Published in *Journal of Cryptographic Engineering*, **6**(3):187–199, 2016.

Inversion-Free Arithmetic on Elliptic Curves Through Isomorphisms

Raveen R. Goundar · Marc Joye

Abstract This paper presents inversion-free formulas for the efficient implementation of a scalar multiplication over elliptic curves. Specifically, it proposes to make use of curve isomorphisms as a way to avoid the computation of inverses in point addition formulas. Interestingly, the presented techniques are independent of the model used to represent the elliptic curve and of the coordinate system used to represent the points. In particular, they apply to affine representations. Further, whereas certain inversion-free techniques are mostly limited to specific scalar multiplication algorithms, the proposed techniques apply to all scalar multiplication algorithms. The so-obtained formulas are well suited to embedded systems and can easily be combined with existing countermeasures to provide secure implementations.

Keywords Elliptic curves · scalar multiplication · isomorphisms · affine coordinates

1 Introduction

Elliptic curve cryptography (ECC) [20, 26] provides high level of security (exponential security) in comparison with the conventional RSA cryptosystem (subexponential security). The National Security Agency (NSA) [30] and the National Institute of Standards and Technology (NIST) [8] have published directives and standards, naming ECC as a primarily strong method for protecting classified and unclassified sensitive documents. Its smaller key size makes ECC particularly attractive for small devices such as smart cards.

The efficiency of ECC is dominated by an operation called *scalar multiplication* (or point multiplication). The problem is, given a point $\boldsymbol{P}$ on an elliptic curve (defined over a finite field) and a scalar k, to generate the point $k\boldsymbol{P}$, that is, $\boldsymbol{P}+\boldsymbol{P}+\cdots+\boldsymbol{P}$ (k times) as cost efficiently as possible. This problem is an obvious analog of the exponentiation [9].

The elliptic curve group operations can be expressed in terms of a number of operations in the definition field. The crucial problem becomes to find the right model to represent an elliptic curve in a way to minimize the number of field operations. Indeed, as an elliptic curve is defined up to birational transformations, there are plenty of possible choices for its representation. However, in order not to explode the number of coordinates and operations, only models of elliptic curves lying in low-dimensional spaces are considered in practice [4]. Furthermore, the basic operations involved in point addition formulas—namely, field addition/subtraction, field multiplication, and field inversion, are not equivalent with each other. In particular, field inversion requires a special attention as it may significantly impact the overall performance [7]. For cryptographic applications, typical ratios for inversion over multiplication in the underlying finite field range from 3 to 100 [12]. For that reason, *inversion-free* point addition formulas are of particular interest. This is classically achieved by resorting to projective representations (including the widely used homogeneous and Jacobian coordinates). Numerous useful forms of elliptic curves using various coordinate systems and their respective costs are compiled in [2].

Our contributions We present a *generic* approach to get rid of the field inversion operation in the evaluation

Raveen R. Goundar
Independent Researcher
12 Birkdale Street, Colebee, NSW 2761, Australia

Marc Joye
Technicolor
175 S San Antonio Road, Los Altos, CA 94022, USA

of a scalar multiplication. Our approach can be seen as a refinement of the so-called co-Z arithmetic developed by Meloni [24]. Co-Z arithmetic was successfully applied to scalar multiplication algorithms (i) built from Euclidean chains (i.e., using the Zeckendorf's representation) in [24], (ii) for precomputation schemes in [22], and (iii) using Montgomery-like ladders in [11]), for Weierstraß elliptic curves over large-characteristic fields with Jacobian coordinates. It was also recently applied to hyperelliptic curves [13].

Basically, our approach is to consider the output of a point addition as a point on some related elliptic curve. For well-chosen related curves (i.e., isomorphic curves), this presents benefits similar to the ones expected from the co-Z-based methods. But whereas the co-Z methods were primarily designed for Jacobian coordinates, the proposed approach provides a general framework and thereby widens the range of applications and offered features. An important advantage of recasting scalar multiplication algorithms with the language of curve isomorphisms resides in its *simplicity*. It makes apparent that Jacobian coordinates appear—in some disguised form—as the natural choice to represent points on Weierstraß elliptic curves. It also simplifies algorithms. For example, Algorithms 6 and 7 in [11] are Montgomery-like ladders using only the X- and Y-coordinates of points represented in Jacobian coordinates. While this stems from an astute observation that Z-coordinates are not involved in co-Z point additions, this again comes naturally using curve isomorphisms. The proposed framework helps to better understand the underlying arithmetic. It naturally applies to a variety of elliptic curve models and scalar multiplication algorithms. The deeper understanding of the inner workings allows one to more easily discover implementation tricks for improved performance and/or security of the resulting algorithms.

Outline of the paper The rest of this paper is organized as follows. The next section is the core of our paper. We describe new addition formulas using curve isomorphisms—some background on elliptic curves can be found in appendix. We then present applications thereof in Sect. 3. A performance analysis and security considerations are provided in Sect. 4. Finally, we conclude the paper in Sect. 5.

2 Inversion-Free Arithmetic

2.1 Warming up

Before describing our method in its full generality, we first make a couple of observations on the Weierstraß model over a field of characteristic $\neq 2, 3$.

Consider the elliptic curve E_1 over a field $\mathbb{K}$, with $\operatorname{char}\mathbb{K} \neq 2, 3$, given by

$$E_1 \colon y^2 = x^3 + ax + b \ . \tag{1}$$

Given two finite points $\boldsymbol{P_1} = (x_1, y_1)$ and $\boldsymbol{P_2} = (x_2, y_2)$ on E_1 such that $\boldsymbol{P_1} \neq \pm\boldsymbol{P_2}$ (i.e., such that $x_1 \neq x_2$), their sum is given by $\boldsymbol{P_3} = \boldsymbol{P_1} + \boldsymbol{P_2} = (x_3, y_3)$ where

$$x_3 = \left(\frac{y_1 - y_2}{x_1 - x_2}\right)^2 - x_1 - x_2 \quad \text{and}$$
$$y_3 = (x_1 - x_3)\left(\frac{y_1 - y_2}{x_1 - x_2}\right) - y_1 \ . \tag{2}$$

The double of $\boldsymbol{P_1} = (x_1, y_1)$, provided that $2y_1 + a_1x_1 + a_3 \neq 0$, is given by $\boldsymbol{P_4} = \boldsymbol{2P_1} = (x_4, y_4)$ where

$$x_4 = \left(\frac{3x_1^2 + a}{2y_1}\right)^2 - 2x_1 \quad \text{and}$$
$$y_4 = (x_1 - x_4)\left(\frac{3x_1^2 + a}{2y_1}\right) - y_1 \ . \tag{3}$$

A common way to avoid inversion is to resort to a projective form for E_1. For example, using Jacobian coordinates, letting $x = X/Z^2$ and $y = Y/Z^3$, Equation (1) becomes

$$E_1 : Y^2 = X^3 + aXZ^4 + bZ^6 \ . \tag{4}$$

A finite point $\boldsymbol{P_1} = (x_1, y_1) \in E_1$ is then represented in Jacobian coordinates as a triplet $(X_1 : Y_1 : Z_1)$ with $X_1 = x_1{Z_1}^2$ and $Y_1 = y_1{Z_1}^3$, for any $Z_1 \in \mathbb{K}^*$. There is another way to view the Jacobian representation of point $\boldsymbol{P_1}$. As detailed in Appendix A (taking $r = s = t = 0$), for any $u \in \mathbb{K}^*$, elliptic curve E_1 is $\mathbb{K}$-isomorphic to elliptic curve

$$E_u \colon y^2 = x^3 + au^4\,x + bu^6 \tag{5}$$

via the inverse mappings

$$\Psi_u \colon E_1 \xrightarrow{\sim} E_u, \begin{cases} \boldsymbol{O} \mapsto \boldsymbol{O} \\ (x, y) \mapsto (u^2x, u^3y) \end{cases}$$

and

$$\Psi_u^{-1} \colon E_u \xrightarrow{\sim} E_1, \begin{cases} \boldsymbol{O} \mapsto \boldsymbol{O} \\ (\tilde{x}, \tilde{y}) \mapsto (u^{-2}\tilde{x}, u^{-3}\tilde{y}) \end{cases} \ .$$

With this view in mind, we can see point (X_1, Y_1) as a point on the isomorphic curve E_{Z_1} (compare Eq. (5) with Eq. (4)).

Meloni observed in [24] that, on a short Weierstraß elliptic curve E_1 (*cf.* Eq. (4)), two finite points $\boldsymbol{P_1} = (X_1 : Y_1 : Z)$ and $\boldsymbol{P_2} = (X_2 : Y_2 : Z)$ given in Jacobian coordinates and sharing the same Z-coordinate can be added faster to get $\boldsymbol{P_3} = \boldsymbol{P_1} + \boldsymbol{P_2} = (X_3 : Y_3 : Z_3) \in E_1$. Since two such points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ can be regarded as two points (X_1, Y_1) and (X_2, Y_2) given in affine coordinates on a same isomorphic curve E_Z (*cf.* Eq. (5)), we are led to the following observations:

1. Defining $\varphi := x_1 - x_2$, we get from the above addition formula (Eq. (2))

$$\varphi^2 x_3 = (y_1 - y_2)^2 - \varphi^2 x_1 - \varphi^2 x_2 \quad \text{and}$$
$$\varphi^3 y_3 = (\varphi^2 x_1 - \varphi^2 x_3)(y_1 - y_2) - \varphi^3 y_1 \ .$$

   In other words, given points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ on E_1 (i.e., on E_Z with $Z = 1$), one can easily obtain point $\tilde{\boldsymbol{P_3}} := \Psi_\varphi(\boldsymbol{P_1} + \boldsymbol{P_2}) = (\varphi^2 x_3, \varphi^3 y_3)$ on the isomorphic elliptic curve E_φ. It is worth remarking that no inversion is required in the evaluation of $\tilde{\boldsymbol{P_3}}$. We let iADD denote the operation of getting $\tilde{\boldsymbol{P_3}} \in E_\varphi$.
2. A similar treatment applies to the point doubling operation. Defining now $\varphi := 2y_1$, we get from the doubling formula (Eq. (3))

$$\varphi^2 x_4 = (3x_1^2 + a)^2 - 2\varphi^2 x_1 \quad \text{and}$$
$$\varphi^3 y_4 = (\varphi^2 x_1 - \varphi^2 x_4)(3x_1^2 + a) - \varphi^3 y_1 \ .$$

   Namely, given a point $\boldsymbol{P_1}$ on E_1, one can easily obtain $\tilde{\boldsymbol{P_4}} := \Psi_\varphi(2\boldsymbol{P_1}) = (\varphi^2 x_4, \varphi^3 y_4)$ on E_φ. As for the point addition, it is worth remarking that no inversion is required in the evaluation of $\tilde{\boldsymbol{P_4}}$. We let iDBL denote this operation.

2.2 General description

Let $E_{\mathbb{1}}$ be an elliptic curve over a field $\mathbb{K}$. Consider a family $\{E_{\boldsymbol{\Phi}}\}$ of isomorphic elliptic curves, indexed by some parameter $\boldsymbol{\Phi}$, under isomorphism $\Psi_{\boldsymbol{\Phi}} \colon E_{\mathbb{1}} \xrightarrow{\sim} E_{\boldsymbol{\Phi}}$. Parameter $\boldsymbol{\Phi}$ is a description of the isomorphism; we write $\boldsymbol{\Phi} = \mathrm{desc}(\Psi_{\boldsymbol{\Phi}})$. We let $\mathcal{F}$ denote the set of all possible $\boldsymbol{\Phi}$'s.

To simplify the exposition, we abuse the classical notations for maps. We define three addition operations, iADD, iADDU, and iADDC, given by

$$\begin{cases} \text{iADD} \colon E_{\mathbb{1}} \times E_{\mathbb{1}} \to E_{\boldsymbol{\Phi}} \times \mathcal{F}, \\ \qquad (\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_{\boldsymbol{\Phi}}(\boldsymbol{P_1} + \boldsymbol{P_2}), \boldsymbol{\Phi}\big) \\ \text{iADDU} \colon E_{\mathbb{1}} \times E_{\mathbb{1}} \to E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \times \mathcal{F}, \\ \qquad (\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_{\boldsymbol{\Phi}}(\boldsymbol{P_1} + \boldsymbol{P_2}), \Psi_{\boldsymbol{\Phi}}(\boldsymbol{P_1}), \boldsymbol{\Phi}\big) \\ \text{iADDC} \colon E_{\mathbb{1}} \times E_{\mathbb{1}} \to E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \times \mathcal{F}, \\ \qquad (\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_{\boldsymbol{\Phi}}(\boldsymbol{P_1} + \boldsymbol{P_2}), \\ \qquad\qquad \Psi_{\boldsymbol{\Phi}}(\boldsymbol{P_1} - \boldsymbol{P_2}), \boldsymbol{\Phi}\big) \end{cases} . \quad (6)$$

For efficiency purposes, parameter $\boldsymbol{\Phi}$ is chosen so that, given two different points $\boldsymbol{P_1}$ and $\boldsymbol{P_2}$ on $E_{\mathbb{1}}$, the output of the addition operation does not require field inversions.

We also define two doubling operations, iDBL and iDBLU, given by

$$\begin{cases} \text{iDBL} \colon E_{\mathbb{1}} \to E_{\boldsymbol{\Phi}} \times \mathcal{F}, \\ \qquad \boldsymbol{P_1} \mapsto \big(\Psi_{\boldsymbol{\Phi}}(2\boldsymbol{P_1}), \boldsymbol{\Phi}\big) \\ \text{iDBLU} \colon E_{\mathbb{1}} \to E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \times \mathcal{F}, \\ \qquad \boldsymbol{P_1} \mapsto \big(\Psi_{\boldsymbol{\Phi}}(2\boldsymbol{P_1}), \Psi_{\boldsymbol{\Phi}}(\boldsymbol{P_1}), \boldsymbol{\Phi}\big) \end{cases} . \quad (7)$$

Likewise, parameter $\boldsymbol{\Phi}$ is chosen so that, given a point $\boldsymbol{P_1}$ on $E_{\mathbb{1}}$, the output of the doubling operation does not require field inversions.

More generally, given two elliptic curves $E_{\boldsymbol{\Phi}}$ and $E_{\boldsymbol{\Phi}'}$ being isomorphic to $E_{\mathbb{1}}$, if

$$\Psi_{\boldsymbol{\varphi}} \colon E_{\boldsymbol{\Phi}} \xrightarrow{\sim} E_{\boldsymbol{\Phi}'}$$

denotes the isomorphism between $E_{\boldsymbol{\Phi}}$ and $E_{\boldsymbol{\Phi}'}$ we similarly define the operations

$$\begin{cases} \text{iADD}_{\boldsymbol{\Phi}} \colon E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \to E_{\boldsymbol{\Phi}'} \times \mathcal{F}, \\ \qquad (\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_{\boldsymbol{\varphi}}(\boldsymbol{P_1} + \boldsymbol{P_2}), \boldsymbol{\varphi}\big) \\ \text{iADDU}_{\boldsymbol{\Phi}} \colon E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \to E_{\boldsymbol{\Phi}'} \times E_{\boldsymbol{\Phi}'} \times \mathcal{F}, \\ \qquad (\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_{\boldsymbol{\varphi}}(\boldsymbol{P_1} + \boldsymbol{P_2}), \Psi_{\boldsymbol{\varphi}}(\boldsymbol{P_1}), \boldsymbol{\varphi}\big) \\ \text{iADDC}_{\boldsymbol{\Phi}} \colon E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \to E_{\boldsymbol{\Phi}'} \times E_{\boldsymbol{\Phi}'} \times \mathcal{F}, \\ \qquad (\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_{\boldsymbol{\varphi}}(\boldsymbol{P_1} + \boldsymbol{P_2}), \\ \qquad\qquad \Psi_{\boldsymbol{\varphi}}(\boldsymbol{P_1} - \boldsymbol{P_2}), \boldsymbol{\varphi}\big) \\ \text{iDBL}_{\boldsymbol{\Phi}} \colon E_{\boldsymbol{\Phi}} \to E_{\boldsymbol{\Phi}'} \times \mathcal{F}, \\ \qquad \boldsymbol{P_1} \mapsto \big(\Psi_{\boldsymbol{\varphi}}(2\boldsymbol{P_1}), \boldsymbol{\varphi}\big) \\ \text{iDBLU}_{\boldsymbol{\Phi}} \colon E_{\boldsymbol{\Phi}} \to E_{\boldsymbol{\Phi}'} \times E_{\boldsymbol{\Phi}'} \times \mathcal{F}, \\ \qquad \boldsymbol{P_1} \mapsto \big(\Psi_{\boldsymbol{\varphi}}(2\boldsymbol{P_1}), \Psi_{\boldsymbol{\varphi}}(\boldsymbol{P_1}), \boldsymbol{\varphi}\big) \end{cases} . \quad (8)$$

Subscript $\boldsymbol{\Phi}$ in the operator definition indicates that input points belong to the elliptic curve $E_{\boldsymbol{\Phi}}$.

Example 1 We illustrate the notation for general Weierstraß elliptic curves; see Appendix A. In this case, we have:

$$E_{\mathbb{1}} \colon y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6$$

and

$$\Psi_{\boldsymbol{\varphi}} \colon E_{\boldsymbol{\Phi}} \xrightarrow{\sim} E_{\boldsymbol{\Phi}'} \ , \ (x, y) \longmapsto (u^2 x + r, u^3 y + u^2 s x + t)$$

where $\boldsymbol{\varphi} = (u, r, s, t)$ and $\mathbb{1} = (1, 0, 0, 0)$. We also have $\mathcal{F} = \big\{(U, R, S, T) \in \mathbb{K} \mid U \neq 0\big\}$ where $\mathbb{K}$ is the definition field of $E_{\mathbb{1}}$.

Table 1 Exemplary operation counts for basic addition formulas (M and S denote the cost of a field multiplication and of a field squaring, respectively)

Model	iADD	iADDU	iADDC	iDBL
Weierstraß	4M + 2S	4M + 2S	5M + 3S	1M + 5S
Weierstraß (binary case)[a]	8M + 1S	8M + 1S	12M + 1S	5M + 5S
Twisted Edwards[b]	10M + 1S	12M + 1S	13M + 1S	–

[a] Using Stam's parametrization.
[b] Presented point addition formulas are unified (i.e., they also work for point doubling).

2.3 Applications

The proposed approach is very general and applies to any model of elliptic curves. The Weierstraß model over a field $\mathbb{K}$ of characteristic $\operatorname{char}\mathbb{K} \neq 2, 3$ is sketched in Sect. 2.1. We refer the reader to Appendix B for detailed realizations of the new operations (i.e., iADD, iADDU, iADDC, iDBL, and more) for the Weierstraß model and to Sect. 4.2 for other elliptic curve models.

Table 1 lists various operation counts for certain elliptic curve representations.

3 Efficient Scalar Multiplication

As aforementioned, the basic operation in elliptic curve cryptography is the scalar multiplication. Given a point $\boldsymbol{P}$ on an elliptic curve and a positive scalar k, one has to evaluate $k\boldsymbol{P} = \boldsymbol{P} + \boldsymbol{P} + \cdots + \boldsymbol{P}$.

An addition chain for k is a list of integers, $a_0 = 1, a_1, \ldots, a_\ell = k$ such that for each $i \geqslant 1$, there exists some u and v with $1 \leqslant u, v < i$ and $a_i = a_u + a_v$ [19, § 4.6.3]. Such a chain immediately gives rise to a scalar multiplication algorithm. The question is how long the addition chain is as this determines the running time. Another question is the number of elements that have to be kept in the addition chain to evaluate the next elements as this determines the memory requirements. Efficient scalar multiplication algorithms can therefore be derived from short addition chains wherein the elements are obtained from their direct predecessors and/or some fixed elements.

3.1 Composition of isomorphisms

Applied to our framework, we further require that at each step the two points being added lie on the same elliptic curve. Jumping through curve isomorphisms, we end up with the image of $k\boldsymbol{P}$ on some isomorphic elliptic curve. This resulting point is then converted as a point on the original curve so as to get the expected result, namely $k\boldsymbol{P}$. To do so, we need to explicitly know the isomorphism between the final elliptic curve and the original elliptic curve.

In more detail, if we let $E^{(0)} = E_{\mathbb{1}}$ denote the original elliptic curve, $E^{(i)} = E_{\boldsymbol{\Phi}_i}$ the current elliptic curve at Step i, and $E^{(\ell(k))} = E_{\boldsymbol{\Phi}_{\ell(k)}}$ the final elliptic curve, we have $\boldsymbol{P} \in E^{(0)}$ and

$$\tilde{\boldsymbol{Q}} := k\big((\Psi_{\boldsymbol{\varphi}_{\ell(k)}} \circ \cdots \circ \Psi_{\boldsymbol{\varphi}_i} \circ \cdots \circ \Psi_{\boldsymbol{\varphi}_1})\boldsymbol{P}\big) \in E^{(\ell(k))}$$

or, schematically, as illustrated in Fig. 1.

The isomorphism between the current curve at Step i and the original curve is given by $\Psi_{\boldsymbol{\Phi}_i} = \Psi_{\boldsymbol{\varphi}_i} \circ \cdots \circ \Psi_{\boldsymbol{\varphi}_1}$. Slightly abusing the notation, we also use symbol $\circ$ to denote the operation on the corresponding descriptions, namely $\operatorname{desc}(\Psi_{\boldsymbol{\Phi}_i}) = \boldsymbol{\varphi}_i \circ \cdots \circ \boldsymbol{\varphi}_1$. Since $\tilde{\boldsymbol{Q}} = k\big(\Psi_{\boldsymbol{\Phi}_{\ell(k)}}(\boldsymbol{P})\big) = \Psi_{\boldsymbol{\Phi}_{\ell(k)}}(k\boldsymbol{P})$, the result $\boldsymbol{Q} = k\boldsymbol{P} \in E^{(0)}$ is then given by $\boldsymbol{Q} = \Psi^{-1}_{\boldsymbol{\Phi}_{\ell(k)}}(\tilde{\boldsymbol{Q}})$.

The 'composed' isomorphism $\Psi_{\boldsymbol{\Phi}_{\ell(k)}}$ can be obtained iteratively by observing that $\Psi_{\boldsymbol{\Phi}_i} = \Psi_{\boldsymbol{\varphi}_i} \circ \Psi_{\boldsymbol{\Phi}_{i-1}}$ with $\Psi_{\boldsymbol{\Phi}_0} = \mathrm{Id}$ (i.e., the identity map). Since $\boldsymbol{\Phi}_i = \operatorname{desc}(\Psi_{\boldsymbol{\Phi}_i})$, we get

$$\boldsymbol{\Phi}_i = \boldsymbol{\varphi}_i \circ \boldsymbol{\Phi}_{i-1} \tag{9}$$

with $\boldsymbol{\Phi_0} = \operatorname{desc}(\mathrm{Id}) := \mathbb{1}$.

Example 2 (Example 1 cont'd) In the case of general Weierstraß elliptic curves, we have:

$$\begin{aligned}\Psi_{\boldsymbol{\Phi}_{i-1}} \colon E^{(0)} &\xrightarrow{\sim} E^{(i-1)}, (x, y) \longmapsto \\ &(U_{i-1}{}^2 x + R_{i-1}, U_{i-1}{}^3 y + U_{i-1}{}^2 S_{i-1} x + T_{i-1}) \\ \Psi_{\boldsymbol{\varphi}_i} \colon E^{(i-1)} &\xrightarrow{\sim} E^{(i)}, (x, y) \longmapsto \\ &(u_i{}^2 x + r_i, u_i{}^3 y + u_i{}^2 s_i x + t_i)\end{aligned}$$

where $\boldsymbol{\Phi}_{i-1} = (U_{i-1}, R_{i-1}, S_{i-1}, T_{i-1})$, $\boldsymbol{\varphi}_i = (u_i, r_i, s_i, t_i)$ and $\mathbb{1} = (1, 0, 0, 0)$. Hence, Equation (9) translates into $(U_i, R_i, S_i, T_i) = (u_i, r_i, s_i, t_i) \circ (U_{i-1}, R_{i-1}, S_{i-1}, T_{i-1})$ with

$$\begin{cases} U_i = U_{i-1} u_i \\ R_i = u_i{}^2 R_{i-1} + r_i \\ S_i = u_i S_{i-1} + s_i \\ T_i = u_i{}^3 T_{i-1} + u_i{}^2 s_i R_{i-1} + t_i \end{cases} \tag{10}$$

for $i \geqslant 1$, and $(U_0, R_0, S_0, T_0) = (1, 0, 0, 0)$.

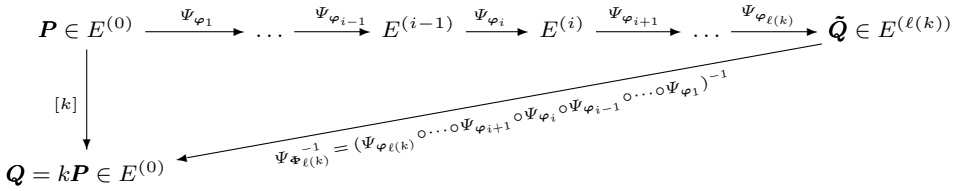

Fig. 1 Composition of isomorphisms

3.2 Classical methods

The classical method for evaluating $\boldsymbol{Q} = k\boldsymbol{P}$ considers the binary representation of scalar k, $k = (k_{n-1}, \ldots, k_0)_2$ with $k_i \in \{0, 1\}$, $0 \leqslant i \leqslant n-1$ [19, § 4.6.3]. Remarkably, it requires a minimal amount of memory and is hence well suited to memory-constrained devices like smart cards. The method relies on the obvious relation that $k\boldsymbol{P} = 2(\lfloor k/2 \rfloor \boldsymbol{P})$ if k is even and $k\boldsymbol{P} = 2(\lfloor k/2 \rfloor \boldsymbol{P}) + \boldsymbol{P}$ if k is odd. Iterating the process yields a left-to-right scalar multiplication algorithm, also known as *double-and-add method*; see Alg. 1. It requires two (point) variables, $\boldsymbol{R_0}$ and $\boldsymbol{R_1}$. Variable $\boldsymbol{R_0}$ acts as an accumulator, and variable $\boldsymbol{R_1}$ is used to store the value of input point $\boldsymbol{P}$.

Algorithm 1 Double-and-add method

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

1: $\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
2: **for** $i = n-1$ down to 0 **do**
3: $\quad \boldsymbol{R_0} \leftarrow 2\boldsymbol{R_0}$
4: $\quad$ **if** $(k_i = 1)$ **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$
5: **end for**
6: **return** $\boldsymbol{R_0}$

Algorithm 2 presents a straightforward implementation of the classical scalar multiplication method with the new addition and doubling formulas. We use a variable $\boldsymbol{\Phi}$ to accumulate [the description of] the current isomorphism with the original curve. This variable is initialized to $\boldsymbol{\Phi} = \mathbb{1}$ (corresponding to the identity map Id). As in Sect. 3.1, symbol $\circ$ denotes the composition of [the description of] elliptic curve isomorphisms.

There is another possible way to implement the double-and-add method (Alg. 1) with elliptic curve isomorphisms. When k_i is equal to 1, variable $\boldsymbol{R_1}$ is added to variable $\boldsymbol{R_0}$. But the content of variable $\boldsymbol{R_1}$ remains invariant throughout the computation: $\boldsymbol{R_1}$ always contains input point $\boldsymbol{P}$. It is therefore not necessary to constantly update it as a point on the current elliptic curve. Instead, at iteration i, its representative on the current elliptic curve, $E_{\boldsymbol{\Phi}}$, can be computed from input point $\boldsymbol{P}$ as $\Psi_{\boldsymbol{\Phi}}(\boldsymbol{P})$.

Algorithm 2 Double-and-add method (with elliptic curve isomorphisms)

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

1: $\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$; $\boldsymbol{\Phi} \leftarrow \mathbb{1}$
2: **for** $i = n-1$ down to 0 **do**
3: $\quad (\boldsymbol{R_0}, \boldsymbol{\varphi}) \leftarrow \mathrm{iDBL}_{\boldsymbol{\Phi}}(\boldsymbol{R_0})$; $\boldsymbol{R_1} \leftarrow \Psi_{\boldsymbol{\varphi}}(\boldsymbol{R_1})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
4: $\quad$ **if** $(k_i = 1)$ **then**
5: $\quad\quad (\boldsymbol{R_0}, \boldsymbol{R_1}, \boldsymbol{\varphi}) \leftarrow \mathrm{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{R_1}, \boldsymbol{R_0})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
6: $\quad$ **end if**
7: **end for**
8: **return** $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

Algorithm 3 Double-and-add method (with elliptic curve isomorphisms) − Version II

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

1: $\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$; $\boldsymbol{\Phi} \leftarrow \mathbb{1}$
2: **for** $i = n-1$ down to 0 **do**
3: $\quad (\boldsymbol{R_0}, \boldsymbol{\varphi}) \leftarrow \mathrm{iDBL}_{\boldsymbol{\Phi}}(\boldsymbol{R_0})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
4: $\quad$ **if** $(k_i = 1)$ **then**
5: $\quad\quad (\boldsymbol{R_0}, \boldsymbol{\varphi}) \leftarrow \mathrm{iADD}_{\boldsymbol{\Phi}}\big(\boldsymbol{R_0}, \Psi_{\boldsymbol{\Phi}}(\boldsymbol{R_1})\big)$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
6: $\quad$ **end if**
7: **end for**
8: **return** $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

Remark 1 There exists a right-to-left variant. The resulting algorithm, known as *add-and-double method*, is depicted in Alg. 7 (in Appendix C). It also requires two (point) variables, $\boldsymbol{R_0}$ and $\boldsymbol{R_1}$, but in this case both act as accumulators. An implementation with the new formulas is presented in Alg. 9 (in Appendix C).

3.3 Montgomery-like ladders

For several elliptic curve models, the point addition formulas of two *distinct* points are independent of the curve parameters. In this case, it is interesting to rely on scalar multiplication algorithms that can be written as a series of iADDU and iADDC operations. We

quote two such algorithms: the Montgomery powering ladder [28] and its dual version [16].

Algorithm 4 Montgomery ladder

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

1: $\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
2: **for** $i = n-1$ down to 0 **do**
3: $\quad b \leftarrow k_i$
4: $\quad (\boldsymbol{R_{1-b}}, \boldsymbol{T}) \leftarrow (\boldsymbol{R_b} + \boldsymbol{R_{1-b}}, \boldsymbol{R_b} - \boldsymbol{R_{1-b}})$
5: $\quad \boldsymbol{R_b} \leftarrow \boldsymbol{R_{1-b}} + \boldsymbol{T}$
6: **end for**
7: **return** $\boldsymbol{R_0}$

The presentation of the algorithm is modified as in [11]. In its original version, the main loop in Algorithm 4 reads as $\boldsymbol{R_{1-b}} \leftarrow \boldsymbol{R_b} + \boldsymbol{R_{1-b}}$, $\boldsymbol{R_b} \leftarrow 2\boldsymbol{R_b}$. Algorithm 4 is easily adapted with the new operations. The value $k_{n-1} = 1$ leads to $(\boldsymbol{R_0}, \boldsymbol{T}) = (\boldsymbol{P}, \boldsymbol{P})$ and then to $\boldsymbol{R_1} = \boldsymbol{P} + \boldsymbol{P}$ in the first iteration of Algorithm 4. This last operation is a point doubling. In order not to have to handle potential special cases, we assume that $k_{n-1} = 1$ and hence start the for-loop at $i = n-2$ and initialize $(\boldsymbol{R_0}, \boldsymbol{R_1})$ with $(\boldsymbol{P}, 2\boldsymbol{P})$. For better performance, this is carried out with the iDBLU operation.

Algorithm 5 Montgomery ladder (with elliptic curve isomorphisms)

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$ with $k_{n-1} = 1$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

1: $(\boldsymbol{R_1}, \boldsymbol{R_0}, \boldsymbol{\Phi}) \leftarrow \mathrm{iDBLU}_{\mathbb{1}}(\boldsymbol{P})$
2: **for** $i = n-2$ down to 0 **do**
3: $\quad b \leftarrow k_i$
4: $\quad (\boldsymbol{R_{1-b}}, \boldsymbol{R_b}, \varphi) \leftarrow \mathrm{iADDC}_{\boldsymbol{\Phi}}(\boldsymbol{R_b}, \boldsymbol{R_{1-b}})$; $\boldsymbol{\Phi} \leftarrow \varphi \circ \boldsymbol{\Phi}$
5: $\quad (\boldsymbol{R_b}, \boldsymbol{R_{1-b}}, \varphi) \leftarrow \mathrm{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{R_{1-b}}, \boldsymbol{R_b})$; $\boldsymbol{\Phi} \leftarrow \varphi \circ \boldsymbol{\Phi}$
6: **end for**
7: **return** $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

The Montgomery ladder (Alg. 4) keeps invariant the difference $\boldsymbol{R_1} - \boldsymbol{R_0}$, which is equal to $\boldsymbol{P}$. Equivalently, variable $\boldsymbol{T}$ ($\leftarrow \boldsymbol{R_b} - \boldsymbol{R_{1-b}}$) in Alg. 4 is equal to $(-1)^{1-b}\boldsymbol{P}$. Therefore, at iteration $i = 0$, variable $\boldsymbol{R_b}$ in our version of the Montgomery ladder (Alg. 5) contains at Line 4 the value of

$$\Psi_{\boldsymbol{\Phi}_{2n-2}}\big((-1)^{1-k_0}\boldsymbol{P}\big) \ .$$

This may allow one to explicitly recover the description of $\Psi_{\boldsymbol{\Phi}_{2n-2}}$ and consequently that of $\Psi_{\boldsymbol{\Phi}_{2n-1}}$ as

$$\boldsymbol{\Phi} := \mathrm{desc}(\Psi_{\boldsymbol{\Phi}_{2n-1}}) = \varphi_{2n-1} \circ \mathrm{desc}(\Psi_{\boldsymbol{\Phi}_{2n-2}}) \ .$$

As a result, we may obtain a Montgomery-like algorithm where there is no need to keep track of the current isomorphism: the iADDC and iADDU operations only need to return the points and not the description of the isomorphism of the resulting curve (i.e., parameter φ; *cf.* Sect. 2.2). This is indicated by symbol $'$ on the operator. This variant of the Montgomery ladder also requires that the iADDC and iADDU operations are independent of the curve parameters;[1] this is indicated by the absence of subscript $\boldsymbol{\Phi}$ in the operator. The resulting algorithm is detailed in Alg. 6.

Algorithm 6 Montgomery ladder (with elliptic curve isomorphisms) − Version II

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$ with $k_{n-1} = 1$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

1: $(\boldsymbol{R_1}, \boldsymbol{R_0}) \leftarrow \mathrm{iDBLU}'_{\mathbb{1}}(\boldsymbol{P})$
2: **for** $i = n-2$ down to 1 **do**
3: $\quad b \leftarrow k_i$; $(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}) \leftarrow \mathrm{iADDC}'(\boldsymbol{R_b}, \boldsymbol{R_{1-b}})$
4: $\quad (\boldsymbol{R_b}, \boldsymbol{R_{1-b}}) \leftarrow \mathrm{iADDU}'(\boldsymbol{R_{1-b}}, \boldsymbol{R_b})$
5: **end for**
6: $b \leftarrow k_0$; $(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}) \leftarrow \mathrm{iADDC}'(\boldsymbol{R_b}, \boldsymbol{R_{1-b}})$
7: Recover $\boldsymbol{\Phi} := \mathrm{desc}(\Psi_{\boldsymbol{\Phi}})$ from $\boldsymbol{R_b} = \Psi_{\boldsymbol{\Phi}}\big((-1)^{1-b}\boldsymbol{P}\big)$
8: $(\boldsymbol{R_b}, \boldsymbol{R_{1-b}}, \varphi) \leftarrow \mathrm{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{R_{1-b}}, \boldsymbol{R_b})$; $\boldsymbol{\Phi} \leftarrow \varphi \circ \boldsymbol{\Phi}$
9: **return** $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

Remark 2 A dual version of the Montgomery ladder was given by Joye in [16]. It scans scalar k from the right to the left; see Alg. 8 in Appendix C. For the same reason that k_{n-1} is supposed to be 1 in the Montgomery ladder, we assume that $k_0 = 1$ in its right-to-left variant. So in Alg. 10 (in Appendix C), we start the for-loop at $i = 2$ and initialize $(\boldsymbol{R_{k_1}}, \boldsymbol{R_{1-k_1}})$ with $(\boldsymbol{P}, 3\boldsymbol{P})$. Again this can be done with the new operations. When $k_0 = 0$, point $\boldsymbol{P}$ needs to be subtracted at the end of the computation to get the correct result.

Joye's double-add ladder (Alg. 8) enjoys a feature analogous to the invariant in the Montgomery ladder. It is easily seen that at iteration i in the for-loop of Algorithm 8, temporary variable $\boldsymbol{T}$ contains the value of $2^i\boldsymbol{P}$. Consequently, in Alg. 10, if $|k|_2$ denotes the binary length of k (i.e., $|k|_2 = n$ if $k_{n-1} = 1$) then, at the exit of the for-loop, one has $\boldsymbol{R_0} + \boldsymbol{R_1} = \Psi_{\boldsymbol{\Phi}}(2^{|k|_2}\boldsymbol{P})$ —and $\boldsymbol{R_0} = \Psi_{\boldsymbol{\Phi}}(k\boldsymbol{P})$. This means that if the point $\boldsymbol{Y} := 2^{|k|_2}\boldsymbol{P}$ is known in advance (e.g., precomputed), then the description of $\Psi_{\boldsymbol{\Phi}}$ may be recovered from $\Psi_{\boldsymbol{\Phi}}(\boldsymbol{Y})$, which is obtained by adding $\boldsymbol{R_0}$ and $\boldsymbol{R_1}$ at the end of the for-loop. This last operation should be performed with an iADDU$'$ operation: $(\boldsymbol{R_1}, \boldsymbol{R_0}) \leftarrow \mathrm{iADDU}'(\boldsymbol{R_0}, \boldsymbol{R_1})$ so that $\boldsymbol{R_1}$ contains $\Psi_{\boldsymbol{\Phi}}(\boldsymbol{Y})$ and $\boldsymbol{R_0}$ contains $\Psi_{\boldsymbol{\Phi}}(k\boldsymbol{P})$. Summing up, when $\boldsymbol{Y}$ is known, Algorithm 10 may be im-

[1] As otherwise, at each step of the for-loop, the curve parameters should be updated with the current value of $\boldsymbol{\Phi}$ for evaluating iADDC/iADDU on the current isomorphic elliptic curve .

plemented with the cheaper iADDU′ and iADDC′ operations.

3.4 Variants and extensions

3.4.1 Handling the neutral element $\boldsymbol{O}$

For certain models (including the popular Weierstraß model), the neutral element (i.e., point $\boldsymbol{O}$) needs a special treatment. This can be circumvented by adequately adapting the initialization step. For the classical left-to-right method, assuming that $k_{n-1} = 1$, we can start the for-loop at $i = n-2$ and set $\boldsymbol{R_0} \leftarrow \boldsymbol{P}$ and $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$ in Algs. 2 and 3 at the initialization step. Similarly, for the right-to-left method, assuming that $k_0 = 1$, we can start the for-loop at $i = 1$ and set $\boldsymbol{R_0} \leftarrow \boldsymbol{P}$ and $\boldsymbol{R_1} \leftarrow 2\boldsymbol{P}$ in Alg. 9 (Appendix C). When $k_0 = 0$, we do the same but subtract $\boldsymbol{P}$ at the end of the computation to get the correct result.

3.4.2 Combined operations

A point doubling-addition is the evaluation of $\boldsymbol{R} = 2\boldsymbol{P} + \boldsymbol{Q}$ [22]. This can be done in two steps as $\boldsymbol{T} \leftarrow \boldsymbol{P} + \boldsymbol{Q}$ followed by $\boldsymbol{R} \leftarrow \boldsymbol{P} + \boldsymbol{T}$. If point $\boldsymbol{R}$ is needed together with updated point $\boldsymbol{P}$, this can be carried out with two consecutive applications of the iADDU operation: $(\boldsymbol{T}, \boldsymbol{P}, \varphi_1) \leftarrow \text{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{P}, \boldsymbol{Q})$; $(\boldsymbol{R}, \boldsymbol{P}, \varphi_2) \leftarrow \text{iADDU}_{\varphi_1 \circ \boldsymbol{\Phi}}(\boldsymbol{P}, \boldsymbol{T})$.

Things are slightly more complex if we want to obtain point $\boldsymbol{R}$ together with updated point $\boldsymbol{Q}$ (rather than $\boldsymbol{P}$) at the end of the computation. This can be performed by an evaluation of iADDU followed by an evaluation of iADDC: $(\boldsymbol{T}, \boldsymbol{P}, \varphi_1) \leftarrow \text{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{P}, \boldsymbol{Q})$; $(\boldsymbol{R}, \boldsymbol{Q}, \varphi_2) \leftarrow \text{iADDC}_{\varphi_1 \circ \boldsymbol{\Phi}}(\boldsymbol{T}, \boldsymbol{P})$. We let $(\boldsymbol{R}, \boldsymbol{Q}, \varphi) \leftarrow \text{iDAU}_{\boldsymbol{\Phi}}(\boldsymbol{P}, \boldsymbol{Q})$ denote the corresponding operation—where $\varphi = \varphi_2 \circ \varphi_1$ represents the isomorphism between the initial elliptic curve and the final one; i.e., $\Psi_\varphi \colon E_{\boldsymbol{\Phi}} \xrightarrow{\sim} E_{\boldsymbol{\Phi}'}$ and

$$\text{iDAU}_{\boldsymbol{\Phi}} \colon E_{\boldsymbol{\Phi}} \times E_{\boldsymbol{\Phi}} \to E_{\boldsymbol{\Phi}'} \times E_{\boldsymbol{\Phi}'} \times \mathcal{F},$$
$$(\boldsymbol{P_1}, \boldsymbol{P_2}) \mapsto \big(\Psi_\varphi(2\boldsymbol{P_1} + \boldsymbol{P_2}), \Psi_\varphi(\boldsymbol{P_2}), \varphi\big) \ .$$

As an illustration, the combined iDAU operation immediately gives rise to an alternative implementation of Joye's double-add algorithm by replacing Lines 5 and 6 in Alg. 10 (in Appendix C) with the single line

$$(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}, \varphi) \leftarrow \text{iDAU}_{\boldsymbol{\Phi}}(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}); \ \boldsymbol{\Phi} \leftarrow \varphi \circ \boldsymbol{\Phi}$$

This can be advantageous for certain parameterizations. For example, when applied to the short Weierstraß model with the formulas given in Appendix B the cost per bit trades two (field) multiplications against two squarings. Similar savings can be obtained for our implementation for Montgomery ladder (i.e., Alg. 6) by defining an operation iACAU′ as the combination of operation iADDC′ followed by operation iADDU′. Lines 3 and 4 in Alg. 6 are then replaced with $(\boldsymbol{R_b}, \boldsymbol{R_{1-b}}) \leftarrow \text{iACAU}'(\boldsymbol{R_b}, \boldsymbol{R_{1-b}})$. Again we refer to Appendix B for an illustration to the short Weierstraß model.

3.4.3 Signed-digit representations

Computing the inverse of a point generally comes almost for free on most elliptic curve models. In that case, it can be worth considering signed-digit representations for scalar k in the computation of $\boldsymbol{Q} = k\boldsymbol{P}$.

So, a common strategy to speed up the evaluation of $\boldsymbol{Q} = k\boldsymbol{P}$ on an elliptic curves is to rely on the *non-adjacent form* (NAF) of scalar k [29]. The NAF is a canonical representation using the set of digits $\{-1, 0, 1\}$ to uniquely represent an integer. The NAF has the smallest Hamming weight; on average, only one third of its digits are nonzero [32]. When the cost of point inversion is negligible, it is therefore interesting to input the NAF representation of k, $k = \sum_{i=0}^{n} k'_i \, 2^i$ with $k'_i \in \{-1, 0, 1\}$, and to adapt the scalar multiplication method accordingly. For example, when applied to Algorithm 1, replacing Line 4 with

if $(k'_i \neq 0)$ **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + (-1)^{k'_i} \boldsymbol{R_1}$

the proportion of point additions roughly drops from $n/2$ to $n/3$. Algorithm 7 (in Appendix C) can be adapted similarly by replacing Line 3 with the above modification.

Other scalar multiplication methods can benefit from signed-digit representations. For example, in [33], Rivain devises a Montgomery-like ladder from the *zero-less signed-digit* (ZSD) expansion [31] (see also [27]). Provided it is odd, scalar $k = \sum_{i=0}^{n-1} k'_i \, 2^i$ is then expressed with digits k'_i in $\{-1, 1\}$ (i.e., without the zero digit). The ZSD of k is easily obtained from its binary expansion, $k = \sum_{i=0}^{n-1} k_i \, 2^i$, as $k'_i = (-1)^{1+k_{i+1}}$ for $0 \leqslant i \leqslant n-2$ and $k'_{n-1} = k_{n-1}$.

3.4.4 Higher-radix methods

It is well known that the classical methods can be improved by using a higher-radix 2^e for a suitable $e > 1$ to represent the digits of scalar k [19]. For example, using a signed-digit representation, the average proportion of point additions during the main loop is approximatively $1/(e+1)$ (see [5] for a precise estimate). These methods, however, require more memory resources since $(2^{e-1} - 1)$ points need to be pre-computed and stored.

4 Discussion

Up to now, inversion-free arithmetic on elliptic curves was achieved by resorting to projective representations. So for example over a field $\mathbb{K}$ with $\operatorname{char}\mathbb{K} \neq 2, 3$, an elliptic curve is usually represented using Jacobian coordinates as triples $(X : Y : Z) \neq (0 : 0 : 0)$ satisfying the equation

$$Y^2 = X^3 + aXZ^4 + bZ^6 \ .$$

Any two triples $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ are said *equivalent* if there exists a nonzero element $\theta \in \mathbb{K}$ such that $X_1 = \theta^2 X_2$, $Y_1 = \theta^3 Y_2$ and $Z_1 = \theta Z_2$. To each finite point $\boldsymbol{P_1} = (x_1, y_1)$ on the short Weierstraß curve $y^2 = x^3 + ax + b$ corresponds an equivalent triple $(x_1 Z^2 : y_1 Z^3 : Z)$ for some $Z \in \mathbb{K} \setminus \{0\}$. The point at infinity corresponds to $Z = 0$.

With the notation of Sect. 2.1, it is interesting to note that a finite point $\boldsymbol{P_1} \in E_1$ with Jacobian coordinates $(X_1 : Y_1 : Z_1)$ can be seen as a point (X_1, Y_1) on the isomorphic Weierstraß elliptic curve

$$E_{Z_1} : y^2 = x^3 + (aZ_1{}^4)x + (bZ_1{}^6) \ .$$

This was already observed in [36, § 3.3] when designing multiplicative coordinate blinding. As explained in Sect. 2.1, adding two points with the same Z-coordinate corresponds to adding two points on a same isomorphic elliptic curve. With this correspondence in mind, it is not too surprising that our algorithms inherit the same good performance (and security features) from the algorithms exploiting co-Z arithmetic [24,11].

But there are some differences. The co-Z scalar multiplication algorithms developed in [24] using the Zeckendorf's representation and in [11] with Montgomery-like ladders ensure that the Z-coordinate of the input points is *automatically* shared. For general scalar multiplication algorithms, the co-Z requirement can be ensured by cross-multiplying the points. Specifically, two input Jacobian points $\boldsymbol{P_1} = (X_1 : Y_1 : Z_1)$ and $\boldsymbol{P_2} = (X_2 : Y_2 : Z_2)$ with $Z_1 \neq Z_2$ are updated with the equivalent representations $\boldsymbol{P_1} = (X_1 Z_2{}^2 : Y_1 Z_2{}^3 : Z)$ and $\boldsymbol{P_2} = (X_2 Z_1{}^2 : Y_2 Z_1{}^3 : Z)$ where $Z = Z_1 Z_2$. The approach using curve isomorphisms require input points to be on the same isomorphic curve. When this is not the case, only *one* point needs to be updated. Another difference is when the employed formulas involve curve parameters, like, for example, the point doubling formulas on Weierstraß elliptic curves. In this case, the required curve parameters on the current isomorphic curve need to be known in order to evaluate the corresponding operation. These differences between the two approaches may result in different performance.

Finally, as aforementioned, scalar multiplication algorithms making use of our framework (i.e., curve isomorphisms) are simpler to understand and thus to analyze. This in turn simplifies the implementation of additional tricks to get further efficiency enhancements.

4.1 Performance analysis

It is useful to introduce some notation. In order to evaluate the elliptic curve arithmetic, we let M and S denote the respective cost of a multiplication and of squaring in the definition field. We do not distinguish between general multiplication and multiplication by curve parameters because, even if chosen as small values, curve parameters are likely to be full-size values after the first hop.

4.1.1 Montgomery-like ladders

As detailed in Appendix B, for a Weierstraß elliptic curve over a field $\mathbb{K}$ with $\operatorname{char}\mathbb{K} \neq 2, 3$, the cost of operations iADD/iADDU, iADDC, and iDBL/iDBLU are of $\underline{4\mathsf{M} + 2\mathsf{S}}$, $\underline{5\mathsf{M} + 3\mathsf{S}}$, and $\underline{1\mathsf{M} + 5\mathsf{S}}$, respectively. As a result and because the iADDC and iADDU operations are independent of the curve parameters, similarly to [11], our implementation of the Montgomery ladder (Alg. 6) leads to a cost per bit of only $(5\mathsf{M} + 3\mathsf{S}) + (4\mathsf{M} + 2\mathsf{S}) = 9\mathsf{M} + 5\mathsf{S}$ —or $\underline{8\mathsf{M} + 6\mathsf{S}}$ by using the iACAU′ operation.

The description of the isomorphism in Step 7 of Algorithm 6 is recovered as

$$\Phi = (-1)^{1-b} \frac{\mathrm{x}(\boldsymbol{P})\,\mathrm{y}(\boldsymbol{R_b})}{\mathrm{y}(\boldsymbol{P})\,\mathrm{x}(\boldsymbol{R_b})} \ .$$

This step can be combined with the final step, $\Psi_{\Phi}^{-1}(\boldsymbol{R_0})$, so as to only compute a single inversion in $\mathbb{K}$.

Furthermore, as in [11], it is also possible to devise a signed variant with the same cost per bit (i.e., $\underline{8\mathsf{M} + 6\mathsf{S}}$); see Alg. 11 (in Appendix C). Yet another option is to instead start with the Joye's double-add ladder for the same cost per bit of $\underline{8\mathsf{M} + 6\mathsf{S}}$; see Alg. 12 (in Appendix C) for an illustration. This presents the advantage of not involving point negations but requires the prior knowledge of point $\boldsymbol{Y} := 2^{|k|_2}\boldsymbol{P}$. We note that it is not always necessary to know the whole representation of $\boldsymbol{Y}$. For example, for the short Weierstraß model the knowledge of the ratio $\mathrm{x}(\boldsymbol{Y})/\mathrm{y}(\boldsymbol{Y})$ is enough to recover the description $\boldsymbol{\Phi}$ of the final isomorphism.

4.1.2 Classical methods

The analysis of the classical methods is more interesting as they are not covered with the previous algorithms

making use of the co-Z arithmetic. As presented in Alg. 3, the main loop of the double-and-add algorithm looks like

$(\boldsymbol{R_0}, \boldsymbol{\varphi}) \leftarrow \mathrm{iDBL}_{\boldsymbol{\Phi}}(\boldsymbol{R_0})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
if $(k_i = 1)$ **then**
  $\boldsymbol{R_0} \leftarrow \mathrm{iADD}_{\boldsymbol{\Phi}}(\boldsymbol{R_0}, \Psi_{\Phi}(\boldsymbol{P}))$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
end if

For the short Weierstraß model, the iDBL operation (*cf.* Appendix B) involves curve parameter a, whose current value is[2] $a\,\Phi^4$. Its evaluation requires $1\mathsf{M} + 2\mathsf{S}$. The same is true for point $\boldsymbol{P} = (x, y)$ in the iADD operation; evaluating $\Psi_{\Phi}(\boldsymbol{P}) = (\Phi^2 x, \Phi^3 y)$ requires $3\mathsf{M}+1\mathsf{S}$. Finally, the definition on the new current curve (i.e., Φ) should be kept track of as $\Phi \leftarrow \varphi \cdot \Phi$; this requires $1\mathsf{M}$ for iDBL and $1\mathsf{M}$ for iADD. Putting all together, the double-and-add algorithm has an (average) cost per bit of $((1\mathsf{M}+5\mathsf{S})+(1\mathsf{M}+2\mathsf{S})+1\mathsf{M})+\frac{1}{2}((4\mathsf{M}+2\mathsf{S})+(3\mathsf{M}+1\mathsf{S})+1\mathsf{M}) = 7\mathsf{M}+8.5\mathsf{S}$. If scalar k in the computation of $k\boldsymbol{P}$ is represented as a NAF, then the (average) cost per bit drops to $\underline{5.67\mathsf{M}+8\mathsf{S}}$.

This is already better than our best implementations of the Montgomery ladder (whatever the ratio S/M). But we can do even better by caching the value of curve parameter a. Instead of evaluating it as $a\,\Phi^4$, we update it as $a \leftarrow a\,\varphi^4$ after every iDBL operation and every iADD operation. The iDBL operation computes the value $8L$, which yields the value of φ^4 as $\varphi^4 = 2(8L)$. Hence, updating a as $a \leftarrow a(\varphi^4)$ only costs $1\mathsf{M}$ after an iDBL operation. The iADD operation computes $C = \varphi^2$ (but not φ^4); hence, updating a requires $1\mathsf{M} + 1\mathsf{S}$. As a result, the total (average) cost per bit for Algorithm 3 reduces to $((1\mathsf{M} + 5\mathsf{S}) + 1\mathsf{M} + 1\mathsf{M}) + \frac{1}{2}((4\mathsf{M}+2\mathsf{S})+(3\mathsf{M}+1\mathsf{S})+(1\mathsf{M}+1\mathsf{S})+1\mathsf{M}) = 7.5\mathsf{M}+7\mathsf{S}$, or $\underline{6\mathsf{M}+6.33\mathsf{S}}$ using a NAF-based representation.

4.2 Universality of the technique

Although our technique using curve isomorphisms was vastly illustrated with the Weierstraß model over a large-characteristic field, it can be applied to any elliptic curve model. There are numerous models for representing elliptic curves. The choice of a model is guided by performance issues (target device, number of field operations, memory usage, etc.) and by security concerns (resistance against certain implementation attacks) [12, 6].

In this section, for the sake of illustration, we study two popular models: the Weierstraß form over binary fields [15] and the (twisted) Edwards form over non-binary fields [3,1,14]. Application to other models is handled similarly. We refer the reader to [2] for further models and their efficient arithmetic.

[2] Since, as presented, in the short Weierstraß model the description of the isomorphism comprises only one parameter, we omit the arrow on $\boldsymbol{\varphi}$ and $\boldsymbol{\Phi}$; and $\circ$ becomes $\cdot$ (field multiplication).

4.2.1 Weierstraß model (binary case)

Although curve parameter a_1 can be chosen as $a_1 = 1$ in the Weierstraß model over a field $\mathbb{K}$ of characteristic 2, it should nevertheless be explicitly included since it will pop up through subsequent isomorphisms. So let $\Psi_{\varphi}\colon E_1 \xrightarrow{\sim} E_{\varphi}, (x, y) \mapsto (\varphi^2 x, \varphi^3 y)$ where

$$E_{\varphi}\colon y^2 + (a_1\varphi)xy = x^3 + (a_2\varphi^2)x^2 + (a_6\varphi^6)$$

with $a_1, a_2, a_6 \in \mathbb{K}$, $a_1 a_6 \neq 0$, and $\varphi \in \mathbb{K}^*$.

For the addition of two distinct points $\boldsymbol{P_1} = (x_1, y_1)$, $\boldsymbol{P_2} = (x_2, y_2) \in E_1$, we define $\varphi = x_1 + x_2$. Then, we have $(\widetilde{x_1}, \widetilde{y_1}) := \Psi_{\varphi}(\boldsymbol{P_1}) = (\varphi^2 x_1, \varphi^3 y_1)$ and $(\widetilde{x_3}, \widetilde{y_3}) := \Psi_{\varphi}(\boldsymbol{P_1} + \boldsymbol{P_2})$ where

$$\begin{aligned}(\widetilde{x_3}, \widetilde{y_3}) = \big((y_1 + y_2)(y_1 + y_2 + (a_1\varphi)) + (a_2\varphi^2) + \varphi^3,\\ (y_1 + y_2)(\widetilde{x_3} + \widetilde{x_1}) + (a_1\varphi)\widetilde{x_3} + \widetilde{y_1}\big)\ ,\end{aligned}$$

whose straightforward evaluation yields a cost of $8\mathsf{M} + 1\mathsf{S}$ for iADD/iADDU. Further, noting that $-\boldsymbol{P_2} = (x_2, y_2 + a_1 x_2)$, the evaluation of $\Psi_{\varphi}(\boldsymbol{P_1} - \boldsymbol{P_2})$ needs an extra $4\mathsf{M}$, leading to a cost of $12\mathsf{M}+1\mathsf{S}$ for iADDC. Note that the formula gives for free the new curve parameters, $a_1\varphi$ and $a_2\varphi^2$, which may be needed for subsequent point additions.

For the doubling of $\boldsymbol{P_1} = (x_1, y_1) \in E_1$, we define $\varphi = a_1 x_1$. Then $(\widetilde{x_1}, \widetilde{y_1}) := \Psi_{\varphi}(\boldsymbol{P_1}) = (\varphi^2 x_1, \varphi^3 y_1)$ and $(\widetilde{x_4}, \widetilde{y_4}) := \Psi_{\varphi}(2\boldsymbol{P_1})$ where

$$(\widetilde{x_4}, \widetilde{y_4}) = \big({x_1}^4 + {a_1}^2 a_6, {x_1}^2(\widetilde{x_4} + \widetilde{x_1}) + a_1\widetilde{x_4}(y_1 + \varphi)\big)\ ,$$

which can be evaluated with $5\mathsf{M}+4\mathsf{S}$. We see here that it can be useful to choose the parameterization proposed by Stam [35, Lemma 1]; i.e., taking $a_6 = 1/{a_1}^2$ in E_1. Hence, for a point $\boldsymbol{P_1} \in E_{\Phi}$ the coordinate $\widetilde{x_4}$ is then given by $\widetilde{x_4} = {x_1}^4 + \Phi^8 = (x_1 + \Phi^2)^4$, which involves only squarings.

4.2.2 Twisted Edwards model

We assume $\mathrm{char}\,\mathbb{K} \neq 2$. We consider the general curve equation [1, p. 401], $y^2 + ax^2 = c^2(1 + dx^2y^2)$. But there is a slight complication because an isomorphism of the form $(x, y) \mapsto (\varphi x, \varphi y)$ would require computing an inverse for evaluating the new d-parameter. This is easily handled by replacing d by d^{-1} in the previous curve equation. We therefore consider the family of curves

induced by the isomorphism $\Psi_\varphi\colon E_1 \xrightarrow{\sim} E_\varphi, (x,y) \mapsto (\varphi x, \varphi y)$ where

$$E_\varphi\colon y^2 + ax^2 = (c\varphi)^2 \left(1 + \tfrac{1}{d\varphi^4}\, x^2y^2\right)$$

with $a, c, d \in \mathbb{K}$, $c \neq 0$, ad a non-square in $\mathbb{K}$, and $\varphi \in \mathbb{K}^*$. We present a unified addition formula. Specialized formulas can be obtained in a similar way.

Let $\boldsymbol{P_1} = (x_1, y_1), \boldsymbol{P_2} = (x_2, y_2) \in E_1$. We have $(\widetilde{x_3}, \widetilde{y_3}) := \Psi_\varphi(\boldsymbol{P_1} + \boldsymbol{P_2})$ where

$$\begin{aligned}(\widetilde{x_3}, \widetilde{y_3}) = \big((x_1y_2 + x_2y_1)(d^2 - dx_1x_2y_1y_2),\\ (y_1y_2 - ax_1x_2)(d^2 + dx_1x_2y_1y_2)\big)\end{aligned}$$

with $\varphi = c(d + x_1x_2y_1y_2)(d - x_1x_2y_1y_2)$. Hence, a unified iADD requires $10\mathsf{M} + 1\mathsf{S}$. Updating $\boldsymbol{P_1}$ as $\Psi_\phi(\boldsymbol{P_1})$ requires an extra $2\mathsf{M}$, leading to $12\mathsf{M} + 1\mathsf{S}$ for a unified iADDU. Finally, since $-\boldsymbol{P_2} = (-x_2, y_2)$, a unified iADDC requires an extra $3\mathsf{M}$ from iADD, leading to $13\mathsf{M} + 1\mathsf{S}$. Note that parameter a is unchanged through the isomorphism Ψ_φ and so can be chosen as a small value (for example, $a = -1$), saving $1\mathsf{M}$ in iADD/iADDU/iADDC.

4.3 Security considerations

For cryptographic applications, in addition of being efficient, implementations should also be secure, including against fault attacks [17] and side-channel attacks [23]. As they are generic, the methods developed in this paper can advantageously be combined with existing countermeasures to get implementations that are at the same time efficient and protected. In particular, they nicely combine with the random curve-isomorphism countermeasure of [18]. Replacing $\mathbb{1}$ with [the description of] a random isomorphism at the initialization step of the scalar multiplication algorithm yields an inexpensive protection against certain DPA-type attacks.

5 Conclusion

This paper introduced elliptic curve isomorphisms as a *generic* way to get inversion-free point addition formulas. The resulting scalar multiplication algorithms are efficient in both speed and memory—*speed*: the underlying point addition formulas do not involve field inversion; and *memory*: the point representation leads to a small memory footprint. They generalize and extend the previous co-Z scalar multiplication algorithms (i.e., beyond the Weierstraß model over large prime fields with Jacobian coordinates).

References

1. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: Progress in Cryptology − AFRICACRYPT 2008. LNCS, vol. 5023, pp. 389–405. Springer (2008)
2. Bernstein, D.J., Lange, T.: Explicit-formulas database. `http://www.hyperelliptic.org/EFD/`
3. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: Advances in Cryptology − ASIACRYPT 2007. LNCS, vol. 4833, pp. 29–50. Springer (2007)
4. Chudnovsky, D.V., Chudnovsky, G.V.: Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Advances in Applied Mathematics 7(4), 385–434 (1986)
5. Cohen, H.: Analysis of the sliding window powering algorithm. Journal of Cryptology 18(1), 63–76 (2005)
6. Cohen, H., Frey, G., Avanzi, R., Doche, C., Lange, T., Nguyen, K., Vercauteren, F. (eds.): Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press (2005)
7. De Win, E., Mister, S., Preneel, B., Wiener, M.J.: On the performance of signature schemes based on elliptic curves. In: Algorithmic Number Theory (ANTS-III). LNCS, vol. 1423, pp. 252–266. Springer (1998)
8. FIPS PUB 186-3: Digital signature standard (DSS). Federal Information Processing Standards Publication (Jul 2009)
9. Gordon, D.M.: A survey of fast exponentiation methods. Journal of Algorithms 27(1), 129–146 (1998)
10. Goundar, R.R., Joye, M., Miyaji, A.: Co-Z addition formulæ and binary ladders on elliptic curves. In: Cryptographic Hardware and Embedded Systems − CHES 2010. LNCS, vol. 6225, pp. 65–79. Springer (2010)
11. Goundar, R.R., Joye, M., Miyaji, A., Rivain, M., Venelli, A.: Scalar multiplication on Weierstraß elliptic curves from co-Z arithmetic. Journal of Cryptographic Engineering 1(2), 161–176 (2011)
12. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer (2004)
13. Hisil, H., Costello, C.: Jacobian coordinates on genus 2 curves. In: Advances in Cryptology − ASIACRYPT 2014. LNCS, vol. 8873, pp. 338–357. Springer (2014)
14. Hışıl, H., Wong, K.K.H., Carter, G., Dawson, E.: Twisted Edwards curves revisited. In: Advances in Cryptology − ASIACRYPT 2008. LNCS, vol. 5350, pp. 326–343. Springer (2008)
15. IEEE Std P1363-2000: Standard specifications for public key cryptography. IEEE Computer Society (2000)
16. Joye, M.: Highly regular right-to-left algorithms for scalar multiplication. In: Cryptographic Hardware and Embedded Systems − CHES 2007. LNCS, vol. 4727, pp. 135–147. Springer (2007)
17. Joye, M., Tunstall, M. (eds.): Fault Analysis in Cryptography. Springer (2012)
18. Joye, M., Tymen, C.: Protections against differential analysis for elliptic curve cryptography: An algebraic approach. In: Cryptographic Hardware and Embedded Systems − CHES 2001. LNCS, vol. 2162, pp. 377–390. Springer (2001)
19. Knuth, D.E.: The Art of Computer Programming: Seminumerical Algorithms, vol. 2. Addison-Wesley, 3rd edn. (1997)
20. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48(177), 203–209 (1987)

21. Longa, P., Gebotys, C.H.: Efficient techniques for high-speed elliptic curve cryptography. In: Cryptographic Hardware and Embedded Systems – CHES 2010. LNCS, vol. 6225, pp. 80–94. Springer (2010)
22. Longa, P., Miri, A.: New composite operations and precomputation for elliptic curve cryptosystems over prime fields. In: Public Key Cryptography – PKC 2008. LNCS, vol. 4939, pp. 229–247. Springer (2008)
23. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks. Springer (2007)
24. Meloni, N.: New point addition formulæ for ECC applications. In: Arithmetic of Finite Fields (WAIFI 2007). LNCS, vol. 4547, pp. 189–201. Springer (2007)
25. Menezes, A.J.: Elliptic Curve Public Key Cryptosystems. Kluwer Academic Publishers (1993)
26. Miller, V.S.: Use of elliptic curves in cryptography. In: Advances in Cryptology – CRYPTO '85. LNCS, vol. 218, pp. 417–426. Springer (1985)
27. Möller, B.: Securing elliptic curve point multiplication against side-channel attacks. In: Information Security (ISC 2001). LNCS, vol. 2200, pp. 324–334. Springer (2001)
28. Montgomery, P.L.: Speeding up the Pollard and elliptic curve methods of factorization. Mathematics of Computation 48(177), 243–264 (1987)
29. Morain, F., Olivos, J.: Speeding up the computations on an elliptic curve using addition-subtraction chains. RAIRO Theoretical Informatics and Applications 24(6), 531–543 (1990)
30. NSA names ECC as the exclusive technology for key agreement and digital signature standards for the U.S. government. Press release (March 2, 2005), announced on February 16, 2005 at the RSA conference
31. Okeya, K., Takagi, T.: The width-w NAF method provides small memory and fast elliptic scalar multiplications secure against side channel attacks. In: Topics in Cryptology – CT-RSA 2003. LNCS, vol. 2612, pp. 328–342. Springer (2003)
32. Reitwiesner, G.W.: Binary arithmetic. Advances in Computers 1, 231–308 (1960)
33. Rivain, M.: Fast and regular algorithms for scalar multiplication over elliptic curves. Cryptology ePrint Archive, Report 2011/338 (2011), `http://eprint.iacr.org/`
34. Silverman, J.H.: The Arithmetic of Elliptic Curves. Springer (1986)
35. Stam, M.: On Montgomery-like representationsfor elliptic curves over GF(2^k). In: Public Key Cryptography – PKC 2003. LNCS, vol. 2567, pp. 240–253. Springer (2003)
36. Tunstall, M., Joye, M.: Coordinate blinding over large prime fields. In: Cryptographic Hardware and Embedded Systems – CHES 2010. LNCS, vol. 6225, pp. 443–455. Springer (2010)

A Mathematical Background

Let $\mathbb{K}$ be a field. An elliptic curve E defined over $\mathbb{K}$ is given by the Weierstraß equation

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \ .$$

The set of points on E together with the formal point at infinity $\boldsymbol{O}$ form a group under the chord-and-tangent law [34, Chapter III].

Any two elliptic curves given the Weierstraß equations

$$E\colon y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6$$

and

$$E'\colon y^2 + a'_1xy + a'_3y = x^3 + a'_2x^2 + a'_4x + a'_6$$

are isomorphic over $\mathbb{K}$ if and only if there exist $u, r, s, t \in \mathbb{K}$, $u \neq 0$, such that the linear change of variables

$$(x, y) \leftarrow (u^2x + r, u^3y + u^2sx + t)$$

transforms E into E' [25, Theorem 2.2]. Such a transformation is said admissible and is the only change of variables fixing $\boldsymbol{O}$ and preserving the Weierstraß form. The corresponding curve parameters are related by

$$\begin{aligned}
ua'_1 &= a_1 + 2s\,,\\
u^2a'_2 &= a_2 - sa_1 + 3r - s^2\,,\\
u^3a'_3 &= a_3 + ra_1 + 2t\,,\\
u^4a'_4 &= a_4 - sa_3 + 2ra_2 - (t + rs)a_1 + 3r^2 - 2st\,,\\
u^6a'_6 &= a_6 + ra_4 + r^2a_2 + r^3 - ta_3 - t^2 - rta_1 \ .
\end{aligned}$$

Two settings are commonly used in cryptographic applications (e.g., see [8,15]): elliptic curves over a large prime field $\mathbb{K}$ and non-supersingular elliptic curves over a large binary field $\mathbb{K}$. When the characteristic of $\mathbb{K}$ is not 2 or 3, one can without loss of generality select $a_1 = a_2 = a_3 = 0$. Likewise, when the characteristic of $\mathbb{K}$ is 2 (binary field), provided that the elliptic curve is non-supersingular, one can select $a_1 = 1$ and $a_3 = a_4 = 0$.

B Short Weierstraß Model

Over a field of characteristic not equal to 2 or 3, the short Weierstraß model can be used to represent the points of an elliptic curve E_1.

We define $E_1 : y^2 = x^3 + ax + b$ and use the notation of Sect. 2.1.

B.1 iADD and iADDU operations

From the addition formula (Eq. (2)), letting $\varphi := x_1 - x_2$, we get

$$\begin{aligned}
\varphi^2x_3 &= (y_1 - y_2)^2 - \varphi^2x_1 - \varphi^2x_2 \quad \text{and}\\
\varphi^3y_3 &= (\varphi^2x_1 - \varphi^2x_3)(y_1 - y_2) - \varphi^3y_1 \ .
\end{aligned}$$

That is, given points $\boldsymbol{P_1} = (x_1, y_1)$ and $\boldsymbol{P_2} = (x_2, y_2)$ on E_1, one can easily obtain $\tilde{\boldsymbol{P_3}} := \Psi_\varphi(\boldsymbol{P_1} + \boldsymbol{P_2}) = (\varphi^2x_3, \varphi^3y_3)$ on E_φ without inversion. In more detail, the evaluation of $\tilde{\boldsymbol{P_3}} = (\widetilde{x_3}, \widetilde{y_3})$ can be done as

$$\begin{aligned}
&\varphi = x_1 - x_2,\ \ C = \varphi^2,\ \ W_1 = x_1C,\ \ W_2 = x_2C,\\
&D = (y_1 - y_2)^2,\ \ A_1 = (W_1 - W_2)y_1,\\
&\widetilde{x_3} = D - W_1 - W_2,\ \ \widetilde{y_3} = (W_1 - \widetilde{x_3})(y_1 - y_2) - A_1 \ .
\end{aligned}$$

We let iADD denote this operation; the cost of which amounts to $\underline{4\mathsf{M} + 2\mathsf{S}}$ —where M and S denote the cost of a field multiplication and of a squaring, respectively.

Obtaining $\tilde{\boldsymbol{P_1}} := \Psi_\varphi(\boldsymbol{P_1}) = (\varphi^2x_1, \varphi^3y_1)$ comes from free during the course of the evaluation of $\tilde{\boldsymbol{P_3}}$. Indeed, we immediately have $\tilde{\boldsymbol{P_1}} = (\widetilde{x_1}, \widetilde{y_1})$ with

$$\widetilde{x_1} = W_1 \quad \text{and} \quad \widetilde{y_1} = A_1 \ .$$

We let iADDU denote the operation of getting $\tilde{\boldsymbol{P_3}}$ together with $\tilde{\boldsymbol{P_1}}$; the total cost of which is $\underline{4\mathsf{M} + 2\mathsf{S}}$.

B.2 iADDC operation

Since $-\boldsymbol{P_2} = (x_2, -y_2)$, it follows that $\boldsymbol{P_1} - \boldsymbol{P_2} = (x_3', y_3')$ satisfies

$$\varphi^2 x_3' = (y_1 + y_2)^2 - \varphi^2 x_1 - \varphi^2 x_2 \quad \text{and}$$
$$\varphi^3 y_3' = (\varphi^2 x_1 - \varphi^2 x_3')(y_1 + y_2) - \varphi^3 y_1 \ .$$

Hence, once $\tilde{\boldsymbol{P}}_{\boldsymbol{3}}$ has been evaluated, the evaluation of $\tilde{\boldsymbol{P}}_{\boldsymbol{3}}' := \Psi_\varphi(\boldsymbol{P_1} - \boldsymbol{P_2}) = (\widetilde{x_3'}, \widetilde{y_3'})$ only requires an additional cost of $1\mathsf{M} + 1\mathsf{S}$, since

$$\widetilde{x_3'} = (y_1 + y_2)^2 - W_1 - W_2 \quad \text{and}$$
$$\widetilde{y_3'} = (W_1 - \widetilde{x_3'})(y_1 + y_2) - A_1 \ .$$

We let iADDC denote the corresponding operation, the total cost of which is $\underline{5\mathsf{M} + 3\mathsf{S}}$.

B.3 iDBL and iDBLU operations

From the doubling formula (Eq. (3)), letting now $\varphi := 2y_1$, we get

$$\varphi^2 x_4 = (3x_1^2 + a)^2 - 2\varphi^2 x_1 \quad \text{and}$$
$$\varphi^3 y_4 = (\varphi^2 x_1 - \varphi^2 x_4)(3x_1^2 + a) - \varphi^3 y_1 \ .$$

(Note here that a is the parameter on the current curve.)

That is, given points $\boldsymbol{P_1}$ on E_1, one can obtain $\tilde{\boldsymbol{P}}_{\boldsymbol{4}} := \Psi_\varphi(2\boldsymbol{P_1}) = (\varphi^2 x_4, \varphi^3 y_4)$ on E_φ. In more detail, the evaluation of $\tilde{\boldsymbol{P}}_{\boldsymbol{4}} = (\widetilde{x_4}, \widetilde{y_4})$ can be done as

$$B = {x_1}^2, \ \ E = {y_1}^2, \ \ L = E^2,$$
$$M = 3B + a, \ \ S = 2((x_1 + E)^2 - B - L),$$
$$\widetilde{x_4} = M^2 - 2S, \ \ \widetilde{y_4} = M(S - \widetilde{x_4}) - 8L \ .$$

We let iDBL denote this operation; the cost of which amounts to $\underline{1\mathsf{M} + 5\mathsf{S}}$.

Moreover, obtaining $\tilde{\boldsymbol{P}}_{\boldsymbol{1}} := \Psi_\varphi(\boldsymbol{P_1}) = (\varphi^2 x_1, \varphi^3 y_1)$ comes from free during the course of the evaluation of $\tilde{\boldsymbol{P}}_{\boldsymbol{4}}$. We have $\tilde{\boldsymbol{P}}_{\boldsymbol{1}} = (\widetilde{x_1}, \widetilde{y_1})$ with $\widetilde{x_1} = S$ and $\widetilde{y_1} = 8L$. The corresponding operation is denoted iDBLU.

B.4 iDAU operation and the likes

Let $\boldsymbol{R} = 2\boldsymbol{P_1} + \boldsymbol{P_2}$ on E_1. One can easily obtain $\tilde{\boldsymbol{R}} := \Psi_\varphi(\boldsymbol{R})$ together with $\tilde{\boldsymbol{P}}_{\boldsymbol{1}} := \Psi_\varphi(\boldsymbol{P_1})$ as $(\boldsymbol{T}, \boldsymbol{V}, \varphi_1) = \text{iADDU}(\boldsymbol{P_1}, \boldsymbol{P_2})$ followed by $(\tilde{\boldsymbol{R}}, \tilde{\boldsymbol{P}}_{\boldsymbol{1}}, \varphi_2) = \text{iADDU}(\boldsymbol{V}, \boldsymbol{T})$, and $\varphi = \varphi_1\varphi_2$. A straightforward implementation requires $2\times(4\mathsf{M}+2\mathsf{S})+1\mathsf{M} = 9\mathsf{M}+4\mathsf{S}$. In a way similar to [10,11], two (field) multiplications can be traded against two squarings using the basic identity $2AB = (A+B)^2 - A^2 - B^2$, which leads to a cost of $\underline{7\mathsf{M} + 6\mathsf{S}}$. Explicitly, if $\boldsymbol{P_1} = (x_1, y_1)$ and $\boldsymbol{P_2} = (x_2, y_2)$ then $\tilde{\boldsymbol{P}}_{\boldsymbol{1}} = (\widetilde{x_1}, \widetilde{y_1})$ and $\tilde{\boldsymbol{R}} = (\widetilde{x_R}, \widetilde{y_R})$ on E_φ where

$$C' = (x_1 - x_2)^2, \ \ W_1' = x_1 C', \ \ W_2' = x_2 C',$$
$$D' = (y_1 - y_2)^2, \ \ A_1' = 2y_1(W_1' - W_2'),$$
$$x_3' = D' - W_1' - W_2', \ \ C = (x_3' - W_1')^2,$$
$$y_3' = (y_1 - y_2 + W_1' - x_3')^2 - D' - C - A_1',$$
$$\widetilde{x_1} = 4W_1' C, \ \ W_2 = 4x_3' C, \ \ \widetilde{y_1} = A_1'(\widetilde{x_1} - W_2),$$
$$D = (A_1' - y_3')^2,$$
$$\widetilde{x_R} = D - \widetilde{x_1} - W_2, \ \ \widetilde{y_R} = (\widetilde{x_1} - \widetilde{x_R})(A_1' - y_3') - \widetilde{y_1},$$
$$\varphi = (x_1 - x_2 + W_1' - x_3')^2 - C' - C \ .$$

In the same way, when one wants $\tilde{\boldsymbol{R}} := \Psi_\varphi(\boldsymbol{R})$ together with $\tilde{\boldsymbol{P}}_{\boldsymbol{2}} := \Psi_\varphi(\boldsymbol{P_2})$, the iDAU operation can be evaluated with $\underline{8\mathsf{M} + 7\mathsf{S}}$ (instead of $10\mathsf{M}+5\mathsf{S}$ from a straightforward application of iADDU followed by iADDC). In more detail, with the same notations as above, one can obtain $\tilde{\boldsymbol{P}}_{\boldsymbol{2}} = (\widetilde{x_2}, \widetilde{y_2})$ and $\tilde{\boldsymbol{R}} = (\widetilde{x_R}, \widetilde{y_R})$ on E_φ together with φ as

$$C' = (x_1 - x_2)^2, \ \ W_1' = x_1 C', \ \ W_2' = x_2 C',$$
$$D' = (y_1 - y_2)^2, \ \ A_1' = 2y_1(W_1' - W_2'),$$
$$x_3' = D' - W_1' - W_2', \ \ C = (x_3' - W_1')^2,$$
$$y_3' = (y_1 - y_2 + W_1' - x_3')^2 - D' - C - A_1',$$
$$W_1 = 4x_3' C, \ \ W_2 = 4W_1' C, \ \ A_1 = y_3'(W_1 - W_2),$$
$$D = (y_3' - A_1')^2,$$
$$\widetilde{x_R} = D - W_1 - W_2, \ \ \widetilde{y_R} = (W_1 - \widetilde{x_R})(y_3' - A_1') - A_1,$$
$$\varphi = (x_1 - x_2 + x_3' - W_1')^2 - C' - C \ , \overline{D} = (y_3' + A_1')^2,$$
$$\widetilde{x_2} = \overline{D} - W_1 - W_2, \ \ \widetilde{y_2} = (y_3' + A_1')(W_1 - \widetilde{x_2}) - A_1 \ .$$

When φ does not need to be returned, we see that one squaring is saved. In other words, iDAU′ can be evaluated with $\underline{8\mathsf{M} + 6\mathsf{S}}$.

For completeness, we describe iACAU′ as the combination of operation iADDC′ followed by the operation iADDU′. A straightforward implementation requires $(5\mathsf{M}+3\mathsf{S})+(4\mathsf{M}+2\mathsf{S}) = 9\mathsf{M}+5\mathsf{S}$. However, we can mimic the trick of [10] by adding the squared difference of the x-coordinates as an input to iACAU′. This allows one to trade $1\mathsf{M}$ against $1\mathsf{S}$, yielding a cost of $\underline{8\mathsf{M} + 6\mathsf{S}}$. A detailed implementation follows.

The input is $\boldsymbol{P_1} = (x_1, y_1)$, $\boldsymbol{P_2} = (x_2, y_2)$, and $C = (x_1 - x_2)^2$, and the output is $(\tilde{\boldsymbol{R}}, \tilde{\boldsymbol{S}}, \tilde{C}) = \text{iACAU}'(\boldsymbol{P_1}, \boldsymbol{P_2}, C)$ with $\tilde{\boldsymbol{R}} = (\widetilde{x_R}, \widetilde{y_R})$, $\boldsymbol{S} = (\widetilde{x_S}, \widetilde{y_S})$ and where

$$(\tilde{\boldsymbol{R}}, \tilde{\boldsymbol{S}}) = \text{iADDU}\big(\text{iADDC}(\boldsymbol{P_1}, \boldsymbol{P_2})\big)$$

and $\tilde{C} = (\widetilde{x_R} - \widetilde{x_S})^2$.

$$W_1 \leftarrow x_1 C, \ \ W_2 \leftarrow x_2 C, \ \ D \leftarrow (y_1 - y_2)^2,$$
$$A_1 \leftarrow y_1(W_1 - W_2),$$
$$x_1' \leftarrow D - W_1 - W_2, \ \ y_1' \leftarrow (y_1 - y_2)(W_1 - x_1') - A_1,$$
$$\overline{D} \leftarrow (y_1 + y_2)^2,$$
$$x_2' \leftarrow \overline{D} - W_1 - W_2, \ \ y_2' \leftarrow (y_1 + y_2)(W_1 - x_2') - A_1,$$
$$C' \leftarrow (x_1' - x_2')^2,$$
$$x_4 \leftarrow x_1' C', \ \ W_2' \leftarrow x_2' C', \ \ D' \leftarrow (y_1' - y_2')^2,$$
$$y_4 \leftarrow 2y_1'(x_4 - W_2'),$$
$$x_3 \leftarrow D' - x_4 - W_2', \ \ \overline{C} \leftarrow (x_3 - x_4)^2,$$
$$y_3 \leftarrow (y_1' - y_2' + x_4 - x_3)^2 - D' - \overline{C} - y_4,$$
$$\widetilde{x_R} \leftarrow 4x_3, \ \ \widetilde{y_R} \leftarrow 4y_3, \ \ \widetilde{x_S} \leftarrow 4x_4, \ \ \widetilde{y_S} \leftarrow 4y_4, \ \ \tilde{C} \leftarrow 16\overline{C} \ .$$

Remark 3 The formulas presented in this section make use of the square-multiply replacement technique. On some architectures, depending on the cost of a field addition, this is counterproductive. We refer the reader to [21] for some dedicated optimizations.

C More Scalar Multiplication Algorithms

We describe in this appendix a number of scalar multiplication algorithms.

C.1 Right-to-left scalar multiplication

We review two scalar multiplication algorithms. They both process the bits of scalar k from the right to the left. Algorithm 7 is the classical right-to-left method [19]. Algorithm 8 is a dual version of the Montgomery ladder. It was proposed in [16].

Algorithm 7 Add-and-double method

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

$\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
for $i = 0$ to $n-1$ **do**
  if $(k_i = 1)$ **then** $\boldsymbol{R_0} \leftarrow \boldsymbol{R_0} + \boldsymbol{R_1}$
  $\boldsymbol{R_1} \leftarrow 2\boldsymbol{R_1}$
end for
return $\boldsymbol{R_0}$

Algorithm 8 Joye's double-add ladder

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

$\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$
for $i = 0$ to $n-1$ **do**
  $b \leftarrow k_i$; $\boldsymbol{T} \leftarrow \boldsymbol{R_{1-b}} + \boldsymbol{R_b}$
  $(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}) \leftarrow (\boldsymbol{T} + \boldsymbol{R_{1-b}}, \boldsymbol{T} - \boldsymbol{R_{1-b}})$
end for
return $\boldsymbol{R_0}$

C.2 Scalar multiplication with elliptic curve isomorphisms

Below are some examples of scalar multiplication algorithms when used with the methodology of elliptic curve isomorphisms.

Algorithm 9 Add-and-double method (with elliptic curve isomorphisms)

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

$\boldsymbol{R_0} \leftarrow \boldsymbol{O}$; $\boldsymbol{R_1} \leftarrow \boldsymbol{P}$; $\boldsymbol{\Phi} \leftarrow \mathbb{1}$
for $i = 0$ to $n-1$ **do**
  if $(k_i = 1)$ **then**
    $(\boldsymbol{R_0}, \boldsymbol{R_1}, \boldsymbol{\varphi}) \leftarrow \mathrm{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{R_1}, \boldsymbol{R_0})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
  end if
  $(\boldsymbol{R_1}, \boldsymbol{\varphi}) \leftarrow \mathrm{iDBL}_{\boldsymbol{\Phi}}(\boldsymbol{R_1})$; $\boldsymbol{R_0} \leftarrow \Psi_{\boldsymbol{\varphi}}(\boldsymbol{R_0})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
end for
return $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

Algorithm 10 Joye's double-add ladder (with elliptic curve isomorphisms)

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$ with $k_0 = 1$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

$(\boldsymbol{R_1}, \boldsymbol{R_0}, \boldsymbol{\Phi}) \leftarrow \mathrm{iDBLU}_{\mathbb{1}}(\boldsymbol{P})$
$(\boldsymbol{R_{1-k_1}}, \boldsymbol{R_{k_1}}, \boldsymbol{\varphi}) \leftarrow \mathrm{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{R_0}, \boldsymbol{R_1})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
for $i = 2$ to $n-1$ **do**
  $b \leftarrow k_i$
  $(\boldsymbol{R_b}, \boldsymbol{R_{1-b}}, \boldsymbol{\varphi}) \leftarrow \mathrm{iADDU}_{\boldsymbol{\Phi}}(\boldsymbol{R_{1-b}}, \boldsymbol{R_b})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
  $(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}, \boldsymbol{\varphi}) \leftarrow \mathrm{iADDC}_{\boldsymbol{\Phi}}(\boldsymbol{R_b}, \boldsymbol{R_{1-b}})$; $\boldsymbol{\Phi} \leftarrow \boldsymbol{\varphi} \circ \boldsymbol{\Phi}$
end for
return $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

Algorithm 11 Montgomery ladder (with elliptic curve isomorphisms) − Version III

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$ with $k_{n-1} = k_0 = 1$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

$(\boldsymbol{R_1}, \boldsymbol{R_0}) \leftarrow \mathrm{iDBLU}'_{\mathbb{1}}(\boldsymbol{P})$; $(\boldsymbol{R_0}, \boldsymbol{R_1}) \leftarrow \mathrm{iADDC}'(\boldsymbol{R_1}, \boldsymbol{R_0})$
for $i = n-2$ down to 1 **do**
  $b \leftarrow k_i \oplus k_{i-1}$; $\boldsymbol{R_1} \leftarrow (-1)^b \boldsymbol{R_1}$
  $(\boldsymbol{R_0}, \boldsymbol{R_1}) \leftarrow \mathrm{iDAU}'(\boldsymbol{R_0}, \boldsymbol{R_1})$
end for
$b \leftarrow k_1$
Recover $\boldsymbol{\Phi} := \mathrm{desc}(\Psi_{\boldsymbol{\Phi}})$ from $\boldsymbol{R_1} = \Psi_{\boldsymbol{\Phi}}\big((-1)^{1-b}\boldsymbol{P}\big)$
return $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$

Algorithm 12 Joye's double-add ladder (with elliptic curve isomorphisms) − Version II

Input: $\boldsymbol{P} \in E(\mathbb{F}_q)$ and $k = (k_{n-1}, \ldots, k_0)_2 \in \mathbb{N}$ with $k_0 = 1$
Output: $\boldsymbol{Q} = k\boldsymbol{P}$

$(\boldsymbol{R_1}, \boldsymbol{R_0}) \leftarrow \mathrm{iDBLU}'_{\mathbb{1}}(\boldsymbol{P})$
$b \leftarrow k_1$; $(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}) \leftarrow \mathrm{iADDU}'(\boldsymbol{R_0}, \boldsymbol{R_1})$
for $i = 2$ down to $n-1$ **do**
  $b \leftarrow k_i$; $(\boldsymbol{R_{1-b}}, \boldsymbol{R_b}) \leftarrow \mathrm{iDAU}'(\boldsymbol{R_{1-b}}, \boldsymbol{R_b})$
end for
$(\boldsymbol{R_1}, \boldsymbol{R_0}) \leftarrow \mathrm{iADDU}'(\boldsymbol{R_0}, \boldsymbol{R_1})$
Recover $\boldsymbol{\Phi} := \mathrm{desc}(\Psi_{\boldsymbol{\Phi}})$ from $\boldsymbol{R_1} = \Psi_{\boldsymbol{\Phi}}\big(2^{|k|_2}\boldsymbol{P}\big)$
return $\Psi_{\boldsymbol{\Phi}}^{-1}(\boldsymbol{R_0})$