Efficient Software Implementation of Laddering Algorithms Over Binary Elliptic Curves

Diego F. Aranha[1(✉)], Reza Azarderakhsh[2], and Koray Karabina[2]

[1] University of Campinas, Campinas, Brazil
dfaranha@ic.unicamp.br
[2] Florida Atlantic University, Boca Raton, USA
{razarderakhsh,kkarabina}@fau.edu

Abstract. Designing efficient and secure implementations of Elliptic Curve Cryptography (ECC) has attracted enormous interest from both theoreticians and practitioners. The main contenders in terms of performance are curves defined over binary extension fields or large prime characteristic fields. In addition to the efficiency requirements, security advantages such as implementation simplicity and resistance to side-channel attacks are receiving increasing attention in research and commercial applications. In this paper, we keep pushing in this direction and study efficient implementation of regular scalar multiplication algorithms for binary curves equipped with efficient endomorphisms. Our focus is on implementing the Galbraith-Lin-Scott (GLS) family of binary curves by exploring the space of different models and laddering algorithms, for their high performance, reasonable implementation simplicity, lower memory consumption and side-channel resistance. Our results demonstrate that laddering implementations can be competitive with window-based methods by obtaining a new speed record for laddering implementations of elliptic curves on high-end Intel processors.

1 Introduction

Secure and efficient implementation of Elliptic Curve Cryptography (ECC) has received a lot of interest by researchers and implementers alike. The security of ECC cryptosystems relies on the hardness of the Elliptic Curve Discrete Logarithm Problem (ECDLP) conjectured as fully exponential, which consists in recovering the scalar $k \in \mathbb{Z}$ from the given points P and $Q = kP$ in some elliptic curve E defined over a finite field $\mathbb{F}_q$.

Scalar multiplication ($k,\ P \rightarrow kP$) is the main operation required when evaluating ECC protocols and corresponds to adding point P to itself $k-1$ times. The performance and security of a curve-based cryptosystem strictly relates to the choice of curve parameters, scalar multiplication algorithm, finite field arithmetic, and implementation quality. Algorithms for scalar multiplication can be broadly classified in *window-based methods*, composed of a precomputation step for computing small multiples of the input point, and a main loop exploiting this precomputation through table lookups; and simpler and more compact *laddering*

S.S. Ali et al. (Eds.): SPACE 2017, LNCS 10662, pp. 74–92, 2017.
https://doi.org/10.1007/978-3-319-71501-8_5

methods [4,23] that execute the same operation across all iterations. These algorithms can be further classified in fixed-base when the input is a known point G (generator), variable-base when the input is unknown point P and double-point when two points P and Q are simultaneously multiplied as $\ell P + mQ$ by scalars ℓ and m. These scenarios typically occur in key generation, key exchange and signature verification, respectively.

In practice, the main contenders in terms of performance are curves defined over large prime characteristic fields (*prime curves*), or over binary extension fields (*binary curves*). Performance is not the only metric, and security advantages such as implementation simplicity and resistance to side-channel attacks are receiving more attention in research and industry. In the prime case, for instance, Edwards curves [3] and Four$\mathbb{Q}$ [7] have provided most of the recent performance and/or security improvements by adopting more conservative or aggressive choices of parameters, respectively. In the binary case, recent advances firmly rely on exploiting efficient endomorphisms and optimized parameters in Koblitz [18,22] and Galbraith-Lin-Scott (GLS) [12,20] curves.

Thanks to many improvements introduced in modern processors as powerful vector instructions, binary curves arguably now enjoy better native support for their underlying field arithmetic in some micro-architectures. Combined with algorithmic developments such as the lambda coordinate system [20,21] and efficient endomorphisms, binary curves currently hold the speed record for the most efficient scalar multiplication in software targeting Intel desktop processors [19]. Despite advances in solving the ECDLP for these curves [10], binary curves are still considered secure for cryptographic applications [23] and were standardized by IEEE [13] and NIST [17].

In this paper, we study the efficient implementation of laddering algorithms for variable-base scalar multiplication under different models of elliptic curves defined over binary extension fields. Laddering algorithms offer some built-in side-channel protection because of their regularity, and implementation friendliness due to the simplicity of not requiring all coordinates of a point to be computed. The Montgomery ladder over the x-coordinate is the most popular algorithm pertaining to this class [16].

We target binary GLS curves equipped with efficient endomorphisms, allowing multi-dimensional laddering algorithms such as the DJB chain proposed by Bernstein in [2] to be used. The DJB algorithm is *uniform*, in the sense that all iterations in the main loop require the same number and type of field operations [6], and can be implemented in an *isochronous* way (constant time) because the number of loop iterations can be made constant. Recently, Azarderakhsh and Karabina proposed the AK laddering algorithm tailored for the computation of point multiplication on elliptic curves with efficiently computable endomorphisms. The AK algorithm can be faster than DJB but has a variable number of loop iterations (with small standard deviation). More recently, AK and DJB laddering algorithms have been employed by Costello et al. for the implementation of point multiplication and x-coodinate only key exchange on elliptic curves defined over prime fields [6]. The main contributions of this work are:

- Concrete strategies for efficiently implementing the Montgomery, DJB and AK variable-base laddering algorithms, minimizing the number of conditional operations for side-channel protection. As long as the efficiency of finite field arithmetic continues to improve due to progress in instruction sets, these overheads become significant and must be handled properly. Also, an efficient uniform algorithm for AK recoding is proposed and evaluated.
- Techniques for converting between binary GLS curves in the Weierstraß model to alternate models such as Huff and Edwards while keeping coefficients compact and efficient to operate with. Faster formulas for evaluating differential addition and doubling operations required by laddering algorithms are presented for these models, applying lazy reduction and other recent techniques from the literature [9].
- A set of GLS curve parameters at the 128-bit security level which maximize the efficiency of the proposed techniques, followed by a state-of-the-art implementation which demonstrate that laddering algorithms can be competitive with window-based methods while setting new speed records for laddering implementations.

The work is organized as follows. In Sect. 2, Weierstraß, Huff and Edwards elliptic curve models are introduced, together with efficient formulae and algorithms for converting from binary GLS curves in the Weierstraß model. In Sect. 3, the Montgomery, DJB and AK laddering algorithms for scalar multiplication are discussed together with improvements. Section 4 presents experimental results and discussion, and Sect. 5 concludes.

2 Binary GLS Curves

Let $E_{W,a,b}$ be an ordinary binary elliptic curve in short Weierstraß form defined by the equation

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

where $a, b \in \mathbb{F}_{2^m}$. The set of affine points $P = (x, y)$ with $x, y \in \mathbb{F}_{\mathbb{F}_{2^m}}$ that satisfy the above equation, together with the point at infinity $\mathcal{O}$, forms an additive abelian group with respect to the elliptic point addition operation. This group is denoted as $E_{W,a,b}(\mathbb{F}_{2^m})$ and its order can be written as $\#E_{W,a,b}(\mathbb{F}_{2^m}) = 2^m - t + 1$, where t is the *trace of Frobenius* and satisfies $|t| \leq 2^m$.

In order to define a Galbraith-Linn-Scott (GLS) curve [11,12], choose a quadratic extension $\mathbb{F}_{2^{2m}}$ of $\mathbb{F}_{2^m}$ as $\mathbb{F}_{2^{2m}} = \mathbb{F}_{2^m}[s]/(s^2 + s + 1)$ and pick a field element $a' \in \mathbb{F}_{2^{2m}}$ such that $Tr(a') = 1$, where Tr is the trace function from $\mathbb{F}_{2^{2m}}$ to $\mathbb{F}_{2^m}$ defined as $Tr : c \mapsto \sum_{i=0}^{2m-1} c^{2^i}$. It follows that $\#E(\mathbb{F}_{2^{2m}}) = (2^m+1)^2-t^2$. Let us define

$$E'/\mathbb{F}_{2^{2m}} : y^2 + xy = x^3 + a'x^2 + b, \tag{2}$$

with $\#E'_{W,a',b}(\mathbb{F}_{2^{2m}}) = (2^m - 1)^2 + t^2$. E' is the quadratic twist of $E_{W,a,b}$ which means that both curves are isomorphic over $\mathbb{F}_{2^{4m}}$ under the endomorphism $\phi : E \to E', (x, y) \mapsto (x, y + \sigma x)$, with $\sigma \in \mathbb{F}_{2^{4m}} \backslash \mathbb{F}_{2^{2m}}$ satisfying $\sigma^2 + \sigma = a + a'$.

Fix $a' = s$ and choose b such that $\#E'_{a',b}(\mathbb{F}_{2^{2m}}) = 2r$, where r is prime with $(2m-1)$ bits. Let $a_1 = b^{-1/8}$, Eq. (2) is isomorphic over $\mathbb{F}_{2^{2m}}$ to

$$E''/\mathbb{F}_{2^{2m}} : Y^2 + a_1XY = X^3 + a_1^2sX^2 + 1/a_1^2, \tag{3}$$

with isomorphism given by $\Phi : E' \to E'', (x, y) \mapsto (a_1^2x, a_1^3y)$ [24].

Let $\pi : E \to E$ be the Frobenius map and let ψ be the composite GLS endomorphism $\psi = \phi\pi\phi^{-1}$ given as $\psi(x, y) = (x^{2^m}, y^{2^m} + x^{2^m}s)$. Hankerson *et al.* showed in [12] that $\psi(P) = \lambda P$ for some $\lambda \in \mathbb{Z}$ satisfying $\lambda^2 + 1 \equiv 0 \pmod{r}$. For $k \in \mathbb{Z}$, the scalar multiplication kP can then be decomposed in $k_1P + k_2\psi(P)$ such that $k \equiv k_1 + k_2\lambda \pmod{r}$.

Parameters. A concrete GLS curve targeting approximately the 128-bit security level can be found by choosing $m = 127$ and binary field $\mathbb{F}_{2^m}$ defined as $\mathbb{F}_2[z]/(f(z) = z^{127} + z^{63} + 1)$. Two possible choices for curve coefficient b defining curves E_1 and E_2, respectively, can be found below:

1. $b_1 =$ `0x5404514441040154401540540515101` in polynomial representation (hexadecimal form) with short 64-bit square root $\sqrt{b} =$ `0xE2DA921E91E38DD1`. This parameter is widely used in the literature [18,20] to optimize a multiplication by b in the Montgomery ladder due to the short square root, and here is chosen for comparison with related work.
2. $b_2 = z^{85} + z^{21}$ in polynomial form with short root $b^{-1/8}$, introduced here to simplify curve coefficients when converted to other curve models.

Both concrete curves E_1 and E_2 have large 253-bit prime subgroup order r and thus satisfy common security requirements for the discrete logarithm setting.

The basic computation in each step of a laddering algorithm (LADD operation) is differential addition (DADD) and doubling (DOUB) evaluated over the base field where the curve is defined. Given points P_1 and P_2 on elliptic curve $E(\mathbb{F}_q)$ with known *difference* $P_0 = P_1 - P_2$, this operation computes point addition $P_1 + P_2$ and point doubling $2P_1$. In general, the formulas can be evaluated over a smaller set of coordinates. Let w be a rational function defined over elliptic curve $E(\mathbb{F}_q)$ given by the fraction of polynomials in the coordinate ring of E, with $w(P) = w(-P)$ [9]. For any points P_1, P_2 given by the values $w(P_1), w(P_2)$ and difference $w(P_1 - P_2)$, differential addition and doubling formulae again compute $w(P_1 + P_2)$ and $w(2P_1)$ in w-coordinates. A projective w-coordinate representation $w(P) = (W : Z)$ of a point P can also be defined to eliminate expensive inversions in curve arithmetic, and the corresponding affine representation can be simply recovered by computing $\frac{W}{Z}$.

Let $\boldsymbol{m}, \boldsymbol{s}, \boldsymbol{d}, \boldsymbol{r}, \boldsymbol{a}$ and $\boldsymbol{i}$ denote the costs of field multiplication, squaring, multiplication by short curve parameter, modular reduction by $f(z)$, addition and inversion in $\mathbb{F}_{2^{2m}}$, respectively. The *lazy reduction* technique evaluates an expression $(ab + cd)$ over a field $\mathbb{F}_q$ by accumulating the multiplication results before modular reduction, incurring a performance trade-off of $(\boldsymbol{a} - \boldsymbol{r})$. Because addition in a binary field is trivial, typically $\boldsymbol{r} > \boldsymbol{a}$ and the technique incurs a

small speedup. Notation $[\cdot]_f$ here denotes an *explicit* modular reduction operation of a double-precision result, implying that multiplication and squaring results are automatically reduced otherwise. In the next subsections, we improve state-of-the-art differential addition and doubling formulas for several binary curve models presented in [9] by using lazy reduction and compare their relative performance.

2.1 Formulae for Weierstraß Curves

In Weierstraß curves, the w-coordinate representation $w(P)$ of a point P can be simplified to the x-coordinate $x(P)$. The formulae below compute the projective w-coordinate differential addition and doubling (LADD) operation among points $P_1 = (X_1 : Z_1)$ and $P_2 = (X_2 : Z_2)$, producing the results $(X_3 : Z_3) = (X_1 : Z_1) + (X_2 : Z_2)$ and $(X_4 : Z_4) = 2(X_1 : Z_1)$, given the difference point in projective coordinates $P_0 = (X_0 : Z_0) = (X_1 : Z_1) - (X_2 : Z_2)$:

$$A = (X_1 + Z_1),\ B = (X_2 + Z_2),\ T = [X_1 \cdot X_2 + Z_1 \cdot Z_2]_f,$$
$$Z_3 = (T + A \cdot B)^2 \cdot X_0,\ X_3 = T^2 \cdot Z_0,$$
$$Z_4 = (a_1 \cdot X_1 \cdot Z_1)^2,\ X_4 = A^4.$$

This formula was improved from [24] by using lazy reduction of $(X_1X_2 + Z_1Z_2)$ and can be used over the curve isomorphic to the set of parameters E_2 defined by Eq. (3). Total cost in this case is $(6\boldsymbol{m} + 5\boldsymbol{s} + \boldsymbol{d} + 5\boldsymbol{a} - \boldsymbol{r})$ by trading an additional modular reduction $(\boldsymbol{r})$ by a double-precision addition (extra $\boldsymbol{a}$). If $Z_0 = 1$, the formulae below can be used instead by switching to López-Dahab coordinates over curve E_1 defined by Eq. (2) with difference point $w(P_0) = x_0$ in affine coordinates [15], costing $(5\boldsymbol{m} + 4\boldsymbol{s} + \boldsymbol{d} + 4\boldsymbol{a} - \boldsymbol{r})$:

$$A = X_1 \cdot Z_2,\ B = X_2 \cdot Z_1,\ T = (X_2)^2,\ U = (Z_2)^2$$
$$X_4 = (T + U\sqrt{b_1})^2,\ Z_4 = T \cdot U$$
$$Z_3 = (A + B)^2,\ X_3 = [x_0 \cdot Z_4 + A \cdot B]_f.$$

Multiplication by b_1 is efficient because b_1 is chosen such that its square root is a 64-bit polynomial.

2.2 Formulae for Edwards Curves

Let $d_1, d_2 \in \mathbb{F}_{2^{2m}}$ with $d_1 \neq 0$ and $d_2 \neq d_1^2 + d_1$, the binary Edwards curve is the non-singular curve

$$E_{E,d_1,d_2} : (x + y)(d_1 + d_2(x + y)) = xy(1 + x)(1 + y). \tag{4}$$

When $Tr(d_2) = 1$, the curve is complete and there are no exceptions to the addition law. The Edwards model is birationally equivalent to the Weierstraß model

$$v^2 + uv = u^3 + (d_1^2 + d_2)u^2 + d_1^4(d_1^4 + d_1^2 + d_2^2) \tag{5}$$

under the map $(x, y) \longmapsto (u, v)$ and its inverse defined by [5]

$$u = (d_1^3 + d_1^2 + d_1 d_2)(x + y)/(xy + d_1(x + y)), \quad (6)$$
$$v = (d_1^3 + d_1^2 + d_1 d_2)(d_1 + 1 + x/(xy + d_1(x + y)).$$

Since the curve used in this work has no rational points of order 4, it is not isomorphic to an Edwards curve with coefficients $d_1 = d_2$ and cannot enjoy the simpler arithmetic in that case.

We still obtain efficient arithmetic by selecting parameters of curve E_2 and choosing curve coefficients $d_1 = (s \cdot z^{84}) \in \mathbb{F}_{2^{2m}}$ and $d_2 = d_1^2 + d_1 + \sqrt{b}/d_1^2 \in \mathbb{F}_{2^{2m}}$. A subfield constant is thus obtained for evaluating the differential addition and doubling formula. Define function $w(x, y) = (x + y)/(d_1(x + y + 1))$ such that $w(P) = w(-P)$ for all affine points except when $x + y = 1$ [9]. Assuming that $w(P_1)$ and $w(P_2)$ are represented in projective coordinates $(W_1 : Z_1)$ and $(W_2 : Z_2)$, respectively, and given precomputed w-coordinate $z_0 = 1/w_0$ of difference point P_0, the formulae below compute differential addition and doubling with cost $(5\boldsymbol{m} + 4\boldsymbol{s} + \boldsymbol{d} + 4\boldsymbol{a} - \boldsymbol{r})$:

$$A = W_1 \cdot Z_1,\ B = W_1 \cdot Z_2,\ C = W_2 \cdot Z_1,$$
$$W_4 = A^2,\ Z_4 = (\sqrt[4]{d_1^4 + d_1^3 + d_1^2 d_2} W_1 + Z_1)^4,$$
$$W_3 = (B + C)^2,\ Z_3 = [B \cdot C + z_0 \cdot W_3]_f.$$

These formulae are faster than the almost complete formular given in [9] by $(\boldsymbol{r} - \boldsymbol{a})$, due to lazy reduction. Compared to the affine Weierstraß formula, it apparently has the same cost, but the multiplication by the curve coefficient is slower because $(d_1^4 + d_1^3 + d_1^2 d_2)$ is a polynomial in $\mathbb{F}_{2^m}$ with degree 91 in our set of parameters E_2.

2.3 Formulae for Huff Curves

Let $h_a \neq h_b \in \mathbb{F}_{2^{2m}}$ the coefficients of the generalized binary Huff curve given by the set of coordinates satisfying the equation

$$E_{H,h_a,h_b} : h_a x(y^2 + fy + 1) = h_b y(x^2 + fx + 1). \quad (7)$$

This equation is birationally equivalent to the Weierstrass curve

$$v(v + (h_a + h_b)fu) = u(u + h_a^2)(u + h_b^2). \quad (8)$$

under the map $(x, y) \longmapsto (u, v)$ and its inverse defined by [8]

$$(x, y) = \left(\frac{h_b(u + h_a^2)}{v}, \frac{h_a(u + h_b^2)}{v + (h_a + h_b)fu} \right), \quad (9)$$
$$(u, v) = \left(\frac{h_a h_b}{xy}, \frac{h_a h_b(h_a xy + h_b)}{x^2 y} \right).$$

In order to convert a GLS curve to the Huff curve model, we adapt the method in [8, Proposition 2] by defining $f = g(z) \cdot s$, where g is a polynomial of small degree such that $Tr(1/f) = Tr(s)$ and $Tr(f^8 b) = 0$. For simplicity, parameters h_a, h_b will be chosen as $h_b = 1$ and $h_a \in \mathbb{F}_{2^m}$ as the solution h_a^2 to equation $t^2 + \frac{1}{f^4\sqrt{(b)}}t = 1$ in a subfield. This guarantees that the constant $\gamma = f^2 \frac{(h_a+h_b)^2}{h_a h_b}$ and its inverse $1/\gamma$ will have small degree and reside in a subfield, allowing fast multiplication by these constants.

We adapt the almost complete formulae from [9] to general binary Huff curves. By choosing $w(x, y) = (xy) \cdot \gamma$, and given the w-coordinate w_0 of difference point P_0 precomputed as $z_0 = 1/w_0$, we propose the following formulae for differential addition and doubling costing ($5\boldsymbol{m} + 4\boldsymbol{s} + \boldsymbol{d} + 4\boldsymbol{a} - \boldsymbol{r}$):

$$A = W_1 \cdot Z_1,\ B = W_1 \cdot Z_2,\ C = W_2 \cdot Z_1,$$
$$W_3 = A^2,\ Z_3 = (W_1/\gamma + Z_1)^4,$$
$$W_3 = (B + C)^2,\ Z_3 = [B \cdot C + z_0 \cdot W_3]_f.$$

These formula are again faster than [9] by ($\boldsymbol{r} - \boldsymbol{a}$) due to lazy reduction. Compared to the Edwards model, this formula is faster in our parameters because the multiplication by curve coefficient with cost $\boldsymbol{d}$ involves a multiplication by a polynomial of degree 54. This is equivalent to the cost of the Weierstraß formulae. There is another advantage of this curve model: it is easy to observe that $w(x, y) = xy\gamma = \frac{h_a}{u}$ for our choice of parameters, thus converting from the x-coordinate Weierstraß representation requires only an inversion and multiplication by a subfield constant. When working over w-coordinates only, this allows the GLS endomorphism to be computed as a simple 2^m-power over $\mathbb{F}_{2^{2m}}$ because h_a lies in a subfield.

At last, there are formulas in the binary Hessian model with this exact same cost [9], but which result in larger curve coefficients after conversion from Weierstrasß for our choices of parameters, so they are not discussed in this work.

3 Laddering Algorithms

Scalar multiplication is the performance-critical operation for protocols based on elliptic curves and the algorithms follow two general ideas. In window-based methods, a table of points containing small multiples $P_i = d_i P$ is precomputed, the scalar is recoded to another representation and the scalar multiplication follows by adding and doubling multiples obtained from the table according to the recoded scalar digits d_i. This strategy usually consumes more memory due to the precomputed table, and side-channel countermeasures are needed to prevent leaks from the recoding process or differences in memory access during table lookups. Laddering methods uniformly iterate a ladder step consisting of point doubling and addition over a smaller set of variables, reducing memory consumption. From the point of view of efficiency and simplicity, almost complete formulae as in the previous section which do not compute all coordinates are

preferable and sufficient to prevent exceptional cases within the laddering. Side-channel countermeasures protect the selection of variables to be updated with conditional operations.

Below we summarize and propose optimizations for three different laddering algorithms: the original Montgomery ladder, Bernstein's double point multiplication algorithm based on the new binary chain and a recent double multiplication algorithm due to Azarderakhsh and Karabina. The algorithms heavily depend on three branchless operations depending on a bit condition: SELECT for selecting among two inputs, CSWAP for conditionally swapping variables, and CCOPY for conditionally copying the input to the output. These conditional operations are usually considered to be cheap, but their cost is becoming increasingly significant due to faster finite field and elliptic curve arithmetic, and more powerful instruction sets. Our proposed versions of the algorithms will then focus on simplifying conditional operations when merging two consecutive loop iterations.

3.1 Montgomery Ladder

A version of the Montgomery scalar multiplication based on the projective w-coordinate representation is given in Algorithm 1. The algorithm receives as input $w(P)$, the affine w-coodinate of P, and the integer scalar k. Two accumulator points $P_1 = (W_1 : Z_1)$ and $P_2 = (W_2 : Z_2)$ are initialized as $w(P)$ and $w(2P)$, respectively, which are doubled and added depending on the current bit of the key. Iteration j starts with accumulators $[w(lP), w((l+1)P)]$, where l is the integer represented by the j leftmost bits of k, and computes $[w(2lP), w((2l+1)P)]$ or $[w((2l+1)P), w((2l+2)P)$. By induction, the last iteration produces $[kP, (k+1)P]$, where the first point is the result and the second point may be useful for recovering the full coordinates of the result. Following previous work, this version merges two consecutive iterations and only performs real swaps when necessary (consecutive bits are different).

When operating over Weierstraß curves with $w(P) = x(P)$, the y-coordinate y_1 of kP can be recovered from $(X_1 : Z_1) = w(P)$, $(X_2 : Z_2) = w((k+1)P)$ and $P = (x, y)$ with the following formula from [15]:

$$y_3 = (x + X_1/Z_1)[(X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)(Z_1Z_2)](xZ_1Z_2)^{-1} + y.$$

Although not explictly mentioned in the literature, this formula can be used to fully recover $kP = (x_3, y_3)$ with cost of $(\boldsymbol{i}+10\boldsymbol{m}+1\boldsymbol{s}+6\boldsymbol{a})$, at a relatively small increase from the cost $(\boldsymbol{i} + \boldsymbol{m})$ of computing $x_3 = X_1/Z_1$:

$$A = Z_1 \cdot Z_2,\ B = (X_1 + x \cdot Z_1),\ C = x \cdot Z_2,\ D = C \cdot X_1,$$
$$E = B \cdot (X_2 + C),\ F = (x^2 + y) \cdot A + E,\ G = (x \cdot A)^{-1},\ H = F \cdot G,$$
$$x_3 = D \cdot G,\ y_3 = y + (x + x_3) \cdot H.$$

3.2 Two-Dimensional DJB Ladder

As described in Sect. 2, a scalar multiplication kP can be computed as k_1P+k_2Q, for $Q = \psi(P)$. Hence, two-dimensional laddering algorithms can be used to evaluate a single scalar multiplication exploiting endomorphisms. We briefly explain

Algorithm 1. Montgomery ladder using differential addition and doubling formulae (LADD). The auxiliary function CSWAP conditionally swaps the two first arguments depending on the value of the third parameter.

Input: $k = (k_{l-1}, \ldots, k_1, k_0) \in \mathbb{Z}$ such that $k > 0$ and $w(P)$, for $P \in E(\mathbb{F}_{2^{2m}})$
Output: $w(kP), w((k+1)P) \in E(\mathbb{F}_{2^{2m}})$
1: $(W_1 : Z_1) \leftarrow w(P)$
2: $(W_2 : Z_2) \leftarrow w(2P)$
3: $p \leftarrow 0$
4: **for** $j \leftarrow l-2$ **downto** 0 **do**
5: $\quad c \leftarrow k_j \oplus p$
6: $\quad p \leftarrow k_j$
7: $\quad (W_1, W_2) \leftarrow \text{CSWAP}(W_1, W_2, c)$
8: $\quad (Z_1, Z_2) \leftarrow \text{CSWAP}(Z_1, Z_2, c)$
9: $\quad ((W_1 : Z_1), (W_2 : Z_2)) \leftarrow \text{LADD}((W_1 : Z_1), (W_2 : Z_2), w(P))$
10: **end for**
11: $(W_1, W_2) \leftarrow \text{CSWAP}(W_1, W_2, p)$
12: $(Z_1, Z_2) \leftarrow \text{CSWAP}(Z_1, Z_2, p)$
13:**return** $(W_1 : Z_1) = w(kP), (W_2 : Z_2) = w((k+1)P)$

Bernstein's double point multiplication algorithm based on the new binary chain [2]. The chain for (k_1, k_2) is computed as follows. Let $(M, N) = (k_1, k_2)$ and $D = k_1 \bmod 2$. $C_D(0,0)$ is defined as the base case $(0,0), (1,0), (0,1), (1,-1)$. For $(M,N) \neq (0,0)$, $C_D(M,N)$ is defined recursively:

$$\begin{aligned} C_D(M,N) = & C_d(\lfloor M/2 \rfloor, \lfloor N/2 \rfloor), \\ & (M + (M+1 \bmod 2), N + (N+1 \bmod 2)), \\ & (M + (M \bmod 2), N + (N \bmod 2)), \\ & (M + (M+D \bmod 2), N + (N+D+1 \bmod 2)), \text{ and} \end{aligned}$$

$$d = \begin{cases} 0 & \text{if } (\lfloor M/2 \rfloor + M, \lfloor N/2 \rfloor + N) \bmod 2 = (0,1) \\ 1 & \text{if } (\lfloor M/2 \rfloor + M, \lfloor N/2 \rfloor + N) \bmod 2 = (1,0) \\ D & \text{if } (\lfloor M/2 \rfloor + M, \lfloor N/2 \rfloor + N) \bmod 2 = (0,0) \\ 1 - D & \text{if } (\lfloor M/2 \rfloor + M, \lfloor N/2 \rfloor + N) \bmod 2 = (1,1). \end{cases}$$

Building the new binary chain for the integers (k_1, k_2) requires a number of $\max(\lceil \log_2 k_1 \rceil, \lceil \log_2 k_2 \rceil)$ iterations, and at the each iteration three vectors are added to the sequence. Given two elliptic curve points $P, Q \in E(\mathbb{F}_q)$, the new binary chain for (k_1, k_2) allows us to compute $k_1 P + k_2 Q$ at a cost of two point additions and one point doubling per iteration. The algorithm generates a chain sequence specifying the input to the doubling and addition operations at each iteration and a sequence of differences which encodes the differences of the points that are the input points to the addition operations at each iteration [1].

Algorithm 2 presents our optimized version of the DJB laddering algorithm. The algorithm starts by computing the chain sequence, returning four bit sequences S_0, S_1, S_2, S_3 representing the recoded versions of the input scalars k_1 and k_2, a value f_a determining the first addition and the correct point f_i among three accumulators to be returned at the end of the algorithm. Accumulators $(W_0 : Z_0), (W_1 : Z_1), (W_2 : Z_2)$ are initialized in projective coordinates using the DADD and DOUB and later keep track of the multiples of P and Q inside the main loop. Accumulator $(W_2 : Z_2)$ starts with value $w(P+2Q)$ or $w(2P+Q)$ depending on the chosen difference point and accumulator $(W_1 : Z_1)$ always starts with $w(2(P+Q))$. Differences w_{P+Q}, w_{P-Q} can be computed in affine coordinates sharing an inversion, to avoid slower projective representation of differences.

At the beginning of each iteration of our implementation, the condition bit t evaluates what operand will be copied to temporary variable $(W_4 : Z_4)$. This is needed because the DOUB operation inside the laddering can receive any of the three accumulators copied to $(W_3 : Z_3)$. Then the correct differences are copied to w_0 and w_1, followed by two differential additions using the chosen differences and a point doubling. These conditions were tediously derived and minimized from the bit combinations in the sequences S_0, S_1, S_2, S_3 to correctly position the inputs and outputs of the curve arithmetic operations. An advantage of this algorithm is always returning $(k+1)P = (k_1+1)P+k_2\psi(P)$ among the two other unused results, in a position depending on the parities of k_1 and k_2. This allows to recover the full coordinates of P using the same formulas in the previous subsection, increasing the scenarios in which the laddering can be applied.

3.3 Two-Dimensional AK Ladder

Let k_1 and k_2 be again two positive integers. In order to compute k_1P+k_2Q, the AK laddering algorithm starts with the initial values $d = k_1$, $e = k_2$, $\vec{R} = (P, Q)$, $\vec{u} = (1, 0)$, $\vec{v} = (0, 1)$, and $\vec{\Delta} = (1, -1)$. Define also $R_u = \vec{u} \cdot \vec{R}$, $R_v = \vec{v} \cdot \vec{R}$, and $R_\Delta = \vec{\Delta} \cdot \vec{R}$. The initial values yield $R_u = P$, $R_v = Q$, $R_\Delta = R_u - R_v = P - Q$, and $dR_u + eR_v = k_1P + k_2Q$, and the values $d, e, \vec{u}, \vec{v}, \vec{\Delta}, R_u, R_v, R_\Delta$ are updated according to the rules in Table 1 so that $dR_u + eR_v = k_1P + k_2Q$ and $R_\Delta = R_u - R_v$ hold, $d, e > 0$, and $(d + e)$ decreases until $d = e$. When $d = e$, we have $k_1P + k_2Q = dR_u + eR_v = d(R_u + R_v)$ which can be computed using a single point multiplication algorithm with base $R_u + R_v$ and scalar d. Note that when $\gcd(k_1, k_2) = 1$, $(d + e)$ in the algorithm will decrease until $d = e = 1$ and we have $k_1P + k_2Q = R_u + R_v$ [1].

Algorithm 3 computes a recoded format for the scalars according to Table 1 in a branchless manner. The recoded sequence stores in each position a value among the four rules in the table. First conditions t and t' are computed in lines 3 and 4 as $d \stackrel{?}{\equiv} e \pmod 2$ and $d \stackrel{?}{\equiv} 0 \pmod 2$, respectively. Variable f is assigned to $|d - e|$ in line 6, values (d, e) are swapped before division by 2 (shifting to the right by 1) if the conditions for rules R'_1 or R'_2 apply, and d conditionally receives f to update the correct value, after which the swapping is restored in line 10. At the end of each iteration, the sequence is increased by one element storing the rule and the current length if incremented.

Algorithm 2. DJB laddering algorithm, employing the DADD and DOUB operations. The CHAIN computation returns recoded scalars and two additional values determining the first addition (f_a) and the correct point (f_i) to be returned at the end of the algorithm. Auxiliary functions SELECT conditionally selects among two arguments and CCOPY copies the input to the destination depending on the last parameter.

Input: Integers $k_1, k_2 > 0$ and $w(P), w(Q)$ for $P, Q \in E(\mathbb{F}_{2^{2m}})$
Output: $w(k_1P + k_2Q) \in E(\mathbb{F}_{2^{2m}})$
1: $S_0, S_1, S_2, S_3, f_a, f_i \leftarrow \text{CHAIN}(k_1, k_2)$
2: $(W_0 : Z_0) \leftarrow w(P + Q)$, $w_P \leftarrow w(P), w_Q \leftarrow w(Q)$
3: $w_{P+Q} \leftarrow w(P + Q), w_{P-Q} \leftarrow w(P - Q)$
4: $(w_P, w_Q) \leftarrow \text{CSWAP}(w_P, w_Q, f_a \stackrel{?}{=} 1)$
5: $(W_2 : Z_2) \leftarrow \text{DADD}((w_{P+Q} : 1), (w_P : 1), w_Q)$
6: $(W_1 : Z_1) \leftarrow \text{DOUB}((W_0 : Z_0))$
7: $(w_P, w_Q) \leftarrow \text{CSWAP}(w_P, w_q, f_a \stackrel{?}{=} 1)$
8: **for** $j \leftarrow \max(\lceil \log_2 k_1 \rceil, \lceil \log_2 k_2 \rceil)$ **downto** 0 **do**
9: $\quad t \leftarrow S_{1,j} \oplus (S_{3,j} \wedge S_{0,j})$
10: $\quad w_0 \leftarrow \text{SELECT}(w_P, w_Q, \neg S_{3,j})$
11: $\quad w_1 \leftarrow \text{SELECT}(w_{P+Q}, w_{P-Q}, S_{2,j})$
12: $\quad (W_4 : Z_4) \leftarrow \text{SELECT}((W_1 : Z_1), (W_0 : Z_0), t)$
13: $\quad (W_3 : Z_3) \leftarrow \text{SELECT}((W_2 : Z_2), (W_4 : Z_4), \neg S_{0,j})$
14: $\quad (W_2 : Z_2) \leftarrow \text{DADD}((W_2 : Z_2), (W_4 : Z_4), w_0)$
15: $\quad (W_0 : Z_0) \leftarrow \text{DADD}((W_1 : Z_1), (W_0 : Z_0), w_1)$
16: $\quad (W_1 : Z_1) \leftarrow \text{DOUB}((W_3 : Z_3))$
17: **end for**
18: $R \leftarrow (W_0 : Z_0)$
19: $R \leftarrow \text{CCOPY}((W_1 : Z_1), f_i \stackrel{?}{=} 1)$
20: $R \leftarrow \text{CCOPY}((W_2 : Z_2), f_i \stackrel{?}{=} 2)$
21: **return** R

The authors of the algorithm discuss in [1] that, if k_1 and k_2 are ℓ-bit integers, then $k_1P + k_2Q$ can on average be computed in about 1.4ℓ point additions and 1.4ℓ point doublings. Moreover addition and doubling operations can be performed using differential addition and differential doubling formulas as the differences of the group elements to be added are known by construction. Algorithm 4 presents our implementation of the AK laddering approach by merging consecutive iterations. Expressions for the conditions determining the conditional operations at the beginning of each iteration were tediously evaluated and minimized to reduce the number of required conditional operations for a correct execution. In contrast with the previous laddering algorithms, the LADD operation now performs the laddering step with difference point in projective coordinates because the differences are not fixed in the AK algorithm, as it can be observed at the end of the conditional swap operations that R_Δ can be among the updated variables.

Algorithm 3. AK recoding, returning the sequence S and its length l according to the rules in Table 1. Auxiliary function CSWAP conditionally swaps the two arguments and SELECT returns one of the arguments based on the condition, respectively.

Input: Integers $k_1, k_2 > 0$
Output: Recoded sequence S and its length l
1: $d \leftarrow k_1, e \leftarrow k_2, i \leftarrow 0$
2: **while** $d \neq e$ **do**
3: $t \leftarrow (d \stackrel{?}{\equiv} e) \pmod 2$
4: $t' \leftarrow d \stackrel{?}{\equiv} 0 \pmod 2$
5: $c \leftarrow (d - e)$
6: $f \leftarrow \text{SELECT}(c, -c, (c < 0))$
7: $(d, e) \leftarrow \text{CSWAP}(d, e, ((c < 0) \wedge t) \vee (\neg t \wedge \neg t'))$
8: $d \leftarrow \text{SELECT}(d, f, t)/2$
9: $d \leftarrow d/2$
10: $(d, e) \leftarrow \text{CSWAP}(d, e, ((c < 0) \wedge t) \vee (\neg t \wedge \neg t'))$
11: $S_i \leftarrow \text{SELECT}(\text{SELECT}(R_1, R'_1, (c < 0)), \text{SELECT}(R_2, R'_2, \neg t'), \neg t)$
12: $i \leftarrow i + 1$
13: **end while**
14: **return** S, i

Table 1. Update rules for double point multiplication in the AK algorithm.

Rule	Condition	d	e	$\vec{u}$	$\vec{v}$	$\vec{\Delta}$	R_u	R_v	R_Δ
R1	$d \equiv e \pmod 2$ and $d > e$	$(d-e)/2$	e	$2\vec{u}$	$\vec{u} + \vec{v}$	$\vec{\Delta}$	$2R_u$	$R_u + R_v$	R_Δ
R1′	$d \equiv e \pmod 2$ and $d < e$	d	$(e-d)/2$	$\vec{u} + \vec{v}$	$2\vec{v}$	$\vec{\Delta}$	$R_u + R_v$	$2R_v$	R_Δ
R2	$d \equiv 0 \pmod 2$	$d/2$	e	$2\vec{u}$	$\vec{v}$	$\vec{u} + \vec{\Delta}$	$2R_u$	R_v	$R_u + R_\Delta$
R2′	$e \equiv 0 \pmod 2$	d	$e/2$	$\vec{u}$	$2\vec{v}$	$\vec{\Delta} + (-\vec{v})$	R_u	$2R_v$	$R_\Delta + (-R_v)$

4 Experimental Results and Discussion

In order to detect what curve model was more promising in terms of performance, we started the implementation from the differential addition and doubling formulas, because the operation counts for the multiple curve models were very similar. We largely followed and reused publicly available code[1] for finite field arithmetic from [19,23] to enjoy optimizations for our high-end target platforms. This implementation employs compiler intrinsics to take advantage of 128-bit vector instructions for binary field arithmetic, especially the carryless multiplier available through instruction `PCLMULQDQ` to accelerate polynomial multiplication. The base binary field was defined as $\mathbb{F}_2^{127} \cong \mathbb{F}_2[z]/(z^{127} + z^{63} + 1)$ and its quadratic extension as $\mathbb{F}_{2^{254}} \cong \mathbb{F}_{2^{127}}[s]/(s^2 + s + 1)$. Curve arithmetic for the two

[1] SUPERCOP: https://bench.cr.yp.to.

Algorithm 4. AK laddering, employing a projective version of the LADD operation. The RECODE computation returns recoded scalars and the sequence length according to the recoding rules. The auxiliary function CSWAP conditionally swaps the two arguments depending on the value of the last condition.

Input: $k_1 > 0, k_2 > 0 \in \mathbb{Z}$ with $gcd(k_1, k_2) = 1$ and $w(P), w(Q)$ for $P, Q \in E(\mathbb{F}_{2^{2m}})$
Output: $w(k_1P + k_2Q) \in E(\mathbb{F}_{2^{2m}})$
1: $(S, l) \leftarrow \text{RECODE}(k_1, k_2)$
2: $R_u \leftarrow w(P), R_v \leftarrow w(Q), R_\Delta \leftarrow w(P - Q)$
3: $b'_0 \leftarrow 0, b'_1 \leftarrow 0, b'_2 \leftarrow 0$
4: **for** $j \leftarrow l - 1$ **downto** 0 **do**
5: $b_0 = (S_j \stackrel{?}{=} R_2), b_1 = (S_j \stackrel{?}{=} R'_1), b_2 = (S_j \stackrel{?}{=} R'_2)$
7: $c_0 \leftarrow b'_0 \oplus b_0$, $c_2 \leftarrow b'_2 \oplus b_2$, $c'_1 \leftarrow (b'_1 \vee b'_2)$
6: $c_1 \leftarrow c'_1 \oplus (b_1 \vee b_2)$
7: $(R_v, R_\Delta) \leftarrow \text{CSWAP}(R_v, R_\Delta, (c_0 \wedge \neg c'_1) \vee (c_2 \wedge c'_1))$
8: $(R_u, R_\Delta) \leftarrow \text{CSWAP}(R_u, R_\Delta, (c_0 \wedge c'_1) \vee (c_2 \wedge \neg c'_1))$
9: $(R_u, R_v) \leftarrow \text{CSWAP}(R_u, R_v, c_1)$
10: $(R_v, R_u) \leftarrow \text{LADD}_P(R_u, R_v, R_\Delta)$
11: $b'_0 \leftarrow b_0, b'_1 \leftarrow b_1, b'_2 \leftarrow b_2$
12: **end for**
13: $(R_u, R_v) \leftarrow \text{CSWAP}(R_u, R_v, b'_1 \vee b'_2)$
14: $(R_u, R_\Delta) \leftarrow \text{CSWAP}(R_u, R_\Delta, b'_2)$
15: $(R_v, R_\Delta) \leftarrow \text{CSWAP}(R_v, R_\Delta, b'_0)$
16: $(R_u, R_v) \leftarrow \text{LADD}_P(R_u, R_v, R_\Delta)$
17: **return** R_u

sets of parameters described in Sect. 2 was implemented on top of the finite field arithmetic and the GLV recoding code for scalar decomposition was extended to work with the new curve parameters. Conditional operations were implemented based on the 128-bit version of the `BLENDV` instruction.

Our target platforms are an Intel Ivy Bridge Core i5-3510M running at 3.1 GHz, an Intel Haswell Core i7-4770 running at 3.4 GHz and an Intel Skylake Core i7-6700K clocked at 4 GHz, all three with Turbo Boost and HyperThreading disabled to make benchmarking more stable. The code was compiled with *gcc* 7.1.1, *icc* 17.0.4 and *clang* 4.0.1 with the optimization flags `-O3 -march=native -fomit-frame-pointer` in the three machines. Performance figures under different compilers were somewhat close, with *clang* producing marginally better results for the vectorized field arithmetic. Hence we decided to report only the numbers for the last compiler.

4.1 Laddering Steps

Table 2 presents our performance numbers for evaluating the differential addition and doubling formulae in the target platforms. Field operations within the routines were carefully scheduled to avoid dependencies and exploit the high throughput of vector instructions in the target platforms. Performance clearly

increases in more recent microarchitecture families due to faster carryless multiplication instruction.

Differential addition and doubling was faster for all curve models in affine coordinates when compared to Weierstraß in projective coordinates due to the smaller number of multiplications, following our operation counts in Sect. 2. The implementations of the Huff model enjoyed a slightly better instruction scheduling for the field operations and were faster than Weierstraß in affine coordinates. The Edwards model was competitive with Weierstraß, but suffers from larger coefficients and an inefficient way of applying the GLS endomorphism spending expensive inversions to convert points from and to Weierstraß coordinates, which makes it less competitive in the big picture. This was much simpler for the Huff model, because our choice of parameters allows the GLS endomorphism to be applied with a single Frobenius application (Sect. 2.3), amounting to one field addition and some cheap word shuffling instructions. We observe that the Huff model was the best representation in terms of performance for the laddering step.

Table 2. Timings in clock cycles for evaluating the LADD operation in the Ivy Bridge, Haswell and Skylake platforms. Numbers were taken as the average of 10^4 executions and cycles were counted with help of the `rdtsc` instruction with TurboBoost and HyperThreading turned off.

Curve model	Cycles on Ivy	Cycles on Haswell	Cycles on Skylake
Weierstraß affine	630	225	168
Weierstraß projective	758	250	149
Huff	621	215	152
Edwards	643	223	178

4.2 Laddering Algorithms

The observations from Table 2 allow us to reduce the combinations of scalar multiplication algorithm and curve model to select only the most promising ones. Because we could not evaluate the GLS endomorphism in the Edwards model efficiently, we did not implement the two-dimensional DJB and AK laddering algorithms in this curve model. The DJB algorithm was then implemented for the Weierstraß and Huff models, where the GLS endomorphism can be efficiently applied, and the AK algorithm was implemented in the projective Weierstraß model due to the restrictions imposed by the difference point changing at every iteration (difference in projective coordinates). We present the execution times for scalar multiplication in Table 3 below. Following [6], implementations are classified in terms of resistance against timing attacks (TAR) in *uniform* (U) where the same number of field operations is executed at every laddering iteration, but the number of iterations may be variable; and *constant-time* (CT) when the two requirements are satisfied. Timings for DJB and AK include recoding routines, although this step is negligible only in the DJB chain.

Table 3. Results from related work and for our implementation for uniform (U) and constant-time (CT) scalar multiplication algorithms over binary and prime curves at the 128-bit security level. Performance figures are presented for Ivy Bridge (I), Haswell (H) or Skylake (S) platforms. Timings for Four$\mathbb{Q}$ in the Skylake processor were obtained by benchmarking code available by [7] in our platform (*). Our best numbers for each platform are highlighted in bold and best numbers overall in italic.

Related work (laddering/window)	Curve	TAR	Cycles on I	Cycles on H	Cycles on S
DJB laddering [6]	prime	CT	148,000	-	-
AK laddering [6]	prime	U	133,000	-	-
Four$\mathbb{Q}$ (window-based) [7]	prime	CT	*69,000*	56,000	46,467*
Montgomery ladder [18,23]	binary	CT	-	70,800	50,823
2-GLV double-and-add [19,20]	binary	CT	114,800	*48,312*	*38,044*
This work (laddering)	Curve	TAR	Cycles on I	Cycles on H	Cycles on S
Montgomery on Weierstraß	binary	CT	142,660	60,838	46,446
Montgomery on Huff	binary	CT	147,914	58,214	44,373
Montgomery on Edwards	binary	CT	150,483	60,083	46,538
DJB on Weierstraß	binary	CT	123,145	**50,851**	39,800
DJB on Huff	binary	CT	**122,541**	51,995	**38,658**
AK on Weierstraß	binary	U	124,267	55,524	41,492

The table demonstrates that binary curves are only competitive in Haswell and Skylake platforms supporting efficient vectorized binary field arithmetic through a very fast carry-less multiplier. Laddering approaches can be competitive with the window-based methods employed in Four$\mathbb{Q}$ [7] and 2-GLV double-and-add [19] if our techniques are employed. For the Weierstraß model, a direct comparison for the Montgomery Ladder algorithm between our implementation and [23] gives a 8.6% speedup on Haswell. We implemented formulas from Sect. 3.1 for y-coordinate recovery and the resulting cost was negligible, amounting to 333 cycles in Haswell and 312 cycles in Skylake, almost 4 times faster than the 1203 Skylake cycles in [23]. We strongly suspect that their implementation uses two inversions for computing both x and y coordinates.

In particular, performance figures for our implementation of the DJB algorithm in the Huff model were very close to speed records presented in [19], being slower by 5% and 1.6% in the Haswell and Skylake platforms, respectively. This is an interesting result, given that the laddering algorithms are simpler to implement with protection against side-channel attacks, and require smaller amounts of storage. These approaches are somewhat penalized by an affine point addition at the beginning of the laddering algorithm to compute difference points $w(P \pm Q)$. The AK laddering algorithm suffers from a slow recoding routine costing 6.4% and 8.7% of the whole scalar multiplication in the two platforms, respectively. This cost comes mostly from the side-channel protections in Algorithm 3 and the penalty could be alleviated if scalars were already generated in

recoded form, given that the constant time requirement is not mandatory. We now discuss application of our techniques in the broader context of key exchange protocols.

4.3 Discussion

Our techniques can be applied for accelerating the curve-based Diffie-Hellman key exchange. In the ephemeral version of the protocol, two parties negotiate a shared key by first generating an ephemeral key pair (a, A) (respectively (b, B)) using a fixed-base scalar multiplication $A = aG$ of generator G (respectively $B = bG$), exchanging the resulting ephemeral public keys A and B and computing the variable-base scalar multiplication of the received public key by the ephemeral private key as $K = abG$.

After restricting the scalar multiplication approaches exclusively to laddering algorithms, there are a few options. The DJB algorithm in the Weierstraß and Huff model is well suited for the fixed-base scalar multiplication, because the affine point addition required for computing $w(G \pm \psi(G))$ can be precomputed and provided together with the curve parameters. The curve models also allow simple recovery of the y-coordinate to allow any receiving party to efficiently evaluate the GLS endomorphism and employ a two-dimensional laddering algorithm for its variable-base scalar multiplication. Notice that this is not true for the AK algorithm, which is more useful for the variable-base multiplication. In the latter case, subscalars can be generated in recoded form to avoid the high cost of the AK recoding. Table 4 reports our timings for implementing the ephemeral Diffie-Hellman key exchange using the proposed optimizations in three scenarios: for comparison with related work, the Montgomery laddering algorithm is used in the Weierstraß model; the DJB algorithm in the Huff model is used for the two scalar multiplications to achieve constant time execution; and side-channel security is relaxed by using the AK algorithm in the Weierstraß model for the second scalar multiplication with previously recoded subscalars.

We restrict the comparison to Haswell and Skylake platforms where binary curves enjoy faster vector instruction sets. Compared to the state-of-the-art in laddering implementations for the Skylake platform, our implementation of the standard Montgomery laddering in the Weierstraß model improves upon [23] by 2.9%, but is not competitive with the window-based method in [19]. The DJB algorithm in the Huff model increases the performance improvement to 21.3% and becomes close to window-based methods. Notice, however, that Four$\mathbb{Q}$ employs a large 7.5 KB precomputed table for accelerating the window-based fixed-base portion of the key exchange protocol, a technique from which we do not benefit in this work. We anticipate such an optimization would reach a new speed record for key exchange implementations in the target platform. The performance for key exchange can be slightly increased by using a combination of the DJB and AK laddering algorithms, if one is willing to sacrifice constant-time execution for uniform execution only.

Table 4. Results for related work and our implementations of the Diffie-Hellman key exchange, using different approaches for instantiating the protocol. Benchmarks are presented for Ivy Bridge (I), Haswell (H) or Skylake (S) platforms. Timings for Four$\mathbb{Q}$ in the Skylake processor were obtained by benchmarking code available by [7] in our platform (*). Our best numbers in each platform are highlighted in bold and best numbers overall in italic.

Related work (laddering/window)	Curve	TAR	Cycles on I	Cycles on H	Cycles on S
Four$\mathbb{Q}$ (window-based) [7]	prime	CT	*104,000*	*88,000*	*74,032**
2-GLV double-and-add [19]	binary	CT	120,000	96,624	76,088
Montgomery ladder [23]	binary	CT	-	-	95,702
This work (laddering)	Curve	TAR	Cycles on I	Cycles on H	Cycles on S
Montgomery on Weierstraß	binary	CT	295,828	121,676	92,890
DJB on Huff	binary	CT	245,682	**101,696**	75,318
DJB + AK on Weierstraß	binary	U	**243,188**	101,769	**74,440**

5 Conclusion

This work presented several contributions. First, we proposed tricks to convert GLS curves to alternative models and obtained parameters optimized for elliptic curve arithmetic. The latest formulas for differential addition and doubling in the Weiertraß, Huff and Edwards models were slightly improved by using lazy reduction and short coefficients allowed by the parameters. The resulting implementations were combined with efficient implementations of the Montgomery, DJB and AK algorithms to obtain efficient scalar multiplication based on laddering, achieving a new speed record in laddering algorithms for high-end Intel desktop processors and performance improvements for executing the Diffie-Hellman key exchange protocol.

As future work, we plan to extend our strategies to the recently proposed twisted μ_4-normal form binary curves [14] to enjoy their efficient arithmetic in the case of elliptic curves with endomorphisms. The entire code for our implementations is available at https://github.com/dfaranha/ladd-gls254 to allow reproducibility and facilitate further improvements by independent researchers.

Acknowledgements. The authors would like to thank the reviewers for their comments. This work is supported in parts by the Intel/FAPESP grant 14/50704-7 under project "Secure Execution of Cryptographic Algorithms", and the grants NIST-60NANB16D246, NSF CNS-1661557, and ARO W911NF-17-1-0311.

References

1. Azarderakhsh, R., Karabina, K.: A new double point multiplication algorithm and its application to binary elliptic curves with endomorphisms. IEEE Trans. Comput. **63**(10), 2614–2619 (2014)
2. Bernstein, D.J.: Differential addition chains, Preprint (2006)

3. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.: High-speed high-security signatures. J. Cryptograph. Eng. **2**(2), 77–89 (2012)
4. Bernstein, D.J., Lange, T.: Montgomery curves and the Montgomery ladder. In: Bos, J.W., Lenstra, A.K. (eds.) Topics In Computational Number Theory Inspired by Peter L. Montgomery. Cambridge University Press (2017, to appear). https://eprint.iacr.org/2017/293
5. Bernstein, D.J., Lange, T., Rezaeian Farashahi, R.: Binary edwards curves. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 244–265. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85053-3_16
6. 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, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_11
7. Costello, C., Longa, P.: Four$\mathbb{Q}$: four-dimensional decompositions on a $\mathbb{Q}$-curve over the mersenne prime. In: Iwata, T., Cheon, J.H. (eds.) ASIACRYPT 2015. LNCS, vol. 9452, pp. 214–235. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48797-6_10
8. Devigne, J., Joye, M.: Binary huff curves. In: Kiayias, A. (ed.) CT-RSA 2011. LNCS, vol. 6558, pp. 340–355. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-19074-2_22
9. Rezaeian Farashahi, R., Hosseini, S.G.: Differential addition on binary elliptic curves. In: Duquesne, S., Petkova-Nikova, S. (eds.) WAIFI 2016. LNCS, vol. 10064, pp. 21–35. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-55227-9_2
10. Galbraith, S.D., Gaudry, P.: Recent progress on the elliptic curve discrete logarithm problem. Des. Codes Crypt. **78**(1), 51–72 (2016)
11. Galbraith, S.D., Lin, X., Scott, M.: Endomorphisms for faster elliptic curve cryptography on a large class of curves. J. Cryptol. **24**(3), 446–469 (2011)
12. 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). http://dx.doi.org/10.1109/TC.2009.61
13. Institute of Electrical and Electronics Engineers: Traditional public-key cryptography (IEEE Std 1363–2000 and 1363a–2004) (2004). http://grouper.ieee.org/groups/1363/
14. Kohel, D.: Twisted μ_4-normal form for elliptic curves. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 659–678. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_23
15. López, J., Dahab, R.: Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. In: Koç, Ç.K., Paar, C. (eds.) CHES 1999. LNCS, vol. 1717, pp. 316–327. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48059-5_27
16. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 243–264 (1987)
17. National Institute of Standards and Technology: Recommended Elliptic Curves for Federal Government Use. NIST Special Publication (1999). http://csrc.nist.gov/groups/ST/toolkit/documents/dss/NISTReCur.pdf
18. Oliveira, T., Aranha, D.F., López, J., Rodríguez-Henríquez, F.: Fast point multiplication algorithms for binary elliptic curves with and without precomputation. In: Joux, A., Youssef, A. (eds.) SAC 2014. LNCS, vol. 8781, pp. 324–344. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-13051-4_20
19. Oliveira, T., Aranha, D.F., Hernandez, J.L., Rodríguez-Henríquez, F.: Improving the performance of the GLS254 curve. In: CHES Rump Session (2016)

20. Oliveira, T., López, J., Aranha, D.F., Rodríguez-Henríquez, F.: Lambda coordinates for binary elliptic curves. In: Bertoni, G., Coron, J.-S. (eds.) CHES 2013. LNCS, vol. 8086, pp. 311–330. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40349-1_18
21. Oliveira, T., López, J., Aranha, D.F., Rodríguez-Henríquez, F.: Two is the fastest prime: lambda coordinates for binary elliptic curves. J. Cryptograph. Eng. **4**(1), 3–17 (2014)
22. Oliveira, T., López, J., Rodríguez-Henríquez, F.: Software implementation of Koblitz curves over quadratic fields. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 259–279. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53140-2_13
23. Oliveira, T., López, J., Rodríguez-Henríquez, F.: The Montgomery ladder on binary elliptic curves. J. Cryptograph. Eng. (2017, to appear). https://eprint.iacr.org/2017/350
24. Stam, M.: On montgomery-like representations for elliptic curves over $GF(2^k)$. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 240–254. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36288-6_18