.J., Chan, W.F., Huang, Z.: FPGA implementation of point multiplication on koblitz curves using kleinian integers. In: Goubin, L., Matsui, M. (eds.) CHES 2006. LNCS, vol. 4249, pp. 445–459. Springer, Heidelberg (2006)
13. Dimitrov, V.S., Jullien, G.A., Miller, W.C.: An Algorithm for Modular Exponentiation. Information Processing Letters 66(3), 155–159 (1998)
14. Doche, C., Habsieger, L.: A tree-based approach for computing double-base chains. In: Mu, Y., Susilo, W., Seberry, J. (eds.) ACISP 2008. LNCS, vol. 5107, pp. 433–446. Springer, Heidelberg (2008)
15. Doche, C., Kohel, D.R., Sica, F.: Double-Base Number System for Multi-Scalar Multiplications, `http://eprint.iacr.org/2008/`
16. Hankerson, D., Menezes, A.J., Vanstone, S.A.: Guide to Elliptic Curve Cryptography. Springer, Heidelberg (2003)
17. Okeya, K., Takagi, T., Vuillaume, C.: Short memory scalar multiplication on koblitz curves. In: Rao, J.R., Sunar, B. (eds.) CHES 2005. LNCS, vol. 3659, pp. 91–105. Springer, Heidelberg (2005)
18. Park, D.J., Sim, S.G., Lee, P.J.: Fast scalar multiplication method using change-of-basis matrix to prevent power analysis attacks on koblitz curves. In: Chae, K.-J., Yung, M. (eds.) WISA 2003. LNCS, vol. 2908, pp. 474–488. Springer, Heidelberg (2004)
19. Solinas, J.A.: Low-weight binary representations for pairs of integers. Combinatorics and Optimization Research Report CORR 2001-41, University of Waterloo (2001)
20. Straus, E.G.: Addition chains of vectors (problem 5125). Amer. Math. Monthly 70, 806–808 (1964)
21. Washington, L.C.: Elliptic Curves. Discrete Mathematics and its Applications (Boca Raton). Chapman & Hall/CRC, Boca Raton (2003); number theory and cryptography