A version of this paper appears in ACISP 2009, LNCS Vol. 5594, pp. 452–468. C. Boyd and J. Gonzalez Nieto (Eds.), Springer-Verlag, 2009. This is an updated version. 5 November 2009.

Jacobi Quartic Curves Revisited

Huseyin Hisil, Kenneth Koon-Ho Wong, Gary Carter, 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 provides new results about efficient arithmetic on (extended) Jacobi quartic form elliptic curves $y^2 = dx^4 + 2ax^2 + 1$. Recent works have shown that arithmetic on an elliptic curve in Jacobi quartic form can be performed solidly faster than the corresponding operations in Weierstrass form. These proposals use up to 7 coordinates to represent a single point. However, fast scalar multiplication algorithms based on windowing techniques, precompute and store several points which require more space than what it takes with 3 coordinates. Also note that some of these proposals require $d = 1$ for full speed. Unfortunately, elliptic curves having 2-times-a-prime number of points, cannot be written in extended Jacobi quartic form if $d = 1$. Even worse the contemporary formulae may fail to output correct coordinates for some inputs. This paper provides improved speeds using fewer coordinates without causing the above mentioned problems. For instance, our proposed point doubling algorithm takes only 2 multiplications, 5 squarings, and no multiplication with curve constants when d is arbitrary and $a = \pm 1/2$.

Keywords: Efficient elliptic curve arithmetic, scalar multiplication, Jacobi model of elliptic curves.

1 Introduction

Cryptology as a computational science has been a driving force behind the arithmetic of elliptic curves in the past few decades. The demand for more speed led researchers to propose new formulae/algorithms/point-representations for several different elliptic curve models. However, the speed limitation for performing arithmetic on elliptic curves —like many other computational problems— is still an open question.

The historical roots of the topic dates back to late 18^{th} and early 19^{th} century: the time of Euler, Abel and Jacobi. An outline of the previous work restricted to the efficient arithmetic on Jacobi quartic curves is as follows. Chudnovsky and Chudnovsky [8] introduced the first inversion-free algorithms for performing group operations using a weighted projective point representation. Billet and Joye [7] used extended Jacobi quartic form for protection against side-channel attacks with a *unified* addition speed record for that time of $10\mathbf{M} + 3\mathbf{S} + 3\mathbf{D}$ for arbitrary a and d. In this paper, $\mathbf{M}$ stands for a field multiplication; $\mathbf{S}$ for a field squaring; $\mathbf{D}$ for a multiplication by a curve constant; $\mathbf{I}$ for a field inversion. This notation is borrowed from [4]. Duquesne [11] improved this operation count by $1\mathbf{M} + 1\mathbf{S}$ with a variant of Billet/Joye unified point addition algorithm. Duquesne's method converts the base point in weighted projective coordinates to a new point representation with 4 coordinates, performs the scalar multiplication within the new coordinate system, and outputs the final result in original weighted projective coordinates. Duquesne's improvement was followed by additional results in [3], [17], and [19]. However, the latter proposals tend to use more space —up to 7 coordinates per point— despite their speed advantage. Further disadvantages have already been mentioned in the abstract. We will extend our discussion on these aspects in §2.

In this paper, we carefully optimize the arithmetic of (extended) Jacobi quartic form targeting more efficient scalar multiplication operations. Our proposal performs faster and uses less space than [11], [3], [17], and [19].

The paper is organized as follows. A review of extended Jacobi quartic form is given in §2. Efficient algorithms/formulae/point-representations are introduced in §3, §4, and §5. Implementation timings are given in §6. We draw our conclusions in §7.

2 Background

This section gives definitions for extended Jacobi quartic curves. Some of the results involved in this section are analogous to our earlier work [18].

Let K be a field with $\mathrm{char}(K) \neq 2$. An extended Jacobi quartic form elliptic curve over K is defined by

$$E_{J,d,a} \colon y^2 = dx^4 + 2ax^2 + 1$$

where $a, d \in K$ with $\Delta = 256d(a^2 - d)^2 \neq 0$. This extended model covers more elliptic curves than Jacobi model $E_{J,k^2,-(k^2+1)/2}$. In particular, Billet and Joye [7] remark that any elliptic curve, E/K, can be written as $E_{J,d,a}/K$ if $E(K)$ has an element of order 2 and provide the transformations between a Weierstrass elliptic curve $y^2 = x^3 + ax + b$ of even order and a weighted projective Jacobi quartic curve. Throughout the paper, we work with arbitrary a and d. The j-invariant of this curve is given by $64d^{-1}(a^2 - d)^{-2}(a^2 + 3d)^3 \in K$.

We first review the most popular addition formulae. Let $(x_1, y_1), (x_2, y_2) \in E_{J,d,a}(K)$. Assuming that (x_3, y_3) is defined we have $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 - dx_1^2x_2^2}, \tag{1}$$

$$y_3 = \frac{(y_1y_2 + 2ax_1x_2)(1 + dx_1^2x_2^2) + 2dx_1x_2(x_1^2 + x_2^2)}{(1 - dx_1^2x_2^2)^2}. \tag{2}$$

A special case of (1) appears in one of Euler's historical works [13]. Formula (1) and (2) is adapted from [7]. With this selection of the algebraic expressions, the identity element becomes the point $(0, 1)$. The negative of a point (x, y) is $(-x, y)$. The point $(0, -1)$ is of order 2. $E_{J,d,a}$ is non-singular provided that $\Delta \neq 0$. On the other hand, the point at infinity $(0\colon 1\colon 0)$ on the projective closure of $E_{J,d,a}$, $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$, is singular. Resolving the singularity in two consecutive *blowup*s yields two points of order 2. These points are defined over K if and only if d is a square in K. The following lemma shows that formulae (1) and (2) are *complete* if d is not a square in K. The term *complete* is used to emphasize that addition formulae are defined for all inputs, cf. [4].

Lemma 2.1. *Let $d, x_1, x_2 \in K$. Assume that d is non-square. Then $dx_1^2x_2^2 \neq 1$.*

Proof. Suppose that $dx_1^2x_2^2 = 1$. So $d, x_1, x_2 \neq 0$. But then $d = (1/(x_1x_2))^2$. □

Lemma 2.1 is similar to Theorem 3.3 of [4]. In the case of $E_{J,d,a}$, the statement of the lemma and its proof are shorter since the curve equation is not involved.

We can prevent $dx_1^2x_2^2 = 1$ even if d is a square in K. Lemma 2.2 states a sufficient condition. This lemma and its proof are similar to Corollary 1 in [18].

Lemma 2.2. *Let $a, d, x_1, y_1, x_2, y_2 \in K$ such that $d(a^2 - d) \neq 0$. Assume that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points of odd order on $E_{J,d,a}$. Then $1 - dx_1^2x_2^2 \neq 0$.*

We provide a proof in Appendix A. By elementary group theory, multiplying a point of even order with a suitable power of 2 yields a point of odd order.

More formulae. Jacobi elliptic functions give rise to many addition formulae, cf. [24], [8], and [7]. The original reference is [20]. The following formulae are equivalent via the algebraic relations $\mathrm{sn}(\cdot)^2 + \mathrm{cn}(\cdot)^2 = 1$ and $k^2\mathrm{sn}(\cdot)^2 + \mathrm{dn}(\cdot)^2 = 1$.

$$\begin{aligned} \mathrm{sn}(u_1+u_2) &= \frac{\mathrm{sn}(u_1)\mathrm{cn}(u_2)\mathrm{dn}(u_2) + \mathrm{cn}(u_1)\mathrm{dn}(u_1)\mathrm{sn}(u_2)}{1 - k^2\mathrm{sn}(u_1)^2\mathrm{sn}(u_2)^2} &\quad (3)\\ &= \frac{\mathrm{sn}(u_1)^2 - \mathrm{sn}(u_2)^2}{\mathrm{sn}(u_1)\mathrm{cn}(u_2)\mathrm{dn}(u_2) - \mathrm{cn}(u_1)\mathrm{dn}(u_1)\mathrm{sn}(u_2)}. &\quad (4)\end{aligned}$$

To see the equivalence, either take the arithmetic cross product and write

$$\mathrm{sn}(u_1)^2\mathrm{cn}(u_2)^2\mathrm{dn}(u_2)^2 - \mathrm{cn}(u_1)^2\mathrm{dn}(u_1)^2\mathrm{sn}(u_2)^2 =$$

$$(\mathrm{sn}(u_1)^2 - \mathrm{sn}(u_2)^2)(1 - k^2\mathrm{sn}(u_1)^2\mathrm{sn}(u_2)^2)$$

—the rest follows when $\mathrm{cn}(\cdot)^2$ is replaced with $1 - \mathrm{sn}(\cdot)^2$ and $\mathrm{dn}(\cdot)^2$ is replaced with $1 - k^2\mathrm{sn}(\cdot)$— or simply run the Maple script

```
> simplify(expand(
      JacobiSN(u1 + u2, k) - (
          (JacobiSN(u1, k)*JacobiCN(u2, k)*JacobiDN(u2, k) +
          JacobiSN(u2, k)*JacobiCN(u1, k)*JacobiDN(u1, k))/
          (1 - k^2*JacobiSN(u1, k)^2*JacobiSN(u2, k)^2))));
0

> simplify(expand(
      JacobiSN(u1 + u2, k) - (
          (JacobiSN(u1, k)^2 - JacobiSN(u2, k)^2)/
          (JacobiSN(u1, k)*JacobiCN(u2, k)*JacobiDN(u2, k) -
          JacobiSN(u2, k)*JacobiCN(u1, k)*JacobiDN(u1, k)))));
0
```

As pointed out in [7], (3) is analogous to (1) via the relation $(\mathrm{sn}(u_i), \mathrm{cn}(u_i)\mathrm{dn}(u_i)) = (x_i, y_i)$ where (x_i, y_i) satisfies $E_{J,k^2,-(k^2+1)/2}$. Similarly, the analog of (4) is given by

$$x_3 = \frac{x_1^2 - x_2^2}{x_1y_2 - y_1x_2}. \quad (5)$$

This formula also holds for the extended curve $E_{J,d,a}$. This formula fails if $(x_1, y_1) = (x_2, y_2)$. This formula is independent of a and d. There are several other ways to derive (5). For instance, one may use the strategy applied in [18] for the derivation of dedicated addition formulae on twisted Edwards curves. Formula (5) is of minimal total degree. Therefore, the Monagan/Pearce minimal total degree algorithm in [23] can be used to derive this same formula (or maybe an alternative formula of same total degree if there exists one) departing from (1) or any other valid formula.

Lemma 2.2 can be rewritten for (5) as follows. The proof is similar to the proof of Lemma 2.2.

Lemma 2.3. *Let $a, d, x_1, y_1, x_2, y_2 \in K$ such that $d(a^2 - d) \neq 0$. Assume that $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points of odd order on $E_{J,d,a}$. Assume that $P \neq Q$. Then $x_1y_2 - y_1x_2 \neq 0$.*

The choices for computing y_3 are abundant. For instance, each of the following formulae computes y_3 (except for a few exceptional inputs):

$$\begin{aligned} y_3 &= \frac{(x_1^2 + x_2^2)(y_1y_2 - 2ax_1x_2) - 2x_1x_2(1 + dx_1^2x_2^2)}{(x_1y_2 - y_1x_2)^2}, &\quad (6)\\ y_3 &= \frac{2(x_1y_1 - x_2y_2) - (x_1y_2 - y_1x_2)(y_1y_2 + 2ax_1x_2)}{(x_1y_2 - y_1x_2)(1 - dx_1^2x_2^2)}, &\quad (7)\\ y_3 &= \frac{x_1y_1(2 + 2ax_2^2 - y_2^2) - x_2y_2(2 + 2ax_1^2 - y_1^2)}{(x_1y_2 - y_1x_2)(1 - dx_1^2x_2^2)}. &\quad (8)\end{aligned}$$

In the printed version of this paper there has been typing errors in (7) and (8). Both formulae are correctly stated in this version.

Relevant Work. Efficient implementations often use inversion-free point doubling and point addition formulae. To the best of our knowledge all such proposals for Jacobi quartic curves reference from weighted projective coordinates which represent the points as $(X\colon Y\colon Z)_{[1,2,1]} = (\lambda X\colon \lambda^2 Y\colon \lambda Z)$ for all nonzero $\lambda \in K$ on the curve $Y^2 = dX^4 + 2aX^2Z^2 + Z^4$. Unlike the homogeneous projective case, this curve is non-singular provided that $\Delta \neq 0$.

Chudnovsky and Chudnovsky [8] proposed two inversion-free point addition and two inversion-free point doubling formulae using a slightly different quartic equation given by

$$E_{\tilde{J},a',b'}\colon y^2 = x^4 + a'x^2 + b'$$

and using weighted projective coordinates. The formulae in [8, (4.10i) on p.418] are analogous to (5) and (6) with the minor detail that the identity element is moved to the point at infinity $(1\colon 1\colon 0)$ on $Y^2 = X^4 + a'X^2Z^2 + b'Z^4$. The arithmetic of this curve is similar to that of $E_{J,d,a}$ due to the symmetry in the right hand side of the weighted projective equations $Y^2 = X^4 + a'X^2Z^2 + b'Z^4$ and $Y^2 = dX^4 + 2aX^2Z^2 + Z^4$.

Billet and Joye [7] proposed a faster inversion-free unified addition algorithm on the curve $Y^2 = dX^4 + 2aX^2Z^2 + Z^4$ based on (1) and (2). The term *unified* is used to emphasize that point addition formulae remain valid when two summands are identical, see [9, §29.1.2]. Keeping in mind that the points at infinity are defined over K if and only if d is a square in K, Lemma 2.1 implies that the Billet/Joye unified point addition algorithm is complete if d is nonsquare. This algorithm needs $10\mathbf{M} + 3\mathbf{S} + 3\mathbf{D}$ for arbitrary a and d. We remark that no faster way of inversion-free general point addition is known to date in $(X\colon Y\colon Z)_{[1,2,1]}$ coordinates. It remains an open question whether it is possible to speed up the addition in weighted $(X\colon Y\colon Z)_{[1,2,1]}$ coordinates. Nevertheless, the speed of the Billet/Joye algorithm was improved by Duquesne in [11] with the proposal of $(X^2\colon XZ\colon Z^2\colon Y)$ coordinates. Duquesne's variant addition algorithm needs $9\mathbf{M} + 2\mathbf{S} + 3\mathbf{D}$ saving $1\mathbf{M} + 1\mathbf{S}$ over the Billet/Joye algorithm by using slightly more space to represent the points. For the case $d = 1$, Bernstein and Lange [3] extended this representation to $(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2)$ and $(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2\colon X^2 + Z^2)$ saving an extra $\mathbf{M} - \mathbf{S}$ (i.e. **M**-**S** trade-off) over Duquesne's algorithm. A more detailed overview of these algorithms and operation counts can be found in the original papers or in the *Explicit-Formulas Database* (EFD) [3] which also reports $1\mathbf{M} + 9\mathbf{S} + 1\mathbf{D}$ doubling algorithm by Bernstein/Lange, $2\mathbf{M} + 6\mathbf{S} + 2\mathbf{D}$ doubling algorithm by Hisil/Carter/Dawson, and $2\mathbf{M} + 6\mathbf{S} + 1\mathbf{D}$ doubling algorithm by Feng/Wu in $(X\colon Y\colon Z)_{[1,2,1]}$. Duquesne coordinates $(X^2\colon XZ\colon Z^2\colon Y)_{[2,2,2,2]}$ use less space than redundant coordinates but need special treatment in the scalar multiplication to obtain the original coordinates $(X\colon Y\colon Z)_{[1,2,1]}$ of the final result. The original representation as $(X\colon Y\colon Z)_{[1,2,1]}$ in [7] uses even less space however this representation has to date been slower than the redundant coordinates.

Hisil, Carter, and Dawson [17] introduced new point doubling formulae together with a fast point doubling algorithm costing only $3\mathbf{M} + 4\mathbf{S}$ in $(X\colon Y\colon Z\colon X^2\colon Z^2)$ with $d = 1$. Roughly at the same time essentially the same formulae were independently derived by Feng and Wu, see EFD [3]. These formulae were adapted to $(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2)$ coordinates with the same operation count in EFD.

Later Hisil, Wong, Carter, and Dawson [19] introduced (for the case $d = 1$) new unified addition formulae which use $7\mathbf{M} + 3\mathbf{S} + 1\mathbf{D}$ in $(X\colon Y\colon Z\colon X^2\colon Z^2\colon XZ)$ and $7\mathbf{M} + 4\mathbf{S} + 1\mathbf{D}$ in $(X\colon Y\colon Z\colon X^2\colon Z^2)$ and newer doubling formulae which need $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$ in $(X\colon Y\colon Z\colon X^2\colon Z^2)$ and $(X\colon Y\colon Z\colon X^2\colon Z^2\colon XZ)$.

The redundant representations such as

$$(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2\colon X^2 + Z^2)_{[1,2,1,2,2,2,2]},$$
$$(X\colon Y\colon Z\colon X^2\colon 2XZ\colon Z^2)_{[1,2,1,2,2,2]},$$

$$(X : Y : Z : X^2 : Z^2 : X^2 + Z^2)_{[1,2,1,2,2,2]},$$
$$(X : Y : Z : X^2 : Z^2)_{[1,2,1,2,2]},$$
$$(X : Y : Z : X^2 : Z^2 : XZ)_{[1,2,1,2,2,2]}$$

help in the development of faster algorithms for performing point operations and their overall performance only slightly differs from each other. However, they all share one serious drawback. They need more space for storing the points in comparison to earlier proposals. Despite the speed advantage of these coordinate systems, the large space requirement makes the practical use of Jacobi quartic form questionable since windowing techniques in scalar multiplication algorithms precompute and store several points.

We aim to solve this disadvantage in subsequent sections. Furthermore we propose faster doubling algorithms.

Even more formulae. All of the affine formulae given in this section involve inversions in K. In cryptographic applications K is finite and computing inverses in a finite field can be very costly in comparison to the multiplication and addition operations. In §3 and §4 we will introduce inversion-free formulae which are simply derived by the adaptation of affine formulae to a suitable projective point representation. However, formulae given so far do not necessarily lead to the fastest inversion-free algorithms to perform the basic operations; point doubling and point addition. Next, we propose new affine point doubling and point addition formulae which help in improving the previous speeds on (extended) Jacobi quartic form.

Let $(x_1, y_1) \in E_{J,d,a}(K)$. Assuming that (x_3, y_3) is defined we have $2(x_1, y_1) = (x_3, y_3)$ where

$$x_3 = \mu x_1, \tag{9}$$
$$y_3 = \mu(\mu - y_1) - 1 \tag{10}$$

with $\mu = 2y_1/(2 + 2ax_1^2 - y_1^2)$. In the derivations of (9) and (10) we were inspired by the results in [4] and [19]. If d is a non-square in K then these point doubling formulae work for all inputs i.e. (x_3, y_3) is defined for all inputs. (If d is a square in K then there exist two points at infinity of order two. The double of these points is $(0, 1)$. If $2 + 2ax_1^2 - y_1^2 = 0$ then (x_1, y_1) is a point of order 4 and the output is a point at infinity.)

Further let $(x_2, y_2) \in E_{J,d,a}(K)$. Assuming that (x_3, y_3) is defined we have $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where x_3 is defined as in (5) and

$$y_3 = \frac{(x_1 - x_2)^2}{(x_1y_2 - y_1x_2)^2}(y_1y_2 - 2ax_1x_2 + 1 + dx_1^2x_2^2) - 1. \tag{11}$$

In addition, if $s \in K$ such that $d = s^2$ then we can also write

$$y_3 = \frac{(1 + sx_1x_2)^2}{(1 - dx_1^2x_2^2)^2}(y_1y_2 + 2ax_1x_2 + sx_1^2 + sx_2^2) - sx_3^2 \tag{12}$$

where x_3 is given by (1).

Formulae (11) and (12) compute the same result as (2), (6), (7), and (8) whenever their denominators are nonzero and neither of the summands is a point at infinity. Formula (12) is defined if $(x_1, y_1) = (x_2, y_2)$. Formula (11) fails if $(x_1, y_1) = (x_2, y_2)$. Both formulae are incomplete, i.e. they fail for a few special inputs.

3 Homogeneous projective coordinates

Projective coordinates are used as basic tools in designing inversion-free algorithms to carry out group arithmetic on elliptic curves. In the case of (extended) Jacobi quartic curves, we consider homogeneous projective coordinates $(X : Y : Z)_{[1,1,1]}$ for efficiency purposes for the first time. From now on we omit the informative subscript $[1, 1, 1]$ for these coordinates.

In homogeneous projective coordinates, $\mathcal{Q}$, each point (x, y) is represented by the triplet $(X : Y : Z)$ which satisfies the projective equation $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$ and corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. The identity element is represented by $(0 : 1 : 1)$. The negative of $(X : Y : Z)$ is $(-X : Y : Z)$. In the following subsection, we provide efficient point doubling formulae.

3.1 Dedicated point doubling in $\mathcal{Q}$

The fastest-so-far three-coordinate point doubling algorithm in [3, `dbl-2007-fw-2`] costs $2\mathbf{M} + 6\mathbf{S} + 1\mathbf{D}$ in $(X : Y : Z)_{[1,2,1]}$. We also remark that this algorithm assumes $d = 1$.

In this section we introduce new efficient doubling formulae. Given $(X_1 : Y_1 : Z_1)$ the point doubling can be performed as $2(X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(2Z_1^2 + 2aX_1^2 - Y_1^2), \\ Y_3 &= 2Y_1^2(Y_1^2 - 2aX_1^2) - (2Z_1^2 + 2aX_1^2 - Y_1^2)^2, \\ Z_3 &= (2Z_1^2 + 2aX_1^2 - Y_1^2)^2. \end{aligned} \tag{13}$$

We obtained these formulae from (9) and (10). With these formulae a point doubling takes $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D}$ where $1\mathbf{D}$ is multiplication with a. These formulae do not depend on d. Therefore keeping d arbitrary has no effect on the cost of (13).

If $a = \pm 1/2$ then a point doubling takes $2\mathbf{M} + 5\mathbf{S}$. Note $2a$ can be rescaled to -1 via the map $(x, y) \mapsto (x/\sqrt{-2a}, y)$ provided that $\sqrt{-2a} \in K$. This map transforms the curve $y^2 = dx^4 + 2ax^2 + 1$ to $y^2 = (d/(4a^2))x^4 - x^2 + 1$. Alternatively, a curve having $a = -1/2$ can be selected without rescaling. We comment that similar arguments apply to the case $a = 1/2$.

For justifications and more on operation counts see `DBL-Q-x` in Appendix B. The proposed algorithm(s) are faster than other three-coordinate point doubling algorithms for Jacobi quartic form.

3.2 Point addition in $\mathcal{Q}$

It would be convenient to give an efficient point addition algorithm for the projective coordinates. However, the fastest point addition algorithms that we could design were quite uncompetitive in comparison to the previous proposals in other coordinate systems. Therefore, we leave this as an open question. As a remedy to this, we will introduce fast point addition algorithms on a new coordinate system in the next section and show that the new point addition algorithms can be efficiently combined with the fast doubling algorithms from §3.1.

4 Extended homogeneous projective coordinates

Jacobi quartic form not only has a rich body of formulae but also allow us to use various efficient point representations. We have already given a detailed review in §2.

This section introduces a new representation and provides efficient algorithms to perform group operations on (extended) Jacobi quartic form elliptic curves. Some of the results in this section are analogous to our earlier work [18].

In the new system a point $(x, y) \in E_{J,d,a}(K)$ is represented by $(X : Y : T : Z)$ where $T = X^2/Z$ and $(X : Y : T : Z)_{[1,1,1,1]} = (\lambda X : \lambda Y : \lambda T : \lambda Z) = (x : y : x^2 : 1)$ for all nonzero $\lambda \in K$. From now on we omit the informative subscript $[1, 1, 1, 1]$ for these coordinates. Each quadruplet $(X : Y : T : Z)$ with $Z \neq 0$ simultaneously satisfy the homogeneous projective equations

$$\begin{cases} X^2 - TZ = 0 \\ Y^2 - dT^2 - 2aX^2 - Z^2 = 0 \end{cases} \tag{14}$$

or simply the homogeneous projective equation

$$Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4 \tag{15}$$

where T is omitted in the latter case. A point representation $(X\colon Y\colon Z)$ satisfying (15) can be converted to the new coordinates by computing $(XZ\colon YZ\colon X^2\colon Z^2)$ with $Z \neq 0$ in $1\mathbf{M}+3\mathbf{S}$ $(XZ = ((X+Z)^2 - X^2 - Z^2)/2)$. This coordinate system will be denoted by $\mathcal{Q}^e$ in the rest of the paper. The identity element is represented by the quadruplet $(0\colon 1\colon 0\colon 1)$. The negative of $(X\colon Y\colon T\colon Z)$ is $(-X\colon Y\colon T\colon Z)$.

4.1 Dedicated point doubling in $\mathcal{Q}^e$

Given $(X_1\colon Y_1\colon T_1\colon Z_1)$ with $Z_1 \neq 0$ satisfying (15), point doubling can be performed as $2(X_1\colon Y_1\colon T_1\colon Z_1) = (X_3\colon Y_3\colon T_3\colon Z_3)$ where X_3, Y_3, and Z_3 are the same as (13) and

$$T_3 \quad = \quad (2X_1Y_1)^2. \tag{16}$$

If $a = -1/2$ then a point doubling takes only $8\mathbf{S}$. Again the formulae do not depend on d. Therefore keeping d arbitrary has no effect on the cost of (16). There are many $\mathbf{M}/\mathbf{S}$ trade-offs possible for doubling in $\mathcal{Q}^e$ when a is arbitrary or when $a = -1/2$. For justifications and more on operation counts see `DBL-Qe-x` in Appendix B.

In §5, we will mix $\mathcal{Q}^e$ with $\mathcal{Q}$ to benefit from faster doubling algorithms proposed in §3.1. In §5, we will use point doubling from this section to develop a double-and-add algorithm.

4.2 Dedicated point addition in $\mathcal{Q}^e$

Given $(X_1\colon Y_1\colon T_1\colon Z_1)$ and $(X_2\colon Y_2\colon T_2\colon Z_2)$ with $Z_1 \neq 0$ and $Z_2 \neq 0$ and $(X_1\colon Y_1\colon T_1\colon Z_1) \neq (X_2\colon Y_2\colon T_2\colon Z_2)$, a dedicated addition can be performed as $(X_1\colon Y_1\colon T_1\colon Z_1) + (X_2\colon Y_2\colon T_2\colon Z_2) = (X_3\colon Y_3\colon T_3\colon Z_3)$ where

$$\begin{aligned}
X_3 &= (X_1Y_2 - Y_1X_2)(T_1Z_2 - Z_1T_2),\\
Y_3 &= (Y_1Y_2 - 2aX_1X_2)(T_1Z_2 + Z_1T_2) - 2X_1X_2(Z_1Z_2 + dT_1T_2),\\
T_3 &= (T_1Z_2 - Z_1T_2)^2,\\
Z_3 &= (X_1Y_2 - Y_1X_2)^2.
\end{aligned} \tag{17}$$

We derived these formulae using (5) and (6) in §2. Without any assumption on the curve constants, Y_3 can alternatively be written as

$$Y_3 \quad = \quad (T_1Z_2 + Z_1T_2 - 2X_1X_2)(Y_1Y_2 - 2aX_1X_2 + Z_1Z_2 + dT_1T_2) - Z_3. \tag{18}$$

We obtained this formula from (11). If $a = -1/2$ then the dedicated addition costs $7\mathbf{M} + 3\mathbf{S} + 2\mathbf{D}$ with the use of (18). For justifications and more on operation counts see `ADD-Qe-x` in Appendix B.

4.3 Unified point addition in $\mathcal{Q}^e$

Given $(X_1\colon Y_1\colon T_1\colon Z_1)$ and $(X_2\colon Y_2\colon T_2\colon Z_2)$ with $Z_1 \neq 0$ and $Z_2 \neq 0$, a unified addition can be performed as $(X_1\colon Y_1\colon T_1\colon Z_1) + (X_2\colon Y_2\colon T_2\colon Z_2) = (X_3\colon Y_3\colon T_3\colon Z_3)$ where

$$\begin{aligned}
X_3 &= (X_1Y_2 + Y_1X_2)(Z_1Z_2 - d\,T_1T_2),\\
Y_3 &= (Y_1Y_2 + 2aX_1X_2)(Z_1Z_2 + d\,T_1T_2) + 2dX_1X_2(T_1Z_2 + Z_1T_2),\\
T_3 &= (X_1Y_2 + Y_1X_2)^2,\\
Z_3 &= (Z_1Z_2 - d\,T_1T_2)^2.
\end{aligned} \tag{19}$$

These formulae are analogous to (1) and (2) hence complete[1] by Lemma 2.1 if d is not a square in K.

Let $s \in K$ such that $d = s^2$. Alternatively, we can write

$$Y_3 = (Z_1Z_2 + dT_1T_2 + 2sX_1X_2)(Y_1Y_2 + 2aX_1X_2 + sT_1Z_2 + sZ_1T_2) - sT_3. \quad (20)$$

We obtained this formula from (12) and following the derivation notes in [19, §2.1]. In this case, the addition is still unified. However, the completeness is lost. Nevertheless, logical checks can be eliminated if the inputs are selected as indicated in Lemma 2.2. As indicated before these formulae do not strictly require $d = 1$.

For justifications and more on operation counts see `UADD-Qe-x` in Appendix B. The new representation is solidly faster than the representation in [7]. The new representation can be equally fast as (or even faster than) the representation [11]. The special treatment in [11] for obtaining the original coordinates is also removed since $(X_3 : Y_3 : T_3 : Z_3)$ satisfies (15). The new representation can be equally fast as the representations in [3], [17], and [19]. However this is achieved by using only 4 coordinates rather than 5, 6, or 7 coordinates.

5 Mixed homogeneous projective coordinates

The construction in this section is the same as [18, §4.3] and is closely linked with [10]. Therefore, we only give a brief outline of the technique. The details can be extracted from the original papers.

Most of the efficient scalar multiplication implementations are based on a suitable combination of signed integer recoding (such as NAF, MOF), fast precomputation and left-to-right sliding fractional-windowing techniques. The resulting algorithm is doubling intensive. Roughly for each bit of the scalar one doubling is performed. Additions are accessed less frequently. Excluding the additions used in the precomputation phase, approximately $l/(w + 1)$ additions are needed where l is the number of bits in the scalar and w is the window length. w is used to control space consumption and optimize the total running time.

An abstract view of the scalar multiplication is composed of several repeated-doublings each followed by a single addition. In our specific case, these operations are performed in the following way:

(i) If a point doubling is followed by another point doubling, use $\mathcal{Q} \leftarrow 2\mathcal{Q}$.

(ii) If a point doubling is followed by a point addition, use

1. $\mathcal{Q}^e \leftarrow 2\mathcal{Q}$ for the point doubling step; followed by,
2. $\mathcal{Q} \leftarrow \mathcal{Q}^e + \mathcal{Q}^e$ for the point addition step.

Suppose that a repeated-doubling phase is composed of m doublings. In (i), $m - 1$ successive doublings in $\mathcal{Q}$ are performed with the fastest `DBL-Q-x` algorithm explained in §3.1 and given in Appendix B. In (ii), the remaining doubling is merged with the single addition phase to yield a combined double-and-add step; a similar approach to [12]. To perform the double-and-add operation we first compute the doubling step with the fastest `DBL-QtoQe-x` algorithm explained in §4.1 and given in Appendix B. This algorithm is suitable to compute $\mathcal{Q}^e \leftarrow 2\mathcal{Q}$ since the inputs are only composed of the coordinates X, Y, Z and the output is still produced in $\mathcal{Q}^e$. We then perform the addition in $\mathcal{Q}^e$ using `ADD-Qe-2` which is explained in §4.2 and given in Appendix B but output only the coordinates of $\mathcal{Q}$. Note that the last operation of `ADD-Qe-2` (i.e. $T_3 \leftarrow T_3^2$) can be confidently removed to save $1\mathbf{S}$ since the result is in $\mathcal{Q}$ (not $\mathcal{Q}^e$).

For instance, if we use `DBL-Q-1` for repeated doubling operations and a combination of `DBL-QtoQe-1` and `ADD-Qe-2` for double-and-add operations then we need only $2\mathbf{M} + 5\mathbf{S}$

[1] If d is not a square in K then the point $(0 : 1 : 0)$ is not defined over K and should be omitted though it seems to satisfy the curve equation (15).

for each doubling and we effectively need $((8\mathbf{S}) + (8\mathbf{M} + 2\mathbf{S} + 2\mathbf{D} - 1\mathbf{S})) - (2\mathbf{M} + 5\mathbf{S}) = 6\mathbf{M} + 4\mathbf{S} + 2\mathbf{D}$ for each addition step.

We should note that the precomputed points are kept in $\mathcal{Q}^e$ which is composed of 4 coordinates rather than 3. On the other hand, we do not need 5, 6, or 7 coordinates as is the case in [3], [17], and [19].

Let P be a point on $Y^2Z^2 = dX^4 + 2aX^2Z^2 + Z^4$. Let $\langle P \rangle$ denote the subgroup generated by P. Lemma 2.2 implies that the new dedicated doubling and unified addition algorithms work for all points in $\langle P \rangle$ if P is of odd order. Lemma 2.1 implies that the new dedicated doubling and unified addition algorithms work for all inputs if d is a not a square in K. Lemma 2.3 implies that new dedicated addition algorithms work for all points in $\langle P \rangle - \{P\}$ if P is of odd order. Note here that identical summands at an addition step of a scalar multiplication algorithm based on successive squaring technique, can only appear at the very first iteration if the scalar is smaller than the order of P.

6 Experimental results

This section provides implementation timings for elliptic curve single-variable-point-single-variable-scalar multiplication. We have used a single core of Intel Core 2 Duo (E6550) processor in our experiments.

Finite field operations. Following the implementation notes from [15] and [14], we have written a hand-crafted finite field layer using `x86-64` instruction set and GCC extended inline assembly. We have designed our field arithmetic layer to serve for fields $\mathbb{F}_p$ where p is of the form $2^{256} - c$ such that c has at most 64 bits. In our experiments we have fixed $c = 587$.

Elliptic curve operations. We have selected `Q-DBL-2` as the doubling algorithm and a combination of `Q-DBL-2` and `Qe-ADD-2` as the double-and-add algorithm. This decision is due to the fact that the cost of additions (in $\mathbb{F}_p$) is not so negligible on `x86-64` processors. This was previously discussed in [15].

Scalar multiplication algorithm. We have implemented Algorithm 3.38 in [16] by modifying Steps 4.3 and Steps 4.4 as we discussed in §5.

Integer recoding. We have used Avanzi's w-LtoR integer recoding algorithm [1] which runs on-the-fly as the main loop of the scalar multiplication is performed. We have determined $w = 5$ to be the optimal window length in our implementation. We have not incorporated fractional windowing techniques [22] to our implementation following the comments in [14].

Lookup table. To accommodate the 5-LtoR technique $3P, 5P, \ldots, 15P$ are precomputed by the sequence of operations $2P, 2P + P, 2P + 3P, \ldots, 2P + 13P$. A new precomputation strategy in [21] is of interest for implementation. We have not implemented this approach yet. In our implementation $\mathbf{I}/\mathbf{M} \approx 121$. Therefore we have not normalized the precomputed values following the analysis [5]. Also following the same reference, we have derived and implemented double and add algorithms in $\mathcal{Q}^e$ with $Z = 1$. These special operations save time in the precomputation.

Table 1 summarizes measured average clock cycles for primitive operations for a single-variable-point-single-variable-scalar multiplication on $E_{J,d,-1/2}$ for some fixed d.

Table 1: 256-bit scalar multiplication on Intel Core 2 Duo (E6550)

Operation	Cycles
Precomputation	17,000
Main loop	345,000
Normalization	14,000
Total	376,000

We should warn the reader that we have detected these cycle counts with our local benchmarking tools. Unfortunately, we have not yet integrated our implementation to the commonly accepted toolkit SUPERCOP, a benchmarking framework within eBACS, the benchmarking project of ECRYPT II [6]. Therefore we do not claim verifiability at this stage.

We should also warn the reader that our implementation is a variable-single-point-variable-single-scalar multiplication. In the case where the base point is fixed the timings can be dramatically improved by using Algorithm 3.44 or Algorithm 3.45 in [16]. Indeed such an approach was used in [14] for Diffie-Hellman key pair generation where the base point is fixed. Note also that our implementation does not incorporate the Galbraith-Lin-Scott (GLS) homomorphism [14] which has been recently shown to yield faster results.

In our implementation, a scalar multiplication on $E_{J,d,-1/2}$ takes approximately $1162\mathbf{M} + 1110\mathbf{S} + 102\mathbf{D}$. In addition, there are approximately 1796 calls to faster field operations (such as addition, subtraction, division by 2, etc.).

As the comparative part of our work, we have also covered Weierstrass ($a = -3$) and twisted Edwards ($a = -1$) curves in our implementation. For the twisted Edwards implementation we have followed [18, §4.3]. We have used doubling formulae from [2]. For the Weierstrass implementation we have collected the most efficient formulae from EFD [3]. In our implementation, a scalar multiplication on the Weierstrass curve $y^2 = x^3 - 3x + b$ for some fixed b with $w = 5$ —optimum— takes approximately $1598\mathbf{M} + 1156\mathbf{S} + 0\mathbf{D}$. In addition, there are approximately 2896 calls to faster field operations (such as addition, subtraction, division by 2, etc.). In our implementation, a scalar multiplication on the twisted Edwards curve $-x^2 + y^2 = 1 + dx^2y^2$ for some fixed d with $w = 6$ —optimum— takes approximately $1202\mathbf{M} + 969\mathbf{S} + 0\mathbf{D}$. In addition, there are approximately 2025 calls to faster field operations (such as addition, subtraction, division by 2, etc.). We have not tested the performance of formulae in [19] yet.

Table 2 summarizes measured average clock cycles for a single-variable-point-single-variable-scalar multiplication on different representations of elliptic curves.

Table 2: 256-bit scalar multiplication on Intel Core 2 Duo (E6550)

Curve	Cycles
Weierstrass ($a = -3$), Jacobian	468,000
Jacobi quartic ($a = -1/2$), $\mathcal{Q}^e$ with $\mathcal{Q}$	376,000
Twisted Edwards ($a = -1$), $\mathcal{E}^e$ with $\mathcal{E}$	362,000

In the printed version of this paper there has been a typing error in the case of Weierstrass curves where the cycle count was reported as 418,000. *The error in* Table 2 *is corrected in this version.*

In this implementation Jacobi quartic curves runs significantly faster than Weierstrass curves and slightly slower than twisted Edwards curves.

7 Conclusion

We introduced new results for performing arithmetic on (extended) Jacobi quartic curves.

In §2, we proved that earlier formulae (1) and (2) in the literature are complete provided that d is a not a square in the underlying field K. We explored several affine formulae some being known to date and some being new.

In §3 and §4, we carefully selected the most suitable affine formulae and then with suitable point representations converted them to projective form. In this context, for the first time we used homogeneous projective coordinates on (extended) Jacobi quartic curves for efficiency purposes and introduced new and faster doubling algorithms. In addition, we introduced a new point representation namely extended homogeneous projective coordinates, $\mathcal{Q}^e$, for Jacobi quartic curves. This coordinate system allows very efficient point addition operations using fewer coordinates in comparison to recent proposals for Jacobi quartic curves.

In §6 we reported our experimental results using state-of-art formulae for Jacobi quartic, twisted Edwards, and Weierstrass curves. In our implementation Jacobi quartic curves are 20%-25% faster than Weierstrass curves.

With our proposal Jacobi quartic curves can provide similar speeds to twisted Edwards curves in variable-point-and-variable-scalar multiplications. The point doubling on a Jacobi quartic curve is slightly faster than that of (twisted) Edwards curves. Note that doubling is the most frequently accessed elliptic curve group operation in variable-point-and-variable-scalar multiplications. On the other hand, Jacobi quartic curves seem to be slower on the double-and-add operation (§5) in comparison to twisted Edwards curves. In overall timings for variable-single-point-variable-single-scalar multiplication Jacobi quartic curves are competitive with (but slightly slower than) twisted Edwards curves. It should be noted here that there are elliptic curves of order 2-times-a-prime where our methods are applicable with the use of extended Jacobi quartic parametrization with $a = -1/2$. However, such curves cannot be written (over the same field) in (twisted) Edwards form.

References

[1] R. M. Avanzi. A note on the signed sliding window integer recoding and its left-to-right analogue. In *SAC 2004*, volume 3357 of *LNCS*, pages 130–143. Springer, 2005.

[2] D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In *AFRICACRYPT 2008*, volume 5023 of *LNCS*, pages 389–405. Springer, 2008.

[3] D. J. Bernstein and T. Lange. Explicit-formulas database. `http://www.hyperelliptic.org/EFD`.

[4] D. J. Bernstein and T. Lange. Faster addition and doubling on elliptic curves. In *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 29–50. Springer, 2007.

[5] D. J. Bernstein and T. Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In *Finite Fields and Applications Fq8*, volume 461 of *Contemporary Mathematics*, pages 1–18. American Mathematical Society, 2008.

[6] D. J. Bernstein and T. Lange. eBACS: ECRYPT benchmarking of cryptographic systems, 2008. `http://bench.cr.yp.to`.

[7] O. Billet and M. Joye. The Jacobi model of an elliptic curve and side-channel analysis. In *AAECC-15*, volume 2643 of *LNCS*, pages 34–42. Springer, 2003.

[8] D. V. Chudnovsky and G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Mathematics*, 7(4):385–434, 1986.

[9] H. Cohen and G. Frey, editors. *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. CRC Press, 2005.

[10] H. Cohen, A. Miyaji, and T. Ono. Efficient elliptic curve exponentiation using mixed coordinates. In *ASIACRYPT'98*, volume 1514 of *LNCS*, pages 51–65. Springer, 1998.

[11] S. Duquesne. Improving the arithmetic of elliptic curves in the Jacobi model. *Information Processing Letters*, 104(3):101–105, 2007.

[12] K. Eisenträger, K. Lauter, and P. L. Montgomery. Fast elliptic curve arithmetic and improved Weil pairing evaluation. In *CT-RSA 2003*, volume 2612 of *LNCS*, pages 343–354. Springer, 2003.

[13] L. Euler. De integratione aequationis differentialis $m\,dx/\sqrt{1-x^4} = n\,dy/\sqrt{1-y^4}$. *Novi Commentarii Academiae Scientiarum Petropolitanae 6*, pages 37–57, 1761. Translated from the Latin by Stacy G. Langton; On the integration of the differential equation $m\,dx/\sqrt{1-x^4} = n\,dy/\sqrt{1-y^4}$; available at `http://home.sandiego.edu/~langton/eell.pdf`.

[14] S. D. Galbraith, X. Lin, and M. Scott. Endomorphisms for faster elliptic curve cryptography on a large class of curves. In *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 518–535. Springer, 2009.

[15] P. Gaudry and E. Thomé. The mpFq library and implementing curve-based key exchanges, SPEED 2007, pp.49–64, 2007. `http://www.loria.fr/~gaudry/publis/mpfq.pdf`.

[16] D. Hankerson, A. J. Menezes, and S. A. Vanstone. *Guide to Elliptic Curve Cryptography.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[17] H. Hisil, G. Carter, and E. Dawson. New formulae for efficient elliptic curve arithmetic. In *INDOCRYPT 2007*, volume 4859 of *LNCS*, pages 138–151. Springer, 2007.

[18] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Twisted Edwards curves revisited. In *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 326–343. Springer, 2008.

[19] H. Hisil, K. K.-H. Wong, G. Carter, and E. Dawson. Faster group operations on elliptic curves. In *Australasian Information Security Conference (AISC 2009), Wellington, New Zealand, January 2009*, volume 98, pages 7–19. Conferences in Research and Practice in Information Technology (CRPIT), 2009.

[20] C. G. J. Jacobi. *Fundamenta nova theoriae functionum ellipticarum.* Sumtibus Fratrum Borntræger, 1829.

[21] P. Longa and C. Gebotys. Novel precomputation schemes for elliptic curve cryptosystems. In *ACNS '09, to appear*, LNCS. Springer, 2009.

[22] B. Möller. Improved techniques for fast exponentiation. In *ICISC 2002*, volume 2587 of *LNCS*, pages 298–312. Springer, 2003.

[23] M. Monagan and R. Pearce. Rational simplification modulo a polynomial ideal. In *ISSAC'06*, pages 239–245. ACM, 2006.

[24] E. T. Whittaker and G. N. Watson. *A Course of Modern Analysis.* Cambridge University Press, 1927.

A Lemma 2.2

Proof. Points at infinity (over the extension of K where they exist) are of order 2. 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) and (2) are complete provided that the points at infinity are not involved, the denominator $1 - dx_1^2x_2^2$ must be nonzero.

An algebraic approach is as follows. Suppose that $1 - dx_1^2x_2^2 = 0$. Then $x_1, x_2 \neq 0$ and we can write $x_1^2 = 1/(dx_2^2)$.

Suppose that $P = \pm Q$. Then $1 - dx_1^2x_2^2 = 1 - dx_1^4 = 1 - dx_2^4 = 0$. It follows that $R = 2P$ and $S = 2Q$ are points at infinity. Therefore P and Q must be of order 4 which contradicts the hypothesis. From now on we assume $P \neq \pm Q$.

We now have $1 - dx_1^4 \neq 0$ and $1 - dx_2^4 \neq 0$. Using the relations $x_1^2 = 1/(dx_2^2)$, $y_2^2 = dx_2^4 + 2ax_2^2 + 1$ and formulae (1) and (2) we get

$$x(R)^2 \quad = \quad \frac{(2x_1y_1)^2}{(1-dx_1^4)^2} = \frac{4(1/(dx_2^2))(d(1/(dx_2^2)^2) + 2a(1/(dx_2^2)) + 1)}{(1-d(1/(dx_2^2))^2)^2} = \frac{(2x_2y_2)^2}{(1-dx_2^4)^2} = x(S)^2,$$

$$
\begin{aligned}
y(R) &= \frac{(1+dx_1^4)(y_1^2+2ax_1^2)+4dx_1^4}{(1-dx_1^4)^2} \\
&= \frac{(1+d(1/(dx_2^2))^2)((d(1/(dx_2^2)^2)+2a(1/(dx_2^2))+1)+2a(1/(dx_2^2)))+4d(1/(dx_2^2))^2}{(1-d(1/(dx_2^2))^2)^2} \\
&= \frac{(1+dx_2^4)(y_2^2+2ax_2^2)+4dx_2^4}{(1-dx_2^4)^2} = y(S).
\end{aligned}
$$

Hence, $R = \pm S$. But then $R \mp S = 2P \mp 2Q = 2(P \mp Q) = (0, 1)$. It follows that $P \mp Q$ is a point of order 2 since $P \neq \pm Q$.

Now either P is a point of even order or Q is a point of even order or both P and Q are points of even order. All conditions contradict the hypothesis. In conclusion $1 - dx_1^2 x_2^2 \neq 0$. $\square$

B Verification scripts

The algorithms in this appendix are designed in a way that X_1-X_2-X_3, Y_1-Y_2-Y_3, T_1-T_2-T_3, and Z_1-Z_2-Z_3 are allowed to be the same registers. The algorithm uses t_i as temporary registers. **a** stands for an addition or a subtraction or a multiplication by 2 or a division by 2.

The following Maple script verifies (1), (2), (5), (6), (7), (8), (9), (10), (11), and (12).

```
a1:=0: a3:=0: a6:=0: a:=-a2/4: d:=(a2^2-4*a4)/16:
W:=(u,v)->v^2+a1*u*v+a3*v-(u^3+a2*u^2+a4*u+a6):
C:=(x,y)->y^2-(d*x^4+2*a*x^2+1):
CtoW:=(x,y)->(2*(a+(y+1)/x^2),4*(a+(y+1)/x^2)/x):
WtoC:=(u,v)->(2*u/v,2*(u-2*a)*u^2/v^2-1):
simplify([W(CtoW(x1,y1))],[C(x1,y1)]); #Check CtoW.
simplify([C(WtoC(u1,v1))],[W(u1,v1)]); #Check WtoC.
simplify([(x1,y1)-WtoC(CtoW(x1,y1))],[C(x1,y1)]); #Check WtoC(CtoW).
simplify([(u1,v1)-CtoW(WtoC(u1,v1))],[W(u1,v1)]); #Check CtoW(WtoC).
ut,vt:=CtoW(x1,y1): simplify([(-x1,y1)-WtoC(ut,-vt-a1*ut-a3)],[C(x1,y1)]); #Check the negation.

##Addition formulae.
unassign('x1','y1','x2','y2'): u1,v1:=CtoW(x1,y1): u2,v2:=CtoW(x2,y2):
L:=(v2-v1)/(u2-u1): u3:=L^2+a1*L-a2-u1-u2: v3:=L*(u1-u3)-v1-a1*u3-a3:
simplify([W(u3,v3)],[C(x1,y1),C(x2,y2)]); x3std,y3std:=WtoC(u3,v3):

x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
x3:=(x1^2-x2^2)/(x1*y2-y1*x2): simplify([x3std-x3],[C(x1,y1),C(x2,y2)]);
y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+
    2*d*x1*x2*(x1^2+x2^2))/((1-d*x1^2*x2^2)^2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
y3:=((x1^2+x2^2)*(y1*y2-2*a*x1*x2)-
    2*x1*x2*(1+d*x1^2*x2^2))/((x1*y2-y1*x2)^2): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
y3:=(2*(x1*y1-x2*y2)-(x1*y2-y1*x2)*(y1*y2+
    2*a*x1*x2))/((x1*y2-y1*x2)*(1-d*x1^2*x2^2)): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
y3:=(x1*y1*(2+2*a*x2^2-y2^2)-x2*y2*(2+2*a*x1^2-
    y1^2))/((x1*y2-y1*x2)*(1-d*x1^2*x2^2)): simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
y3:=(x1-x2)^2/(x1*y2-y1*x2)^2*(y1*y2-
    2*a*x1*x2+1+d*x1^2*x2^2)-1: simplify([y3std-y3],[C(x1,y1),C(x2,y2)]);
y3:=(1+s*x1*x2)^2/(1-d*x1^2*x2^2)^2*(y1*y2+
    2*a*x1*x2+s*x1^2+s*x2^2)-s*x3^2: simplify([y3std-y3],[C(x1,y1),C(x2,y2),d-s^2]);

##Doubling formulae.
unassign('x1','y1'): u1,v1:=CtoW(x1,y1):
L:=(3*u1^2+2*a2*u1+a4-a1*v1)/(2*v1+a1*u1+a3): u3:=L^2+a1*L-a2-2*u1: v3:=L*(u1-u3)-v1-a1*u3-a3:
simplify([W(u3,v3)],[C(x1,y1)]); x3std,y3std:=WtoC(u3,v3):

x3:=mu*x1: simplify([x3std-x3],[C(x1,y1),mu-2*y1/(2+2*a*x1^2-y1^2)]);
y3:=mu*(mu-y1)-1: simplify([y3std-y3],[C(x1,y1),mu-2*y1/(2+2*a*x1^2-y1^2)]);
```

Projective doubling formulae. The following Maple script verifies (13) and (16).

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    X3:=2*X1*Y1*(2*Z1^2+2*a*X1^2-Y1^2):
    Y3:=2*Y1^2*(Y1^2-2*a*X1^2)-(2*Z1^2+2*a*X1^2-Y1^2)^2:
    Z3:=(2*Z1^2+2*a*X1^2-Y1^2)^2:
    T3:=(2*X1*Y1)^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3]); #Check.
```

The following Maple scripts detail the evaluation of (13).

DBL-Q-1, $2\mathbf{M} + 5\mathbf{S} + 7\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: X3:=X3+Y3: t1:=t1-X3:
    Z3:=2*Z3: Y3:=X3*Y3: Y3:=2*Y3: Z3:=Z3-X3: X3:=t1*Z3: Z3:=Z3^2: Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3],[a+1/2]); #Check.
```

DBL-Q-2, $3\mathbf{M} + 4\mathbf{S} + 4\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3:
    X3:=Z3-X3: Z3:=X3^2: Y3:=Y3-Z3: X3:=t1*X3:
simplify([x3-X3/Z3,y3-Y3/Z3],[a+1/2]); #Check.
```

DBL-Q-3, $2\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 8\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: t1:=t1-X3: t1:=t1-Y3: X3:=k*X3:
    X3:=Y3+X3: Z3:=2*Z3: Y3:=X3*Y3: Y3:=2*Y3: Z3:=Z3-X3: X3:=t1*Z3: Z3:=Z3^2: Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3],[k+2*a]); #Check.
```

DBL-Q-4, $3\mathbf{M} + 4\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=k*X3: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3:
    X3:=Z3-X3: Z3:=X3^2: Y3:=Y3-Z3: X3:=t1*X3:
simplify([x3-X3/Z3,y3-Y3/Z3],[k+2*a]); #Check.
```

The following Maple scripts detail the evaluation of (13) and (16).

DBL-Qe-1, DBL-QtoQe-1, $8\mathbf{S} + 13\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: X3:=X3+Y3: T3:=T3-X3: Z3:=2*Z3:
    Z3:=Z3-X3: X3:=T3+Z3: T3:=T3^2: Z3:=Z3^2: X3:=X3^2: X3:=X3-T3: X3:=X3-Z3: Z3:=2*Z3:
    Y3:=2*Y3: Y3:=Y3^2: Y3:=Y3+T3: Y3:=Y3-Z3: T3:=2*T3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

DBL-Qe-2, DBL-QtoQe-2, $1\mathbf{M} + 7\mathbf{S} + 9\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: X3:=X3+Y3: T3:=T3-X3: Z3:=2*Z3:
    Z3:=Z3-X3: X3:=T3*Z3: Z3:=Z3^2: T3:=T3^2: Y3:=2*Y3: Y3:=Y3^2: Y3:=Y3+T3: Y3:=Y3/2:
    Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

DBL-Qe-3, DBL-QtoQe-3, $2\mathbf{M} + 6\mathbf{S} + 6\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=X3+Y3: X3:=X3/2: Z3:=Z3-X3: X3:=T3*Z3:
    Z3:=Z3^2: T3:=T3^2: Y3:=Y3^2: Y3:=Y3+T3: Y3:=Y3/2: Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

DBL-Qe-4, DBL-QtoQe-4, $3\mathbf{M} + 5\mathbf{S} + 4\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3: X3:=Z3-X3:
    Z3:=X3^2: Y3:=Y3-Z3: X3:=X3*T3: T3:=T3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

DBL-Qe-5, DBL-QtoQe-5, $8\mathbf{S} + 2\mathbf{D} + 14\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: T3:=T3-X3: t1:=T3-Y3: X3:=k*X3:
    X3:=X3+Y3: Z3:=2*Z3: Z3:=Z3-X3: T3:=t1^2: X3:=t1+Z3: X3:=X3^2: Z3:=Z3^2: X3:=X3-T3:
    X3:=X3-Z3: Z3:=2*Z3: t1:=k*T3: T3:=2*T3: Y3:=2*Y3: Y3:=Y3^2: Y3:=Y3+t1: Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

DBL-Qe-6, DBL-QtoQe-6, $1\mathbf{M} + 7\mathbf{S} + 1\mathbf{D} + 12\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: t1:=t1-X3: t1:=t1-Y3: X3:=k*X3: X3:=X3+Y3: Z3:=2*Z3:
    Y3:=X3*Y3: Z3:=Z3-X3: T3:=t1^2: X3:=t1+Z3: Z3:=Z3^2: Y3:=2*Y3: Y3:=Y3-Z3: X3:=X3^2: X3:=X3-T3: X3:=X3-Z3:
    X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

`DBL-Qe-7`, `DBL-QtoQe-7`, $1\mathbf{M} + 7\mathbf{S} + 2\mathbf{D} + 10\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1^2: t1:=t1-X3: t1:=t1-Y3: X3:=k*X3: X3:=X3+Y3: Z3:=2*Z3:
    Z3:=Z3-X3: X3:=t1*Z3: T3:=t1^2: Z3:=Z3^2: Y3:=2*Y3: Y3:=Y3^2: t1:=k*T3: Y3:=Y3+t1: Y3:=Y3/2: Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

`DBL-Qe-8`, `DBL-QtoQe-8`, $2\mathbf{M} + 6\mathbf{S} + 1\mathbf{D} + 8\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1+Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: T3:=T3^2: T3:=T3-X3: T3:=T3-Y3: X3:=k*X3: X3:=Y3+X3: Z3:=2*Z3:
    Y3:=X3*Y3: Y3:=2*Y3: Z3:=Z3-X3: X3:=T3*Z3: Z3:=Z3^2: Y3:=Y3-Z3: T3:=T3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

`DBL-Qe-9`, `DBL-QtoQe-9`, $2\mathbf{M} + 6\mathbf{S} + 2\mathbf{D} + 6\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    t1:=X1*Y1: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: X3:=k*X3: X3:=X3+Y3: X3:=X3/2: Z3:=Z3-X3: X3:=t1*Z3: Z3:=Z3^2:
    T3:=t1^2: Y3:=Y3^2: t1:=k*T3: Y3:=Y3+t1: Y3:=Y3/2: Y3:=Y3-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

`DBL-Qe-10`, `DBL-QtoQe-10`, $3\mathbf{M} + 5\mathbf{S} + 1\mathbf{D} + 4\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: mu:=2*y1/(2+2*a*x1^2-y1^2): x3:=mu*x1: y3:=mu*(mu-y1)-1:
    T3:=X1*Y1: X3:=X1^2:Y3:=Y1^2: Z3:=Z1^2: X3:=k*X3: X3:=X3+Y3: X3:=X3/2: Y3:=Y3*X3: X3:=Z3-X3: Z3:=X3^2:
    Y3:=Y3-Z3: X3:=T3*X3: T3:=T3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

Projective dedicated addition formulae. The following Maple script verifies (17).

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1^2-x2^2)/(x1*y2-y1*x2):
y3:=((x1^2+x2^2)*(y1*y2-2*a*x1*x2)-2*x1*x2*(1+d*x1^2*x2^2))/((x1*y2-y1*x2)^2):
    X3:=(X1*Y2-Y1*X2)*(T1*Z2-Z1*T2):
    Y3:=(Y1*Y2-2*a*X1*X2)*(T1*Z2+Z1*T2)-2*X1*X2*(Z1*Z2+d*T1*T2):
    T3:=(T1*Z2-Z1*T2)^2:
    Z3:=(X1*Y2-Y1*X2)^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3]); #Check.
```

The following Maple script verifies (17) when Y_3 is computed using (18).

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1^2-x2^2)/(x1*y2-y1*x2):
y3:=((x1-x2)^2)/((x1*y2-y1*x2)^2)*(y1*y2-2*a*x1*x2+1+d*x1^2*x2^2)-1:
    X3:=(X1*Y2-Y1*X2)*(T1*Z2-Z1*T2):
    T3:=(T1*Z2-Z1*T2)^2:
    Z3:=(X1*Y2-Y1*X2)^2:
    Y3:=(T1*Z2+Z1*T2-2*X1*X2)*(Y1*Y2-2*a*X1*X2+Z1*Z2+d*T1*T2)-Z3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3]); #Check.
```

The following Maple scripts detail the evaluation of (17) when Y_3 is computed using (18).

`ADD-Qe-1`, $7\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 19\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1^2-x2^2)/(x1*y2-y1*x2):
y3:=((x1-x2)^2)/((x1*y2-y1*x2)^2)*(y1*y2-2*a*x1*x2+1+d*x1^2*x2^2)-1:
    t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3:
    t3:=T3+t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3:
    Y3:=Y3+t1: t1:=2*t2: t1:=t3-t1: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3+T3: X3:=X3^2: Z3:=Z3^2: Y3:=Y3-Z3:
    X3:=X3-Z3: T3:=T3^2: X3:=X3-T3: X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

`ADD-Qe-2`, $8\mathbf{M} + 2\mathbf{S} + 2\mathbf{D} + 15\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1^2-x2^2)/(x1*y2-y1*x2):
y3:=((x1-x2)^2)/((x1*y2-y1*x2)^2)*(y1*y2-2*a*x1*x2+1+d*x1^2*x2^2)-1:
    t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3:
    t3:=T3+t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3:
    Y3:=Y3+t1: t1:=2*t2: t1:=t3-t1: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3*T3: Z3:=Z3^2: Y3:=Y3-Z3: T3:=T3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

ADD-Qe-3, $7\mathbf{M} + 3\mathbf{S} + 3\mathbf{D} + 19\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1^2-x2^2)/(x1*y2-y1*x2):
y3:=((x1-x2)^2)/((x1*y2-y1*x2)^2)*(y1*y2-2*a*x1*x2+1+d*x1^2*x2^2)-1:
    t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3:
    t3:=T3+t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3:
    Y3:=Y3+t1: t1:=2*t2: t1:=t3-t1: t2:=k*t2: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3+T3: X3:=X3^2: Z3:=Z3^2:
    Y3:=Y3-Z3: X3:=X3-Z3: T3:=T3^2: X3:=X3-T3: X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

ADD-Qe-4, $8\mathbf{M} + 2\mathbf{S} + 3\mathbf{D} + 15\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1^2-x2^2)/(x1*y2-y1*x2):
y3:=((x1-x2)^2)/((x1*y2-y1*x2)^2)*(y1*y2-2*a*x1*x2+1+d*x1^2*x2^2)-1:
    t1:=T1+Z1: t2:=d*T2: t2:=t2+Z2: t1:=t1*t2: t2:=Z1*T2: T3:=T1*Z2: t1:=t1-T3: t3:=d*t2: t1:=t1-t3:
    t3:=T3+t2: T3:=T3-t2: Z3:=X1-Y1: t2:=X2+Y2: Z3:=Z3*t2: t2:=X1*X2: Y3:=Y1*Y2: Z3:=Z3-t2: Z3:=Z3+Y3:
    Y3:=Y3+t1: t1:=2*t2: t1:=t3-t1: t2:=k*t2: Y3:=Y3+t2: Y3:=t1*Y3: X3:=Z3*T3: Z3:=Z3^2: Y3:=Y3-Z3: T3:=T3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

Projective unified addition formulae. The following Maple script verifies (19). If d is a non-square in K then these formulae work for all inputs.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/((1-d*x1^2*x2^2)^2):
    X3:=(X1*Y2+Y1*X2)*(Z1*Z2-d*T1*T2):
    Y3:=(Y1*Y2+2*a*X1*X2)*(Z1*Z2+d*T1*T2)+2*d*X1*X2*(T1*Z2+Z1*T2):
    T3:=(X1*Y2+Y1*X2)^2:
    Z3:=(Z1*Z2-d*T1*T2)^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3]); #Check.
```

The following Maple scripts detail the evaluation of (19).

UADD-Qe-5, $8\mathbf{M} + 3\mathbf{S} + 2\mathbf{D} + 17\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/((1-d*x1^2*x2^2)^2):
    t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3:
    Z3:=Z3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=d*t1:
    t1:=t1*X3: t1:=2*t1: Y3:=Y3-X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3+T3: X3:=X3^2: T3:=T3^2: Z3:=Z3^2:
    X3:=X3-T3: X3:=X3-Z3: X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

UADD-Qe-6, $9\mathbf{M} + 2\mathbf{S} + 2\mathbf{D} + 13\mathbf{a}$, assumes $a = -1/2$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/((1-d*x1^2*x2^2)^2):
    t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3:
    Z3:=Z3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=d*t1:
    t1:=t1*X3: t1:=2*t1: Y3:=Y3-X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3*T3: T3:=T3^2: Z3:=Z3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[a+1/2]); #Check.
```

UADD-Qe-7, $8\mathbf{M} + 3\mathbf{S} + 3\mathbf{D} + 17\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/((1-d*x1^2*x2^2)^2):
    t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2:  t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3:
    Z3:=Z3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=d*t1:
    t1:=t1*X3: t1:=2*t1: X3:=k*X3: Y3:=Y3-X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3+T3: X3:=X3^2: T3:=T3^2:
    Z3:=Z3^2: X3:=X3-T3: X3:=X3-Z3: X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

UADD-Qe-8, $9\mathbf{M} + 2\mathbf{S} + 3\mathbf{D} + 13\mathbf{a}$, assumes $k = -2a$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((y1*y2+2*a*x1*x2)*(1+d*x1^2*x2^2)+2*d*x1*x2*(x1^2+x2^2))/((1-d*x1^2*x2^2)^2):
    t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t1:=t1-T3: t1:=t1-Z3: T3:=d*T3: t2:=Z3+T3:
    Z3:=Z3-T3: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: T3:=T3-Y3: t1:=d*t1:
    t1:=t1*X3: t1:=2*t1: X3:=k*X3: Y3:=Y3-X3: Y3:=Y3*t2: Y3:=Y3+t1: X3:=Z3*T3: T3:=T3^2: Z3:=Z3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a]); #Check.
```

The following Maple script verifies (19) when Y_3 is computed using (20).

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((1+s*x1*x2)^2)/((1-d*x1^2*x2^2)^2)*(y1*y2+2*a*x1*x2+s*x1^2+s*x2^2)-s*x3^2:
    X3:=(X1*Y2+Y1*X2)*(Z1*Z2-d*T1*T2):
    T3:=(X1*Y2+Y1*X2)^2:
    Z3:=(Z1*Z2-d*T1*T2)^2:
    Y3:=(Z1*Z2+d*T1*T2+2*s*X1*X2)*(Y1*Y2+2*a*X1*X2+s*T1*Z2+s*Z1*T2)-s*T3:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[d-s^2]); #Check.
```

The following Maple scripts detail the evaluation of (19) when Y_3 is computed using (20).

UADD-Qe-1, $7\mathbf{M} + 3\mathbf{S} + 1\mathbf{D} + 18\mathbf{a}$, assumes $k = -2a$, assumes $d = 1$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((1+s*x1*x2)^2)/((1-d*x1^2*x2^2)^2)*(y1*y2+2*a*x1*x2+s*x1^2+s*x2^2)-s*x3^2:
    t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3: t1:=t1-t2: T3:=X1+Y1:
    t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: t3:=T3-X3: t3:=t3-Y3: T3:=t3^2: Y3:=Y3+t1: t1:=2*X3:
    t1:=t1+t2: t2:=k*X3: Y3:=Y3-t2: Y3:=Y3*t1: Y3:=Y3-T3: X3:=t3+Z3: X3:=X3^2: Z3:=Z3^2: X3:=X3-Z3:
    X3:=X3-T3: X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a,s-1,d-1]); #Check.
```

UADD-Qe-2, $8\mathbf{M} + 2\mathbf{S} + 1\mathbf{D} + 14\mathbf{a}$, assumes $k = -2a$, assumes $d = 1$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((1+s*x1*x2)^2)/((1-d*x1^2*x2^2)^2)*(y1*y2+2*a*x1*x2+s*x1^2+s*x2^2)-s*x3^2:
    t1:=T1+Z1: t2:=T2+Z2: T3:=T1*T2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3: t1:=t1-t2: T3:=X1+Y1:
    t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: t3:=T3-X3: t3:=t3-Y3: T3:=t3^2: Y3:=Y3+t1: t1:=2*X3:
    t1:=t1+t2: t2:=k*X3: Y3:=Y3-t2: Y3:=Y3*t1: Y3:=Y3-T3: X3:=t3*Z3: Z3:=Z3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a,s-1,d-1]); #Check.
```

UADD-Qe-3, $7\mathbf{M} + 3\mathbf{S} + 5\mathbf{D} + 18\mathbf{a}$, assumes $k = -2a$, assumes $d = s^2$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((1+s*x1*x2)^2)/((1-d*x1^2*x2^2)^2)*(y1*y2+2*a*x1*x2+s*x1^2+s*x2^2)-s*x3^2:
    t1:=s*T1: t2:=s*T2: T3:=t1*t2: t1:=t1+Z1: t2:=t2+Z2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3:
    t1:=t1-t2: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: t3:=T3-Y3: Y3:=Y3+t1:
    T3:=t3^2: t1:=2*s: t1:=t1*X3: t1:=t1+t2: t2:=k*X3: Y3:=Y3-t2: Y3:=Y3*t1: t1:=s*T3: Y3:=Y3-t1:
    X3:=t3+Z3: X3:=X3^2: Z3:=Z3^2: X3:=X3-Z3: X3:=X3-T3: X3:=X3/2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a,d-s^2]); #Check.
```

UADD-Qe-4, $8\mathbf{M} + 2\mathbf{S} + 5\mathbf{D} + 14\mathbf{a}$, assumes $k = -2a$, assumes $d = s^2$.

```
x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2: T1:=X1^2/Z1: T2:=X2^2/Z2:
x3:=(x1*y2+y1*x2)/(1-d*x1^2*x2^2):
y3:=((1+s*x1*x2)^2)/((1-d*x1^2*x2^2)^2)*(y1*y2+2*a*x1*x2+s*x1^2+s*x2^2)-s*x3^2:
    t1:=s*T1: t2:=s*T2: T3:=t1*t2: t1:=t1+Z1: t2:=t2+Z2: Z3:=Z1*Z2: t1:=t1*t2: t2:=Z3+T3: Z3:=Z3-T3:
    t1:=t1-t2: T3:=X1+Y1: t3:=X2+Y2: X3:=X1*X2: Y3:=Y1*Y2: T3:=T3*t3: T3:=T3-X3: t3:=T3-Y3: Y3:=Y3+t1:
    T3:=t3^2: t1:=2*s: t1:=t1*X3: t1:=t1+t2: t2:=k*X3: Y3:=Y3-t2: Y3:=Y3*t1: t1:=s*T3: Y3:=Y3-t1:
    X3:=t3*Z3: Z3:=Z3^2:
simplify([x3-X3/Z3,y3-Y3/Z3,X3^2/Z3-T3],[k+2*a,d-s^2]); #Check.
```