Fast point multiplication algorithms for binary elliptic curves with and without precomputation

Thomaz Oliveira[1], Diego F. Aranha[2], Julio López[2⋆], and Francisco Rodríguez-Henríquez[1]

[1] Computer Science Department, CINVESTAV-IPN
[2] Institute of Computing, University of Campinas

Abstract. In this paper we introduce new methods for computing constant-time variable-base point multiplications over the Galbraith-Lin-Scott (GLS) and the Koblitz families of elliptic curves. Using a left-to-right double-and-add and a right-to-left halve-and-add Montgomery ladder over a GLS curve, we present some of the fastest timings yet reported in the literature for point multiplication. In addition, we combine these two procedures to compute a multi-core protected scalar multiplication. Furthermore, we designed for the first time a regular τ-adic scalar expansion for Koblitz curves. As a result, using the regular recoding approach, we set the speed record for a single constant-time point multiplication on standardized binary elliptic curves at the 128-bit security level.

Keywords: binary elliptic curves, scalar multiplication, software implementation

1 Introduction

From the cryptographic perspective, one of the most interesting consequences of the Snowden revelations, is the increased awareness about the importance of implementing security protocols that offer the Perfect Forward Secrecy (PFS) property. The PFS property guarantees that in a given protocol, none of its past short term session keys can be derived from the long term server's private key. One tangible example of this situation is the recent announcement by the Internet Engineering Task Force that the Transport Layer Security (TLS) protocol version 1.3, will no longer include cipher suites based on RSA key transport primitives [33]. Instead, the client-server secret key establishment will be performed via either the Ephemeral Diffie-Hellman or the Elliptic Curve Ephemeral Diffie-Hellman (ECDHE) methods. Because of the significant performance advantage of the latter over the former, it is anticipated that in the years to come, ECDHE will be the favorite choice for establishing a TLS shared secret.

The specifications of all the TLS protocol versions [8–10] include support for prime and binary field elliptic curve cryptographic primitives. In the case of binary elliptic curves, the TLS protocol supports a selection of several standardized random curves as well as Koblitz curves [23] at the 80-, 128-, 192- and 256-bit security levels. Koblitz curves allow performance improvements, due to the availability of the Frobenius automorphism τ. Also, their generation is inherently *rigid* (in the SafeCurves sense [2]), where the only degree of freedom in the curve generation process consists in choosing a suitable prime degree extension m that produces a curve with almost-prime order. This severely limits the possibility of "1-in-a-million attacks" [34] aiming to reach a weak curve after testing many random seeds.

Being point multiplication the single most important operation of (hyper) elliptic curve cryptography, considerable effort has been directed towards achieving fast and compact software/hardware implementations of it. A major result that has influenced the latest implementations was found in 2009, when Galbraith, Lin and Scott (GLS), building on a previous technique introduced by Gallant, Lambert and Vanstone (GLV) [14], constructed efficient endomorphisms for a class of elliptic curves

⋆ The author was supported in part by the Intel Labs University Research Office.

defined over the quadratic field $\mathbb{F}_{q^2}$, where q is a prime number [13]. Taking advantage of this finding, the authors of [13] performed a 128-bit security level point multiplication that took $326,000$ clock cycles on a 64-bit processor. Since then, a steady stream of algorithmic and technological advances has translated into a significant reduction in the number of clock cycles required to compute a (hyper) elliptic curve constant-time variable-base-point multiplication at the 128-bit security level [1, 11, 24, 5, 4, 16, 37].

The authors of [24, 11] targeted a twisted Edwards GLV-GLS curve defined over $\mathbb{F}_{p^2}$, with $p = 2^{127} - 5997$. That curve is equipped with a fourth degree endomorphism allowing a fast point multiplication computation that required just $92,000$ clock cycles on an Ivy Bridge processor [11]. Bos *et al.* [5] and Bernstein *et al.* [1], presented an efficient point multiplication on the Kummer surface associated with the Jacobian of a genus 2 curve defined over a field generated by the prime $p = 2^{127} - 1$. Each iteration of the Montgomery ladder presented in [1] costs roughly 25 field multiplications, which implemented on a Haswell processor permits to compute a point multiplication in $72,000$ clock cycles.

In 2014, Oliveira *et al.* introduced the λ-projective coordinate system that leads to faster binary field elliptic curve arithmetic [30, 31]. The authors applied that coordinate system into a binary GLS curve that admits a second degree endomorphism and a fast field arithmetic associated with the quadratic field extension of the binary field $\mathbb{F}_{2^{127}}$. When implemented on a Haswell processor, this approach permits to perform one constant-time point multiplication computation in just $60,000$ clock cycles.

Contributions of this paper. This work presents new methods aimed to perform fast constant-time variable-base-point multiplication computation for both, random and Koblitz binary elliptic curves of the form $y^2 + xy = x^3 + ax^2 + b$. In the case of random binary elliptic curves, we introduce a novel right-to-left variant of the classical Montgomery-López-Dahab ladder algorithm presented in [25], which efficiently adapted the original ladder idea introduced by Peter Montgomery in his 1987 landmark paper [26]. The new variant presented in this work does not require point doublings, but instead, it uses the efficient point halving operation available on binary elliptic curves. In contrast with the algorithm presented in [25] that does not admit the benefit of precomputed tables, our proposed variant *can* take advantage of this technique, a feature that could be proved valuable for the fixed-base-point multiplication scenario. Moreover, we show that our new right-to-left Montgomery ladder formulation can be nicely combined with the classical ladder to attain a high parallel acceleration factor for a constant-time multi-core implementation of the point multiplication operation. As a second contribution, we present a procedure that adapts the regular scalar recoding of [21], to the task of producing a regular τ-NAF scalar recoding for Koblitz curves. To the best of our knowledge, a regular τ-NAF scalar recoding has not been reported in the open literature before this work. This approach allows us to achieve a performance that sets a speed record for constant-time point multiplication on standardized binary elliptic curves at the 128-bit security level.

The remainder of this paper is organized as follows. In Section 2 we give a short description of the GLS and Koblitz curves, their arithmetic and their security. In Section 3 we present new variants of the Montgomery ladder for binary elliptic curves. Then, in Section 4, we introduce a regular τ-NAF recoding amenable for producing protected point multiplication implementations on Koblitz curves. In Section 5, we present our experimental implementation results and finally, we draw our conclusions in Section 6.

2 Mathematical background

2.1 Quadratic field arithmetic

A binary extension field $\mathbb{F}_q$, $q = 2^m$, can be constructed by taking an m-degree polynomial $f(x) \in \mathbb{F}_2[x]$ irreducible over $\mathbb{F}_2$, where the field elements in $\mathbb{F}_q$ are the set of binary polynomials of degree

less than m. Quadratic extensions of a binary extension field can be built using a degree two monic polynomial $g(u) \in \mathbb{F}_2[u]$ irreducible over $\mathbb{F}_q$. In this case, the field $\mathbb{F}_{q^2}$ is isomorphic to $\mathbb{F}_q[u]/(g(u))$ and its elements can be represented as $a_0 + a_1 u$, with $a_0, a_1 \in \mathbb{F}_q$. Operations in the quadratic extension are performed coefficient-wise. For instance, the multiplication of two elements $a, b \in \mathbb{F}_{q^2}$ is computed at the cost of three multiplications in the base field using the customary Karatsuba formulation,

$$\begin{aligned} a \cdot b &= (a_0 + a_1 u) \cdot (b_0 + b_1 u) \\ &= (a_0 b_0 + a_1 b_1) + (a_0 b_0 + a_1 b_1 + (a_0 + a_1) \cdot (b_0 + b_1))u, \end{aligned} \tag{1}$$

with $a_0, a_1, b_0, b_1 \in \mathbb{F}_q$.

In [30, 31], the authors developed an efficient software library for the field $\mathbb{F}_{2^m}$ and its quadratic extension $\mathbb{F}_{2^{2m}}$, with $m = 127$, generated by means of the irreducible trinomials $f(x) = x^{127} + x^{63} + 1$ and $g(u) = u^2 + u + 1$, respectively. The computational cost of the field arithmetic in the quadratic extension field gets significantly reduced by using that towering approach. To be more concrete, let M and m denote the cost of one field multiplication and squaring over $\mathbb{F}_{q^2}$ and $\mathbb{F}_q$, respectively. The execution of the arithmetic library of [31] on the Sandy Bridge and Haswell microprocessors yields a ratio M/m of just 2.23 and 1.51, respectively. These experimental results are considerably better than the theoretical ratio $M/m = 3$ that one could expect from the Karatsuba formulation of Eq (1). The aforementioned performance speedup can be explained from the fact that the towering field approach permits a much better usage of the processor's pipelined execution unit, which potentially can improve the speed of one 64-bit carry-less multiplication[3] from 7 clock cycles to the maximum achievable throughput of just 2 clock cycles [12]. Moreover, the nice form of the irreducible polynomial $f(x)$ permits a highly efficient reduction that could not be achieved by a direct approach that would define the field $\mathbb{F}_{2^{2m}}$ using a $2m$-degree polynomial irreducible over $\mathbb{F}_2$.

2.2 GLS binary elliptic curves

Let $E_{a,b}(\mathbb{F}_{q^2})$ denote the additive abelian group formed by the point at infinity $\mathcal{O}$ and the set of affine points $P = (x, y)$ with $x, y \in \mathbb{F}_{q^2}$ that satisfy the ordinary binary elliptic curve equation given as,

$$E : y^2 + xy = x^3 + ax^2 + b, \tag{2}$$

defined over $\mathbb{F}_{q^2=2^{2m}}$, with $a \in \mathbb{F}_{q^2}$ and $b \in \mathbb{F}^*_{q^2}$. Let $\#E_{a,b}(\mathbb{F}_{q^2})$ denote the size of the group $E_{a,b}(\mathbb{F}_{q^2})$, and let us assume that $E_{a,b}(\mathbb{F}_{q^2})$ includes a subgroup $\langle P \rangle$ of prime order r.

The point multiplication operation, denoted by $Q = kP$, corresponds to adding P to itself $k - 1$ times, with $k \in [0, r - 1]$. The average cost of computing kP by a random n-bit scalar k using the traditional double-and-add method is about $nD + \frac{n}{2}A$, where D and A are the cost of doubling and adding a point, respectively. If the elliptic curve E of Eq. (2) is equipped with a non-trivial efficiently computable endomorphism ψ such that $\psi(P) = \delta P \in \langle P \rangle$, for some $\delta \in [2, r - 2]$. Then the point multiplication can be computed *à la* GLV as,

$$Q = kP = k_1 P + k_2 \psi(P) = k_1 P + k_2 \cdot \delta P,$$

where the subscalars $|k_1|, |k_2| \approx n/2$, can be found by solving a closest vector problem in a lattice [13]. Having split the scalar k into two parts, the computation of $kP = k_1 P + k_2 \psi(P)$ can be performed by applying simultaneous multiple point multiplication techniques [18] that translates into a saving of half of the doublings required by the execution of a single point multiplication kP.

Inspired by the GLS technique of [13], Hankerson, Karabina and Menezes presented in [17] a family of binary GLS curves defined over the field $\mathbb{F}_{q^2}$, with $q = 2^m$, which admits a two-dimensional

[3] corresponding to the Intel's PCLMULQDQ instruction.

endomorphism. This endomorphism can be computed at the inexpensive cost of just three additions in $\mathbb{F}_q$. Furthermore, by carefully choosing the elliptic curve parameters a, b of Eq. (2), the authors of [17] showed that it is possible to find members of that family of GLS curves with an almost-prime group order of the form $\#E_{a,b}(\mathbb{F}_{q^2}) = hr$, with $h = 2$ and where r is a $(2m-1)$-bit prime number.

Security of GLS curves Given a point $Q \in \langle P \rangle$, the Elliptic Curve Discrete Logarithm Problem (ECDLP) consists of finding the unique integer $k \in [0, r-1]$ such that $Q = kP$. To the best of our knowledge, the most powerful attack for solving the ECDLP on binary elliptic curves was presented in [32] (see also [20, 35]), with an associated computational complexity of $O(2^{c \cdot m^{2/3} \log m})$, where $c < 2$, and where m is a prime number. This is worse than generic algorithms with time complexity $O(2^{n/2})$ for all prime field extensions m less than $N = 2000$, a bound that is well above the range used for performing elliptic curve cryptography [32]. On the other hand, since the elliptic curve of Eq. (2) is defined over a quadratic extension of the field $\mathbb{F}_q$, the generalized Gaudry-Hess-Smart (gGHS) attack [15, 19] to solve the ECDLP on the curve E, applies. To prevent this attack, it suffices to verify that the constant b of $E_{a,b}(\mathbb{F}_{q^2})$ is not weak. Nevertheless, the probability that a randomly selected $b \in \mathbb{F}_q^*$ is a weak parameter, is negligibly small [17].

2.3 Koblitz curves

A Koblitz curve, also known as an anomalous binary curve or subfield curve, is defined as the set of affine points $P = (x, y) \in \mathbb{F}_q \times \mathbb{F}_q$, $q = 2^m$, that satisfy the Weierstraβ equation $E_a : y^2 + xy = x^3 + ax^2 + 1$, $a \in \{0, 1\}$, together with a point at infinity denoted by $\mathcal{O}$. In λ-affine coordinates where the points are represented as $P = (x, \lambda = x + \frac{y}{x})$, $x \neq 0$, the λ-affine form of the above equation becomes [31], $(\lambda^2 + \lambda + a)x^2 = x^4 + 1$. A Koblitz curve forms an abelian group denoted as $E_a(\mathbb{F}_{2^m})$ of order $2(2-a)r$, for an odd prime r, where its group law is defined by the point addition operation.

Frobenius map. Since their introduction in [23], Koblitz curves were extensively studied for their additional structure that allows, in principle, a performance speedup in the point multiplication computation. The Frobenius map $\tau : E_a(\mathbb{F}_q) \to E_a(\mathbb{F}_q)$ defined by $\tau(\mathcal{O}) = \mathcal{O}$, $\tau(x, y) = (x^2, y^2)$, is a curve automorphism satisfying $(\tau^2 + 2)P = \mu\tau(P)$ for $\mu = (-1)^{1-a}$ and all $P \in E_a(\mathbb{F}_q)$. By solving the equation $\tau^2 + 2 = \mu\tau$, the Frobenius map can be seen as the complex number $\tau = \frac{\mu + \sqrt{-7}}{2}$. Notice that in λ-coordinates the Frobenius map action remains the same, because, $\tau(x, \lambda) = (x^2, \lambda^2) = (x^2, x^2 + \frac{y^2}{x^2})$, which corresponds to the λ-representation of $\tau(x, y)$. Let $\mathbb{Z}[\tau]$ be the ring of polynomials in τ with coefficients in $\mathbb{Z}$. Since the Frobenius map is highly efficient, as long as it is possible to convert an integer scalar k to its τ-representation $k = \sum_{i=0}^{l-1} u_i \tau^i$, its action can be exploited in a point multiplication computation by adding multiples $u_i\tau^i(P)$, with $u_i\tau^i \in \mathbb{Z}[\tau]$. Solinas [36] proposed exactly that, namely, a τ-adic scalar recoding analogous to the signed digit scalar *Non-Adjacent Form* representation.

Security of Koblitz curves From the security point of view, it has been argued that the availability of additional structure in the form of endomorphisms can be a potential threat to the hardness of elliptic curve discrete logarithms [3], but limitations observed in approaches based on isogeny walks is evidence contrariwise [22]. Furthermore, the generation of Koblitz curves satisfy by definition the *rigidity* property. Constant-time compact implementations for Koblitz curves are also easily obtained by specializing the Montgomery-López-Dahab ladder algorithm [25] for $b = 1$, although we show below that this is not the most efficient constant-time implementation strategy possible. Another practical advantage is the adoption of Koblitz curves by several standards bodies [27], which guarantee interoperability and availability of implementations in many hardware and software platforms.

3 New Montgomery ladder variants

This Section presents algorithms for computing the scalar multiplication through the Montgomery ladder method. Here, we let P be a point in a binary elliptic curve and k a scalar of bit length n. Our objective is to compute $Q = kP$.

Algorithm 1 Left-to-right Montgomery ladder [26]

Input: $P = (x, y)$, $k = (1, k_{n-2}, \ldots, k_1, k_0)$
Output: $Q = kP$
1: $R_0 \leftarrow P$; $R_1 \leftarrow 2P$;
2: **for** $i = n - 2$ **downto** 0 **do**
3: **if** $k_i = 1$ **then**
4: $R_0 \leftarrow R_0 + R_1$; $R_1 \leftarrow 2R_1$
5: **else**
6: $R_1 \leftarrow R_0 + R_1$; $R_0 \leftarrow 2R_0$
7: **end if**
8: **end for**
9: **return** $Q = R_0$

Algorithm 1 describes the classical left-to-right Montgomery ladder approach for point multiplication [26], whose key algorithmic idea is based on the following observation. Given a base point P and two input points R_0 and R_1, such that their difference, $R_0 - R_1 = P$, is known, the x-coordinates of the points, $2R_0$, $2R_1$ and $R_0 + R_1$, is fully determined by the x-coordinates of P, R_0 and R_1.

More than one decade after its original proposal in [26], López and Dahab presented in [25] an optimized version of the Montgomery ladder, which was specifically crafted for the efficient computation of point multiplication on ordinary binary elliptic curves. In this scenario, compact formulae for the point addition and point doubling operations of Algorithm 1 can be derived from the following result.

Lemma 1 ([25]). *Let $P = (x, y)$, $R_1 = (x_1, y_1)$, and $R_0 = (x_0, y_0)$ be elliptic curve points, and assume that $R_1 - R_0 = P$, and $x_0 \neq 0$. Then, the x-coordinate of the point $(R_0 + R_1)$, x_3, can be computed in terms of x_0, x_1, and x as follows,*

$$x_3 = \begin{cases} x + \frac{x_0 \cdot x_1}{(x_0 + x_1)^2} & R_0 \neq \pm R_1 \\ x_0^2 + \frac{b}{x_0^2} & R_0 = R_1 \end{cases} \tag{3}$$

Moreover, the y-coordinate of R_0 can be expressed in terms of P, and the x-coordinates of R_0, R_1 as,

$$y_0 = x^{-1}(x_0 + x)\left[(x_0 + x)(x_1 + x) + x^2 + y\right] + y \tag{4}$$

Let us denote the projective representation of the points R_0, R_1 and $R_0 + R_1$, without considering their y-coordinates as, $R_0 = (X_0, -, Z_0)$ $R_1 = (X_1, -, Z_1)$ and $R_0 + R_1 = (X_3, -, Z_3)$. Then, for the case $R_0 = R_1$, Lemma 1 implies,

$$\begin{cases} X_3 = X_0^4 + b \cdot Z_0^4 \\ Z_3 = X_0^2 \cdot Z_0^2 \end{cases} \tag{5}$$

Furthermore, for the case $R_0 \neq \pm R_1$, one has that,

$$\begin{cases} Z_3 = (X_0 \cdot Z_1 + X_1 \cdot Z_0)^2 \\ X_3 = x \cdot Z_3 + (X_0 \cdot Z_1) \cdot (X_1 \cdot Z_0) \end{cases} \tag{6}$$

From Equations (5) and (6) it follows that the computational cost of each ladder step in Algorithm 1 is of 5 multiplications, 1 multiplication by the curve b-constant, 4 or 5 squarings[4] and 3 additions over the binary extension field where the elliptic curve has been defined.

In the rest of this Section, we will present a novel right-to-left formulation of the classical Montgomery ladder.

3.1 Right-to-left double-and-add Montgomery-LD ladder

Algorithm 2 presents a right-to-left version of the classical Montgomery ladder procedure. At the end of the i-th iteration, the points in the variables R_0, R_1 are, $R_0 = 2^{i+1}P$, and $R_1 = \ell P + \frac{P}{2}$, where ℓ is the integer represented by the i rightmost bits of the scalar k. The variable R_2 maintains the relationship, $R_2 = R_0 - R_1$ from the initialization (step 1), until the execution of the last iteration of the main loop (steps 2-9). This comes from the fact that at each iteration, if $k_i = 1$, then the difference $R_0 - R_1$ remains unchanged. If otherwise, $k_i = 0$, then both R_2 and R_0 are updated with their respective original values plus R_0, which ensures that $R_2 = R_0 - R_1$, still holds. Notice however that, although the difference $R_2 = R_0 - R_1$, is known, it may vary throughout the iterations.

Algorithm 2 Montgomery-LD double-and-add scalar multiplication (right-to-left)

Input: $P = (x, y)$, $k = (k_{n-1}, k_{n-2}, \ldots, k_1, k_0)$
Output: $Q = kP$
1: $R_0 \leftarrow P$; $R_1 \leftarrow \frac{P}{2}$; $R_2 \leftarrow \frac{P}{2} = (R_0 - R_1)$;
2: **for** $i = 0$ **to** $n - 1$ **do**
3: **if** $k_i = 1$ **then**
4: $R_1 \leftarrow R_1 + R_0$;
5: **else**
6: $R_2 \leftarrow R_2 + R_0$;
7: **end if**
8: $R_0 \leftarrow 2R_0$;
9: **end for**
10: **return** $Q = R_1 - \frac{P}{2}$

As stated in Lemma 1, the point additions of steps 4 and 6 in Algorithm 2 can be computed using the x-coordinates of the points R_0, R_1 and R_2, according to the following analysis. If $k_i = 1$, then the x-coordinate of $R_0 + R_1$ is a function of the x-coordinates of R_0, R_1 and R_2, because $R_2 = R_0 - R_1$. If $k_i = 0$, the x-coordinate of $R_2 + R_0$ is a function of the x-coordinates of the points R_0, R_1 and R_2, because $R_0 - R_2 = R_0 - (R_0 - R_1) = R_1$. Hence, considering the projective representation of the points $R_0 = (X_0, -, Z_0)$, $R_1 = (X_1, -, Z_1)$, $R_2 = (X_2, -, Z_2)$ and $R_0 + R_1 = (X_3, -, Z_3)$, where all the y-coordinates are ignored, and assuming $R_0 \neq \pm R_1$, we have,

$$\begin{cases} T = (X_0 \cdot Z_1 + X_1 \cdot Z_0)^2 \\ Z_3 = Z_2 \cdot T \\ X_3 = X_2 \cdot T + Z_2 \cdot (X_0 \cdot Z_1) \cdot (X_1 \cdot Z_0) \end{cases} \tag{7}$$

From Equations (5) and (7), it follows that the computational cost of each ladder step in Algorithm 2 is of 7 multiplications, 1 multiplication by the curve b-constant, 4 or 5 squarings and 3 additions over the binary field where the elliptic curve lies.

[4] Either $b = 1$ or $\sqrt{b}$ is precomputed. Formula (5) can also be computed as $Z_3 = (X_0 \cdot Z_0)^2$ and $X_3 = (X_0^2 + \sqrt{b} \cdot Z_0^2)^2$

Although conceptually simple, the above method has several algorithmic and practical shortcomings. The most important one is the difficulty to recover, at the end of the algorithm, the y-coordinate of R_1, as in none of the available points (R_0, R_1 and R_2) the corresponding y-coordinate is known. This may force the decision to use complete projective formulae for the point addition and doubling operations of steps 4, 6 and 8, which would be costly. Finally, we stress that to guarantee that the case $R_0 = R_2$ will never occur, it is sufficient to initialize R_1 with $\frac{P}{2}$, and perform an affine subtraction at the end of the main loop (step 10).

In the following subsection we present a halve-and-add right-to-left Montgomery ladder algorithm that alleviates the above shortcomings while still achieving a competitive performance.

3.2 Right-To-Left halve-and-add Montgomery-LD ladder

Algorithm 3 Montgomery-López-Dahab halve-and-add (right-to-left)

Input: $P = (x, y)$, $k' = (k'_{n-1}, k'_{n-2}, \ldots, k'_1, k'_0)$
Output: $Q = kP$
1: **Precomputation:** $x(P_i)$, where $P_i = \frac{P}{2^i}$, for $i = 0, \ldots, n$
2: $R_1 \leftarrow P_n$; $R_2 \leftarrow P_n$;
3: **for** $i = 0$ **to** $n - 1$ **do**
4: $R_0 \leftarrow P_{n-1-i}$;
5: **if** $k'_i = 1$ **then**
6: $R_1 \leftarrow R_0 + R_1$;
7: **else**
8: $R_2 \leftarrow R_0 + R_2$;
9: **end if**
10: **end for**
11: $R_1 \leftarrow R_1 - P_n$
12: **return** R_1

Algorithm 3 presents a right-to-left Montgomery ladder procedure similar to Algorithm 2, but in this case, all the point doubling operations are substituted with point halvings. A left-to-right approach using halve-and-add with Montgomery ladder was published in [29], however, the method requires one inversion per iteration, which degrades its efficiency due to the cost of this operation.

As in any halve-and-add procedure, an initial step before performing the actual computation consists of processing the scalar k such that it can be equivalently represented with negative powers of two. To this end, one first computes $k' \equiv 2^{n-1}k \mod r$, with $n = |r|$. This implies that, $k \equiv \sum_{i=1}^{n} k'_{n-i}/2^{i-1} \mod r$ and therefore, $kP = \sum_{i=1}^{n} k'_{n-i}(\frac{1}{2^{i-1}}P)$. Then, in the first step of Algorithm 3, n halvings of the base point P are computed. We stress that all the precomputed points $P_i = \frac{P}{2^i}$, for $i = 0, \ldots, n$ can be stored in affine coordinates. In fact, just the x-coordinate of each one of the above n points must be stored (with the sole exception of the point P_n, whose y-coordinate is also computed and stored).

As in the preceding algorithm notice that at the end of the i-th iteration, the points in the variables R_0, R_1 are, $R_0 = \frac{P}{2^{n-i-1}}$, and $R_1 = \ell P + P_n$, where in this case ℓ is the integer represented as, $\ell = \sum_{j=0}^{i} \frac{k'_j}{2^{n-j}} \mod r$. Notice also that the variable R_2 maintains the relationship, $R_2 = R_0 - R_1$, until the execution of the last iteration of the main loop (steps 3-10). This comes from the fact that at each iteration, if $k_i = 1$, then the difference $R_0 - R_1$ remains unchanged. If otherwise, $k_i = 0$, then both R_2 and R_0 are updated with their respective original values plus R_0, which ensures that $R_2 = R_0 - R_1$, still holds.

Since at every iteration, the values of the points R_0, R_1 and $R_0 - R_1$, are all known, the compact point addition formula (7) can be used. In practice, this is also possible because the y-coordinate of the output point kP can be readily recovered using Equation 4, along with the point $2P$. Moreover, since the points in the precomputed table were generated using affine coordinates, it turns out that the z-coordinate of the point R_0 is always 1 for all the iterations of the main loop. This simplifies (7) as,

$$\begin{cases} T = (X_0 \cdot Z_1 + X_1)^2 \\ Z_3 = Z_2 \cdot T \\ X_3 = X_2 \cdot T + Z_2 \cdot (X_0 \cdot Z_1) \cdot (X_1) \end{cases} \tag{8}$$

Hence, the computational cost per iteration of Algorithm 3 is of 5 multiplications, 1 squarings, 2 additions and one point halving over the binary field where the elliptic curve lies.

GLS Endomorphism The efficient computable endomorphism provided by the GLS curves can be used to implement the 2-GLV method on the Algorithm 3. As a result, only $n/2$ point halving operations must be computed. Besides the speed improvement, the 2-GLV method reduces to a half the number of precomputed points that must be stored.

3.3 Multi-core Montgomery ladder

As proposed in [37], by properly recoding the scalar, one can efficiently compute the scalar multiplication in a multi-core environment. Specifically, given a scalar k of size n, we fix a constant t which establishes how many scalar bits will be processed by the double-and-add, and by the halve-and-add procedures. This is accomplished by computing, $k' = 2^t k \bmod r$, which yields,

$$k = \underbrace{\frac{k'_0}{2^t} + \frac{k'_1}{2^{t-1}} + \cdots + \frac{k'_{t-1}}{2^1}}_{halve-and-add} + \underbrace{\frac{k'_t}{2^0} + 2^1 k'_{t+1} + 2^2 k'_{t+2} + \cdots + 2^{(n-1)-t} k'_{n-1}}_{double-and-add}$$

In a two-core setting, it is straightforward to combine the left-to-right and right-to-left Montgomery ladder procedures of Algorithms 1 and 3, and distribute them to both cores. In this scenario, the number of necessary pre-computed halved points reduces to $\sim \frac{n}{4}$. In a four-core platform, we can apply the GLS endomorphism to the left-to-right Montgomery ladder (Algorithm 1). Even though the GLV technique is ineffective for the classical Montgomery algorithm (due to the fact that we cannot share the point doublings between the base point and its endomorphism), the method does permits an efficient splitting of the algorithm workload into two cores. In this way, one can use the first two cores for computing t-digits of the GLV subscalars k_1 and k_2 by means of Algorithm 3, while we allocate the other two cores to compute the rest of the scalar's bits using Algorithm 1, as shown in Algorithm 6 (see Appendix A).

3.4 Cost comparison of Montgomery ladder variants

Table 1 shows the computational costs associated to the Montgomery ladder variants described in this Section. The values t_2 and t_4 represent the constant t chosen for the two- and four-core implementations, respectively.[5] All Montgomery ladder algorithms require a basic post-computation cost to retrieve the y-coordinate, which demands ten multiplications, one squaring and one inversion. Due to the application of the GLV technique, the Montgomery-LD-2-GLV halve-and-add version (corresponding to Algorithm 3), requires some few extra operations, namely, the subtraction of

[5] In our implementations (see subsection 5.3 below), the values used for the parameters t_2 and t_4 ranged from 53 to 55.

Table 1. Montgomery-LD algorithms cost comparison. In this table, M, M_a, M_b, S, I denote the following field operations: multiplication, multiplication by the curve a-constant, multiplication by the curve b-constant, squaring and inversion. The point halving operation is denoted by H.

	Method			Cost
1-core	Alg. 1: Montgomery-LD		pre/post	$10M + 1S + 1I$
	(double-and-add, left-to-right)		sc. mult.	$n(5M + 1M_b + 4S)$
	Alg. 3: Montgomery-LD-2-GLV		pre/post	$48M + 1M_a + 13S + 3I$
	(halve-and-add, right-to-left)		sc. mult.	$(\frac{n}{2} + 1)H + n(5M + 1S)$
2-core	Montgomery-LD-2-GLV	core I	pre/post	$25M + 1M_a + 5S + 2I$
	(double-and-add, left-to-right)		sc. mult.	$(n - t_2)(5M + 1M_b + 4S)$
	Montgomery-LD-2-GLV	core II	pre/post	$46M + 2M_a + 12S + 2I$
	(halve-and-add, right-to-left)		sc. mult.	$(\frac{t_2}{2} + 1)H + t_2(5M + 1S)$
	Overhead			$15M + 5S + 1I$
4-core	Montgomery-LD-2-GLV	cores	pre/post	$10M + 1S + 1I$
	(double-and-add, left-to-right)	I & II	sc. mult.	$(\frac{n}{2} - t_4)(5M + 1M_b + 4S)$
	Montgomery-LD-2-GLV	cores	pre/post	$16M + 1M_a + 4S + 1I$
	(halve-and-add, right-to-left)	III & IV	sc. mult.	$(\frac{t_4}{2} + 1)H + t_4(5M + 1S)$
	Overhead			$34M + 1M_a + 12S + 1I$

a point and the addition of two accumulators, which is performed using the López-Dahab (LD) projective coordinate formulae. In the end, it is necessary one extra inversion to convert the point representation from LD-projective coordinates to affine coordinates.

In the case of the parallel versions, the overhead is given by the post-computation done in one single core. The exact costs are mainly determined by the accumulator additions that are performed via full and mixed LD-projective formulae. In all of the timings reported below, we consider the LD-projective to affine coordinate transformation cost.

4 A new regular τ-adic approach

4.1 Recoding in τ-adic form

The recoding approach proposed by Solinas finds an element $\rho \in \mathbb{Z}[\tau]$, of as small norm as possible, such that $\rho \equiv k \pmod{\frac{\tau^m - 1}{\tau - 1}}$. A τ-adic expansion with average non-zero density $\frac{1}{3}$ can be obtained by repeatedly dividing ρ by τ and assigning the remainders to the digits u_i to obtain $k = \sum_{i=0}^{i=l-1} u_i \tau^i$. An alternative approach that does not involve multi-precision divisions, is to compute an element $\rho' = k \text{ partmod } (\text{mod } \frac{\tau^m - 1}{\tau - 1})$ by performing a partial reduction procedure. A width-w τ-NAF expansion with non-zero density $\frac{1}{w+1}$, where at most one of any w consecutive coefficients is non-zero, can also be obtained by repeatedly dividing ρ' by τ^w and assigning the remainders to the digit set $\{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$, for $\alpha_i = i \bmod \tau^w$. Under reasonable assumptions, this window-based recoding has length $l \leq m + 1$.

In this section, a regular recoding version of the (width-w) τ-NAF expansion is derived. The security advantages of such recoding are the predictable length and locations of non-zero digits in the expansion. This eliminates any side-channel information that an attacker could possibly collect regarding the operation executed at any iteration of the scalar multiplication algorithm (point doubling/Frobenius map or point addition). As long as querying a precomputed table of points to select the second operand of a point addition takes constant time, the resulting algorithm should be resistant against any timing-based side-channel attacks.

Let us first consider the integer recoding proposed by Joye and Tunstall [21]. They observed that any odd integer i in the interval $[0, 2^w)$ can be written as $i = 2^{w-1} + (-(2^{w-1} - i))$. Repeatedly dividing an odd n-bit integer $k - ((k \bmod 2^w) - 2^{w-1})$ by 2^{w-1} maintains the parity and assigns the

remainders to the digit set $\{\pm 1, \ldots, \pm(2^{w-1}-1)\}$, producing an expansion of length $\lceil 1+\frac{n}{w-1} \rceil$ with non-zero density $\frac{1}{w-1}$. Our solution for the problem of finding a regular τ-adic expansion employs the same intuition as explained next.

Let $\phi_w : \mathbb{Z}[t] \rightarrow \mathbb{Z}_{2^m}$ be a surjective ring homomorphism induced by $\tau \mapsto t_w$, for $t_w^2 + 2 \equiv \mu t_w \pmod{2^w}$, with kernel $\{\alpha \in \mathbb{Z}[\tau] : \tau^w \text{ divides } \alpha\}$. An element $i = i_0 + i_1\tau$ from $\mathbb{Z}[\tau]$ with odd integers $i_0, i_1 \in [0, 2^w)$ satisfies the analogous property $\phi_w(i) = 2^{w-1} + (-(2^{w-1} - \phi_w(i)))$. Repeated division of $(r_0 + r_1\tau) - (((r_0 + r_1\tau) \bmod \tau^w) - \tau^{w-1})$ by τ^{w-1}, correspondingly of $\phi_w(\rho') = (r_0 + r_1 t_w) - ((r_0 + r_1 t_w \bmod 2^w) - 2^{w-1})$ by 2^{w-1}, obtains remainders that belong to the set $\{0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}\}$. The resulting expansion has always length $\lceil 1 + \frac{m+2}{w-1} \rceil$ and non-zero density $\frac{1}{w-1}$. Algorithm 4 presents the recoding process for any $w \geq 2$.

Algorithm 4 Regular width-w τ-recoding for n-bit scalar

Input: w, t_w, $\alpha_u = \beta_u + \gamma_u\tau$ for $u = \{\pm 1, \pm 3, \pm 5, \ldots, \pm 2^{w-1} - 1\}, \rho = r_0 + r_1\tau \in \mathbb{Z}[\tau]$ with odd r_0, r_1

Output: $\rho = \sum_{i=0}^{\lceil \frac{n+2}{w-1} \rceil} u_i \tau^{i(w-1)}$

1: **for** $i \leftarrow 0$ **to** $\lceil \frac{n+2}{w-1} \rceil$ - 1 **do**
2: **if** $w = 2$ **then**
3: $u_i \leftarrow ((r_0 - 2r_1) \mod 4) - 2$
4: $r_0 \leftarrow r_0 - u_i$
5: **else**
6: $u \leftarrow (r_0 + r_1 t_w \bmod 2^w) - 2^{w-1}$
7: **if** $u > 0$ **then** $s \leftarrow 1$ **else** $s \leftarrow -1$
8: $r_0 \leftarrow r_0 - s\beta_u, r_1 \leftarrow r_1 - s\gamma_u, u_i \leftarrow \alpha_u$
9: **end if**
10: **for** $j \leftarrow 0$ **to** $(w-2)$ **do**
11: $t \leftarrow r_0, r_0 \leftarrow \mu r_0/2, r_1 \leftarrow -t/2$
12: **end for**
13: **end for**
14: **if** $r_0 \neq 0$ **and** $r_1 \neq 1$ **then**
15: $u_i \leftarrow r_0 + r_1\tau, i \leftarrow i + 1$
16: **else**
17: **if** $r_1 \neq 0$ **then**
18: $u_i \leftarrow r_1, i \leftarrow i + 1$
19: **else**
20: $u_i \leftarrow r_0, i \leftarrow i + 1$
21: **end if**
22: **end if**

4.2 Left-to-right regular approach

Algorithm 5 presents a complete description of a regular scalar multiplication approach that uses as a building block the regular width-w τ-recoding recoding procedure just described.

For benchmarking purposes we also included a baseline implementation of the customary Montgomery López-Dahab ladder. This allows easier comparisons with related work and permits to evaluate the impact of partial reduction in the field arithmetic performance (cf. subsection 5.2).

5 Implementation issues and results

In this Section, we discuss several implementation issues. We also present our experimental results and we compare them against state-of-the-art protected point multiplication implementations at the 128-bit security level.

5.1 Mechanisms to achieve a constant-time GLS-Montgomery ladder implementation

To protect the previously described algorithms against timing attacks, we observed the following precautions,

Algorithm 5 Protected scalar multiplication

Input: $P = (x, \lambda)$, $k \in \mathbb{Z}$, width w
Output: $Q = kP$

Compute $\rho' = r_0 + r_1\tau = k$ partmod $(\text{mod } \frac{\tau^m - 1}{\tau - 1})$
if $2|r_0$ **then**, $r'_0 = r_0 + 1$
if $2|r_1$ **then**, $r'_1 = r_1 + 1$
Compute width-w length-l regular τ-adic of $r'_0 + r'_1\tau$ as $\sum_{i=0}^{\lceil \frac{n+3}{w-1} \rceil} u_i \tau^{i(w-1)}$ (Algorithm 4)
for $i \in \{1, \ldots, 2^{w-1} - 1\}$ **do**
    Compute $P_u = \alpha_u P$

$Q \leftarrow \mathcal{O}$
for $i = l - 1$ **downto** 0 **do**
    $Q \leftarrow \tau^{w-1}(Q)$
    Perform a linear pass to recover P_{u_i}
    $Q \leftarrow Q + P_{u_i}$
end for
return $Q = Q - (r'_0 - r_0)P - (r'_1 - r_1)\tau(P)$.

Branchless code The main loop, the pre- and post-computation phases are implemented by a completely branch-free code.

Data veiling To guarantee a constant memory access pattern in the main loop of the Montgomery ladder algorithms, we proposed an efficient data veiling method, as described in Algorithm 7 of Appendix B. Algorithm 7 evaluates the actual and the previous scalar bits to decide whether the variables containing the Montgomery-LD accumulators values should or should not be masked. This strategy saves a considerable portion of the computational effort associated to Algorithm 1 of [4].

Field arithmetic Two of the base field arithmetic operations over $\mathbb{F}_q$ were implemented through look-up tables, namely, the half-trace and the multiplicative inverse operations. The half-trace is used to perform the point halving primitive, which is required in the pre-computation phase of the Montgomery-LD halve-and-add algorithm. The multiplicative inverse is one of the operations in the y-coordinate retrieval procedure, at the end of the Montgomery ladder algorithms. Also, whenever post-computational additions are necessary, inverses must be performed to convert a point from LD-projective to affine coordinates.

Although we are aware of the existence of protocols that consider the base point as a secret information [6], in which case one *could not* consider that our software provides protection against timing attacks, in the vast majority of protocols, the base point is public. Consequently, any attacks aimed at the two field operations mentioned above would be pointless.

5.2 Mechanisms to achieve a constant-time Koblitz implementation

Implementing Algorithm 5 in constant time needs some care, since all of its building blocks must be implemented in constant time.

Finite field arithmetic. Modern implementations of finite field arithmetic can make extensive use of vector registers, removing timing variances due to the cache hierarchy. For our illustrative implementation of curve NIST-K283, we closely follow the arithmetic described in Bluhm-Gueron [4], adopting the partial reduction improved proposed by Negre-Robert [28].

Integer recoding. All the branches in Algorithm 4 need to be eliminated by conditional execution statements to protect leakage of the scalar k through secret-dependent branching. Moreover, to remove the remaining sign-related branches, multiple precision integer arithmetic must be implemented in two's complement representation. If two constants, say β_u, γ_u, are stored in a precomputed table, then they need to be recovered by a linear pass across the table in constant time. Finally, the partial reduction step producing ρ' must also be implemented in constant time by removing all of its branches. Notice that the requirement for r_0, r_1 to be odd is not a problem, since partial reduction can be modified to always result in odd integers, with a possible correction at the end of the scalar multiplication by performing a (protected) conditional subtraction of points.

5.3 Results

Our implementation was mainly designed for the Intel Haswell processor family, which supports vectorial sets such as SSE and AVX, a carry-less multiplication and some bit manipulation instructions. The programming was done in C with the support of assembly inline code. The compilation was performed via GCC version 4.7.3 with the flags `-m64 -march=core-avx2 -mtune=core-avx2 -O3 -fomit-frame-pointer -funroll-loops`. Finally, the timings were collected on an Intel Core i7-4700MQ, with the Turbo Boost and Hyperthreading features disabled[6].

Table 2 presents the experimental timings obtained for the most prominent building blocks required for computing the point multiplication operation on the GLS and Koblitz binary elliptic curves.

Table 2. Timings (in clock cycles) for the elliptic curve operations in the Intel Haswell platform.

Elliptic curve operation	GLS $E/\mathbb{F}_{2^{254}}$	
	cycles	op/M[1]
Halving	184	4.181
Montgomery-LD D&A (left-to-right) Addition (Eq. (6))	161	3.659
Montgomery-LD H&A (right-to-right) Addition (Eq. (8))	199	4.522
Montgomery-LD Doubling[7] (Eq. (5))	95	2.159

Elliptic curve operation	Koblitz $E/\mathbb{F}_{2^{283}}$	
	cycles	op/M[1]
Frobenius	70	1.235
Integer τ-adic recoding (Alg. 4) ($w = 5$)	8,900	156.863
Point addition	602	10.588

[1] Ratio to multiplication.

We present in Table 3 a comparison of our timings against a selection of state-of-the-art implementations of the point multiplication operation on binary and prime elliptic curves. Due to the Montgomery-LD point doubling efficiency, which costs 49% less than a point halving, the GLS-Montgomery-LD-double-and-add achieved the fastest timing in the one-core setting, with 70,800 clock cycles. This is 13% faster than the performance obtained by the GLS-Montgomery-LD-halve-and-add algorithm. In the known-base point setting, we can ignore the GLS-Montgomery-LD-halve-and-add pre-computation expenses associated with its table of halved points. In that case, we can

[6] We pretend to submit our software to the ECRYPT Benchmarking of Cryptographic Systems (eBACS) SUPERCOP toolkit in the near future.

[7] The flexibility for finding a curve b-constant, provided by the GLS curves, allow us to have a small $\sqrt{b}$ (see Appendix C). As a consequence, we used the Eq. (5) alternative formula.

compute the scalar multiplication in an estimated time of 44,600 clock cycles using a table of just 4128 bytes.

Furthermore, the GLS-Montgomery-LD-halve-and-add is crucial for implementing the multi-core versions of the Montgomery ladder. When compared with our one-core double-and-add implementation, Table 3 reports a speedup of 1.36 and 2.03 in our two- and four-core Montgomery ladder versions, respectively. Here, besides the overhead costs commented in Section 3, we can clearly perceive the usual multicore management penalty. Finally, we observe that our GLS-Montgomery-LD-double-and-add surpasses by 48%, 40% and 2% the Montgomery ladder implementations of [4] (Random), [4] (Koblitz) and [1], respectively.

As for our Koblitz implementations, the fast τ endomorphism allows us to have a regular-recoding implementation that outperforms a standard Montgomery ladder for Koblitz curves by 18%. In addition, our fastest Koblitz code surpasses by 15% the recent implementation reported in [4] [8]. Finally, note that, in spite of the fact that the τ endomorphism is 26% faster than the Montgomery-LD point doubling, the superior efficiency of the GLS quadratic field arithmetic produces faster results for the GLS Montgomery ladder algorithms.

Table 3. Timings (in clock cycles) for 128-bit level scalar multiplication with timing-attack resistance in the Intel Ivy Bridge (I) and Haswell (H) architectures.

	Method	Cycles	Arch
State-of-the-art implementations	Montgomery-DJB-chain (prime) [7]	148,000	I
	Random-Montgomery-LD ladder (binary) [4]	135,000	H
	Genus-2-Kummer (prime) [5]	122,000	I
	Koblitz-Montgomery-LD ladder (binary) [4]	118,000	H
	Twisted-Edwards-4-GLV (prime) [11]	92,000	I
	Genus-2-Kummer Montgomery ladder (prime) [1]	72,200	H
	GLS-2-GLV double-and-add (λ) [31]	**60,000**	H
Our Work	Koblitz-Montgomery-LD double-and-add (left-to-right)	122,000	H
	Koblitz-regular τ-and-add (left-to-right, $w = 5$)	100,000	H
	GLS-Montgomery-LD-2-GLV halve-and-add (**Algorithm 3**)	80,800	H
	GLS-Montgomery-LD double-and-add (**Algorithm 1**)	70,800	H
	2-core GLS-Montgomery-LD-2-GLV halve-and-add/double-and-add	52,000	H
	4-core GLS-Montgomery-LD-2-GLV halve-and-add/double-and-add (**Algorithm 6**)	34,800	H

6 Conclusion

We presented several algorithms that permit to compute a constant-time high-security point multiplication operation over two families of binary elliptic curves, namely, the GLS and the Koblitz curves. Although this work was completely focused on a high-end desk computation of the variable-base point multiplication, the possibility of applying Algorithm 3 to the fixed-base point multiplication setting is highly appealing since that procedure requires a comparatively small pre-computed table of roughly $2n \cdot (n + 1)$ bits for computing a point multiplication at the n-bit security level. The above combined with the Montgomery ladder unique feature of performing all the computations using only two point coordinates, should be attractive for deployments of public key cryptography on constrained computing environments.

[8] We could not reproduce the timing of 118,000 cycles with the code available from [4], which indicates that TurboBoost could be possibly turned on on their benchmarks. Considering this, our implementation of Koblitz-Montgomery-LD becomes 9% faster than [4], reflecting the savings from partial reduction, and the speedup achieved by the Koblitz-regular implementation increases to 26%.

References

1. D. J. Bernstein, C. Chuengsatiansup, T. Lange, and P. Schwabe. Kummer strikes back: new DH speed records. Cryptology ePrint Archive, Report 2014/134, 2014. `http://eprint.iacr.org/`.
2. D. J. Bernstein and T. Lange. SafeCurves: choosing safe curves for elliptic-curve cryptography. `http://safecurves.cr.yp.to`.
3. D. J. Bernstein and T. Lange. Security dangers of the NIST curves. Invited talk, International State of the Art Cryptography Workshop, Athens, Greece, 2013.
4. M. Bluhm and S. Gueron. Fast Software Implementation of Binary Elliptic Curve Cryptography. Cryptology ePrint Archive, Report 2013/741, 2013. `http://eprint.iacr.org/`.
5. J. W. Bos, C. Costello, H. Hisil, and K. Lauter. Fast Cryptography in Genus 2. In T. Johansson and P. Q. Nguyen, editors, *Advances in Cryptology - EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 194–210. Springer, 2013.
6. S. Chatterjee, K. Karabina, and A. Menezes. A New Protocol for the Nearby Friend Problem. In M. G. Parker, editor, *Cryptography and Coding, 12th IMA International Conference, Cryptography and Coding 2009*, volume 5921 of *LNCS*, pages 236–251. Springer, 2009.
7. C. Costello, H. Hisil, and B. Smith. Faster Compact Diffie-Hellman: Endomorphisms on the x-line. In P. Nguyen and E. Oswald, editors, *Advances in Cryptology EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 183–200. Springer Berlin Heidelberg, 2014.
8. T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), Jan. 1999. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176.
9. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346 (Proposed Standard), Apr. 2006. Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176.
10. T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008.
11. A. Faz-Hernández, P. Longa, and A. H. Sanchez. Efficient and Secure Algorithms for GLV-Based Scalar Multiplication and Their Implementation on GLV-GLS Curves. In J. Benaloh, editor, *Topics in Cryptology - CT-RSA 2014*, volume 8366 of *LNCS*, pages 1–27. Springer, 2014.
12. A. Fog. Instruction Tables: List of Instruction Latencies, Throughputs and Micro-operation Breakdowns for Intel, AMD and VIA CPUs., Accessed: May 14 2014. Available at: `http://www.agner.org/optimize/instruction_tables.pdf`.
13. S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for Faster Elliptic Curve Cryptography on a Large Class of Curves. In A. Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 518–535. Springer, 2009.
14. R. P. Gallant, R. J. Lambert, and S. A. Vanstone. Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms. In J. Kilian, editor, *Advances in Cryptology CRYPTO 2001*, volume 2139 of *LNCS*, pages 190–200. Springer Berlin Heidelberg, August 2001.
15. P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of Weil descent on elliptic curves. *Journal of Cryptology*, 15:19–46, March 2002.
16. S. Gueron and V. Krasnov. Fast Prime Field Elliptic Curve Cryptography with 256 Bit Primes. Cryptology ePrint Archive, Report 2013/816, 2013. `http://eprint.iacr.org/`.
17. D. Hankerson, K. Karabina, and A. Menezes. Analyzing the Galbraith-Lin-Scott Point Multiplication Method for Elliptic Curves over Binary Fields. *Computers, IEEE Transactions on*, 58(10):1411 – 1420, October 2009.
18. D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.
19. F. Hess. Generalising the GHS Attack on the Elliptic Curve Discrete Logarithm Problem. *LMS Journal of Computation and Mathematics*, 7:167–192, June 2004.
20. Y.-J. Huang, C. Petit, N. Shinohara, and T. Takagi. Improvement of Faugère et al.'s Method to Solve ECDLP. In K. Sakiyama and M. Terada, editors, *Advances in Information and Computer Security - IWSEC 2013*, volume 8231 of *LNCS*, pages 115–132. Springer, 2013.
21. M. Joye and M. Tunstall. Exponent Recoding and Regular Exponentiation Algorithms. In B. Preneel, editor, *Progress in Cryptology – AFRICACRYPT 2009*, volume 5580 of *LNCS*, pages 334–349. Springer Berlin Heidelberg, 2009.

22. A. H. Koblitz, N. Koblitz, and A. Menezes. Elliptic curve cryptography: The serpentine course of a paradigm shift. *Journal of Number Theory*, 131(5):781 – 814, 2011. Elliptic Curve Cryptography.
23. N. Koblitz. CM-curves with good cryptographic properties. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, volume 576 of *LNCS*, pages 279–287. Springer, 1991.
24. P. Longa and F. Sica. Four-Dimensional Gallant-Lambert-Vanstone Scalar Multiplication. *Journal of Cryptology*, 27(2):248–283, 2014.
25. J. López and R. Dahab. Fast Multiplication on Elliptic Curves over GF(2^{m}) without Precomputation. In Çetin Kaya Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99*, volume 1717 of *LNCS*, pages 316–327. Springer, 1999.
26. P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of Computation*, 48:243–264, 1987.
27. National Institute of Standards and Technology. Recommended Elliptic Curves for Federal Government Use. NIST Special Publication, 1999. `http://csrc.nist.gov/csrc/fedstandards.html`.
28. C. Nègre and J.-M. Robert. Impact of Optimized Field Operations AB,AC and AB+CD in Scalar Multiplication over Binary Elliptic Curve. In *Progress in Cryptology AFRICACRYPT 2013*, volume 7918 of *LNCS*, pages 279–296. Springer Berlin Heidelberg, 2013.
29. C. Nègre and J.-M. Robert. New Parallel Approaches for Scalar Multiplication in Elliptic Curve over Fields of Small Characteristic. 2013. `http://hal.archives-ouvertes.fr/docs/00/90/84/63/PDF/parallelization-ecsm8.pdf`.
30. T. Oliveira, J. López, D. F. Aranha, and F. Rodríguez-Henríquez. Lambda Coordinates for Binary Elliptic Curves. In G. Bertoni and J.-S. Coron, editors, *Cryptographic Hardware and Embedded Systems - CHES 2013*, volume 8086 of *LNCS*, pages 311–330. Springer, 2013.
31. T. Oliveira, J. López, D. F. Aranha, and F. Rodríguez-Henríquez. Two is the fastest prime: lambda coordinates for binary elliptic curves. *J. Cryptographic Engineering*, 4(1):3–17, 2014.
32. C. Petit and J.-J. Quisquater. On Polynomial Systems Arising from a Weil Descent. In X. Wang and K. Sako, editors, *Advances in Cryptology - ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 451–466. Springer, 2012.
33. J. Salowey. Confirming Consensus on removing RSA key Transport from TLS 1.3. Transport Layer Security working group of the IETF Mailing List, May 3 2014.
34. M. Scott. Re: NIST announces set of Elliptic Curves. `https://groups.google.com/forum/message/raw?msg=sci.crypt/mFMukSsORmI/FpbHDQ6hM_MJ`, 1999.
35. M. Shantz and E. Teske. Solving the Elliptic Curve Discrete Logarithm Problem Using Semaev Polynomials, Weil Descent and Gröbner Basis Methods – an Experimental Study. Cryptology ePrint Archive, Report 2013/596, 2013. `http://eprint.iacr.org/`.
36. J. A. Solinas. Efficient Arithmetic on Koblitz Curves. *Designs, Codes and Cryptography*, 19(2-3):195–249, 2000.
37. J. Taverne, A. Faz-Hernández, D. F. Aranha, F. Rodríguez-Henríquez, D. Hankerson, and J. López. Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. *Journal of Cryptographic Engineering*, 1:187–199, November 2011.

A Multi-core Montgomery ladder

Here we present the four-core GLS-Montgomery-LD ladder algorithm. Given t_4 the integer constant that establish the workload of each algorithm, $P \in E(\mathbb{F}_{q^2})$, and the scalar k represented as $k_1 + k_2 \cdot \delta$ using the GLS-GLV method, cores I and II are both responsible for computing $\lfloor \frac{n}{2} \rfloor - t_4$ bits of the subscalars k_1 and k_2 using the Montgomery-LD double-and-add method. In turn, the cores III and IV both compute t_4 bits of k_1 and k_2 with the Montgomery-LD halve-and-add algorithm. In the end, on a single core, it is necessary to add all the accumulators Q_i, for $i = 0 \ldots 3$.

Algorithm 6 Parallel Montgomery ladder scalar multiplication (four-core)

Input: $P \in E(\mathbb{F}_{q^2})$, scalar k of bit length n, integer constant t_4
Output: $Q = kP$
$k' \leftarrow 2^{t_4} k$
Represent $k' = k'_1 + k'_2 \lambda$, where $\psi(P) = \lambda P$

Core I

{Initialization}
$R_0 \leftarrow \mathcal{O}, R_1 \leftarrow P$
for $i = \lceil \frac{n}{2} \rceil$ **downto** t_4 **do**
  $b \leftarrow k'_{1,i} \in \{0, 1\}$
  $R_{1-b} \leftarrow R_{1-b} + R_b$
  $R_b \leftarrow 2R_b$
end for
$Q_0 \leftarrow R_0$
{Barrier}

Core II

{Initialization}
$R_0 \leftarrow \mathcal{O}, R_1 \leftarrow P$
for $i = \lceil \frac{n}{2} \rceil$ **downto** t_4 **do**
  $b \leftarrow k'_{2,i} \in \{0, 1\}$
  $R_{1-b} \leftarrow R_{1-b} + R_b$
  $R_b \leftarrow 2R_b$
end for
$Q_1 \leftarrow R_0$
{Barrier}

Core III

{Precomputation}
for $i = 1$ **downto** $t_4 + 1$ **do**
  $P_i \leftarrow \frac{P}{2^i}$
end for
{Initialization}
$R_1 \leftarrow P_{t_4+1}$, $R_2 \leftarrow P_{t_4+1}$
for $i = 0$ **downto** $t_4 - 1$ **do**
  $R_0 \leftarrow P_{t_4-i}$
  $b \leftarrow k'_{1,i} \in \{0, 1\}$
  $R_{2-b} \leftarrow R_{2-b} + R_0$
end for
$Q_2 \leftarrow R_1 - P_{t_4+1}$
{Barrier}

Core IV

{Precomputation}
for $i = 1$ **downto** $t_4 + 1$ **do**
  $P_i \leftarrow \frac{P}{2^i}$
end for
{Initialization}
$R_1 \leftarrow P_{t_4+1}$, $R_2 \leftarrow P_{t_4+1}$
for $i = 0$ **downto** $t_4 - 1$ **do**
  $R_0 \leftarrow P_{t_4-i}$
  $b \leftarrow k'_{2,i} \in \{0, 1\}$
  $R_{2-b} \leftarrow R_{2-b} + R_0$
end for
$Q_3 \leftarrow R_1 - P_{t_4+1}$
{Barrier}

return $Q = Q_0 + Q_2 + \psi(Q_1 + Q_3)$

B Memory access pattern

The following data veiling algorithm allows us to use the bits k_{i-1} and k_i to decide if the R_0 and R_1 accumulators will or will not be swapped. As a result, it is not necessary to reapply the procedure at the end of each iteration.

Algorithm 7 Data veiling algorithm

Input: Scalar digits k_i and k_{i-1}, Montgomery-LD accumulators R_0 and R_1
$mask \leftarrow 0 - (k_{i-1} \oplus k_i)$
$tmp \leftarrow R_0 \oplus R_1$
$tmp \leftarrow tmp \wedge mask$
$R_0 \leftarrow R_0 \oplus tmp$
$R_1 \leftarrow R_1 \oplus tmp$
return R_0, R_1

C GLS elliptic curve parameters

For achieving a greater benefit from the multiplication by the b-constant in the Montgomery-LD doubling formula

$$X_3 = {X_0}^4 + b{Z_0}^4 = ({X_0}^2 + \sqrt{b}{Z_0}^2)^2$$

we carefully selected a GLS curve with a 64-bit b-parameter square-root. As a result, we saved two carry-less multiplication and a dozen of SSE instructions per field multiplication. Next, we describe the parameters, as polynomials represented in hexadecimal, for our GLS curve $E_{a,b}/\mathbb{F}_{q^2} : y^2 + xy = x^3 + ax^2 + b$.

- $a = u$.
- $b = 0x540451444104015441015405405151 01$.
- $\sqrt{b} = 0xE2DA921E91E38DD1$.

Finally, the 253-bit prime order r of the main subgroup of $E_{a,b}/\mathbb{F}_{q^2}$ is,

$$r = 0x1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF$$
$$FFFA6B89E49D3FECD828CA8D66BF4B88ED5$$