Twisted Edwards Curves Revisited

Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, and Ed Dawson

Information Security Institute,
Queensland University of Technology, QLD, 4000, Australia
{h.hisil, kk.wong, g.carter, e.dawson}@qut.edu.au

Abstract. This paper introduces fast algorithms for performing group operations on twisted Edwards curves, pushing the recent speed limits of Elliptic Curve Cryptography (ECC) forward in a wide range of applications. Notably, the new addition algorithm uses[1] $8\mathbf{M}$ for suitably selected curve constants. In comparison, the fastest point addition algorithms for (twisted) Edwards curves stated in the literature use $9\mathbf{M}+1\mathbf{S}$. It is also shown that the new addition algorithm can be implemented with four processors dropping the effective cost to $2\mathbf{M}$. This implies an effective speed increase by the full factor of 4 over the sequential case. Our results allow faster implementation of elliptic curve scalar multiplication. In addition, the new point addition algorithm can be used to provide a natural protection from side channel attacks based on simple power analysis (SPA).

Keywords: Efficient elliptic curve arithmetic, unified addition, side channel attack, SPA.

1 Introduction

Edwards curves are drawing increasing attention with their low cost and memory friendly arithmetic in cryptographic applications. Recently, there has been a rapid development of Edwards curves and their use in cryptology. An outline of the previous work that closely relates to twisted Edwards curves is as follows.

- Building on the historical results of Euler and Gauss, Edwards introduced a normal form for elliptic curves and stated the addition law in [13]. These curves are defined by $x^2 + y^2 = c^2 + c^2x^2y^2$.
- Bernstein and Lange introduced a more general version of these curves defined by $x^2 + y^2 = c^2(1 + dx^2y^2)$ or simply $x^2 + y^2 = 1 + dx^2y^2$ together with the first algorithms for computing the group operations on projective coordinates in [5]. For instance, the addition costs $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$ with $c = 1$. Here, and in the rest of this paper, multiplication by a curve constant is denoted by $\mathbf{D}$. With the definitions in [5], these curves are today known as the Edwards curves.

[1] $\mathbf{M}$: Field multiplication, $\mathbf{S}$: Field squaring, $\mathbf{I}$: Field inversion.

– Bernstein and Lange introduced the inverted Edwards coordinates in [6] which reduce the cost for the group operations on Edwards curves. For instance, the addition costs $9\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$.
– Bernstein, Birkner, Joye, Lange, and Peters introduced twisted Edwards curves $ax^2 + y^2 = 1 + dx^2y^2$ in [1], a generalization of Edwards curves.

In this paper, the speed of the arithmetic of twisted Edwards curves is increased by a suitable point representation. The new system is called *extended twisted Edwards coordinates* which adds an auxiliary coordinate to twisted Edwards coordinates. Despite the computational overhead of the additional coordinate, we develop faster ways of performing point addition since the new formulae are composed of polynomial expressions with lower total degrees. We show that the increase in the number of coordinates comes with an increase in the level of parallelism which is exploited for further improvements. We also provide optimizations for the scalar multiplication by mixing extended twisted Edwards coordinates with twisted Edwards coordinates.

The paper is organized as follows. A review of twisted Edwards curves together with some new results is given in Section 2. The new point representation is introduced in Section 3. Several applications of the new achievements are given in Section 4. We draw our conclusions in Section 5.

2 Twisted Edwards Curves

In what follows some terms related to the group law on elliptic curves will be extensively used. In particular, the term *unified* is used to emphasize that point addition formulae remain valid when two input points are identical, see [10, Section 29.1.2]. Therefore, unified addition formulae can be used for point doubling. The term *complete* is used to emphasize that addition formulae are defined for all inputs, see [5]. The term *readdition* is used to emphasize that a point addition has already taken place and some of the previously computed data is cached, see [5]. The term *mixed addition* refers to adding an affine point to a point in some projective representation, see [11]. We adapt the notation from [11], [5], and [1].

Let K be a field of odd characteristic. In [5], Bernstein and Lange introduce Edwards curves defined by $x^2 + y^2 = c^2(1 + dx^2y^2)$ where $c, d \in K$ with $cd(1 - dc^4) \neq 0$. In [1], this form is generalized to twisted Edwards form defined by

$$E_{E,a,d}\colon ax^2 + y^2 = 1 + dx^2y^2$$

where $a, d \in K$ with $ad(a - d) \neq 0$. Edwards curves are then a special case of twisted Edwards curve where a can be rescaled to 1. We next review some formulae regarding the group law on twisted Edwards curves which will be used with slight modifications in Section 3.

Affine addition formulae for twisted Edwards curves in [1] (also see [13], [5]):

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1y_1x_2y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1y_1x_2y_2}\right) = (x_3, y_3). \qquad (1)$$

The point $(0,1)$ is the identity element and the point $(0,-1)$ is of order 2. The negative of a point (x,y) is $(-x,y)$. For further facts such as the resolution of singularities or the points at infinity or the coverage of these curves or the group structure, we refer the reader to the original reference [1]. Also see [13], [5], [4], [6], and [3].

In [5] (where $a=1$) and later in [1], it was proven that if d is not a square in K and a is a square in K then these formulae are complete. In Theorem 1, with reasonable assumptions, we show that it is possible to prevent exceptions in the addition formulae even if d is a square in K or a is not a square in K. We should note that this statement should not be considered as a recommendation for selecting d a square in K and/or a a non-square in K. The desired properties for a and d may change depending on the target application. We will recall Theorem 1 in Section 4.

Theorem 1. *Let K be a field of odd characteristic. Let $E_{E,a,d}$ be a twisted Edwards curve defined over K. Let $P=(x_1,y_1)$ and $Q=(x_2,y_2)$ be points on $E_{E,a,d}$. Assume that P and Q are of odd order. It follows that $1-dx_1x_2y_1y_2\neq 0$ and $1+dx_1x_2y_1y_2\neq 0$.*

Proof. In [5] (where $a=1$) and later in [1], it is proven that the points at infinity (over the extension of K where they exist) are of even order. Assume that P and Q are of odd order. Thus, P, Q and $P+Q$ cannot be the points at infinity. Since the formulae (1) are complete (see [1]) provided that the points at infinity are not involved, the denominators of (1); $1-dx_1x_2y_1y_2$ and $1+dx_1x_2y_1y_2$ must be nonzero. □

Affine doubling formulae (independent of d) for twisted Edwards curves deduced from [1] (also see [5], [2], [3]):

$$2(x_1,y_1)=\left(\frac{2x_1y_1}{y_1^2+ax_1^2},\frac{y_1^2-ax_1^2}{2-y_1^2-ax_1^2}\right)=(x_3,y_3). \tag{2}$$

The exceptional cases and how to prevent them are analogous to formulae (1).

Affine addition formulae (independent of d) for twisted Edwards curves adapted from our preprint [17]: Consider the relations obtained by the curve equation; $ax_1^2+y_1^2=1+dx_1^2y_1^2,\ ax_2^2+y_2^2=1+dx_2^2y_2^2$. After straight forward eliminations, we express a and d in terms of x_1,x_2,y_1,y_2 as follows,

$$a=\frac{(x_1^2y_1^2-x_2^2y_2^2)-y_1^2y_2^2(x_1^2-x_2^2)}{x_1^2x_2^2(y_1^2-y_2^2)},\qquad d=\frac{(x_1^2-x_2^2)-(x_1^2y_2^2-y_1^2x_2^2)}{x_1^2x_2^2(y_1^2-y_2^2)}.$$

Ignoring any exceptions that can be introduced by these rational expressions, substitutions in the addition formulae (1) yield

$$\begin{aligned}x_3&=\frac{x_1y_2+y_1x_2}{1+\frac{(x_1^2-x_2^2)-(x_1^2y_2^2-y_1^2x_2^2)}{x_1^2x_2^2(y_1^2-y_2^2)}x_1y_1x_2y_2}=\frac{x_1x_2(y_1^2-y_2^2)}{x_1y_1-x_2y_2-y_1y_2(x_1y_2-y_1x_2)}\\&=\frac{x_1y_1+x_2y_2}{y_1y_2+\frac{(x_1^2y_1^2-x_2^2y_2^2)-y_1^2y_2^2(x_1^2-x_2^2)}{x_1^2x_2^2(y_1^2-y_2^2)}x_1x_2}=\frac{x_1y_1+x_2y_2}{y_1y_2+ax_1x_2},\end{aligned}$$

$$y_3 = \frac{y_1y_2 - \frac{(x_1^2y_1^2 - x_2^2y_2^2) - y_1^2y_2^2(x_1^2 - x_2^2)}{x_1^2x_2^2(y_1^2 - y_2^2)}x_1x_2}{1 - \frac{(x_1^2 - x_2^2) - (x_1^2y_2^2 - y_1^2x_2^2)}{x_1^2x_2^2(y_1^2 - y_2^2)}x_1y_1x_2y_2} = \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2}.$$

The addition formulae (independent of d) are then as follows,

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_1 + x_2y_2}{y_1y_2 + ax_1x_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2}\right) = (x_3, y_3). \tag{3}$$

The formulae given by (3) produce the same outputs as the addition formulae (1). However, these formulae fail for point doubling. In addition, there are exceptional cases even if d is a not a square in K and a is a square in K. The following theorem states these points explicitly.

Theorem 2. *Let K be a field of odd characteristic. Let $E_{E,a,d}$ be a twisted Edwards curve defined over K. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points on $E_{E,a,d}$. Assume that P is fixed.*

If $x_1 = 0$ or $y_1 = 0$ then $y_1y_2 + ax_1x_2 = 0$ if and only if $Q \in S_x$ where $S_x = \{(y_1/\sqrt{a}, -x_1\sqrt{a}), (-y_1/\sqrt{a}, x_1\sqrt{a})\}$. Similarly, $x_1y_2 - y_1x_2 = 0$ if and only if $Q \in S_y$ where $S_y = \{(x_1, y_1), (-x_1, -y_1)\}$.

Otherwise (i.e. $x_1 \neq 0$ and $y_1 \neq 0$), S_x and S_y are given by

$$S_x = \left\{\left(\frac{y_1}{\sqrt{a}}, -x_1\sqrt{a}\right), \left(-\frac{y_1}{\sqrt{a}}, x_1\sqrt{a}\right), \left(\frac{1}{x_1\sqrt{a\,d}}, -\frac{\sqrt{a}}{y_1\sqrt{d}}\right), \left(-\frac{1}{x_1\sqrt{a\,d}}, \frac{\sqrt{a}}{y_1\sqrt{d}}\right)\right\},$$

$$S_y = \left\{(x_1, y_1), (-x_1, -y_1), \left(\frac{1}{y_1\sqrt{d}}, \frac{1}{x_1\sqrt{d}}\right), \left(-\frac{1}{y_1\sqrt{d}}, -\frac{1}{x_1\sqrt{d}}\right)\right\}.$$

Proof. $\Rightarrow$: The set of all solutions to the system of equations $y_1y_2 + ax_1x_2 = 0, ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2, ax_2^2 + y_2^2 = 1 + dx_2^2y_2^2$ gives S_x. The set of all solutions to the system of equations $x_1y_2 - y_1x_2 = 0, ax_1^2 + y_1^2 = 1 + dx_1^2y_1^2, ax_2^2 + y_2^2 = 1 + dx_2^2y_2^2$ gives S_y. Clearly, all solutions are distinct since $(0, 0)$ is not on the curve.

$\Leftarrow$: Trivial, by substitution. □

Theorem 2 shows that suitable selection of a and d are not enough to eliminate all exceptional cases. Therefore the formulae given by (3) are not complete. Nevertheless, the exceptional inputs have a special property given by the following lemma.

Lemma 1. *Let $K, E_{E,a,d}, P, Q$ be defined as in Theorem 2. Assume that P is a fixed point of odd order. Assume that $Q \in S_x \cup S_y - \{P\}$. Then Q is of even order.*

Proof. The proof is given in Appendix-A. □

We now provide a practical solution to prevent exceptional cases. We will recall Corollary 1 in Section 4.

Corollary 1. *Let $E_{E,a,d}$ be a twisted Edwards curve defined over K. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be points on $E_{E,a,d}$. Assume that P and Q are of odd order with $P \neq Q$. It follows that $y_1y_2 + ax_1x_2 \neq 0$ and $x_1y_2 - y_1x_2 \neq 0$.*

Proof. The proof follows from Theorem 2 and Lemma 1. □

Cryptographic applications involving elliptic curve scalar multiplication typically use points of prime order. If this is the case, Corollary 1 shows that the addition formulae given by (3) are exception-free for distinct input points. Furthermore, extending K cannot introduce any exception. Of course, one can still choose arbitrary points as the input at the expense of exception handling or leave the exceptions unhandled. However, this can lead active attackers to succeed in exceptional point attacks, see [19]. As a general solution, a suitable randomization technique can be used. For various randomization techniques, a comprehensive reference is [10, chapter 29].

The rest of the paper is about cryptographic applications. Therefore, we now further assume that K is finite. In some implementations the ratio $\mathbf{I}/\mathbf{M}$ is quite large. For this reason, a natural strategy is to prevent the frequent use of field inversions and a classical solution is using projective coordinates.

At this stage, consider the homogenous projective coordinates in [1]. In this system, each point (x, y) on $ax^2 + y^2 = 1 + dx^2y^2$ is represented as the triplet $(X : Y : Z)$ which corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. These triplets satisfy the homogenous projective equation

$$(aX^2 + Y^2)Z^2 = Z^4 + dX^2Y^2. \tag{4}$$

The curve defined by (4) is the projective closure of the curve $ax^2 + y^2 = 1 + dx^2y^2$. The identity element is represented by $(0 : 1 : 1)$. The negative of $(X : Y : Z)$ is $(-X : Y : Z)$. For all nonzero $\lambda \in K$, $(X : Y : Z) = (\lambda X : \lambda Y : \lambda Z)$. We denote this system by $\mathcal{E}$. The choice of $\mathcal{E}$ leads to inversion-free very efficient point addition algorithms recently proposed in [1, Section 6].

3 Extended Twisted Edwards Coordinates

To gain more speed, it is convenient to introduce an auxiliary coordinate $t = xy$ to represent a point (x, y) on $ax^2 + y^2 = 1 + dx^2y^2$ in extended affine coordinates (x, y, t). One can pass to the projective representation using the map $(x, y, t) \mapsto (x : y : t : 1)$. For all nonzero $\lambda \in K$, $(X : Y : T : Z) = (\lambda X : \lambda Y : \lambda T : \lambda Z)$ which satisfies (4) and corresponds to the extended affine point $(X/Z, Y/Z, T/Z)$ with $Z \neq 0$. The auxiliary coordinate T has the property $T = XY/Z$. This point representation is named *extended twisted Edwards coordinates* and is denoted by $\mathcal{E}^e$. The identity element is represented by $(0 : 1 : 0 : 1)$. The negative of $(X : Y : T : Z)$ is $(-X : Y : -T : Z)$. Given $(X : Y : Z)$ in $\mathcal{E}$ passing to $\mathcal{E}^e$ can be performed in $3\mathbf{M} + 1\mathbf{S}$ by computing (XZ, YZ, XY, Z^2). Given $(X : Y : T : Z)$ in $\mathcal{E}^e$ passing to $\mathcal{E}$ is cost-free by simply ignoring T.

3.1 Unified Addition in $\mathcal{E}^e$

Given $(X_1 : Y_1 : T_1 : Z_1)$ and $(X_2 : Y_2 : T_2 : Z_2)$ with $Z_1 \neq 0$ and $Z_2 \neq 0$, a unified addition can be performed as $(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) =$

$(X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 + Y_1X_2)(Z_1Z_2 - dT_1T_2), \\ Y_3 &= (Y_1Y_2 - aX_1X_2)(Z_1Z_2 + dT_1T_2), \\ T_3 &= (Y_1Y_2 - aX_1X_2)(X_1Y_2 + Y_1X_2), \\ Z_3 &= (Z_1Z_2 - dT_1T_2)(Z_1Z_2 + dT_1T_2). \end{aligned} \tag{5}$$

These unified formulae are derived from the addition formulae (1). We deduce from [5] and [1] that these formulae are also complete when d is not a square in K and a is a square in K. The operations can be performed with a $9\mathbf{M} + 2\mathbf{D}$ algorithm given by

$$\begin{gathered} A \leftarrow X_1 \cdot X_2, \quad B \leftarrow Y_1 \cdot Y_2, \quad C \leftarrow dT_1 \cdot T_2, \quad D \leftarrow Z_1 \cdot Z_2, \\ E \leftarrow (X_1 + Y_1) \cdot (X_2 + Y_2) - A - B, \quad F \leftarrow D - C, \quad G \leftarrow D + C, \\ H \leftarrow B - aA, \quad X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G. \end{gathered}$$

An $8\mathbf{M}+2\mathbf{D}$ mixed addition algorithm can then be derived by setting $Z_2 = 1$. This means that we are adding $(X_1 : Y_1 : T_1 : Z_1)$ and an extended affine point (x_2, y_2, x_2y_2) which is equally written as $(x_2 : y_2 : x_2y_2 : 1)$.

Choosing curve constants with extremely small sizes or extremely low (or high) hamming weight can be used to eliminate the computational overhead of a field multiplication. For instance see [9], [7], [12]. See also [1, Section 7] for an alternative strategy for the selection of constants. When using $\mathcal{E}^e$ the situation is even better if $a = -1$; we save $1\mathbf{M} + 1\mathbf{D}$ rather than just $1\mathbf{D}$. Consider a twisted Edwards curve given by

$$ax^2 + y^2 = 1 + dx^2y^2.$$

The map $(x, y) \mapsto (x/\sqrt{-a}, y)$ defines the curve,

$$-x^2 + y^2 = 1 + (-d/a)x^2y^2.$$

This map can be constructed if $-a$ is a square in K. It is worth pointing out here that the curve $-x^2 + y^2 = 1 + (-d/a)x^2y^2$ corresponds to the Edwards curve $x^2 + y^2 = 1 + (d/a)x^2y^2$ via the map $(x, y) \mapsto (ix, y)$ if $i \in K$ with $i^2 = -1$. For such curves a $10\mathbf{M} + 1\mathbf{S} + 1\mathbf{D}$ point addition algorithm is given in [4, add-2007-bl-4].

After a renaming of the constant $-d/a$ to d', the point addition on the twisted Edwards curve $-x^2 + y^2 = 1 + d'x^2y^2$ can now be performed with an $8\mathbf{M} + 1\mathbf{D}$ algorithm given by

$$\begin{gathered} A \leftarrow (Y_1 - X_1) \cdot (Y_2 - X_2), \quad B \leftarrow (Y_1 + X_1) \cdot (Y_2 + X_2), \quad C \leftarrow kT_1 \cdot T_2, \\ D \leftarrow 2Z_1 \cdot Z_2, \quad E \leftarrow B - A, \quad F \leftarrow D - C, \quad G \leftarrow D + C, \\ H \leftarrow B + A, \quad X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G \end{gathered}$$

where $k = 2d'$. The optimization that leads to the removal of the extra multiplication is similar to the optimizations in [23] and [4, add-2007-bl-4]. A $7\mathbf{M} + 1\mathbf{D}$ mixed addition algorithm can be derived by setting $Z_2 = 1$.

In the case $a = -1$, we comment that it is possible to save two additions by further extending the coordinates to $(X : Y : T : Z : Y-X : Y+X)$. Alternatively, $(Y_2 - X_2), (Y_2 + X_2), 2Z_2$, and $k = 2d'$ can be cached to save two additions and two multiplications by 2 when performing readdition. We do not claim that these cachings are very useful in practice. On the other hand, a caching of kT_2 leads to readdition in $8\mathbf{M}$ rather than $8\mathbf{M} + 1\mathbf{D}$. This can save time if $\mathbf{D}$ is large. As a consequence, readdition with $Z_2 = 1$ needs $7\mathbf{M}$ rather than $7\mathbf{M} + 1\mathbf{D}$. Similar arguments can be easily extended over the other algorithms in Section 3 when appropriate.

3.2 Dedicated Addition in $\mathcal{E}^e$

Given the representations $(X_1 : Y_1 : T_1 : Z_1)$ and $(X_2 : Y_2 : T_2 : Z_2)$ of distinct points with $Z_1 \neq 0$ and $Z_2 \neq 0$, the point addition can be performed as $(X_1 : Y_1 : T_1 : Z_1) + (X_2 : Y_2 : T_2 : Z_2) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Y_2 - Y_1X_2)(T_1Z_2 + Z_1T_2), \\ Y_3 &= (Y_1Y_2 + aX_1X_2)(T_1Z_2 - Z_1T_2), \\ T_3 &= (T_1Z_2 + Z_1T_2)(T_1Z_2 - Z_1T_2), \\ Z_3 &= (Y_1Y_2 + aX_1X_2)(X_1Y_2 - Y_1X_2). \end{aligned} \tag{6}$$

These formulae are independent of the curve constant d. These formulae are analogous to the addition formulae (3). The operations can be performed with a $9\mathbf{M} + 1\mathbf{D}$ algorithm given by

$$\begin{gathered} A \leftarrow X_1 \cdot X_2, \quad B \leftarrow Y_1 \cdot Y_2, \quad C \leftarrow Z_1 \cdot T_2, \quad D \leftarrow T_1 \cdot Z_2, \\ E \leftarrow D + C, \quad F \leftarrow (X_1 - Y_1) \cdot (X_2 + Y_2) + B - A, \quad G \leftarrow B + aA, \\ H \leftarrow D - C, \quad X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G. \end{gathered}$$

An $8\mathbf{M} + 1\mathbf{D}$ mixed addition algorithm can be derived by setting $Z_2 = 1$.

For the case $a = -1$, the operations can be performed with an $8\mathbf{M}$ algorithm given by

$$\begin{gathered} A \leftarrow (Y_1 - X_1) \cdot (Y_2 + X_2), \quad B \leftarrow (Y_1 + X_1) \cdot (Y_2 - X_2), \quad C \leftarrow 2Z_1 \cdot T_2, \\ D \leftarrow 2\,T_1 \cdot Z_2, \quad E \leftarrow D + C, \quad F \leftarrow B - A, \quad G \leftarrow B + A, \\ H \leftarrow D - C, \quad X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \quad T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G. \end{gathered}$$

A $7\mathbf{M}$ mixed addition algorithm can be derived by setting $Z_2 = 1$. A parallel version of the dedicated addition algorithm is given in Section 4.4 for the case $a = -1$.

3.3 Dedicated Doubling in $\mathcal{E}^e$

Given $(X_1 : Y_1 : T_1 : Z_1)$ with $Z_1 \neq 0$, point doubling can be performed as $2(X_1 : Y_1 : T_1 : Z_1) = (X_3 : Y_3 : T_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 - Y_1^2 - aX_1^2), \\ Y_3 &= (Y_1^2 + aX_1^2)(Y_1^2 - aX_1^2), \\ T_3 &= 2X_1Y_1(Y_1^2 - aX_1^2), \\ Z_3 &= (Y_1^2 + aX_1^2)(2Z_1^2 - Y_1^2 - aX_1^2). \end{aligned} \tag{7}$$

These formulae are independent of the curve constant d. These are essentially the same formulae from [1] plus the formula $T_3 = 2X_1Y_1(Y_1^2 - aX_1^2)$ which increases the number of multiplications needed to compute a point doubling by 1. The operations can be performed with a $4\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ algorithm given by

$$\begin{gathered} A \leftarrow X_1^2, \quad B \leftarrow Y_1^2, \quad C \leftarrow 2Z_1^2, \quad D \leftarrow aA, \quad E \leftarrow (X_1 + Y_1)^2 - A - B, \\ G \leftarrow D + B, \quad F \leftarrow G - C, \quad H \leftarrow D - B, \quad X_3 \leftarrow E \cdot F, \quad Y_3 \leftarrow G \cdot H, \\ T_3 \leftarrow E \cdot H, \quad Z_3 \leftarrow F \cdot G. \end{gathered}$$

This algorithm is similar to $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ point doubling algorithm in [1]. The slowing down from $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ to $4\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ will be remedied in Section 4.3 by mixing $\mathcal{E}^e$ with $\mathcal{E}$. A parallel version of the doubling algorithm is given in Section 4.4 for the case $a = -1$.

3.4 More Formulae

Since we have two different addition formulae for computing x_3 and another two for y_3, it is possible to produce hybrid addition formulae from (1) and (3). The hybrid formulae are given by

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_1 + x_2y_2}{y_1y_2 + ax_1x_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1y_1x_2y_2} \right) = (x_3, y_3), \tag{8}$$

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1y_2 + y_1x_2}{1 + dx_1y_1x_2y_2}, \frac{x_1y_1 - x_2y_2}{x_1y_2 - y_1x_2} \right) = (x_3, y_3). \tag{9}$$

We comment that $\mathcal{E}^e$ analogs of (8) and (9) lead to similar speeds.

4 Applications

We provide further optimizations targeting scalar multiplication operations, nP where n is an integer called the *scalar* and P is the *base point* multiplied by the scalar.

The impact of the new unified addition algorithms in $\mathcal{E}^e$ for preventing side channel attacks is discussed in Section 4.1. Parallel versions of the $8\mathbf{M}+1\mathbf{D}$ unified addition in $\mathcal{E}^e$ are provided in Section 4.2. The speed of scalar multiplication on twisted Edwards curves is increased by mixing $\mathcal{E}^e$ with $\mathcal{E}$ in Section 4.3. A

parallel implementation of fast scalar multiplication in $\mathcal{E}^e$ is explained in Section 4.4. When parallelization is desired the algorithms in Section 4.2 and Section 4.4 help to reduce significantly the effective cost of scalar multiplication. Other applications appear in Section 4.5.

4.1 Defeating SPA Attacks

It is well known that a scalar multiplication algorithm can gain SPA protection when unified additions are used as the only group operation, see [10, Section 29.1.2] for instance. From Section 3.4 we know that the unified addition costs $9\mathbf{M}+2\mathbf{D}$ in $\mathcal{E}^e$. For the case $a = -1$ the cost drops to $8\mathbf{M}+1\mathbf{D}$. Both results are faster than all the other unified addition algorithms known to date. Assuming that $\mathbf{S} = 0.8\mathbf{M}$ and $\mathbf{D} \approx 0$, the $8\mathbf{M} + 1\mathbf{D}$ algorithm is approximately 17.5%, 22.5%, 35%, 50%, 55%, 82.5%, 97.5% faster than the best results in [17], [6], [5], [20], [7], [22], [8], respectively. Note, if $\mathbf{S} = \mathbf{M}$ most speedups will be even more significant. Furthermore, both unified addition algorithms are complete for suitably selected parameters, see section 2 for pointers. The completeness is a stronger property than the unification, see [5, p.2].

Another approach to a protected scalar multiplication is using the Montgomery ladder with Montgomery curves or Kummer surfaces. Montgomery's algorithm for Montgomery curves in [23] use $5\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ per scalar bit. Gaudry/Lubicz algorithm for Kummer surfaces (genus 1, odd characteristic case) in [16] use $3\mathbf{M} + 6\mathbf{S} + 3\mathbf{D}$ per scalar bit. We will only provide comparisons with Montgomery curves in the rest of the paper. Assuming that an optimized protected scalar multiplication algorithm uses 1.2 unified additions per scalar bit, scalar multiplication using the $8\mathbf{M} + 1\mathbf{D}$ algorithm then requires $(8\mathbf{M}+1\mathbf{D}) \times 1.2 = 9.6\mathbf{M}+1.2\mathbf{D}$ per scalar bit. Assuming that $0.67\mathbf{M} \leq \mathbf{S} \leq \mathbf{M}$ and $0 < \mathbf{D} \leq \mathbf{M}$, this will be approximately 6% to 25% slower[2] than Montgomery curves. However, we will show in Section 4.2 that the $8\mathbf{M} + 1\mathbf{D}$ algorithm can be faster on parallel implementations. When designing the parallel algorithms we try exploiting all inherent parallelism. If an $\mathbf{M}$ is performed in parallel with a $\mathbf{D}$ and/or an $\mathbf{S}$ then the cost is counted as an effective $1\mathbf{M}$.

4.2 Defeating SPA Attacks in Parallel Environments

A useful feature of the $8\mathbf{M} + 1\mathbf{D}$ unified addition algorithm is that it is highly parallelizable. In this section, targeting parallel environments, we explain how a protected scalar multiplication using the $8\mathbf{M} + 1\mathbf{D}$ unified addition in $\mathcal{E}^e$ can perform faster than a protected scalar multiplication based on the Montgomery ladder [23]. For details on the ladder algorithm and Montgomery curves, we refer the reader to [23] and [21]. See [18] and [15] for preventing side channel attacks in parallel environments using general elliptic curves.

The Montgomery curve $E_{M,A,B}$ is defined by $By^2 = x^3 + Ax^2 + x$ with $B(A^2 - 4) \neq 0$. Given the projective coordinates of two points $(X_m : Z_m)$ and

[2] The ratios $\mathbf{S}/\mathbf{M}$ and $\mathbf{D}/\mathbf{M}$ are fixed equally for both cases.

$(X_n : Z_n)$ and also $(X_{m-n} : Z_{m-n}) = (X_m : Z_m) - (X_n : Z_n)$; $(X_{m+n}, : Z_{m+n}) = (X_m : Z_m) + (X_n : Z_n)$ is given in [23] by

$$X_{m+n} = Z_{m-n}((X_m - Z_m)(X_n + Z_n) + (X_m + Z_m)(X_n - Z_n))^2,$$
$$Z_{m+n} = X_{m-n}((X_m - Z_m)(X_n + Z_n) - (X_m + Z_m)(X_n - Z_n))^2.$$

Dedicated doubling formulae (which can be faster than the addition) are used to compute X_{2n} and Z_{2n} given in [23] by

$$4X_nZ_n = (X_n + Z_n)^2 - (X_n - Z_n)^2,$$
$$X_{2n} = (X_n + Z_n)^2(X_n - Z_n)^2,$$
$$Z_{2n} = (4X_nZ_n)((X_n - Z_n)^2 + ((A + 2)/4)(4X_nZ_n)).$$

The doubling algorithm uses $2\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ and the addition algorithm uses $3\mathbf{M} + 2\mathbf{S}$ assuming that $Z_{m-n} = 1$. The total cost of a doubling and an addition is then $5\mathbf{M}+4\mathbf{S}+1\mathbf{D}$. In a sequential environment it is convenient to consider the addition and doubling operations as a single composite operation. This approach is given in [4]. To follow the same notation rename

$$[(A + 2)/4, X_{m-n}, Z_{m-n}, X_m, Z_m, X_n, Z_n, X_{2n}, Z_{2n}, X_{m+n}, Z_{m+n}]$$

as $[a24, X_1, Z_1, X_2, Z_2, X_3, Z_3, X_4, Z_4, X_5, Z_5]$. Assuming that $Z_1 = 1$, a $5\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ Montgomery differential-addition-and-doubling algorithm is given in [4, mladd-1987-m] by

$$A \leftarrow X_2 + Z_2, \quad AA \leftarrow A^2, \quad B \leftarrow X_2 - Z_2, \quad BB \leftarrow B^2,$$
$$E \leftarrow AA - BB, \quad C \leftarrow X_3 + Z_3, \quad D \leftarrow X_3 - Z_3, \quad DA \leftarrow D \cdot A,$$
$$CB \leftarrow C \cdot B, \quad X_5 \leftarrow (DA + CB)^2, \quad Z_5 \leftarrow X_1 \cdot (DA - CB)^2,$$
$$X_4 \leftarrow AA \cdot BB, \quad Z_4 \leftarrow E \cdot (BB + a24E).$$

2-Processor Montgomery addition and doubling. In [21], it is observed that the doubling and the addition phases of the Montgomery ladder algorithm can be performed independently. From this, it is clear that one of the processors needs $2\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ and the other needs $3\mathbf{M} + 2\mathbf{S}$ to perform doubling and addition, respectively. Since $3\mathbf{M} + 2\mathbf{S} \geq 2\mathbf{M} + 2\mathbf{S} + 1\mathbf{D}$ we conclude that one round of computing a doubling and an addition can be done in an effective $3\mathbf{M} + 2\mathbf{S}$. Alternatively, we can parallelize the "mladd-1987-m" algorithm in [4]. This approach also yields an effective $3\mathbf{M} + 2\mathbf{S}$. See Appendix-B. The ladder algorithm then uses $3\mathbf{M} + 2\mathbf{S}$ per scalar bit.

2-Processor twisted Edwards $(a = -1)$ **unified addition in** $\mathcal{E}^e$**.** We now investigate the $8\mathbf{M} + 1\mathbf{D}$ unified addition algorithm. We can split the computational task into 9 steps with a full utilization of 2 processors. The unified addition can then be performed with an effective $4\mathbf{M} + 1\mathbf{D}$ algorithm.

Cost	Step	Processor 1	Processor 2
	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 - X_2$
	2	$R_3 \leftarrow Y_1 + X_1$	$R_4 \leftarrow Y_2 + X_2$
1**M**	3	$R_5 \leftarrow R_1 \cdot R_2$	$R_6 \leftarrow R_3 \cdot R_4$
1**M**	4	$R_7 \leftarrow T_1 \cdot T_2$	$R_8 \leftarrow Z_1 \cdot Z_2$
1**D**	5	$R_7 \leftarrow kR_7$	$R_8 \leftarrow 2R_8$
	6	$R_1 \leftarrow R_6 - R_5$	$R_2 \leftarrow R_8 - R_7$
	7	$R_3 \leftarrow R_8 + R_7$	$R_4 \leftarrow R_6 + R_5$
1**M**	8	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_3 \cdot R_4$
1**M**	9	$T_3 \leftarrow R_1 \cdot R_4$	$Z_3 \leftarrow R_2 \cdot R_3$

Assuming that an optimized SPA protected scalar multiplication algorithm uses 1.2 unified additions per scalar bit, we have the cost estimate $(4\mathbf{M}+1\mathbf{D})\times 1.2 = 4.8\mathbf{M}+1.2\mathbf{D}$ per scalar bit (for each of 2 processors). The fastest system is determined by the ratios $\mathbf{S}/\mathbf{M}$ and $\mathbf{D}/\mathbf{M}$. For instance, if $\mathbf{S}=\mathbf{M}$ and $\mathbf{D}\approx 0$ then twisted Edwards ($a=-1$) curves are approximately 4.2% faster than Montgomery curves. On the other hand, using Montgomery curves still seems to be preferable since the ladder algorithm needs less memory and it is not affected by changes in the ratio $\mathbf{D}/\mathbf{M}$. Note also that $\mathbf{S}<\mathbf{M}$ in some applications.

We omit details for the 3-processor case which can be derived with similar approaches.

4-Processor Montgomery addition and doubling. The Montgomery addition and doubling does not *nicely* fit the 4-processor setting. For instance the "mladd-1987-m" algorithm in [4] seems to be *quite* uncompetitive even if we exploit all inherent parallelism. A quick investigation shows that we can perform a doubling and addition in an effective $2\mathbf{M}+2\mathbf{S}$. See Appendix-B. The ladder algorithm then uses $2\mathbf{M}+2\mathbf{S}$ per scalar bit.

4-Processor twisted Edwards ($a=-1$) unified addition in $\mathcal{E}^e$. We can split the computational task into 5 sequential steps among 4 processors. The unified addition can then be performed with an effective $2\mathbf{M}+1\mathbf{D}$ algorithm.

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 - X_2$	$R_3 \leftarrow Y_1 + X_1$	$R_4 \leftarrow Y_2 + X_2$
1**M**	2	$R_5 \leftarrow R_1 \cdot R_2$	$R_6 \leftarrow R_3 \cdot R_4$	$R_7 \leftarrow T_1 \cdot T_2$	$R_8 \leftarrow Z_1 \cdot Z_2$
1**D**	3	*idle*	*idle*	$R_7 \leftarrow kR_7$	$R_8 \leftarrow 2R_8$
	4	$R_1 \leftarrow R_6 - R_5$	$R_2 \leftarrow R_8 - R_7$	$R_3 \leftarrow R_8 + R_7$	$R_4 \leftarrow R_6 + R_5$
1**M**	5	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_3 \cdot R_4$	$T_3 \leftarrow R_1 \cdot R_4$	$Z_3 \leftarrow R_2 \cdot R_3$

Following the assumption from the 2-processor case we have the cost estimate $(2\mathbf{M}+1\mathbf{D})\times 1.2 = 2.4\mathbf{M}+1.2\mathbf{D}$ per scalar bit. If $\mathbf{S}=\mathbf{M}$ and $\mathbf{D}\approx 0$ then twisted Edwards ($a=-1$) curves are approximately 66.7% faster than Montgomery curves. If $\mathbf{S}=0.8\mathbf{M}$ and $\mathbf{D}=0.25\mathbf{M}$ then twisted Edwards ($a=-1$) curves are approximately 33.3% faster. If $\mathbf{S}=0.8\mathbf{M}$ and $\mathbf{D}=\mathbf{M}$ then twisted Edwards ($a=-1$) curves are approximately 5.9% faster.

Assuming $\mathbf{D}\approx 0$, we estimate that a "256-bit, sliding window, 4-NAF" scalar multiplication on twisted Edwards ($a=-1$) curves will require approximately $602\mathbf{M}$ for each of 4 processors, depending on the analysis in [5, Section 5].

Consider the field multiplication operation kR_7 in Step 3. The finite field arithmetic can be implemented building on integer arithmetic. Treating field elements k as a $4n$-bit integer and R_7 as an integer, we fix $k_1, k_2, k_3, k_4 \in \left[0, 2^n - 1\right]$ such that $k = k_0 + 2^n k_1 + 2^{2n} k_2 + 2^{3n} k_3$. Now, kR_7 can be obtained as $k_0R_7 + 2^n(k_1R_7) + 2^{2n}(k_2R_7) + 2^{3n}(k_3R_7)$ by computing k_iR_7 in parallel. The rest of the computation for obtaining kR_7 can be practically negligible (depending on the application). Here, the 3 additions to obtain kR_7 and $R_8 \leftarrow 2R_8$ can be put in a new parallel step. Furthermore if $\#K$ is a special prime allowing very fast modular reduction (such as NIST primes) then the cost of casting the integer kR_7 to K (i.e. the modular reduction) can also be practically negligible (depending on the application). This method leads to a better utilization of processors and can be used for decreasing $\mathbf{D}$. Even if k is of the *full size* (i.e. $\mathbf{D}=\mathbf{M}$), this

technique fixes each k_i to a quarter of the size of k (i.e. $\mathbf{D}$ is close to $0.25\mathbf{M}$ if schoolbook multiplication and fast reduction are being used). Alternatively, fixing n to the word size of the underlying hardware (or maybe to the size of a compiler-supported data type) can be advantageous in some applications. The same method can be adapted to the 2-processor case.

The parallel implementation of $\mathcal{E}^e \leftarrow \mathcal{E}^e + \mathcal{E}^e$ is easier than the Montgomery case because all processors perform similar tasks at each step. In addition, the implementation does not require a special field squaring circuit to gain better timings.

2×2-Processor Montgomery addition and doubling. If the doubling operation is assigned to a team of two processors and the addition operation is assigned to another team of two processors, the $2\mathbf{M}+2\mathbf{S}$ figure can be improved to $2\mathbf{M}+1\mathbf{S}$. See Appendix-B. Here, we make the assumption that the addition-team and the doubling-team work in an unsynchronized fashion and perform the synchronization at the end (of each round); we are not claiming that the implementation of this is easy. Even with this assumption twisted Edwards ($a = -1$) curves can still be faster. For instance, if $\mathbf{S} = \mathbf{M}$ and $\mathbf{D} \approx 0$ then twisted Edwards ($a = -1$) curves are approximately 25% faster than Montgomery curves.

4.3 Fast Scalar Multiplication

In [11], Cohen, Miyaji, and Ono introduced the modified Jacobian coordinates and studied other systems in the literature, namely affine, projective, Jacobian, and Chudnovsky Jacobian coordinates. To gain better timings they proposed a technique of carefully mixing these coordinates. We follow a similar approach. Note, the notations $\mathcal{E}$ and $\mathcal{E}^e$ follow the notation introduced in [11].

On twisted Edwards curves, the speed of scalar multiplications which involve point doublings can be increased by mixing $\mathcal{E}^e$ with $\mathcal{E}$. The following technique replaces (slower) doublings in $\mathcal{E}^e$ with (faster) doublings in $\mathcal{E}$. In the execution of a scalar multiplication:

(i) If a point doubling is followed by another point doubling, use $\mathcal{E} \leftarrow 2\mathcal{E}$.
(ii) If a point doubling is followed by a point addition, use
   1. $\mathcal{E}^e \leftarrow 2\mathcal{E}$ for the point doubling step; followed by,
   2. $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$ for the point addition step.

$\mathcal{E} \leftarrow 2\mathcal{E}$ is performed using $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ doubling algorithm in [1]. The details of the other operations are given below.

$\mathcal{E}^e \leftarrow 2\mathcal{E}$ using (7):

(i) In Section 3 it was noted that passing from $(X : Y : Z)$ to $(X : Y : T : Z)$ (i.e. passing from $\mathcal{E}$ to $\mathcal{E}^e$) can be performed in $3\mathbf{M}+1\mathbf{S}$. From this, it might seem at the first glance that computing $\mathcal{E}^e \leftarrow 2\mathcal{E}$ will more costly than expected. However, the doubling algorithm for (7) does not use the input T_1 and so it can be used for $\mathcal{E}^e \leftarrow 2\mathcal{E}$ without modification.
(ii) Theorem 1 implies that Z_1 and Z_3 are always nonzero if the base point is of odd order. Alternatively, careful selection of a and d also guarantees that Z_1 and Z_3 are always nonzero regardless of the order of the base point, see [1].

$\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$ based on (either) (5) or (6):

(i) Observe that one field multiplication can be saved by not computing T_3. This can be regarded as a remedy to the extra field multiplication which appears in $\mathcal{E}^e \leftarrow 2\mathcal{E}$ while computing T_3.

(ii) If (6) is used (without computing T_3), scalar multiplication is independent of d. Indeed $\mathcal{E} \leftarrow 2\mathcal{E}$ (see [1]) and $\mathcal{E}^e \leftarrow 2\mathcal{E}$ (see Section 3.3) are also independent of d. Formulae (6) save time if $\mathbf{D}$ is large. In addition, Corollary 1 implies that Z_1, Z_2 and Z_3 are always nonzero if the base point is of odd order.

(iii) If (5) is used (without computing T_3), the curve constant d will be involved in the calculations. Using the concept of readdition discussed in Section 3.4, one can also achieve similar performance in comparison to the case of (6). In addition, Theorem 1 implies that Z_1, Z_2 and Z_3 are always nonzero if the base point is of odd order. Alternatively, careful selection of a and d also guarantees that Z_1, Z_2 and Z_3 are always nonzero regardless of the order of the base point, see [1].

In Table 1, a comparison is made for the speeds that can be achieved under different $\mathbf{S}/\mathbf{M}$ and $\mathbf{D}/\mathbf{M}$ scenarios. These estimates are based on the analysis in [5, Section 5]. To gain the best speed, we assume that ($a = -1$). To make the cost estimation easier (without sacrificing the accuracy), we can consider the cost of $\mathcal{E}^e \leftarrow 2\mathcal{E}$ as $3\mathbf{M}+4\mathbf{S}$ by pushing the extra multiplication to the operation count of $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$. In this case, the relevant costs for various additions based on the formulae (6) are as follows. Addition: $8\mathbf{M}$; readdition: $8\mathbf{M}$; readdition with $Z_2 = 1$: $7\mathbf{M}$; mixed addition (i.e. addition with $Z_2 = 1$ reasonably denoted by $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{A}^e$): $7\mathbf{M}$. As a special case, we also include cost estimates for the Montgomery ladder [23] which require $5\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ per scalar bit. The rows are sorted with respect to the column (.8, 0) in descending order. The headers (e.g. (.8, .5)) of columns 2 to 7 fix the ratios $\mathbf{S}/\mathbf{M}$ and $\mathbf{D}/\mathbf{M}$, respectively. (Of course, $\mathbf{D}/\mathbf{M} = 0$ should be regarded as $\mathbf{D}/\mathbf{M} \approx 0$ when it appears.)

Table 1. Cost estimates ($\mathbf{M}$) for fast scalar multiplication, 256-bit. (The Montgomery ladder algorithm for Montgomery curves and "sliding window, 4-NAF" method for Edwards, inverted Edwards, and mixed twisted Edwards coordinates).

System	(1,1)	(.8, 1)	(1, .5)	(.8, .5)	(1, 0)	(.8, 0)
Montgomery Ladder, [23]	2560	2355	2432	2227	2304	2099
Edwards, [5]	2351	2139	2326	2115	2301	2090
Inverted Edwards, [6]	2552	2341	2402	2191	2251	2040
Twisted Edwards ($a = -1$), mixed	2152	1951	2152	1951	2152	1951

It is also convenient to consider $\mathcal{E}^e \leftarrow 2\mathcal{E}$ followed by $\mathcal{E} \leftarrow \mathcal{E}^e + \mathcal{E}^e$ as a single composite operation as $\mathcal{E} \leftarrow 2\mathcal{E} + \mathcal{E}^e$ where $\mathcal{E}^e$ is the base point. See [14] for a similar approach in affine Weierstrass coordinates.

4.4 Fast Scalar Multiplication in Parallel Environments

It is natural to ask whether the speed of the protected scalar multiplication discussed in Section 4.2 can be increased by using a fast dedicated doubling algorithm. Unfortunately mixing $\mathcal{E}^e$ with $\mathcal{E}$ does not seem to be helpful in parallel environments for increasing the speed. Nevertheless, $\mathcal{E}^e \leftarrow 2\mathcal{E}^e$ can be performed with an effective $1\mathbf{M} + 1\mathbf{S}$ algorithm, as follows.

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	*idle*	*idle*	*idle*	$R_1 \leftarrow X_1 + Y_1$
1**S**	2	$R_2 \leftarrow X_1^2$	$R_3 \leftarrow Y_1^2$	$R_4 \leftarrow Z_1^2$	$R_5 \leftarrow R_1^2$
	3	$R_6 \leftarrow R_2 + R_3$	$R_7 \leftarrow R_2 - R_3$	$R_4 \leftarrow 2R_4$	*idle*
	4	*idle*	$R_1 \leftarrow R_4 + R_7$	*idle*	$R_2 \leftarrow R_6 - R_5$
1**M**	5	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_6 \cdot R_7$	$T_3 \leftarrow R_2 \cdot R_6$	$Z_3 \leftarrow R_1 \cdot R_7$

This is essentially the same algorithm as in Section 3.3. It is easy to deduce that the 2-processor point doubling needs an effective $2\mathbf{M} + 2\mathbf{S}$. Point addition $\mathcal{E}^e \leftarrow \mathcal{E}^e + \mathcal{E}^e$ can be performed with an effective $2\mathbf{M}$ algorithm, as follows.

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	$R_1 \leftarrow Y_1 - X_1$	$R_2 \leftarrow Y_2 + X_2$	$R_3 \leftarrow Y_1 + X_1$	$R_4 \leftarrow Y_2 - X_2$
1**M**	2	$R_5 \leftarrow R_1 \cdot R_2$	$R_6 \leftarrow R_3 \cdot R_4$	$R_7 \leftarrow Z_1 \cdot T_2$	$R_8 \leftarrow T_1 \cdot Z_2$
	3	*idle*	*idle*	$R_7 \leftarrow 2R_7$	$R_8 \leftarrow 2R_8$
	4	$R_1 \leftarrow R_8 + R_7$	$R_2 \leftarrow R_6 - R_5$	$R_3 \leftarrow R_6 + R_5$	$R_4 \leftarrow R_8 - R_7$
1**M**	5	$X_3 \leftarrow R_1 \cdot R_2$	$Y_3 \leftarrow R_3 \cdot R_4$	$T_3 \leftarrow R_1 \cdot R_4$	$Z_3 \leftarrow R_2 \cdot R_3$

This is essentially the same algorithm as in Section 3.2. It is easy to deduce that the 2-processor point doubling needs an effective $4\mathbf{M}$. One may prefer using the parallel version of the addition formulae (1) which comes at the expense of multiplication by d. See the discussions about readdition in Section 3.4 and partitioning k in Section 4.2. Assuming $\mathbf{S} = 0.8\mathbf{M}$ and $\mathbf{D} \approx 0$, we estimate that "256-bit, sliding window, 4-NAF" scalar multiplication using $\mathcal{E}^e$ will require approximately $552\mathbf{M}$ for each of 4 processors, depending on the analysis in [5, Section 5].

4.5 Other Applications

Point addition intensive operations bring out the full power of the new addition algorithms. Therefore, we will consider the batch signature verification algorithm in this section.

There is a vast literature on the optimization of special exponentiation techniques. A general references is [10]. An example to the case of scalar multiplication is computing $\sum n_i P_i$ with fixed base point(s) or fixed scalar(s). In [5, Section 7], cost estimations for selected applications about $\sum n_i P_i$ are provided for several curve models. The expected increases in speed for twisted Edwards curves can be deduced from [5] by simply substituting the new operation counts. For instance, the batch signature verification technique in [24] attributed to Bos-Coster is summarized in [5, Section 5] for one variant of the ElGamal signature system. The cost estimates for this operation are given in Table 2 in comparison to Edwards coordinates and inverted Edwards coordinates.

Table 2. Cost estimates (**M**) for batched verification of 100 ElGamal signatures, 256-bit.

System	(1,1)	(.8, 1)	(1, .5)	(.8, .5)	(1, 0)	(.8, 0)
Edwards, [5]	302	297	289	284	276	271
Inverted Edwards, [6]	276	271	264	259	251	246
Twisted Edwards ($a = -1$), $\mathcal{E}^e$	201	201	201	201	201	201

5 Conclusion

In this work, a new point representation $\mathcal{E}^e$ is introduced for twisted Edwards curves. We derive efficient and highly parallel group operations and discuss alternative ways of preventing exceptional cases. We then provide performance estimates and comparisons for different implementation scenarios.

Defeating SPA Attacks. We provide two fast unified addition algorithms which cost $9\mathbf{M} + 2\mathbf{D}$ and $8\mathbf{M} + 1\mathbf{D}$. The latter case is at least 22% faster than all the other unified addition methods stated in the literature. These formulae are even 17.5% faster than our preliminary result in [17].

Defeating SPA Attacks in Parallel Environments. We provide an effective $2\mathbf{M} + 1\mathbf{D}$ unified point addition algorithm on a 4-processor environment. We further showed that twisted Edwards ($a = -1$) curves can be faster up to 66.7% than Montgomery curves in this parallel environment.

Fast Scalar Multiplication. We first handle single-scalar multiplication. We explain how to perform fast scalar multiplication by mixing $\mathcal{E}^e$ with twisted Edwards coordinates $\mathcal{E}$, improving the current relevant literature bounds by approximately 4%-18%. We then point out that multi-scalar multiplications profit even more from the faster point additions in $\mathcal{E}^e$.

Fast Scalar Multiplication in Parallel Environments. We also point to the parallel versions of fast scalar multiplication offering a speed increase by a factor of 3.54 (using 4 processors) over the optimized sequential case.

In conclusion, we have pushed the recent speed limits of Elliptic Curve Cryptography forward in a wide range of applications. Building on our observations we recommend using $\mathcal{E}^e$ (and mixing $\mathcal{E}^e$ with $\mathcal{E}$ when useful) for speeding up the scalar multiplication in several different settings.

Acknowledgement

The authors thank Tanja Lange and anonymous referees for very useful comments and suggestions.

References

1. Bernstein, D.J., Birkner, P., Joye, M., Lange, T., Peters, C.: Twisted Edwards curves. In: AFRICACRYPT 2008. Volume 5023 of LNCS., Springer (2008) 389–405

2. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: Optimizing double-base elliptic-curve single-scalar multiplication. In: INDOCRYPT 2007. Volume 4859 of LNCS., Springer (2007) 167–182
3. Bernstein, D.J., Birkner, P., Lange, T., Peters, C.: ECM using Edwards curves. Cryptology ePrint Archive, Report 2008/016 (2008) `http://eprint.iacr.org/`.
4. Bernstein, D.J., Lange, T.: *Explicit-formulas database* (2007) `http://www.hyperelliptic.org/EFD`.
5. Bernstein, D.J., Lange, T.: Faster addition and doubling on elliptic curves. In: ASIACRYPT 2007. Volume 4833 of LNCS., Springer (2007) 29–50
6. Bernstein, D.J., Lange, T.: Inverted Edwards coordinates. In: AAECC-17. Volume 4851 of LNCS., Springer (2007) 20–27
7. Billet, O., Joye, M.: The Jacobi model of an elliptic curve and side-channel analysis. In: AAECC-15. Volume 2643 of LNCS., Springer (2003) 34–42
8. Brier, E., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: PKC 2002. Volume 2274 of LNCS., Springer (2002) 335–345
9. Brier, E., Joye, M.: Fast point multiplication on elliptic curves through isogenies. In: AAECC-15. Volume 2643 of LNCS., Springer (2003) 43–50
10. Cohen, H., Frey, G., eds.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. CRC Press (2005)
11. Cohen, H., Miyaji, A., Ono, T.: Efficient elliptic curve exponentiation using mixed coordinates. In: ASIACRYPT'98. Volume 1514 of LNCS., Springer (1998) 51–65
12. Doche, C., Icart, T., Kohel, D.R.: Efficient scalar multiplication by isogeny decompositions. In: PKC 2006. Volume 3958 of LNCS., Springer (2006) 191–206
13. Edwards, H.M.: A normal form for elliptic curves. Bulletin of the AMS **44**(3) (2007) 393–422
14. Eisenträger, K., Lauter, K., Montgomery, P.L.: Fast elliptic curve arithmetic and improved Weil pairing evaluation. In: CT-RSA 2003. Volume 2612 of LNCS., Springer (2003) 343–354
15. Fischer, W., Giraud, C., Knudsen, E.W., Seifert, J.P.: Parallel scalar multiplication on general elliptic curves over $\mathbb{F}_p$ hedged against non-differential side-channel attacks. Cryptology ePrint Archive, Report 2002/007 (2002) `http://eprint.iacr.org/`.
16. Gaudry, P., Lubicz, D.: The arithmetic of characteristic 2 Kummer surfaces. Cryptology ePrint Archive, Report 2008/133 (2008) `http://eprint.iacr.org/`.
17. Hisil, H., Wong, K., Carter, G., Dawson, E.: Faster group operations on elliptic curves. Cryptology ePrint Archive, Report 2007/441 (2007) `http://eprint.iacr.org/`.
18. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. In: PKC 2002. Volume 2274 of LNCS., Springer (2002) 280–296
19. Izu, T., Takagi, T.: Exceptional procedure attack on elliptic curve cryptosystems. In: PKC 2003. Volume 2567 of LNCS., Springer (2003) 224–239
20. Joye, M., Quisquater, J.J.: Hessian elliptic curves and side-channel attacks. In: CHES 2001. Volume 2162 of LNCS., Springer (2001) 402–410
21. Joye, M., Yen, S.M.: The Montgomery powering ladder. In: CHES 2002. Volume 2523 of LNCS., Springer (2003) 291–302
22. Liardet, P.Y., Smart, N.P.: Preventing SPA/DPA in ECC systems using the Jacobi form. In: CHES 2001. Volume 2162 of LNCS., Springer (2001) 391–401
23. Montgomery, P.L.: Speeding the Pollard and elliptic curve methods of factorization. Mathematics of Computation **48**(177) (1987) 243–264
24. de Rooij, P.: Efficient exponentiation using precomputation and vector addition chains. In: EUROCRYPT'94. (1994) 389–399

A Proof of Lemma 1

Proof. Note that the points at infinity are of even order, see [1]. Assume that $P = (x_1, y_1)$ is of odd order. Thus, P is not one of the points at infinity. Assume that $Q \in S_x \cup S_y - \{P\}$. If Q were one of the points at infinity it would have even order and the claim follows. Note also that $P \neq Q$ and $P \neq -Q$ since $P, -P \notin S_x \cup S_y - \{P\}$. Instead of a further case by case analysis on $S_x \cup S_y - \{P\}$, we will prove the lemma with a general approach. The proof has two parts.

In the first part we will prove that all points in S_x are of even order. Assume that $Q = (x_2, y_2)$ is an element of S_x. By Theorem 2, $ax_1x_2 + y_1y_2 = 0$.

Suppose that $x_1 = 0$. Since P is of odd order $P \neq (0, -1)$ and consequently $P = (0, 1)$. By Theorem 2, $Q = (\pm 1/\sqrt{a}, 0)$. Since $4(\pm 1/\sqrt{a}, 0) = (0, 1)$, Q is of even order as desired.

Assume from now on that $x_1 \neq 0$. We can write $x_2 = -y_1y_2/(ax_1)$ since x_1 is nonzero. Let $M = 2P$ and $N = 2Q$. Since P is of odd order, so is M. Therefore, M is not one of the points at infinity. We can assume that N is not one of the points at infinity; for otherwise Q is of even order as desired. Using the relation $x_2 = -y_1y_2/(ax_1)$ and formula (3) for computing x_3 we get

$$x(N) = \frac{2x_2y_2}{y_2^2 + ax_2^2} = \frac{2(-y_1y_2/(ax_1))y_2}{y_2^2 + a(-y_1y_2/(ax_1))^2} = -\frac{2x_1y_1}{y_1^2 + ax_1^2} = -x(M).$$

The denominators $y_1^2 + ax_1^2$ and $y_2^2 + ax_2^2$ must be nonzero since M and N are not points at infinity. By the curve definition we have

$$y = \pm\sqrt{(1 - ax^2)/(1 - dx^2)}.$$

So $y(M) = \pm y(N)$ since $|x(M)| = |x(N)|$.

$y(M) = -y(N)$ implies that $M - N = (0, -1)$, a point of order 2. Then $2(M - N) = 2(2P - 2Q) = 4(P - Q) = (0, 1)$. So $P - Q$ is a point of order 4.

$y(M) = y(N)$ implies that $M + N = (0, 1)$, the identity. Then $M + N = 2P + 2Q = 2(P + Q) = (0, 1)$. So $P + Q$ is a point of order 2 since $P \neq -Q$.

In conclusion, we have $P \pm Q$ of even order for all situations. Since P is of odd order, $Q \in S_x$ must be of even order.

In the second part of the proof we will prove that all points in $S_y - \{P\}$ are of even order. Assume that $Q = (x_2, y_2)$ is an element of $S_y - \{P\}$. By Theorem 2, $x_1y_2 - y_1x_2 = 0$.

Suppose that $x_1 = 0$. Since P is of odd order $P \neq (0, -1)$ and consequently $P = (0, 1)$. By Theorem 2, $Q = (0, -1)$. Then Q is of even order as desired.

Assume from now on that $x_1 \neq 0$. We can write $y_2 = y_1x_2/x_1$ since x_1 is nonzero. Let $M = 2P$ and $N = 2Q$. Since P is of odd order, so is M. Therefore, M is not one of the points at infinity. We can assume that N is not one of the points at infinity; for otherwise Q is of even order as desired. Using the relation $y_2 = y_1x_2/x_1$ and formula (3) for computing x_3 we get

$$x(N) = \frac{2x_2y_2}{y_2^2 + ax_2^2} = \frac{2x_2(y_1x_2/x_1)}{(y_1x_2/x_1)^2 + ax_2^2} = \frac{2x_1y_1}{y_1^2 + ax_1^2} = x(M).$$

The denominators $y_1^2 + ax_1^2$ and $y_2^2 + ax_2^2$ must be nonzero since M and N are not points at infinity. By the curve definition $y(M) = \pm y(N)$ since $|x(M)| = |x(N)|$.

$y(M) = -y(N)$ implies that $M + N = (0, -1)$, a point of order 2. Then $2(M + N) = 2(2P + 2Q) = 4(P + Q) = (0, 1)$. So $P + Q$ is a point of order 4.

$y(M) = y(N)$ implies that $M - N = (0, 1)$, the identity. Then $M - N = 2P - 2Q = 2(P - Q) = (0, 1)$. So $P - Q$ is a point of order 2 since $P \neq Q$.

In conclusion, we have $P \pm Q$ of even order for all situations. Since P is of odd order, $Q \in S_y - \{P\}$ must be of even order.

In summary, all points in $S_x \cup S_y - \{P\}$ are of even order provided that P is of odd order. □

B Parallel algorithms

This appendix contains parallel algorithms for Montgomery addition and doubling discussed in Section 4.2.

2-processor Montgomery differential-addition-and-doubling. Effective $3\mathbf{M} + 2\mathbf{S}$, assumption $Z_1 = 1$, adapted from [4, mladd-1987-m].

Cost	Step	Processor 1	Processor 2
	1	$R_1 \leftarrow X_2 + Z_2$	$R_2 \leftarrow X_2 - Z_2$
	2	$R_3 \leftarrow X_3 + Z_3$	$R_4 \leftarrow X_3 - Z_3$
1**S**	3	$R_5 \leftarrow R_1^2$	$R_6 \leftarrow R_2^2$
	4	$R_7 \leftarrow R_5 - R_6$	*idle*
1**M**	5	$R_1 \leftarrow R_1 \cdot R_4$	$R_2 \leftarrow R_2 \cdot R_3$
	6	$R_3 \leftarrow R_1 + R_2$	$R_4 \leftarrow R_1 - R_2$
1**S**	7	$X_5 \leftarrow R_3^2$	$R_2 \leftarrow R_4^2$
1**M**	8	$R_8 \leftarrow a24R_7$	$X_4 \leftarrow R_5 \cdot R_6$
	9	$R_8 \leftarrow R_6 + R_8$	*idle*
1**M**	10	$Z_4 \leftarrow R_7 \cdot R_8$	$Z_5 \leftarrow X_1 \cdot R_2$

4-processor Montgomery differential-addition-and-doubling. Effective $2\mathbf{M} + 2\mathbf{S}$, adapted from [4, mladd-1987-m].

Cost	Step	Processor 1	Processor 2	Processor 3	Processor 4
	1	$R_1 \leftarrow X_2 + Z_2$	$R_2 \leftarrow X_2 - Z_2$	$R_3 \leftarrow X_3 + Z_3$	$R_4 \leftarrow X_3 - Z_3$
1**S**	2	$R_5 \leftarrow R_1^2$	$R_6 \leftarrow R_2^2$	*idle*	*idle*
	3	$R_7 \leftarrow R_5 - R_6$	*idle*	*idle*	*idle*
1**M**	4	$R_1 \leftarrow R_1 \cdot R_4$	$R_2 \leftarrow R_2 \cdot R_3$	$R_8 \leftarrow a24R_7$	*idle*
	5	$R_3 \leftarrow R_1 + R_2$	$R_4 \leftarrow R_1 - R_2$	$R_8 \leftarrow R_6 + R_8$	*idle*
1**S**	6	$X_5 \leftarrow R_3^2$	$R_2 \leftarrow R_4^2$	*idle*	*idle*
1**M**	7	$Z_5 \leftarrow X_1 \cdot R_2$	$X_4 \leftarrow R_5 \cdot R_6$	$Z_4 \leftarrow R_7 \cdot R_8$	*idle*

2×2-processor Montgomery differential-addition and Montgomery doubling. Effective $2\mathbf{M} + 1\mathbf{S}$. Using the notation from [4].

2-processor Montgomery Addition

Cost	Step	Processor 1	Processor 2
	1	$R_0 \leftarrow X_2 - Z_2$	$R_1 \leftarrow X_3 + Z_3$
	2	$R_2 \leftarrow X_2 + Z_2$	$R_3 \leftarrow X_3 - Z_3$
1**M**	3	$R_0 \leftarrow R_0 \cdot R_1$	$R_2 \leftarrow R_2 \cdot R_3$
	4	$R_1 \leftarrow R_0 + R_2$	$R_3 \leftarrow R_0 - R_2$
1**S**	5	$R_0 \leftarrow R_1^2$	$R_2 \leftarrow R_3^2$
1**M**	6	$X_5 \leftarrow Z_1 \cdot R_0$	$Z_5 \leftarrow X_1 \cdot R_2$

2-processor Montgomery Doubling

Cost	Step	Processor 1	Processor 2
	1	$R_4 \leftarrow X_2 + Z_2$	$R_5 \leftarrow X_2 - Z_2$
1**S**	2	$R_4 \leftarrow R_4^2$	$R_5 \leftarrow R_5^2$
	3	$R_6 \leftarrow R_4 - R_5$	*idle*
1**D**	4	$R_7 \leftarrow a24R_6$	*idle*
	5	$R_7 \leftarrow R_5 + R_7$	*idle*
1**M**	6	$X_4 \leftarrow R_4 \cdot R_5$	$Z_4 \leftarrow R_6 \cdot R_7$

The effective cost of addition is $2\mathbf{M} + 1\mathbf{S}$ (even if $Z_1 = 1$). The effective cost of doubling is $1\mathbf{M}+1\mathbf{S}+1\mathbf{D}$. Since $2\mathbf{M}+1\mathbf{S} \geq 1\mathbf{M}+1\mathbf{S}+1\mathbf{D}$ the overall effective cost is $2\mathbf{M} + 1\mathbf{S}$ depending on the assumption in Section 4.2.