J Cryptogr Eng (2018) 8:241–258
https://doi.org/10.1007/s13389-017-0163-8

SPECIAL ISSUE ON MONTGOMERY ARITHMETIC

The Montgomery ladder on binary elliptic curves

Thomaz Oliveira[1] · **Julio López**[2] · **Francisco Rodríguez-Henríquez**[1]

Received: 5 September 2016 / Accepted: 11 April 2017 / Published online: 26 April 2017

Abstract In this survey paper, we present a careful analysis of the Montgomery ladder procedure applied to the computation of the constant-time point multiplication operation on elliptic curves defined over binary extension fields. We give a general view of the main improvements and formula derivations that several researchers have contributed across the years, since the publication of Peter Lawrence Montgomery seminal work in 1987. We also report a fast software implementation of the Montgomery ladder applied on a Galbraith–Lin–Scott (GLS) binary elliptic curve that offers a security level close to 128 bits. Using our software, we can execute the ephemeral Diffie–Hellman protocol in just 95,702 clock cycles when implemented on an Intel Skylake machine running at 4 GHz.

Keywords Elliptic curves · Scalar multiplication · Software implementation

1 Introduction

The focus of this paper is on the study of the Montgomery ladder procedure for computing the scalar multiplication operation on elliptic curves defined over binary extension fields. For the important case of the Montgomery ladder applied to elliptic curves that lie on prime fields, the reader is referred to the previous article included in this special issue.

In this paper, a binary elliptic curve will be defined as the set of affine points $(x, y) \in \mathbb{F}_q \times \mathbb{F}_q$, $q = 2^m$ that satisfy the Weierstrass equation,

$$E_B : y^2 + xy = x^3 + ax^2 + b, \tag{1}$$

together with a point at infinity denoted by $\mathcal{O}$. The set of points on a binary elliptic curve forms an abelian group denoted as $E_B(\mathbb{F}_q)$ of order $\#E_B(\mathbb{F}_q) = h \cdot r$, where r is a large prime and the co-factor h is usually a small integer number. The group law of $E_B(\mathbb{F}_q)$ is defined by the point addition operation. Let $\langle P \rangle$ be an additively written subgroup in E_B of prime order r, and let k be a positive integer such that $k \in [1, r-1]$. Then, the elliptic curve scalar multiplication operation computes the multiple $Q = kP$, which corresponds to the point resulting of adding P to itself, $k-1$ times.

In [49], Peter Lawrence Montgomery famously presented a procedure that allows one to compute an elliptic curve scalar multiplication using only the x-coordinate of the involved points. Although Montgomery conceived his ladder for accelerating Lenstra's ECM factorization algorithm [41], in this paper we will address the application of this procedure for achieving efficient elliptic curve-based cryptography.

The key algorithmic idea of the Montgomery ladder procedure is that given the x-coordinates of the points kP and $(k+1)P$, one can apply specialized addition and doubling formulas to compute the x-coordinates of the points $(2k+1)P$ and $2kP$. Moreover, Montgomery presented a special form of elliptic curves that lie on large characteristic fields, now known as Montgomery curves, which were particularly well suited for computing the scalar multiplication at a cost of about six field multiplications and four squarings

T. Oliveira and F. Rodríguez-Henríquez: The authors acknowledge partial support from the CONACyT project 180421.
J. López: The author was supported in part by the Intel Labs University Research Office and by a research productivity scholarship from CNPq Brazil.

✉ Francisco Rodríguez-Henríquez
francisco@cs.cinvestav.mx

[1] Computer Science Department, CINVESTAV-IPN, Mexico City, Mexico

[2] Institute of Computing, University of Campinas, Campinas, Brazil

per ladder step. The case for binary elliptic curves, however, was not addressed in [49].

In fact, a few years after its publication, the convenience of using the Montgomery ladder procedure for binary elliptic curves remained unclear. For instance, due to the lack of an efficient projective coordinate formulation of the binary elliptic curve arithmetic required at each ladder step, the Montgomery ladder was deemed to be inefficient in [1,46].

It was only one decade later, after the publication of Montgomery's landmark paper that it became apparent that this approach could be as well applied to the binary elliptic curve case efficiently. In [43], López and Dahab presented compact projective formulae for the elliptic curve arithmetic of the Montgomery ladder, by representing the points as, $P = (X, -, Z)$, in an analogous way as Montgomery did it ten years before in [49]. This projective point representation avoids all the costly field multiplicative inversions. It was shown in [43] that the computational cost of a binary elliptic curve ladder step is of five field multiplications, one multiplication by a constant, four squarings and three additions.

Furthermore, the authors of [43] presented a formula that permits to recover the y-coordinate of the points involved in the Montgomery ladder procedure. This y-coordinate retrieval formula opened the door for the full computation of elliptic curve scalar multiplications using the Montgomery ladder procedure.[1]

Binary GLS curves

Inspired in the Galbraith–Lin–Scott (GLS) elliptic curves introduced in [21], Hankerson, Karabina and Menezes reported in [26] a family of binary GLS curves defined over the quadratic field $\mathbb{F}_{q^2}$, with $q = 2^m$. GLS curves are cryptographically interesting mainly because they come equipped with a two-dimensional endomorphism. By carefully choosing the elliptic curve parameters a, b of Eq. (1), the authors of [26] found instances 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.

Taking advantage of the two-dimensional endomorphism ψ associated with GLS curves, a point multiplication can be computed using the Gallant–Lambert–Vanstone (GLV) approach presented in [24] as,

$$Q = kP = k_1 P + k_2 \psi(P) = k_1 P + k_2 \cdot \delta P, \tag{2}$$

where the subscalars $|k_1|, |k_2| \approx n/2$, with $n = \lceil \log_2(r) \rceil$, can be found by using lattice techniques [21].

λ coordinates for binary curves

In λ-affine coordinates [52] the points are represented as $P = (x, \lambda)$, $x \neq 0$ and, $\lambda = x + \frac{y}{x}$. The λ-affine form of the Weierstrass Eq. (1) becomes,

$$E_\lambda : (\lambda^2 + \lambda + a)x^2 = x^4 + 1.$$

λ point representation provides efficient point addition, doubling and halving formulas. In [52], Oliveira et al. applied this coordinate system into a binary GLS curve defined over the quadratic extension of the binary field $\mathbb{F}_{2^{127}}$. When implemented on a Haswell processor, this approach permits to compute a constant-time variable-point multiplication in just 48, 312 clock cycles [53], which is the current speed record for an elliptic curve designed to offer about 128 bits of security level.

Security of Binary elliptic curves

The Pollard's rho algorithm can be used to solve the discrete logarithm problem (DLP) over an elliptic curve subgroup of a prime order r with an average running time given as, $\frac{4}{3}\sqrt{\frac{r}{2}}$ [20]. In the case of binary elliptic curves, by applying the *negation map*, an extra small improvement acceleration (which reduces the attack complexity by less than 2 bits), to the above estimate can be achieved [7,63]. Using the Pollard's rho algorithm, the authors of [7] were able to compute the current elliptic curve DLP record computation on a binary elliptic curve defined over the field $\mathbb{F}_{2^{127}}$.

Given an ordinary binary elliptic curve satisfying Eq. (1), the Gaudry–Hess–Smart (GHS) attack [15,23,25,27,47] attempts to find an algebraic curve C of a relatively small genus g, such that the target elliptic curve group is contained in the Jacobian of C. In this case, the original elliptic curve discrete logarithm problem can be transferred into the Jacobian of C defined over $\mathbb{F}_{2^l}$, with $l|m$. If the genus of such a curve C is not too large, nor too short, then the DLP would be easier to solve in that Jacobian, due to the availability of a relatively efficient index calculus strategy for that group.

In general, however, the GHS strategy is difficult to implement because of the large genus of suitable curves C. This difficulty is so serious that the authors of [25,47] reported that the GHS attack fails, i.e., the Pollard's rho attack is more effective, for all binary elliptic curves defined over $\mathbb{F}_{2^m}$, where $m \in [160, 600]$ is a prime number. Further, in [44], it was proved that the GHS attack fails for most of the composite extensions in the range $m \in [160, 600]$.

New lines of research for solving the DLP over binary curves were presented by Semaev in 2004 [59], by introducing summation-polynomial methods (sometimes also referred as Semaev's polynomials). Since then, several researchers have attempted to attack the DLP on binary elliptic curves using this approach [18,22,60]. However, the current status of these attempts are based on Gröbner basis assumptions that are still not well understood, which in some cases have led to contradictory behaviors, even for tiny experiments [28].

[1] The analogous of this recovering formula has been recently applied in the context of hyperelliptic curves [13,56].

For a comprehensive survey of recent progress in the computation of the elliptic curve discrete problem in characteristic two, the reader is referred to the paper by Galbraith and Gaudry [20]. All in all, it is not hyperbolic to claim that at the moment the most ominous threat to the security of binary elliptic curves is the announced advent of quantum computers, only that this peril is shared by all elliptic curve-based cryptography.

Aim of this paper

The aim of this paper is twofold. First, we would like to present a recount of some of the main research findings for the Montgomery ladder applied on binary elliptic curves. We also present a short summary of some of the most relevant software and hardware implementations reported for this procedure. Second, we report a state-of-the-art software implementation of the Montgomery ladder on a GLS binary curve that can take advantage of pre-computation when the base point is known. Using this approach, we can execute the ephemeral Diffie–Hellman protocol in just 95,702 clock cycles when implemented on an Intel Skylake machine running at 4 GHz.

The remainder of this survey is organized as follows. In Sect. 2, we describe the Montgomery ladder and by using a division polynomial approach, we re-discover the point addition and doubling formulas required for performing a Montgomery ladder step. We also re-discover the y-coordinate retrieval formula as it was originally reported in [43]. In Sect. 3 we describe a Montgomery ladder procedures that admit off-line pre-computation, which was a long-standing open problem that was solved in [51] by Oliveira et al. In Sect. 4, we report a software implementation of a Montgomery ladder point multiplication that targets the Intel Skylake processor. In Sect. 5, we present a selection of the most interesting implementations of the Montgomery ladder point multiplication in software and hardware platforms; in this section, we also present the useful common-Z trick that permits valuable area savings for light hardware implementations, and a brief description of multi-dimensional Montgomery ladders. Finally, we draw our concluding remarks in Sect. 6.

2 Montgomery ladders on binary elliptic curves

We begin this section by describing the Montgomery ladder algorithm. Then, we use the division polynomial technique, which was suggested by Victor Miller in [48], in order to rediscover the point addition and doubling formulas, as well as the y-coordinate retrieval formula, as they were reported in [43].

2.1 The Montgomery ladder point multiplication algorithm

Algorithm 1 describes the classical left-to-right Montgomery ladder approach for point multiplication [49], whose key algorithmic idea is based on the following observation.

Algorithm 1 Left-to-right Montgomery ladder [49]

Input: $P = (x, y)$, $k = (1, k_{n-2}, \ldots, k_1, k_0)_2$
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** $Q = R_0$

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 $R_0 + R_1$, can be fully determined by the x-coordinates of P, R_0 and R_1.

Notice that at each iteration of Algorithm 1, the variable R_0 is updated as

$$R_0 = \begin{cases} R_0 + R_1 = 2R_0 + P & \text{if } k_i = 1, \\ 2R_0 & \text{otherwise.} \end{cases}$$

From the above equation, one can easily see that R_0 is updated in the same manner as it would be updated by the left-to-right double-and-add algorithm for point multiplication. Furthermore, notice that Algorithm 1 updates R_1 as

$$R_1 = \begin{cases} 2R_1 & \text{if } k_i = 1, \\ R_0 + R_1 & \text{otherwise.} \end{cases}$$

This maintains the invariant relationship $R_1 - R_0 = P$ throughout all the algorithm execution.

The formulas to compute the point addition and point doubling operations included in Steps 4 and 6 of Algorithm 1, are carefully analyzed in the following two subsections.

2.2 Montgomery point addition, Montgomery point doubling, and y-coordinate retrieval formulae

López and Dahab presented in [43] an efficient version of the Montgomery ladder procedure applied to binary elliptic curves, deriving compact formulas for the point addition

and point doubling operations of Algorithm 1. Moreover, the authors of [43] presented for the first time a retrieval formula that allows to recover the y-coordinate of the points involved in the Montgomery ladder procedure.

Lemma 1 ([43]) *Let $P = (x, y)$, $R_1 = (x_1, y_1)$, and $R_0 = (x_0, y_0)$ be binary elliptic curve points satisfying Eq. (1), 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*

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

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 the points R_0, R_1 and $R_0 + R_1$, be represented in projective coordinates (with $x = \frac{X}{Z}$) as $R_0 = (X_0, Z_0)$, $R_1 = (X_1, Z_1)$, $R_2 = 2R_0 = (X_2, Z_2)$ and $R_3 = R_0 + R_1 = (X_3, Z_3)$. Notice that according with the spirit of the Montgomery ladder, the y-coordinates of all the involved points are ignored.

Then, from Lemma 1, it follows that the point doubling operation ($R_2 = 2R_0$) can be computed as

$$\begin{aligned} X_2 &= X_0^4 + b \cdot Z_0^4 = (X_0^2 + \sqrt{b} \cdot Z_0^2)^2 \\ Z_2 &= X_0^2 \cdot Z_0^2. \end{aligned} \tag{5}$$

Furthermore, the point addition operation defined as $R_3 = R_0 + R_1$, with $R_0 \neq \pm R_1$, can be computed as

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

Each Montgomery ladder step of Algorithm 1 involves the computation of one point addition and point doubling. Using Eqs. (5)–(6), it follows that each ladder step of Algorithm 1 can be computed at a cost of five field multiplications, one multiplication by a constant, four squarings and three additions.

In the following subsection, we present an alternative approach for deriving Lemma 1, which to the best of our knowledge has not been presented in the open literature before.

2.3 Division polynomials

In his seminal paper [48], Victor Miller suggested that an approach based on division polynomials could yield efficient formulae for the computation of the scalar multiplication. Unfortunately, he did not present a concrete algorithm. Besides its theoretical interest, this approach has the potential advantage of allowing a natural derivation of ladder-like formulae (cf. with Proposition 1).[2] Recently, the authors of [12] presented a division polynomial analysis that allows them to find scalar multiplication formulae for both, binary and prime field elliptic curves. However, the authors of [12] obtained a rather disappointing ladder step cost of 44 field multiplications for binary elliptic curves, which is considerably more expensive than the ladder step cost reported in [43] and in this subsection.

In the following, we revisit division polynomials as they apply to the case of binary curves, showing that they are effective for deriving ladder recursive formulas used in point multiplication computations.

A division polynomial $p_k \in \mathbb{F}_2[x, y, a, b]$ is a nonzero homogeneous polynomial of total weight $k^2 - 1$. Given an order-r point $P = (x_P, y_P)$ that belongs to the non-supersingular binary elliptic curve: $y^2 + xy = x^3 + ax^2 + b$, one has that $rP = \mathcal{O}$, if and only if $p_r(x, y) = 0$. The following Proposition summarizes the main properties that these polynomials satisfy. In particular, given a point $P = (x_P, y_P)$, the coordinates of its multiple $kP = (x_k, y_k)$ for $k \in [1, r - 1]$ can be computed in terms of division polynomials.

Proposition 1 (see [35,39])

1. *The division polynomials p_k can be computed recursively:*
   $p_1 = 1$, $p_2 = x$, $p_3 = x^4 + x^3 + b$, and $p_4 = x^6 + x^2 b$. For $k \geq 5$, define,

$$\begin{aligned} p_{2k+1} &= p_{k+2} p_k^3 + p_{k-1} p_{k+1}^3; \\ p_{2k} &= (p_{k+2} p_k p_{k-1}^2 + p_{k-2} p_k p_{k+1}^2)/x. \end{aligned}$$

2. *For $k \geq 1$, $p_{2k} = x_k p_k^4$, where x_k is the x-coordinate of kP.*
3. *For $k \geq 2$, the coordinates of $kP = (x_k, y_k)$ are given by*

$$\begin{aligned} x_k &= x + \frac{p_{k+1} p_{k-1}}{p_k^2}, \\ y_k &= y + x_k + \frac{p_{k+1}^2 p_{k-2}}{x p_k^3} + (x^2 + y)\frac{p_{k+1} p_{k-1}}{x p_k^2}. \end{aligned}$$

Let $Q = kP$. In the following, we will derive formulas for computing the x-coordinates x_{2k+1} and x_{2k} of $(2k + 1)P = 2Q + P$ and $2kP$, using division polynomials.

[2] Division polynomials also play a crucial role for computing isogenies of elliptic curves [37].

The Montgomery ladder allows one to compute the x-coordinate of kP by repeatedly computing either, (x_{2k}, x_{2k+1}) or $(x_{2k+1}, x_{2(k+1)})$ from (x_k, x_{k+1}). Interestingly, by following this approach, we will be able to re-discover the formulas given in [43] for the binary Montgomery ladder algorithm.

Proposition 2 *Given an integer $k \geq 1$, an elliptic point $P(x, y)$, the x-coordinate of kP can be computed recursively using the formulas:*

$$x_{2k+1} = x + \frac{x_k x_{k+1}}{(x_k + x_{k+1})^2}, \quad x_{2k} = x_k^2 + \frac{b}{x_k^2}.$$

Proof By Proposition 1, the x-coordinate of $(2k+1)P$ can be computed using the formula

$$x_{2k+1} = x + \frac{p_{2k} p_{2(k+1)}}{p_{2k+1}^2}.$$

The recursive formula p_{2k+1} from Proposition 1 can be expressed as follows

$$\begin{aligned} p_{2k+1} &= p_{k+2} p_k^3 + p_{k-1} p_{k+1}^3 \\ &= \frac{p_{k+2} p_k}{p_{k+1}^2} p_k^2 p_{k+1}^2 + \frac{p_{k-1} p_{k+1}}{p_k^2} p_k^2 p_{k+1}^2 \\ &= (x + x_{k+1} + x + x_k) p_k^2 p_{k+1}^2 \\ &= (x_k + x_{k+1}) p_k^2 p_{k+1}^2. \end{aligned}$$

Now, by using the recurrence relations $p_{2k} = x_k p_k^4$ and $p_{2(k+1)} = x_{k+1} p_{k+1}^4$, x_{2k+1} can be written in terms of x, x_k and x_{k+1} as

$$\begin{aligned} x_{2k+1} &= x + \frac{p_{2k} p_{2(k+1)}}{p_{2k+1}^2} \\ &= x + \frac{x_k p_k^4 x_{k+1} p_{k+1}^4}{(x_k + x_{k+1})^2 p_k^4 p_{k+1}^4} \\ &= x + \frac{x_k x_{k+1}}{(x_k + x_{k+1})^2}. \end{aligned}$$

It is well known that there exists an efficient formula for computing the x-coordinate of $2P$ in terms of the x-coordinate of P, which corresponds to $x_{2k} = x_k^2 + \frac{b}{x_k^2}$.[3]

This completes the proof. □

In the following, the y-coordinate retrieval function of the point kP is re-discovered.

Proposition 3 *Given an elliptic point $P(x, y)$, the y-coordinate of kP can be computed in terms of x, y, x_k and x_{k+1}.*

[3] For the sake of completeness, we present in Appendix A, the derivation of the point doubling formula using division polynomials.

Proof From Proposition 1, we have:

$$\begin{aligned} y_k &= y + x_k + \frac{p_{k+1}^2 p_{k-2}}{x p_k^3} + (x^2 + y) \frac{p_{k+1} p_{k-1}}{x p_k^2} \\ &= y + x_k + \frac{p_{k+1}^2 p_{k-1}^2 p_{k-2} p_k}{x p_k^4 p_{k-1}^2} \frac{p_{k+1} p_{k-1}}{x p_k^2} \\ &= y + x_k + \frac{(x + x_k)^2 (x + x_{k-1})}{x} + x^2 + y + \left(x + \frac{y}{x}\right) x_k \\ &= x_k + (x + x_k)^2 + \frac{(x + x_k)^2}{x} x_{k-1} + x^2 + \left(x + \frac{y}{x}\right) x_k \\ &= x_k + x_k^2 + \left(x + \frac{y}{x}\right) x_k + \frac{(x + x_k)^2}{x} \left(x_{k+1} + \frac{x x_k}{(x + x_k)^2}\right) \\ &= x_k^2 + \left(x + \frac{y}{x}\right) x_k + \frac{(x + x_k)^2}{x} x_{k+1} \end{aligned}$$

□

Remark 1 In terms of the λ representation of a point $P = (x, y)$, i.e, $\lambda = x + \frac{y}{x}$, the formula y_k can be expressed as: $x_{k+1} = \frac{x x_k (\lambda_k + \lambda)}{(x + x_k)^2}$.

3 Montgomery ladders with pre-computation

Since Montgomery proposed his ladder strategy in 1987, there was no published work on a variant of this algorithm admitting an efficient off-line pre-computation phase. Pre-computation is a customary acceleration technique (at the price of memory storage) that is especially valuable for the so-called fixed point, also referred to as known-point scenario, where the base point being processed by a scalar multiplication algorithm is established in advance. It was only until 2014, when Oliveira et al. introduced in [51] a ladder variant that admits an off-line pre-computation phase. This feature can be applied to further accelerate the main on-line point multiplication.

In this section, we analyze and compare several efficient ladder algorithms that calculate a constant-time scalar multiplication by pre-computing multiples of the input point which are later processed in a right-to-left fashion.

3.1 Background

Let $f(x) \in \mathbb{F}_2[x]$ be a monic polynomial of prime degree m, irreducible over $\mathbb{F}_2$. Then, $\mathbb{F}_{2^m} \cong \mathbb{F}_2[x]/f(x)$ is a binary extension field of 2^m elements.

Next, consider a binary elliptic curve $E/\mathbb{F}_{2^m}$ and its cyclic additively written group of points $E(\mathbb{F}_{2^m})$ of order $\#E(\mathbb{F}_{2^m}) = h \cdot r$, with prime r. Then we have a subgroup

$G \subset E(\mathbb{F}_{2^m})$ of order r, where $n = \lceil \log_2 r \rceil$, and for any point $P \in G$, with $P \neq \mathcal{O}$, P is a generator of G. That is, $G = \langle P \rangle$.

The scalar multiplication algorithms discussed in this section receive as input a generator point $P \in G$, and an n-bit scalar $k \in \{1, \ldots, r-1\}$ and return the point $Q = kP$, which, as mentioned in the introduction of this article, is the process of adding P to itself $k-1$ times.

3.2 A right-to-left Montgomery ladder

In addition to the traditional left-to-right approach presented in Algorithm 1, one can compute the Montgomery ladder by processing the scalar k from the least to the most significant bit. This method is denominated Montgomery right-to-left double-and-add, and it is shown in Algorithm 2.

Algorithm 2 Montgomery right-to-left double-and-add [51]

Input: $P = (x, y) \in G$, $k = (k_{n-1}, k_{n-2}, \ldots, k_1, k_0)_2 \in \mathbb{Z}_r$
Output: $Q = khP$

1: **Initialization:** Select an order-h point $S \in E(\mathbb{F}_{2^m}) \setminus G$
2: $R_0 \leftarrow P$, $R_1 \leftarrow S$, $R_2 \leftarrow P - S$
3: **for** $i \leftarrow 0$ **to** $n-1$ **do**
4: **if** $k_i = 1$ **then**
5: $R_1 \leftarrow R_0 + R_1$
6: **else**
7: $R_2 \leftarrow R_0 + R_2$
8: **end if**
9: $R_0 \leftarrow 2R_0$
10: **end for**
11: **return** $Q = hR_1$

The rationale of Algorithm 2 can be explained as follows. At the beginning of the iteration i, with $i \in \{0, \ldots, n-1\}$, R_0 stores the point $2^i P$,[4] which will be added to the accumulator R_1 if the scalar digit k_i is equal to one.

As discussed in Sect. 2, in order to perform the Montgomery addition (Step 5), we need to know the difference between the accumulator R_1 and the point R_0. However, unlike the classical left-to-right ladder, this difference is not fixed, but may vary at each iteration; more specifically, it will change whenever the digit k_i is equal to zero. For that reason, we have to occasionally update and store this varying difference in the point R_2 (Step 7).

In other words, if the bit k_i is one, R_0 is added to the accumulator R_1 and their difference does not need to be updated because the addition is compensated in Step 9 as $2R_0 = R_0 + R_0$. On the other hand, if the digit k_i is zero, R_0 is not added to the accumulator. As a consequence, the difference between R_1 and R_0 increases by R_0, and R_2 must be updated.

There are (at least) three particularities in the right-to-left algorithm worth to be mentioned. First, we should guarantee that, at every iteration, $R_0 \neq R_1$ and $R_0 \neq R_2$, so that the point additions (Steps 5 and 7) are valid. We can achieve this by initializing R_1 as a point $S \in E(\mathbb{F}_{2^m}) \setminus G$ of order h. At the end of the algorithm, the point can be subtracted (Step 11).[5][6] Note that this issue can be disregarded for curves whose addition law's formulas are complete (e.g., binary Edwards curves [8]).

Second, given that the difference point $R_2 = R_0 - R_1$ is volatile and must be updated via a Montgomery addition, its coordinates should be represented in projective coordinates, where the Z-coordinate will generally have a value different than one. As a result, the Montgomery addition $R_3 = R_0 + R_1$ (with $R_2 = R_0 - R_1$) is more expensive when compared with Eq. (6), and it is calculated as,

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

costing six field multiplications, one squaring and two additions. In the case where the multiplication by the curve parameter $c = \sqrt{b}$ is cheap, it is preferred to compute the addition via the following formula

$$\begin{aligned} T &= (X_0 + Z_0) \cdot (X_1 + Z_1) \\ A &= X_0 \cdot Z_0 \\ B &= X_1 \cdot Z_1 \\ Z_3 &= Z_2 \cdot (T + A + B)^2 \\ X_3 &= X_2 \cdot (A + c \cdot B)^2, \end{aligned} \tag{8}$$

which costs five field multiplications, one multiplication by the parameter c, two squarings and five additions.

Finally, after the execution of the n iterations of Algorithm 2, the y-coordinates of the points R_0, R_1 and R_2 are unknown. Therefore, it is not possible to retrieve the y-coordinate of the resulting point Q using the approach given in [43]. This hindrance is of no concern for the Diffie–Hellman key exchange and digital signature protocols, since their outputs are derived only from the x-coordinates of the resulting points [30].

In the fixed-point scenario, it is indeed possible to retrieve the y-coordinate, since we can pre-compute the y-coordinate of the last assignation to R_0: $R_0 \leftarrow 2^n P$. Nonetheless, if the

[4] According to Algorithm 2, at the end of the iteration $n-1$, $2^n P$ is assigned to R_0. The point $2^n P$ is never used in the unknown-point scenario; therefore, this assignation can be avoided in practical implementations.

[5] In binary fields, the value of h can be as low as 2. Thus, the process of subtracting S is merely a Montgomery point doubling.

[6] In fixed-point scenarios, the y-coordinate of R_1 can be retrieved. Consequently, Step 11 can be alternatively computed as $Q = R_1 - S$.

point P is known beforehand, Algorithm 2 can be further optimized by pre-computing and storing the multiples $2^i P$. This approach is analyzed in the next section.

3.3 Montgomery ladders with pre-computation

Algorithm 2 can be adapted by adding an initial pre-computation phase where the multiples $2^i P$, with $i \in \{0, \ldots, n\}$, are calculated and stored in the internal memory. This approach is shown in Algorithm 3.

Algorithm 3 Montgomery right-to-left double-and-add with pre-computation

Input: $P = (x, y) \in G, k = (k_{n-1}, k_{n-2}, \ldots, k_1, k_0)_2 \in \mathbb{Z}_r$
Output: $Q = khP$

1: **Pre-computation:** Calculate $P_i = 2^i P$, for $i \in \{0, \ldots, n\}$
2: **Initialization:** Select an order-h point $S \in E(\mathbb{F}_{2^m}) \setminus G$
3: $R_0 \leftarrow P$, $R_1 \leftarrow S$, $R_2 \leftarrow P - S$
4: **for** $i \leftarrow 0$ **to** $n - 1$ **do**
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: $R_0 \leftarrow P_{i+1}$
11: **end for**
12: **return** $Q = hR_1$

Concisely, Algorithm 3 pre-computes in Step 1 the multiples $2^i P$ that are used in the ladder iterations. Those multiples are computed through the Montgomery doubling formula (5), which returns the projective coordinates X and Z. For that reason, it is required a memory storage of approximately $2mn$ bits.[7] Next, instead of performing on-line point doublings in Step 10, the algorithm just access the appropriate point P_i from the lookup table pre-computed in Step 1.

The most immediate advantage of this algorithm is that, in the fixed-point scenario, one can save all the point doubling computations at the price of memory storage and two extra field multiplications per iteration, which are embedded in the modified Montgomery addition, Eq. (7). In addition, one can further optimize Algorithm 3 by applying the GLV method discussed in the introduction of this paper.

Let us consider a binary curve $E/\mathbb{F}_{2^m}$ equipped with an efficiently computable endomorphism ψ that allows degree-2 decompositions (e.g., binary GLS curves [26]), where $k \equiv k_0 + k_1\delta \pmod r$, with $|k_1|, |k_2| \approx n/2$ and $\psi(P) = \delta P$. Then, Algorithm 3 can be combined with the GLV method, as shown in Algorithm 4.

[7] In the fixed-point setting, one can store the multiples in the affine form. As a result, only mn bits of storage is needed.

[8] If the order h is prime, then one can return instead the point $Q = h(R_{1,0} + \psi(R_{1,1}))$.

Algorithm 4 Montgomery 2-GLV right-to-left double-and-add with pre-computation

Input: $P = (x, y) \in G, k \in \mathbb{Z}_r$
Output: $Q = khP$

1: **Pre-computation:** Calculate $P_i = 2^i P$, for $i \in \{0, \ldots, \frac{n}{2}\}$
2: **Initialization:** Decompose the scalar k as $k \equiv k_0 + k_1\delta \pmod r$ via the GLV method. Select an order-h point $S \in E(\mathbb{F}_{2^m}) \setminus G$
3: $R_0 \leftarrow P$, $R_{1,j} \leftarrow S$, $R_{2,j} \leftarrow P - S$, for $j \in \{0, 1\}$
4: **for** $i \leftarrow 0$ **to** $\frac{n}{2} - 1$ **do**
5: **for** $j \leftarrow 0$ **to** 1 **do**
6: **if** $k_{i,j} = 1$ **then**
7: $R_{1,j} \leftarrow R_0 + R_{1,j}$
8: **else**
9: $R_{2,j} \leftarrow R_0 + R_{2,j}$
10: **end if**
11: **end for**
12: $R_0 \leftarrow P_{i+1}$
13: **end for**
14: **return** $Q = (hR_{1,0}) + \psi(hR_{1,1})$ [8]

Here, the number of point doublings were reduced by half, which results in less computing time and less required storage resources; here, only nm bits of memory space are necessary. Note that even in the unknown-point scenario, one can profit from this algorithm, by exchanging a smaller number of point doublings with a more expensive point addition. The feasibility of this trade will be evaluated in Sect. 3.6.

In the remaining of this section, it is shown how to use the point halving operation to reduce the cost of the right-to-left point additions.

3.4 The point halving operation

In 1999, Knudsen [34] and Schroppel [57,58] proposed the application of the point halving operation to compute scalar multiplications. Given a point $P \in E(\mathbb{F}_{2^m})$, the point halving operation consists of computing a point R such that $P = 2R$. The interested reader is referred to [3,19,51,52] for a discussion on how to implement point halvings efficiently.

Let $k \in \mathbb{Z}_r$ be a scalar of n bits. Then, in order to perform a scalar multiplication with point halvings, k must be first manipulated as $k' = 2^{n-1}k \mod r$. Consequently, $k \equiv \sum_{i=1}^{n} k'_{n-i}/2^{i-1} \pmod r$, and therefore, $kP = \sum_{i=1}^{n} k'_{n-i}(\frac{P}{2^{i-1}})$. This implies that we can compute $Q = kP$ by performing consecutive point halvings on P. This method is called halve-and-add and is presented in Algorithm 5.

Computing the scalar multiplication via point halvings is usually more efficient than the traditional double-and-add algorithm. This is because we need about five field multiplications and four squarings to compute the point doubling with λ-projective coordinates, while only one half-trace, one square root and one field multiplication are required to calculate a point halving. For that reason, the halve-and-add

Algorithm 5 Halve-and-add as described in [19]

Input: $P = (x, y) \in G, k \in \mathbb{Z}_r$
Output: $Q = kP$

1: **Initialization:** Compute scalar $k' = 2^{n-1}k \mod r = (k'_{n-1}, k'_{n-2}, \ldots, k'_1, k'_0)_2$
2: $Q \leftarrow \mathcal{O}$
3: **for** $i \leftarrow n-1$ **downto** 0 **do**
4: **if** $k'_i = 1$ **then**
5: $Q \leftarrow Q + P$
6: **end if**
7: $P \leftarrow P/2$
8: **end for**
9: **return** Q

Table 1 Ratio between the cost of selected finite field $\mathbb{F}_{2^{254}}$ arithmetic operations and the field multiplication

Operation	Ratio op./*mul*
Multiplication (*mul*)	1.00
Squaring (*sqr*)	0.62
Square root (*sqt*)	0.62
Half-trace (*hft*) (quadratic equation solver)	4.58

became a popular approach for implementing fast point multiplication algorithms on binary elliptic curves.[9]

To illustrate the performance of the point halving operation, a concrete comparison using our software implementation of the field $\mathbb{F}_{2^{254}}$ arithmetic in a 64-bit Intel Skylake processor is reported next.

In Table 1, we present the ratio of the costs, measured in clock cycles, between the finite field elementary arithmetic functions required for computing the point doubling and point halving, and the field multiplication, which is the most crucial operation in the design of efficient point multiplication implementations.

According to the estimations presented in the previous paragraphs, doubling a point using λ-projective coordinates would cost 7.48 *mul*, while a point halving costs only 6.20 *mul*, which is about 17.11% faster. Another advantage of the point halving function is that its output is given in λ-affine coordinates. As a result, we can compute faster point additions in right-to-left point multiplication algorithms, since the Z-coordinates of the points $P/2^i$ will all be equal to 1.

3.5 An efficient halve-and-add Montgomery ladder

Applying the point halving to the Montgomery ladder context is not straightforward. The main reason being the fact that the only known efficient formulas for computing $R = P/2$ assume that the input point is given in affine coordinates. On the other hand, Montgomery ladder algorithms manipulate accumulator points in projective coordinates in order to avoid the costly field multiplicative inversions.

The authors of [50] partially solved this problem by converting the accumulator point from projective to affine representation at every ladder step. Nevertheless, this approach is too expensive, since it requires one inversion per iteration, with an associated cost of about 29.38 *mul*, which becomes prohibitive for most scenarios.

The Montgomery halve-and-add method presented in [51] was the first to apply the point halving operation on Montgomery ladders without the necessity of performing the expensive inversion function over finite fields. This was achieved by combining the pre-computation approach of Algorithm 3 with the halve-and-add procedure shown in Algorithm 5. The resulting procedure is presented in Algorithm 6.

Algorithm 6 Montgomery halve-and-add (right-to-left)

Input: $P = (x, y) \in G, k \in \mathbb{Z}_r$
Output: $Q = khP$

1: **Pre-computation:** Compute and store only the x-coordinates of $P_i = \frac{P}{2^i}$, for $i \in \{0, \ldots, n-1\}$
2: **Initialization:** Compute scalar $k' = 2^{n-1}k \mod r = (k'_{n-1}, k'_{n-2}, \ldots, k'_1, k'_0)_2$. Select an order-$h$ point $S \in E(\mathbb{F}_{2^m}) \setminus G$
3: $R_1 \leftarrow S$, $R_2 \leftarrow P_{n-1} - S$
4: **for** $i \leftarrow 0$ **to** $n-1$ **do**
5: $R_0 \leftarrow P_{n-1-i}$
6: **if** $k'_i = 1$ **then**
7: $R_1 \leftarrow R_0 + R_1$
8: **else**
9: $R_2 \leftarrow R_0 + R_2$
10: **end if**
11: **end for**
12: **return** $Q = hR_1$

In the main loop, right after pre-computing n halved points (Step 1), one can manipulate them in a backwards fashion, by simulating a Montgomery right-to-left double-and-add algorithm with pre-computation. The main advantage of this approach is that the coordinates of $R_0 = P_i$ are represented in λ-affine coordinates, that is, Z_0 is always equal to 1. As a result, the Montgomery addition formula [see Eq. (7)] can be simplified as shown the following equation:

$$\begin{aligned} T &= (X_0 \cdot Z_1 + X_1)^2 \\ Z_3 &= Z_2 \cdot T \\ X_3 &= X_2 \cdot T + Z_2 \cdot (X_0 \cdot Z_1) \cdot X_1, \end{aligned} \tag{9}$$

which costs five field multiplications, one squaring and two additions. Similarly, Eq. (9) leads to a faster addition, if the multiplication by the elliptic curve parameter b is cheap.

Given that R_0 is represented in λ-affine coordinates, we can retrieve the R_1 y-coordinate in the fixed- and variable-point scenarios. At the end of the algorithm, we only have

[9] It is not known how to efficiently half a point in elliptic curves defined over prime fields.

Table 2 Point arithmetic costs in terms of field multiplications over the field $\mathbb{F}_{2^{254}}$

Operation	Cost	
	$\mathbb{F}_{2^{254}}$ functions	$\mathbb{F}_{2^{254}}$ mult.
Montgomery point doubling (*DBL*) (Eq. (5))	1.6 $mul^{\dagger}$ + 3 sqr	3.46 mul
Point halving (*HLV*) [34,57]	1 mul + 1 sqt + 1 hft	6.20 mul
Montgomery point addition (*ADA*) (Eq. (6))	4 mul + 1 sqr	4.62 mul
Montgomery point addition (*ADF*) (Eq. (7))	6 mul + 1 sqr	6.62 mul
Montgomery point addition (*ADM*) (Eq. (9))	5 mul + 1 sqr	5.62 mul

† The multiplication by the term $\sqrt{b}$ [see Eq. (5)] can be performed via one multiplication in the base field $\mathbb{F}_{2^{127}}$ [51], which costs about 0.60 mt

to compute $R_0 \leftarrow 2P$[10] and apply the retrieval function of [43].

The Montgomery halve-and-add algorithm can be combined with the 2-GLV decomposition method as well. In that case, the number of pre-computed halved points would be reduced by a factor of two. Given that the halved points are represented in λ-affine coordinates, the required pre-computing memory space of this approach is of only $\frac{mn}{2}$ bits.

3.6 Comparison

In this subsection, we compare the algorithms and operations previously discussed. First, in Table 2, we show an analysis of the costs of different point arithmetic operations used in Montgomery ladders in terms of the field arithmetic functions described in Table 1 and the field multiplication. The field additions were disregarded since, for binary fields, this operation can be implemented via an exclusive or logical operator, which is nearly free of cost in high-end desktop architectures.[11]

At first glance, Table 2 shows that substituting Montgomery point doublings by point halvings is not a good idea, since the latter costs 2.74 mul more than the former. However, in the 2-GLV method, both operations are shared in the 'parallel' computation of k_0P and k_1P (see Algorithm 4). Since only $\frac{n}{2}$ of such operations are required, we can conclude that the cost per iteration of *DBL* and *HLV* in the 2-GLV algorithms are 1.73 mul and 3.10 mul, respectively. Now, the HLV is only 1.37 mul more expensive than DBL. Given that the Montgomery addition for halve-and-add (ADM) is cheaper than the point addition for double-and-add (ADF) by 15.11%, we can expect similar timings for the two approaches.

In Table 3, we report the cost estimates, in terms of the point operations listed in the previous table and also in terms of field multiplications, for the ladder algorithms described in this section considering the variable-point scenario.

In the first place, it is important to stress that, for the unknown-point scenario, the pre-computation phase does not bring any improvement (besides the fact that it supports the halve-and-add approach), since this technique only transfers the work of computing the multiples of the base point P from the main loop to the initial part of the algorithm.

Also, in such scenario, it is convenient to consider the 2-GLV method, since it results in significant improvements on the double-and-add and halve-and-add algorithms. Regarding the former, it improves Algorithms 2 and 3 by 22.69%, while Algorithm 6 is improved by 26.23%.

Finally, Table 3 shows that the classical double-and-add left-to-right ladder is the state-of-the-art approach for the variable-point setting, surpassing the 2-GLV double-and-add and halve-and-add ladder procedures by a tiny difference of 3.23 and 7.22%, respectively.

In Table 4, we examine the fixed-point setting. In this scenario, the most efficient cases are clearly the Montgomery ladder algorithms with a pre-computation phase. Because of its faster point addition (ADM), the halve-and-add surpasses the right-to-left double-and-add method by 15.11%. Moreover, the 2-GLV halve-and-add approach achieves the most efficient storage requirement within ladders based on pre-computation. A memory comparison is given in Table 5.

When computing one scalar multiplication with scalars of approximately 256 bits on curves defined over a field of the same size, the 2-GLV halve-and-add method requires about 4.28 KB, while 8.31 KB are necessary in the double-and-add analogue. This represents storage space savings of more than 48%.

Hence, we can conclude that the Montgomery halve-and-add method is the most efficient in terms of CPU speed and memory consumption in the fixed-point scenario. In the variable-point setting, the classical double-and-add left-to-right algorithm is still the most convenient alternative.

[10] In Algorithm 6, R_0 is updated after R_1 and R_2.

[11] The latency and the throughput of the vector instruction `pxor` in current desktop architectures are of 1 and 0.33 clock cycles, respectively [31].

Table 3 Montgomery ladder scalar multiplication costs in the variable-point scenario. See Table 2 for the definition of the point arithmetic operations

Ladder algorithm		Cost per iteration	
		Point arithmetic	$\mathbb{F}_{2^{254}}$ mult.
Double-and-add left-to-right (Algorithm 1)		$DBL + ADA$	8.08 *mul*
Double-and-add right-to-left	No-GLV (Algorithm 2)	$DBL + ADF$	10.08 *mul*
	2-GLV	$\frac{1}{2}DBL + ADF$	8.35 *mul*
Double-and-add right-to-left with pre-comp.	No-GLV (Algorithm 3)	$DBL + ADF$	10.08 *mul*
	2-GLV (Algorithm 4)	$\frac{1}{2}DBL + ADF$	8.35 *mul*
Halve-and-add right-to-left	No-GLV (Algorithm 6)	$HLV + ADM$	11.82 *mul*
	2-GLV	$\frac{1}{2}HLV + ADM$	8.72 *mul*

Table 4 Montgomery ladder scalar multiplication costs in the fixed-point scenario. See Table 2 for the definition of the point arithmetic operations

Ladder algorithm		Cost per iteration	
		Point arithmetic	$\mathbb{F}_{2^{254}}$ mult.
Double-and-add left-to-right (Algorithm 1)		$DBL + ADA$	8.08 *mul*
Double-and-add right-to-left	No-GLV (Algorithm 2)	$DBL + ADF$	10.08 *mul*
	2-GLV	$\frac{1}{2}DBL + ADF$	8.35 *mul*
Double-and-add right-to-left with pre-comp.	No-GLV (Algorithm 3)	ADF	6.62 *mul*
	2-GLV (Algorithm 4)	ADF	6.62 *mul*
Halve-and-add right-to-left	No-GLV (Algorithm 6)	ADM	5.62 *mul*
	2-GLV	ADM	5.62 *mul*

Table 5 Memory requirement (in bits) for different Montgomery ladder algorithms. Assume an n-bit scalar and a binary finite field of 2^m elements

Algorithm		Memory required (in bits)†				
		P_i	R_0	R_1	R_2	Total
Double-and-add left-to-right (Algorithm 1)		m	$2m$	$2m$	×	$5m$
Double-and-add right-to-left	No-GLV (Algorithm 2)	×	$2m$	$2m$	$2m$	$6m$
	2-GLV	×	$2m$	$4m$	$4m$	$10m$
Double-and-add right-to-left w/ pre-comp.	No-GLV (Algorithm 3)	$2nm$	$2m$	$2m$	$2m$	$(2n+6)m$
	2-GLV (Algorithm 4)	nm	$2m$	$4m$	$4m$	$(n+10)m$
Halve-and-add right-to-left	No-GLV (Algorithm 6)	nm	m	$2m$	$2m$	$(n+5)m$
	2-GLV	$\frac{nm}{2}$	m	$4m$	$4m$	$(\frac{n}{2}+9)m$

† The points required to recover the y-coordinate of $Q = kP$ are not considered in this summary

4 A software implementation of Montgomery ladder-based elliptic curve protocols

In this section, software implementations for different 128-bit secure Montgomery ladder scalar multiplication algorithms on binary GLS curves are presented.[12]

4.1 Field and curve parameters

The base field $\mathbb{F}_{2^{127}} \cong \mathbb{F}_2[x]/(f(x))$ was generated using the irreducible polynomial,

$$f(x) = x^{127} + x^{63} + 1.$$

The quadratic field $\mathbb{F}_{2^{254}} \cong \mathbb{F}_{2^{127}}[u]/(g(u))$ is constructed by means of the degree-2 irreducible polynomial,

[12] The implementation to be described here closely follows the one presented by Oliveira et al. in [51].

$g(u) = u^2 + u + 1$.

In the following, the parameters of the binary GLS curve

$$E_{a,b}/\mathbb{F}_{2^{254}} : y^2 + xy = x^3 + ax^2 + b,$$

are presented. All the polynomials representing the field elements will be given using integer hexadecimal numbers:

$a = \texttt{0x1}u,$
$b = \texttt{0x5404514441040154410154054051510}1.$

The rationale for selecting the b parameter was based on the X-coordinate computation in the Montgomery projective doubling $R_3 = 2R_1$. Let us recall that its formula requires one multiplication by the square root of b,

$$X_3 = X_1^4 + b \cdot Z_1^4 = (X_1^2 + \sqrt{b} \cdot Z_1^2)^2.$$

Since our software architecture provides a 64-bit carry-less multiplier, we selected b such that its square root $\sqrt{b} =$ `0xE2DA921E91E38DD1` is also of a size of 64 bits. Thanks to this choice, we could save one carry-less multiplication and many logical operations in the field multiplication by this constant.

Lastly, our base point P belongs to a subgroup $G \subset E_{a,b}(\mathbb{F}_{2^{254}})$ of prime order r of about 253 bits given by

$r = \texttt{0x1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF}$
$\texttt{A6B89E49D3FECD828CA8D66BF4B88ED5}.$

The coordinates of $P = (x, y)$ are,

$x = \texttt{0x4A21A3666CF9CAEBD812FA19DF9A3380}$
$+ \texttt{0x358D7917D6E9B5A7550B1B083BC299F3} \cdot u,$
$y = \texttt{0x6690CB7B914B7C4018E7475D9C2B1C13}$
$+ \texttt{0x2AD4E15A695FD54011BA179D5F4B44FC} \cdot u.$

4.2 Constant-time implementation

The Montgomery ladder technique is quite appropriate for implementing a constant-time scalar multiplication algorithm, since the number of operations executed in each iteration of the main loop does not depend on the scalar digits k_i. Namely, in terms of operations, the algorithm structure is inherently constant. There is no need to perform any extra computation in order to process the scalar digits regularly.

Nevertheless, the digits k_i actually determine the order of how the points R_i are loaded from and stored to the memory. This issue can be exploited by cache-based attacks to obtain sensitive information from the scalar k [36]. For that reason, it is necessary to design a data veil mechanism to secure such vulnerability.

Consider the Montgomery left-to-right double-and-add procedure (Algorithm 1). We can reduce the two possible Montgomery additions (Steps 4 and 6) to the following pattern:

$$R_{k_i} \leftarrow R_{k_i} + R_{1-k_i}, \; R_{1-k_i} \leftarrow 2R_{1-k_i}, \quad \text{with } k_i \in \{0, 1\}.$$

Now, let $\mathcal{M}_0$ and $\mathcal{M}_1$ be two memory locations in the RAM whose contents will probably be cached during the execution of the scalar multiplication algorithm. Next, we rewrite the Montgomery additions in terms of fixed memory locations as

$$\mathcal{M}_0 \leftarrow \mathcal{M}_0 + \mathcal{M}_1, \; \mathcal{M}_1 \leftarrow 2\mathcal{M}_1.$$

Then, instead of having two possible set of instructions with different operands (depending on the scalar bit k_i), we have only one set, written as operations over constant memory locations.

As a result, the only thing that will vary with the bits k_i are the contents of $\mathcal{M}_0$ and $\mathcal{M}_1$. That is, when the digit k_i is zero, $\mathcal{M}_0 \leftarrow R_0$ and $\mathcal{M}_1 \leftarrow R_1$; otherwise, $\mathcal{M}_0 \leftarrow R_1$ and $\mathcal{M}_1 \leftarrow R_0$. Nonetheless, since the only data holding places that we have are $\mathcal{M}_0$ and $\mathcal{M}_1$, then these assignments are, in practice, data swaps.

Our data veil mechanism initializes $\mathcal{M}_0 \leftarrow R_0$ and $\mathcal{M}_1 \leftarrow R_1$. Also, it sets a flag named `switch` with the value 1. After that, at each loop iteration i, we compute

$$\texttt{switch} \leftarrow \texttt{switch} \oplus k_i.$$

This updated flag generates a 128-bit mask which determines via logical instructions whether the contents of the memory locations $\mathcal{M}_0$ and $\mathcal{M}_1$ will be swapped or not.

The procedure is described in Algorithm 7. Here, $[\alpha, \beta]$ denotes a 128-bit vector register that stores two 64-bit values α and β. Also, the symbols $\wedge$ and $\oplus$ represents the logical instructions 'and' (`pand`) and exclusive or (`pxor`), respectively. The logical conjunction with negation ($\neg$) of the first operand is implemented through the instruction `pandn`.

Algorithm 7 Data veil procedure

Input: Memory locations $\mathcal{M}_0$, $\mathcal{M}_1$, `switch` flag, bit k_i
Output: The contents of the points R_0, R_1 stored in $\mathcal{M}_0$, $\mathcal{M}_1$ according to k_i
1: `one` $\leftarrow$ [0x1, 0x1]
2: `switch` $\leftarrow$ `switch` $\oplus\, k_i$
3: `mask` $\leftarrow$ [`switch`, `switch`] - `one`
4: $\mathcal{M}_0 \leftarrow$ (`mask` $\wedge\, \mathcal{M}_0$) $\oplus$ ($\neg$`mask` $\wedge\, \mathcal{M}_1$)
5: $\mathcal{M}_1 \leftarrow$ (`mask` $\wedge\, \mathcal{M}_1$) $\oplus$ ($\neg$`mask` $\wedge\, \mathcal{M}_0$)
6: Proceed to the Step 6 or 4 of Algorithm 1

Table 6 Point arithmetic timings (in clock cycles)

Operation	Cycles
λ-projective point doubling [52]	135
Point halving (*HLV*) [34,57]	177
Montgomery point doubling (*DBL*) (Eq. (5))	66
Montgomery point addition (*ADA*) (Eq. (6))	133
Montgomery point addition (*ADM*) (Eq. (9))	157

For right-to-left Montgomery ladders, the data veil countermeasure is similar. The only difference is that, in these algorithms, the addition operation is fixed as

$$\mathcal{M}_0 \leftarrow R_0 + \mathcal{M}_0.$$

Then, the content of one of the points R_1, R_2 is temporarily stored in an auxiliary memory location.

4.3 Timings

The software implementation presented in this work was designed for 64-bit desktop architectures equipped with 64-bit carry-less multipliers and 128-bit vector instructions. The library was benchmarked in an Intel Core i7-6700K 4.00 GHz (Skylake) machine, coded in GNU11 C/Assembly and compiled with the *clang* frontend for the *LLVM* compiler version 3.8 with the following optimization flags:

```
-march=skylake -O3 -fomit-frame-pointer.
```

First, we report in Table 6 the timings for the distinct point arithmetic functions presented in Table 2.

Contrary to the estimations presented in Sect. 3.4, the λ-projective point doubling is 23.73% faster than the point halving. One possible explanation is that, in practice, the latency and throughput of the carry-less multiplier instruction (`pclmulqdq`) overcomes the cost of multiple memory access on pre-computed tables, inherent in the half-trace (hft) computation. This observation could also explain the fact that the point halving operation is about 2.68 times as costly as the computation of one Montgomery point doubling.

Also, the extra field multiplication in the ADM addition, when compared to the ADA corresponds to only 24 clock cycles, which amounts to an extra cost of 15.29%.

Next, we compare in Table 7 the timings of the different phases of the classical Montgomery left-to-right double-and-add and the halve-and-add approach introduced in Sect. 3.4.

Here we can verify that, in the variable-point scenario, the traditional Montgomery double-and-add is the fastest approach, surpassing the Montgomery halve-and-add with and without the 2-GLV method by 23.91 and 41.20%, respectively.

Table 7 Montgomery ladder scalar multiplication timings (in clock cycles). The terms 'Pre' and '`Mxy`' stand for the pre-computation phase and the y-coordinate retrieval function, respectively

Montgomery ladder-based algorithm	Cycles			
	Pre	Main loop	`Mxy`	Total
Double-and-add left-to-right (Algorithm 1)	n/a	50823	1203	52026
Halve-and-add right-to-left, No-GLV (Algorithm 6)	42395	44879		88477
Halve-and-add right-to-left, 2-GLV	22288			68370

In fixed-point scenarios, such as the one that occurs in the key pair generation phase of the Diffie–Hellman key agreement protocol [30], the halves of the base point can be pre-computed off-line. In this situation, the halve-and-add algorithm has an on-line cost of just 46, 082 clock cycles (by disregarding Table 7 column 'Pre'), which is about 1.13 faster than the double-and-add method. Furthermore, this implies that the full computation of the Diffie–Hellman protocol, whose two main operations are one fixed- and one variable-point multiplication, can be computed in less than 100, 000 clock cycles [cf. with Table 8].

5 Survey of papers implementing the Montgomery ladder for binary elliptic curves

In this section, we present a selection of several research papers that have implemented the Montgomery ladder on binary elliptic curves. Our survey includes both hardware and software implementations that presented a number of research novelties in the ladder computation with respect to efficiency and/or side-channel protections.

5.1 The Montgomery ladder on Binary Edwards curves

Let $d_1, d_2 \in \mathbb{F}_q$, such that $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$. The Edwards form of a binary elliptic curve is defined as,[13]

$$E_{BE} : d_1(x+y)+d_2(x^2+y^2) = xy+xy(x+y)+x^2y^2. \quad (10)$$

[13] We stress that binary Weierstrass and Edward curves are birational equivalent [8].

In binary Edwards curves, an affine point can be represented using w-coordinates by using the mapping $(x, y) \rightarrow (w)$, where $w = x + y$. However, in order to minimize the field multiplicative inversion operations, the most efficient formulas for Edward curves are defined using a mixed coordinate point representation where an extra coordinate Z, is defined so that $(x, y) \rightarrow (w) \rightarrow (W, Z)$, where $w = x + y = \frac{W}{Z}$.

Binary Edward curves were presented by Bernstein et al. at CHES 2008 [8]. The authors reported complete formulas for point addition and doubling that allow one to compute a Montgomery ladder step at the price of five field multiplications, two multiplications by a constant and four squarings. In [38], the authors corrected the formulas presented by Kim et al in [33], thus reporting more efficient formulas than the ones given in [8]. Indeed, by choosing $d_1 = d_2$ in Eq. (10), the authors of [38] reported a ladder step cost of just five field multiplications, one multiplication by a constant and four squarings. Remarkably, this result exactly matches the cost of the ladder step for the Weierstrass form of binary elliptic curves.

Algorithm 8 Montgomery ladder step: point addition and doubling

Input: $P_1 = (X_1, Z_1), P_2 = (X_2, Z_2)$ a base point $P = (x, y)$.
Output: $P_1 = P_1 + P_2$.
1: $T_1 \leftarrow x$;
2: $X_1 \leftarrow X_1 \cdot Z_2$;
3: $Z_1 \leftarrow Z_1 \cdot X_2$;
4: $T_2 \leftarrow X_1 \cdot Z_1$;
5: $Z_1 \leftarrow Z_1 + X_1$;
6: $Z_1 \leftarrow Z_1^2$
7: $X_1 \leftarrow Z_1 \cdot T_1$;
8: $X_1 \leftarrow X_1 + T_2$;
9: **return** (X_1, Z_1).

Input: $P_2 = (X_2, Z_2)$ a curve parameter $c = \sqrt{b}$.
Output: $P_2 = 2 \cdot P_2$.
10: $T_1 \leftarrow c$;
11: $X_2 \leftarrow X_2^2$;
12: $Z_2 \leftarrow Z_2^2$;
13: $T_1 \leftarrow T_1 \cdot Z_2$;
14: $T_1 \leftarrow X_2 + T_1$;
15: $Z_2 \leftarrow X_2 \cdot Z_2$;
16: $X_2 \leftarrow T_1^2$;
17: **return** (X_2, Z_2).

5.2 Common-Z trick

As shown in Algorithm 8, in a conventional implementation of the Montgomery ladder one requires seven registers $(x, X_1, Z_1, X_2, Z_2, T_1, T_2)$ to compute a ladder step. However, when dealing with light hardware implementations that must be deployed over constrained and highly constrained environments, registers can take up to 80% of the gate area [40]. So, there is a compelling reason for reducing the number of registers and simplify the register file management.

In [45], Meloni proposed formulas for the common-Z projective coordinate system. His formulas were derived in the context of elliptic curves defined over prime fields. Soon after, Lee et al. presented in [40] an adaptation of this idea to the context of binary elliptic curves, which is based on the following observation.

Two different points $P_1 = (X_1, Z_1), P_2 = (X_2, Z_2)$ can be forced to have the same Z-coordinate by applying the following trick,

$$\begin{aligned} X_1 &\leftarrow X_1 \cdot Z_2; \\ X_2 &\leftarrow X_2 \cdot Z_1; \\ Z &\leftarrow Z_1 \cdot Z_2. \end{aligned} \tag{11}$$

By assuming that the points P_1 and P_2 share the same Z-coordinate, one has that the original Montgomery formulation for point addition [cf. Eq. (6)],

$$\begin{aligned} Z_{ADD} &= (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2; \\ X_{ADD} &= x \cdot Z_{ADD} + X_1 \cdot Z_2 \cdot X_2 \cdot Z_1, \end{aligned}$$

becomes,

$$\begin{aligned} Z_{ADD} &= (X_1 + X_2)^2; \\ X_{ADD} &= x \cdot Z_{ADD} + X_1 \cdot X_2, \end{aligned}$$

whenever $Z_1 = Z_2$. Now, since at each ladder step a point doubling must be computed, one needs to re-apply Eq. (11) to assure that the point that has just been doubled (either P_1 or P_2), still shares the same Z-coordinate with the other point (either P_2 or P_1).

Algorithm 9 computes a Montgomery ladder step using seven field multiplications, four squarings and three additions. This has to be compared with the cost of the original formulation, whose cost is of six field multiplications, five squarings and three additions.

In other words, by trading one field multiplication by a field squaring, one is able to reduce the number of registers from seven to just six. This saving is usually quite valuable for hardware implementations over constrained environments.

Algorithm 9 Common-Z point addition and doubling trick [40]

Input: $P_1 = (X_1, Z), P_2 = (X_2, Z)$ a base point $P = (x, y)$. A curve parameter $c = \sqrt{b}$.
Output: $P_1 = P_1 + P_2, P_2 = 2 \cdot P_2$.
1: $T_2 \leftarrow X_1 + X_2$;
2: $T_2 \leftarrow T_2^2$;
3: $T_1 \leftarrow X_1 \cdot X_2$;
4: $X_1 \leftarrow x$;
5: $X_1 \leftarrow T_2 \cdot X_1$;
6: $X_1 \leftarrow X_1 + T_1$;
7: $X_2 \leftarrow X_2^2$;
8: $Z \leftarrow Z^2$;
9: $T_1 \leftarrow c$;
10: $T_1 \leftarrow Z \cdot T_1$;
11: $Z \leftarrow Z \cdot X_2$;
12: $X_2 \leftarrow X_2 + T_1$;
13: $X_2 \leftarrow X_2^2$;
14: $X_1 \leftarrow X_1 \cdot Z$;
15: $X_2 \leftarrow X_2 \cdot T_2$;
16: $Z \leftarrow Z \cdot T_2$;
17: **return** $(X_1, Z), (X_2, Z)$.

5.3 Multi-dimensional Montgomery ladders

Several elliptic curve-based protocols must compute simultaneously two or more scalar multiplications. This situation is fairly common in a number of protocols proposed in the literature. For example, in the case of the ECDSA verification procedure [2], one step in the procedure implies the computation of $u_1P + u_2Q$. Moreover, as it was mentioned in the previous sections, for those elliptic curves having a two-dimensional endomorphism, a scalar multiplication can be calculated by means of the GLV trick, as the simultaneous computation of two scalar multiplications each one of them associated with two subscalars that have half the size of the original scalar [See Eq. (2) in the Introduction].

By using *differential addition chains*, Bernstein presented in [5], a Montgomery ladder algorithm that can compute this two-dimensional scalar multiplication operation more efficiently than a straightforward approach that would compute the above operation by performing two independent Montgomery ladders. The basic idea in [5] is that at each ladder step, exactly two Montgomery additions and one Montgomery doubling must be calculated, where the three points whose x-coordinates are computed at each iteration have differences of the form $\pm P \pm Q$.

It is noticed that the procedure presented in Algorithm 4 exhibits also a regular pattern of exactly two Montgomery point additions and one Montgomery point doubling per ladder step. However, Algorithm 4 can only be used in the context where a computation *à la* GLV of the form $Q = kP = k_1P + k_2\psi(P) = k_1P + k_2 \cdot \delta P$, must be performed. One advantage of Algorithm 4 over the procedure presented in [5], is that the former admits a pre-computation step for the fixed-point scalar multiplication scenario.

In [11,29,61], the case where one wants to compute three or more scalar multiplications simultaneously was addressed. The applications of such simultaneous scalar multiplications emerge on several protocols and possibly, when one wants to perform batch elliptic curve operations more efficiently.

5.4 Side-channel attacks on Montgomery ladders

Due to its regular pattern execution and the absence of dummy operations, Montgomery ladders are inherently protected against several side-channel attacks in both software and hardware implementations. In the case of software implementations, the interested reader is referred to [6,9] and the discussion given in this work in Sect. 4.2 for protective algorithmic countermeasures.

In the case of hardware implementations, adversaries have much more room for launching attacks that in some cases might be devastating [16,17,32]. In [17], the authors reported a careful analysis of the known side-channel attacks and countermeasures for elliptic curve implementations on hardware platforms. As it happens, it is often true that a given countermeasure against one specific attack might make the design vulnerable to other kinds of attacks. In [16], Karaklajić et al. discuss a number of fault attacks against Montgomery ladder hardware implementations. The authors also state that the computation of the Montgomery ladder *without* y-coordinate retrieval can actually protect a hardware implementation against several attacks, such as safe error and sign change attacks.

5.5 Fast and compact Montgomery ladder hardware implementations

Montgomery ladders over binary elliptic curves is a favorite choice for hardware designers. The main reasons of this lie in the advantage of providing basic protection against simple power analysis (SPA) [17], the high efficiency of the Montgomery ladder step, namely, only five multiplications and four squarings are required, and a total of seven registers are needed if the Lopez–Dahab algorithm [43] is implemented in a conventional way.

Finally, for many protocols, such as the elliptic curve Diffie–Hellman protocol, the y-coordinate recovering is not necessary, and the absence of the y-coordinate computation can provide further protection against certain fault attacks.

Among the fastest and most compact hardware accelerators that use the Montgomery ladder on binary elliptic curves, we can mention [42,55] and [38,40], respectively.

5.5.1 Fast designs

In [55], the authors presented a hardware accelerator that computes scalar multiplications on binary elliptic curves defined over $\mathbb{F}_{2^{233}}$ that hardly offers a security level of 112 bits. Their design can compute a scalar multiplication in 12.5 μS. In [42], the authors targeted NIST standardized binary curves defined over the fields $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$, and $\mathbb{F}_{2^{283}}$. For the latter field, the pipelined architecture presented in [42] achieved a latency of less than 11 μS.

In [54], the authors present a design that manipulates the clock signal so that their design can operate at its maximum frequency at different steps of the Montgomery ladder point multiplication. This design achieves a performance of just 6.84 μS for a scalar multiplication computed over a binary elliptic curve defined over the field $\mathbb{F}_{2^{233}}$.

In [4], Ay et al. report a hardware accelerator that computes a constant-time variable-base point multiplication over a GLS elliptic curve that lies in the field $\mathbb{F}_{2^{2\cdot 127}}$. The prime order subgroup of this curve offers a security level of about 127 bits. The authors report a timing delay of just 3.98 μS for computing one scalar multiplication on an Xilinx Kintex-7 FPGA device running at 253 MHz.

5.5.2 Compact designs

The authors of [40] reported a hardware architecture that can compute a scalar multiplication on Edwards curves defined over the fields $\mathbb{F}_{2^{163}}$, $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$. The area and time performance reported by the authors rank among the most efficient hardware accelerators for constrained environments. The authors in [38] adopted the aforementioned common-Z trick to reduce the number of registers required to carry out one Montgomery ladder step.

5.6 Fast Montgomery ladder software implementations

Table 8 presents the performance timings of some of the fastest Montgomery ladders recently reported in the open literature. For the sake of a broader perspective, several Montgomery ladders operating over prime fields have been included in this comparison. Notice also that this comparison cannot be quite fair, since the software libraries included in Table 8 were implemented in different Intel processors, with different capabilities.

Taking into account this consideration, we decided to measure timings of our code in various machines that represent the architectures in which the recent works on Montgomery ladders were implemented: Intel Core i7-2600K 3.40 GHz (Sandy Bridge), Intel Core i7-4700MQ 2.40 GHz (Haswell) and Intel Core i5-5200U 2.20 GHz (Broadwell).

The Intel Skylake software implementation described in this work achieves the fastest Montgomery ladder performance for a constant-time scalar multiplication at the 128-bit security level. Furthermore, from the timings reported in Table 7, it can be seen that the execution of the ephemeral Diffie–Hellman protocol can be computed by our software in just 95,702 clock cycles.[14]

At last, the second part of Table 8 shows that the timings for the Sandy Bridge are significantly above the ones in the other architectures. For instance, the cost for computing the Montgomery double-and-add is 3.16 times higher when compared to the Skylake architecture. The main point here is that, compared to the other architectures, the cost of the carry-less multiplication instruction in the Sandy Bridge machines is more than twice [31]. As a result, to achieve optimal timings in this architecture, the field arithmetic functions have to be properly designed with alternative instructions.

Table 8 Timings (in clock cycles) for 128-bit-level Montgomery ladder variable-point multiplication software implementations in the Intel Sandy Bridge (Sbr), Ivy Bridge (Ivy), Haswell (Has), Broadwell (Brd) and Skylake (Skl) architectures

Method	Cycles	Arch.
State-of-the-art implementations		
Curve2251-Montgomery ladder (**binary**) [62]	225,000	Sbr
NIST B233-Montgomery-Parallel (**binary**) [50]	149,117	Sbr
Montgomery-DJB-chain (**prime**) [14]	148,000	Ivy
Random-Montgomery-LD ladder (**binary**) [9]	135,000	Has
Genus-2-Kummer (**prime**) [10]	122,000	Ivy
Koblitz-Montgomery-LD double-and-add (**binary**) [51]	122,000	Has
Koblitz-Montgomery-LD ladder (**binary**) [9]	118,000	Has
GLS-Montgomery-LD-2-GLV halve-and-add (**binary**) [51]	80,800	Has
Genus-2-Kummer Montgomery ladder (**prime**) [6]	72,200	Has
GLS-Montgomery-LD double-and-add (**binary**) [51]	70,800	Has

Method	**Cycles**			
	Sbr	Has	Brd	Skl
This work				
Montgomery left-to-right double-and-add (**Algorithm** 1)	164,532	66,076	52,020	52,026
Montgomery 2-GLV halve-and-add (**Algorithm** 6)	157,504	77,668	66;760	68,370
Montgomery 2-GLV halve-and-add fixed point (**Algorithm** 6)	130,872	56,080	44,492	46.082
Ephemeral Diffie–Hellman (**Algorithm** 6 & **Alg.** 1)†	290,388	119,572	94,128	95,702

† The costs for retrieving the y-coordinate in the architectures Sbr, Has and Brd are 2508, 1292 and 1192 cc, respectively

6 Conclusion

Since the publication in 1987 of the Montgomery ladder procedure, there has been an ever increasing number of research works devoted to analyze and present algorithmic improvements to Peter Montgomery's original idea.

In this survey paper, we have striven to present a panoramic review of the main algorithmic ingredients associated with Montgomery ladders and their applications to the fast, side-channel secure and constant-time implementation of the binary elliptic curve scalar multiplication.

[14] As the Diffie–Hellman protocol does not need the y-coordinate of any point, the y-coordinate retrieval function was not considered in this estimation.

Appendix: Further formula derivations using division polynomials

A.1 Point doubling formula

The point doubling formula $x_{2k} = x_k^2 + \frac{b}{x_k^2}$ can be obtained from division polynomials.

$$\begin{aligned} p_{2k+1} &= p_{k+2}p_k^3 + p_{k-1}p_{k+1}^3 \\ &= \frac{p_{k+2}p_k}{p_{k+1}^2}p_k^2p_{k+1}^2 + \frac{p_{k-1}p_{k+1}}{p_k^2}p_k^2p_{k+1}^2 \\ &= (x + x_{k+1} + x + x_k)p_k^2p_{k+1}^2 \\ &= (x_k + x_{k+1})p_k^2p_{k+1}^2. \end{aligned}$$

Similarly, $p_{2(k-1)+1} = (x_{k-1} + x_k)p_{k-1}^2p_k^2$. Then,

$$\begin{aligned} x_{2k} &= x + \frac{p_{2k-1}p_{2k+1}}{p_{2k}^2} \\ &= x + \frac{(x_{k-1} + x_k)p_{k-1}^2p_k^2(x_k + x_{k+1})p_k^2p_{k+1}^2}{(x_kp_k^4)^2} \\ &= x + \frac{(x_k + x_{k-1})(x_{k+1} + x_k)(x + x_k)^2}{x_k^2} \\ &= x + \left(x_k + x_{k+1} + \frac{xx_k}{(x + x_k)^2}\right)(x_{k+1} + x_k)\frac{(x + x_k)^2}{x_k^2} \\ &= x_k^2 + x^2 + x_{k+1}^2 + \frac{x^2x_{k+1}^2}{x_k^2} + \frac{xx_{k+1}}{x_k} \\ &= x_k^2 + x^2 + 1\frac{x^2[(\lambda_k + \lambda)^2 + (\lambda_k + \lambda)](x_k + x)^2}{(x + x_k)^4} \\ &= x_k^2 + x^2 + \frac{x^2(x_{2k} + x_2)}{(x + x_k)^2}; /x_2 = (x^4 + b)/x^2 \end{aligned}$$

Now, from the relation $x_{2k}(x + x_k)^2 = (x + x_k)^4 + x^2(x_{2k} + x_2)$, one gets $x_{2k} = x_k^2 + \frac{b}{x_k^2}$.

A.2 x-coordinate addition of the points $(k+1)P$ and $(k-1)P$

Proposition 4 *The x-coordinates x_k, x_{k-1}, x and x_{k+1} satisfy the relation:*

$$x_{k+1} + x_{k-1} = \frac{xx_k}{(x + x_k)^2}.$$

Proof This relation follows directly from both relations: $p_{2k} = (p_{k+2}p_kp_{k-1}^2 + p_{k-2}p_kp_{k+1}^2)/x$ and $p_{2k} = x_kp_k^4$. □

References

1. Agnew, G.B., Mullin, R.C., Vanstone, S.A.: An implementation of elliptic curve cryptosystems over $\mathbb{F}_{2^{155}}$. IEEE J. Sel. Areas Commun. **11**(5), 804–813 (1993)
2. ANSI X9.62:2005. Public key cryptography for the financial services industry: The elliptic curve digital signature algorithm (ECDSA). American National Standards Institute (2005)
3. Aranha, D.F., López, J., Hankerson, D.: Efficient software implementation of binary field arithmetic using vector instruction sets. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010, LNCS, vol. 6212, pp. 144–161. Springer (2010)
4. Ay, A.U., Öztürk, E., Rodríguez-Henríquez, F., Savas E.: Design and implementation of a constant-time FPGA accelerator for fast elliptic curve cryptography. In: Athanas, P. M., Cumplido, R., Feregrino C., Sass R. (eds.) International Conference on ReConFigurable Computing and FPGAs, ReConFig 2016 pp. 1–8. IEEE (2016)
5. Bernstein, D.J.: Differential addition chains. http://cr.yp.to/ecdh/diffchain-20060219.pdf. Accessed Mar 2017
6. Bernstein, D.J., Chuengsatiansup, C., Lange, T., Schwabe, P.: Kummer strikes back: new dh speed records. In: Sarkar, P., Iwata, T. (eds.) ASIACRYPT 2014, LNCS, vol. 8873, pp. 317–337. Springer (2014)
7. Bernstein, D. J., Engels, S., Lange, T., Niederhagen, R., Paar, C., Schwabe, P., Zimmermann, R.: Faster discrete logarithms on FPGAs. Cryptology ePrint Archive, Report 2016/382, 2016. http://eprint.iacr.org/2016/382
8. Bernstein, D.J., Lange, T., Farashahi, R.: Binary Edwards curves. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008, LNCS, vol. 5154, pp. 244–265. Springer (2008)
9. Bluhm, M., Gueron, S.: Fast software implementation of binary elliptic curve cryptography. J. Cryptogr. Eng. **5**(3), 215–226 (2015)
10. Bos, J.W., Costello, C., Hisil, H., Lauter, K.E.: Fast Cryptography in genus 2. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013, LNCS, vol. 7881, pp. 194–210. Springer (2013)
11. Brown, D.R.L.: Multi-dimensional montgomery ladders for elliptic curves. IACR Cryptology ePrint Archive, Report 2006/220. http://eprint.iacr.org/2006/220 (2006)
12. Chen, B., Hu, C., Zhao, C-A.: A note on scalar multiplication using division polynomials. IACR Cryptology ePrint Archive, Report 2015/284. http://eprint.iacr.org/2015/284 (2015)
13. Chung, P.N., Costello, C., Smith, B.: Fast, uniform, and compact scalar multiplication for elliptic curves and genus 2 Jacobians with applications to signature schemes. (2015). arXiv:1510.03174
14. Costello, C., Hisil, H., Smith, B.: Faster compact Diffie–Hellman: endomorphisms on the x-line. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014, LNCS, vol. 8441, pp. 183–200. Springer (2014)
15. Enge, A., Gaudry, P.: A general framework for subexponential discrete logarithm algorithms. Acta Arith. **102**, 83103 (2002)
16. Fan, J., Guo, X., De Mulder, E., Schaumont, P., Preneel, B., Verbauwhede, I.: State-of-the-art of secure ECC Implementations: A survey on known side-channel attacks and countermeasures. In: IEEE International Symposium on Hardware-Oriented Security and Trust (HOST 2010), pp. 76–87. IEEE (2010)
17. Fan, J., Verbauwhede, I.: An updated survey on secure ECC implementations: Attacks, countermeasures and cost. In: Naccache, D. (ed.) Cryptography and Security: LNCS, vol. 6805, pp. 265–282. Springer (2012)
18. Faugère, J. Perret, L., Petit, C., Renault G.: Improving the complexity of index calculus algorithms in elliptic curves over binary fields. In: EUROCRYPT 2012, LNCS, vol. 7237, p. 2744. Springer (2012)

19. Fong, K., Hankerson, D., López, J., Menezes, A.: Field inversion and point halving revisited. IEEE Trans. Comput. **53**(8), 1047–1059 (2004)
20. Galbraith, S.D., Gaudry, P.: Recent progress on the elliptic curve discrete logarithm problem. Des. Codes Cryptogr. **78**(1), 51–72 (2016)
21. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. J. Cryptol. **24**, 446–469 (2011)
22. Galbraith, S. D., Gebregiyorgis, S. W.: Summation polynomial algorithms for elliptic curves in characteristic two. In: INDOCRYPT 2014, LNCS, vol. 8885, pages 409427. Springer (2014)
23. Galbraith, S. D., Smart, N. P.: A Cryptographic application of weil descent. In Cryptography and Coding, LNCS, vol. 1746, p. 191200. Springer (1999)
24. Gallant, R.P., Lambert, R.J., Vanstone, S.A.: faster point multiplication on elliptic curves with efficient endomorphisms. In: Kilian, J. (ed.) CRYPTO 2001, LNCS, vol. 2139, pp. 190–200. Springer (2001)
25. Gaudry, P., Hess, F., Smart, N.P.: Constructive and destructive facets of Weil descent on elliptic curves. J. Cryptol. **15**, 1946 (2002)
26. Hankerson, D., Karabina, K., Menezes, A.: Analyzing the Galbraith-Lin-Scott point multiplication method for elliptic curves over binary fields. IEEE Trans. Comput. **58**(10), 1411–1420 (2009)
27. Hess, F.: Generalising the GHS attack on the elliptic curve discrete logarithm problem. LMS J. Comput. Math. **7**, 167192 (2004)
28. Huang, Y.-J., Petit, C., Shinohara, N., Takagi, T.: On generalized first fall degree assumptions. IACR Cryptol. ePrint Arch. **2015**, 358 (2015)
29. Hutchinson, A., Karabina, K.: Constructing multidimensional differential addition chains and their applications. IACR Cryptol. ePrint Arch. **2017**, 311 (2017)
30. Igoe, K., McGrew, D.A., Salter, M.: Fundamental elliptic curve cryptography algorithms. RFC 6090. https://rfc-editor.org/rfc/rfc6090.txt. (2015)
31. Intel corporation: Intel intrinsics guide. https://software.intel.com/sites/landingpage/IntrinsicsGuide/. Accessed Mar 2017
32. Karaklajić, D., Fan, J., Schmidt, J-M., Verbauwhede, I.: Low-cost fault detection method for ECC using Montgomery powering ladder. Design, automation and test in Europe, DATE 2011 pp. 1016–1021. IEEE (2011)
33. Kim, K.H., lee, C.O., Nègre, C.: Binary Edwards curves revisited. In: Meier, W., Mukhopadhyay, D. (eds.) INDOCRYPT 2014, LNCS, vol. 8885, pp. 393–408. Springer (2014)
34. Knudsen, E.: Elliptic scalar multiplication using point halving. In: Lam, K.Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 99, LNCS, vol. 1716, pp. 135–149. Springer (1999)
35. Koblitz, N.: Constructing elliptic curve cryptosystems in characteristic 2. In: Menezes, A., Vanstone, S. (eds.) CRYPTO 90, LNCS, pp. 156–167. Springer (1991)
36. Kocher, P. C.: Timing attacks on implementations of Diffie–Hellman, RSA, DSS, and other systems. In: CRYPTO 96, LNCS, vol. 1109, pp. 104–113. Springer (1996)
37. Koher, D.: Endomorphism rings of elliptic curves over finite fields. PhD thesis, University of California Berkeley, (1996). http://echidna.maths.usyd.edu.au/kohel/pub/thesis.pdf. Accessed Apr 2017
38. Koziel, B., Azarderakhsh, R., Mozaffari Kermani, M.: Low-resource and fast binary edwards curves cryptography. In: Biryukov, A., Goyal, V. (eds.) INDOCRYPT 2015, LNCS, vol. 9462, pp. 347–369. Springer (2015)
39. Lang, S.: Elliptic Curves Diophantine Analysis. Springer, New York, USA (1978)
40. Lee, Y.K., Sakiyama, K., Batina, L., Verbauwhede, I.: Elliptic-curve-based security processor for RFID. IEEE Trans. Comput. **57**(11), 1514–1527 (2008)
41. Lenstra Jr., H.W.: Factoring integers with elliptic curves. Ann. Math. **126**(3), 649–673 (1987)
42. Li, L., Li, S.: High-performance pipelined architecture of elliptic curve scalar multiplication over GF(2^m). IEEE Trans. VLSI Syst. **24**(4), 1223–1232 (2016)
43. López, J., Dahab, R.: Fast multiplication on elliptic curves over GF(2^m) without precomputation. In: Koç, Ç.K., Paar, C. (eds.) (eds.) CHES 99, LNCS, vol. 1717, pp. 316–327. Springer (1999)
44. Maurer, M., Menezes, A., Teske, E.: Analysis of the GHS weil descent attack on the ECDLP over characteristic two finite fields of composite degree. In INDOCRYPT 2001, LNCS, vol. 2247, p. 195213. Springer (2001)
45. Meloni, N.: New point addition formulae for ECC applications. In: Carlet, C., Sunar, B. (eds.) WAIFI 2007, LNCS, vol. 4547, pp. 189–201. Springer (2007)
46. Menezes, A., Vanstone, S.A.: Elliptic curve cryptosystems and their implementations. J. Cryptol. **6**(4), 209–224 (1993)
47. Menezes, A., Qu, M: Analysis of the Weil descent attack of Gaudry, Hess and Smart. In CT-RSA 2001, vol. 2020 of LNCS, p. 308318. Springer (2001)
48. Miller, V.S.: Use of Elliptic curves in cryptography, advances in cryptology. In: Williams, H.C. (ed.) CRYPTO 85, LNCS, vol. 218, pp. 417–426. Springer (1986)
49. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Math.Comput. **48**, 243–264 (1987)
50. Nègre, C., Robert, J.-M.: New parallel approaches for scalar multiplication in elliptic curve over fields of small characteristic. IEEE Trans. Comput. **64**(10), 2875–2890 (2015)
51. Oliveira, T., Aranha, D.F., López-Hernández, J., Rodríguez-Henríquez, F.: Fast point multiplication algorithms for binary elliptic curves with and without precomputation. In: Joux, A., Youssef, A. M. (eds.) SAC 2014, LNCS, vol. 8781, pp. 324–344. Springer (2014)
52. Oliveira, T., Aranha, D.F., López-Hernández, J., Rodríguez-Henríquez, F.: Two is the fastest prime: Lambda coordinates for binary elliptic curves. J. Cryptogr. Eng. **4**(1), 3–17 (2014)
53. Oliveira, T., Aranha, D.F., López-Hernández, J., Rodríguez-Henríquez, F.: Improving the performance of the GLS254. http://tinyurl.com/CHES16-Rump
54. Rashidi, B., Sayedi, S.M., Farashahi, R.R.: High-speed hardware architecture of scalar multiplication for binary elliptic curve cryptosystems. Microelectron. J. **52**, 49–65 (2016)
55. Rebeiro, C., Sinha Roy, S., Mukhopadhyay, D.: Pushing the limits of high-speed GF(2^m) Elliptic curve scalar multiplication on FPGAs. In: Prouff, E., Schaumont, P. (eds.) CHES 2012, LNCS, vol. 7428, pp. 494–511. Springer (2012)
56. Renes, J., Schwabe, P., Smith, B., Batina, L.: μ Kummer: Efficient hyperelliptic signatures and key exchange on microcontrollers. In: Gierlichs, B., Poschmann, A. Y. (eds.) CHES 2016, LNCS, vol. 9813, pp. 301–320. Springer (2016)
57. Schroeppel, R.: Elliptic curve point halving wins big. In: 2nd Midwest Arithmetical Geometry in Cryptography Workshop (2000)
58. Schroeppel, R.: Automatically solving equations in finite fields. US patent 2002/0055962 A1 (2002)
59. Semaev, I.: Summation polynomials and the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Report 2004/031, 2004. http://eprint.iacr.org/2004/031
60. Semaev, I: New algorithm for the discrete logarithm problem on elliptic curves. Cryptology ePrint Archive, Report 2015/310, 2015. http://eprint.iacr.org/2015/310
61. Subramanya Rao, S.R.: Three dimensional montgomery ladder, differential point tripling on montgomery curves and point quintupling on Weierstrass' and Edwards curves. In: Pointcheval, D.,

Nitaj, A., Rachidi T. (eds.) AFRICACRYPT 2016, LNCS, vol. 9645, pp. 84–106. Springer (2016)
62. Taverne, J., Faz-Hernández, A., Aranha, D.F., Rodríguez-Henríquez, F., Hankerson, D., López, J.: Speeding scalar multiplication over binary elliptic curves using the new carry-less multiplication instruction. J. Cryptogr. Eng. **1**(3), 187–199 (2011)
63. Wenger, E., Wolfger, P.: Harder, better, faster, stronger: Elliptic curve discrete logarithm computations on FPGAs. J. Cryptogr. Eng. **6**(4), 287297 (2016)