Faster Pairings on Special Weierstrass Curves

Craig Costello*, Huseyin Hisil, Colin Boyd,
Juan Manuel Gonzalez Nieto, and Kenneth Koon-Ho Wong
Information Security Institute, Queensland University of Technology,
GPO Box 2434, Brisbane QLD 4001, Australia
{craig.costello,h.hisil,c.boyd,j.gonzaleznieto,kk.wong}@qut.edu.au

Abstract

This paper presents efficient formulas for computing cryptographic pairings on the curve $y^2 = cx^3 + 1$ over fields of large characteristic. We provide examples of pairing-friendly elliptic curves of this form which are of interest for efficient pairing implementations.

Keywords: Tate pairing, Miller's algorithm, elliptic curves.

1 Introduction

Bilinear pairings have found many applications in cryptography, such as the identity-based encryption scheme of Boneh and Franklin [9], the one-round tripartite key agreement scheme of Joux [17] and the short signature scheme of Boneh, Lynn and Shacham [10]. To implement pairing-based protocols in practice, it is necessary to match curves which are pairing-friendly with an efficient pairing algorithm. The most efficient method of computing pairings is Miller's algorithm [21]. Each iteration of this process requires three significant computations: (i) point operations, i.e. point doubling and/or point addition; (ii) Miller line function computations and (iii) updating the Miller function value. In this paper we explore the j-invariant zero curve $y^2 = cx^3 + 1$ and provide new formulas that facilitate a faster pairing computation on this curve by decreasing the number of computationally expensive field operations encountered in stage (ii).

For pairing computations with even embedding degree k, the curve $y^2 = cx^3 + 1$ allows the Miller doubling stage to be computed in $(k + 3)\mathbf{m} + 5\mathbf{s} + 1\mathbf{M} + 1\mathbf{S}$, where $\mathbf{m}$ and $\mathbf{s}$ denote the costs of multiplication and squaring in the base field while $\mathbf{M}$ and $\mathbf{S}$ denote the costs of multiplication and squaring in the extension field of degree k. For the more general j-invariant zero curve $y^2 = x^3 + b$, the fastest Miller doubling operation count recorded to date is

*This author acknowledges funding from the Queensland Government Smart State PhD Scholarship.

$(k+3)\mathbf{m}+8\mathbf{s}+1\mathbf{M}+1\mathbf{S}$ [1], meaning that the special curve $y^2 = cx^3+1$ offers an advantage of $3\mathbf{s}$ at the doubling stage.

We provide practically useful examples of the curve $y^2 = cx^3+1$ for different embedding degrees. For the majority of embedding degrees $k \leq 50$, the curve generation technique we adopt achieves ρ-values similar to the best values presented in [13]. We draw comparisons between the curve we employ and other special curves and discuss where this curve model would be optimal in practice.

The remainder of this paper is organised as follows. §2 gives a brief overview of pairings. §3 explains our search for a faster Weierstrass model and efficient group operations. §4 presents the optimization of the new formulas for the computation of the Tate pairing. §5 discusses curve generation and provides some practical examples and §6 summarizes our contributions and compares them with the literature. In the appendices, we share our scripts that verify the main claims of §3 and §4. The appendices also provide more intrinsic details on the realization of the proposed formulas.

2 Background on pairings

This section gives a brief background on pairings. Galbraith gives a more comprehensive survey [14].

Let $\mathbb{F}_q$ be a finite field with $q = p^n$ elements where $p \geq 5$ is prime and let E be an elliptic curve defined over $\mathbb{F}_q$. Let $\mathcal{O}$ denote the identity on E. Let r be a large prime that is coprime to q such that $r|\#E(\mathbb{F}_q)$ and let k be the embedding degree of E with respect to r. For practical purposes we assume that $k > 1$. We call $\mathbb{F}_q$ the base field and $\mathbb{F}_{q^k}$ the extension field. Let $f_{i,P} \in \mathbb{F}_q(E)$ be a function with divisor $\mathrm{div}(f_{i,P}) = i(P) - ([i]P) - (i-1)(\mathcal{O})$.

The Tate pairing. Choose a point $P \in E(\mathbb{F}_q)[r]$, this implies $\mathrm{div}(f_{r,P}) = r(P) - r(\mathcal{O})$. Let $Q \in E(\mathbb{F}_{q^k})/rE(\mathbb{F}_{q^k})$ and let μ_r denote the group of r-th roots of unity in $\mathbb{F}^*_{q^k}$. The reduced Tate pairing e_r [4] is defined as

$$e_r \colon (P, Q) \mapsto f_{r,P}(Q)^{(q^k-1)/r} \in \mu_r.$$

Miller's algorithm [21] computes the paired value iteratively by taking advantage of the fact that $f_{i+j,P}$ can be written as $f_{i+j,P} = f_i \cdot f_j \cdot l/v$, where l and v are the lines used in the computation of $[i]P + [j]P = [i+j]P$. That is, l is the line that intersects E at $[i]P$, $[j]P$ and $-[i+j]P$, and v is the vertical line that intersects E at both $[i+j]P$ and $-[i+j]P$. This enables us to compute the function $f_{2i,P}$ from $f_{i,P}$ directly by evaluating the lines that are used in point doubling of P. Similarly, we can compute the function $f_{i+1,P}$ from $f_{i,P}$ so that $f_{r,P}$ can be computed in $\log_2 r$ steps, as summarised in Algorithm 1.

There are many other optimizations which speed up the computation of the Miller loop in certain settings, including the denominator elimination technique [4], uses of efficiently computable endomorphisms [24], [15], and loop shortening techniques [2], [16], [3], [20], [26], [19], [25].

Algorithm 1 Miller's algorithm

Input: $P \in E(\mathbb{F}_{q^k})[r]$, $Q \in E(\mathbb{F}_{q^k})$, $r = (r_{m-1} \dots r_1 r_0)_2$ with $r_{m-1} = 1$.
Output: $f_{r,P}(Q) \leftarrow f_{\text{var}}$.

1: $R \leftarrow P$, $f_{\text{var}} \leftarrow 1$.
2: **for** $i = m - 2$ down to 0 **do**
3: Compute lines l_{dbl} and v_{dbl} for doubling R.
4: $R \leftarrow [2]R$.
5: $f_{\text{var}} \leftarrow f_{\text{var}}^2 \cdot l_{\text{dbl}}(Q)/v_{\text{dbl}}(Q)$.
6: **if** $r_i = 1$ **then**
7: Compute lines l_{add} and v_{add} for adding R and P.
8: $R \leftarrow R + P$.
9: $f_{\text{var}} \leftarrow f_{\text{var}} \cdot l_{\text{add}}(Q)/v_{\text{add}}(Q)$.
10: **end if**
11: **end for**
12: **return** f_{var}.

3 Choice of curve

In this section we specify the choice of curve that facilitates an efficient iteration of the Miller loop.

Let E be a Weierstrass form elliptic curve $y^2 = x^3 + ax + b$. Let (x_1, y_1) be a point in $E(\mathbb{F}_q) - \{\mathcal{O}\}$. We then have $(x_1, y_1) + (x_1, -y_1) = \mathcal{O}$. Further let (x_2, y_2) be a point in $E(\mathbb{F}_q) - \{\mathcal{O}\}$ such that $y_2 \neq 0$ and $(x_2, y_2) \neq (x_1, -y_1)$. We then have $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = \lambda^2 - x_1 - x_2, \tag{1}$$
$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{2}$$

with
$$\lambda = \begin{cases} (y_1 - y_2)/(x_1 - x_2) & \text{if } (x_1, y_1) \neq (x_2, y_2) \\ (3x_1^2 + a)/(2y_1) & \text{if } (x_1, y_1) = (x_2, y_2) \end{cases}.$$

In the literature, addition using (1) and (2) in the case $(x_1, y_1) = (x_2, y_2)$ is named point doubling. Similarly the case $(x_1, y_1) \neq (x_2, y_2)$ is named point addition. We shall follow the same nomenclature.

In our experiments we have observed that it is possible to rewrite the doubling formulas as follows provided that $b \neq 0$ is a square in $\mathbb{F}_q$ such that $c^2 = b$. We have $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = x_1(\mu - \mu^2) + a\sigma, \tag{3}$$
$$y_3 = (y_1 - c)\mu^3 + a\delta - c \tag{4}$$

with $\mu = (y_1 + 3c)/(2y_1)$, $\sigma = (a - 3x_1^2)/(2y_1)^2$, $\delta = (3x_1(y_1 - 3c)(y_1 + 3c) - a(9x_1^2 + a))/(2y_1)^3$. Computer aided proofs of the correctness of formulas (3) and (4) are provided in Appendix A.

In the derivation of these formulas we have consulted [22]. The new point doubling formulas strike us with an interesting property: the total degrees[1] of x_3

[1]The total degree is defined as the sum of the degrees of the numerator and denominator of a rational function.

and y_3 are lower than those of the original point doubling formulas. Furthermore the total degrees of the new formulas are minimal. This can be verified using Algorithm 2 of [22, §4]. In particular, the total degree of x_3 and y_3 drops from 6 to 5 and from 9 to 7, respectively.

The evaluation of lower degree functions often requires less field operations. However, it seems that the original point doubling formulas still win in affine coordinates. On the other hand, we will eventually be forced to switch to homogeneous projective or Jacobian coordinates in order to prevent costly inversions. Therefore it is worthwhile to check operation counts on these coordinates. We will delay the details until §4.

If we work on the elliptic curve $y^2 = x^3 + c^2$, i.e. $a = 0$, the formulas (3) and (4) become much simpler. In addition, in order to prevent the computational disadvantage of field operations with c in doubling formulas we prefer to work with another representation of the same curve given by $y^2 = cx^3 + 1$. This curve is isomorphic over $\mathbb{F}_q$ to the Weierstrass curve $v^2 = u^3 + c^2$. The isomorphism from $y^2 = cx^3 + 1$ to $v^2 = u^3 + c^2$ is given by $\sigma\colon (x, y) \mapsto (u, v) = (cx, cy)$ with the inverse $\sigma^{-1}\colon (u, v) \mapsto (x, y) = (u/c, v/c)$.

Again, we denote the identity on $y^2 = cx^3 + 1$ by $\mathcal{O}$ and point negation is performed by negating the y coordinate. Using the same notation as in the original formulas, we have $[2](x_1, y_1) = (x_3, y_3)$ where

$$x_3 = x_1(\mu - \mu^2), \tag{5}$$
$$y_3 = (y_1 - 1)\mu^3 - 1 \tag{6}$$

with $\mu = (y_1 + 3)/(2y_1)$ and we have $(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$ where

$$x_3 = c^{-1}\lambda^2 - x_1 - x_2, \tag{7}$$
$$y_3 = \lambda(x_1 - x_3) - y_1 \tag{8}$$

with $\lambda = (y_1 - y_2)/(x_1 - x_2)$. The point $(0, 1)$ is of order 3. Computer aided proofs of the correctness of formulas (5), (6), (7), and (8) are provided in Appendix B.

4 Tate pairing computation on $y^2 = cx^3 + 1$

In this section we further investigate the arithmetic of $y^2 = cx^3 + 1$ to assist efficient computation of the Tate pairing. We first derive suitable line equations to compute the Miller value at both the doubling and addition stages. We then eliminate unnecessary computations before converting all computations to projective representation to avoid inversions. We provide several appendices that verify our claims.

Barreto *et al.* [6] show that it is possible to eliminate costly operations in Miller's algorithm provided the point where the Miller function is evaluated is chosen suitably. In the Tate pairing, the vertical line functions v (v_{dbl} and v_{add}) in Algorithm 1 are evaluated at the point $Q = (x_Q, y_Q)$. These vertical line functions take the form $v = x_R - x_Q$, where $R = (x_R, y_R)$ is the intermediate point in Algorithm 1. The computations in Miller's algorithm can be simplified

if v takes a value in a proper subfield $\mathbb{F}_{q^d} \subset \mathbb{F}_{q^k}$. When computing the Tate pairing on curves with even embedding degrees $k = 2d$, we choose Q to enable this simplification by choosing a point Q' on the quadratic twist E' of E and mapping Q' to Q under the twisting isomorphism, meaning that $x_Q \in \mathbb{F}_{q^d}$ and $y_Q = \tilde{y_Q}\sqrt{\nu}$, where $\tilde{y_Q} \in \mathbb{F}_{q^d}$ and ν is some quadratic non-residue in $\mathbb{F}_{q^d}$.

The Miller values. If we derive the line equations arising from the addition of (x_1, y_1) and (x_2, y_2) we obtain

$$g_{\text{add}} = c\frac{\lambda(x_2 - x_Q) - y_2 + y_Q}{c(x_1 + x_2 + x_Q) - \lambda^2} \tag{9}$$

where $\lambda = (y_1 - y_2)/(x_1 - x_2)$ and $g_{\text{add}} = l_{\text{add}}(Q)/v_{\text{add}}(Q)$ (refer to Line 9 of Algorithm 1). This formula shares several common subexpressions with (7) and (8).

For the case $(x_1, y_1) = (x_2, y_2)$, we propose a new formula for the line computation which uses several shared common subexpressions with the new point doubling formulas (5) and (6). The new formula is given by

$$g_{\text{dbl}} = \frac{2cy_1(x_1 - x_Q)^2}{x_1^2(3cx_Q) - y_1^2 + 3 + 2y_1y_Q}, \tag{10}$$

where $g_{\text{dbl}} = l_{\text{dbl}}(Q)/v_{\text{dbl}}(Q)$ (refer to Line 5 of Algorithm 1). Furthermore, if $(x_1, y_1) = -(x_2, y_2)$ we have

$$g_{\text{vert}} = -c(x_1 - x_Q). \tag{11}$$

Computer aided proofs of the correctness of our formulas are provided in Appendix C.

Irrelevant factors. We now focus on eliminating the terms in equations (9) and (10) by adopting the denominator elimination technique [7]. Recall that y_Q is the only element that appears in the formulas above[2] that is in the full extension field $\mathbb{F}_{q^k}$. We immediately notice that the denominator of g_{add} in equation (9) is completely contained in $\mathbb{F}_{q^d}$ and can therefore be eliminated, to give

$$g'_{\text{add}} = (y_1 - y_2)(x_2 - x_Q) - (x_1 - x_2)(y_2 - y_Q). \tag{12}$$

With identical reasoning we can omit the numerator of g_{dbl} in equation (10). These eliminations are standard. Now, observe that since y_Q is of the form $y_Q = \tilde{y_Q}\sqrt{\nu}$, we can write the denominator as $1/(t_1 + t_2\sqrt{\nu})$ where $t_1 = x_1^2(3cx_Q) - y_1^2 + 3$ and $t_2 = 2y_1\tilde{y_Q}$. If the Miller value is computed in this fashion there will be an inversion at the end of the Miller loop. Even worse, both the numerator and the denominator of f_{var} would have to be updated at each iteration of the Miller loop since the addition step produces a non-trivial numerator. To prevent this we multiply the numerator and the denominator of

[2]The point (x_2, y_2) represents $P \in E(\mathbb{F}_q)$ and the point (x_1, y_1) represents $R \in E(\mathbb{F}_q)$ in Algorithm 1, a multiple of P, so that $x_1, x_2, y_1, y_2 \in \mathbb{F}_q$.

$1/(t_1+t_2\sqrt{\nu})$ by the conjugate expression $t_1-t_2\sqrt{\nu}$ to give $(t_1-t_2\sqrt{\nu})/(t_1^2-t_2^2\nu)$. Since $t_1^2 - t_2^2\nu \in \mathbb{F}_{q^d}$ we can simply omit the denominator to give

$$g'_{\text{dbl}} = x_1^2(3cx_Q) - y_1^2 + 3 - 2y_1y_Q. \tag{13}$$

It also follows that if $(x_1, y_1) = -(x_2, y_2)$ we have $g'_{\text{vert}} = 1$. If r is odd, the Miller loop always finishes in this fashion so we ignore the point addition in the final iteration.

We next present point doubling and point addition formulas together with their associated line formulas in homogeneous projective coordinates. Our experiments gave the best results in homogeneous coordinates rather than Jacobian coordinates for doubling and additions. While additions generally favour projective coordinates it is interesting to note that also doublings on this curve are faster in projective coordinates. In particular the number of field operations for the doubling is $4\mathbf{m} + 3\mathbf{s}$ while the best known doubling speeds so far are $2\mathbf{m} + 5\mathbf{s}$ but in Jacobian coordinates. So this representation achieves the best addition speed and the best doubling speed (up to some $\mathbf{m}/\mathbf{s}$ tradeoffs) in *the same* coordinate system.

Homogeneous projective coordinates. In homogeneous projective coordinates each point (x, y) is represented by the triplet $(X : Y : Z)$ which satisfies the projective equation $Y^2Z = cX^3 + Z^3$ and corresponds to the affine point $(X/Z, Y/Z)$ with $Z \neq 0$. The identity element is represented by $(0 : 1 : 0)$. The negative of $(X : Y : Z)$ is $(X : -Y : Z)$.

Point doubling with line computation. Given $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ the point doubling can be performed as $[2](X_1 : Y_1 : Z_1) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= 2X_1Y_1(Y_1^2 - 9Z_1^2), \\ Y_3 &= (Y_1 - Z_1)(Y_1 + 3Z_1)^3 - 8Y_1^3Z_1, \\ Z_3 &= 8Y_1^3Z_1. \end{aligned} \tag{14}$$

These formulas are derived from (5) and (6) in Section 3. Point doubling without line computation needs $4\mathbf{m} + 3\mathbf{s}$ using the following sequence of operations.

$$A = Y_1^2, \quad B = Z_1^2, \quad C = (Y_1 + Z_1)^2 - A - B, \quad Z_3 = 4A \cdot C,$$
$$X_3 = 2X_1 \cdot Y_1 \cdot (A - 9B), \quad Y_3 = (A - 3B + C) \cdot (A + 9B + 3C) - Z_3.$$

The line formula derived from (13) is given by

$$\begin{aligned} g''_{\text{dbl}} &= X_1^2(3cx_Q) - Y_1^2 + 3Z_1^2 - 2Y_1Z_1y_Q \\ &= E \cdot (3cx_Q) - A + 3B - 2C \cdot y_Q \end{aligned} \tag{15}$$

where $E = X_1^2$.

Assume that $3cx_Q$ is precomputed. If Q is chosen according to the discussion at the start of this section, then multiplication with $3cx_Q$ or with y_Q counts as $(k/2)\mathbf{m}$.

The point doubling with line computation needs $(k+3)\mathbf{m}+5\mathbf{s}$ if k is even. In this operation count we have further exploited an additional $\mathbf{m}/\mathbf{s}$ tradeoff when calculating $2X_1Y_1$ in the point doubling formulas, which can now be computed as $(X_1+Y_1)^2-E-A$.

See Appendix D for furter justifications and details on the operation scheduling.

Point addition with line computation. Given $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : Z_2)$ with $Z_1 \neq 0$ and $Z_2 \neq 0$ and $(X_1 : Y_1 : Z_1) \neq (X_2 : Y_2 : Z_2)$, an addition can be performed as $(X_1 : Y_1 : Z_1) + (X_2 : Y_2 : Z_2) = (X_3 : Y_3 : Z_3)$ where

$$\begin{aligned} X_3 &= (X_1Z_2 - Z_1X_2)(Z_1Z_2(Y_1Z_2 - Z_1Y_2)^2 - c(X_1Z_2 + Z_1X_2)(X_1Z_2 - Z_1X_2)^2), \\ Y_3 &= (Y_1Z_2 - Z_1Y_2)(c(2X_1Z_2 + Z_1X_2)(X_1Z_2 - Z_1X_2)^2 - Z_1Z_2(Y_1Z_2 - Z_1Y_2)^2) - \\ &\quad cY_1Z_2(X_1Z_2 - Z_1X_2)^3, \\ Z_3 &= cZ_1Z_2(X_1Z_2 - Z_1X_2)^3. \end{aligned} \tag{16}$$

These formulas are derived from (1) and (2) in Section 3. Point addition without line computation needs $12\mathbf{m}+2\mathbf{s}+1\mathbf{c}$ if Z_2 is arbitrary and $9\mathbf{m}+2\mathbf{s}+1\mathbf{c}$ if $Z_2 = 1$. Note that $\mathbf{c}$ stands for a multiplication with c.

The line formula derived from (12) is given by

$$\begin{aligned} g''_{\mathrm{add}} &= (Y_1Z_2 - Z_1Y_2)(X_2 - x_QZ_2) - \\ &\quad (X_1Z_2 - Z_1X_2)Y_2 + (X_1Z_2 - Z_1X_2)Z_2y_Q. \end{aligned} \tag{17}$$

Assuming that Q is chosen according to the discussion at the start of this section, multiplication with $(X_2 - x_QZ_2)$ or with Z_2y_Q counts as $(k/2)\mathbf{m}$ each. Assume that $Z_2 = 1$. Point addition with line computation needs $(k+10)\mathbf{m} + 2\mathbf{s} + 1\mathbf{c}$ if k is even. Assume that Z_2 is arbitrary. Assume that $(X_2 - x_QZ_2)$ and Z_2y_Q are precomputed. The point addition with line computation needs $(k+13)\mathbf{m} + 2\mathbf{s} + 1\mathbf{c}$ if k is even.

The algorithm that we use for the point addition part is a slightly modified version of Cohen/Miyaji/Ono algorithm [12]. We omit details here and refer to Appendix D for justifications and details on the operation scheduling.

5 Curve Generation

This section discusses generating pairing-friendly curves of the form $y^2 = cx^3+1$. We also point out a minor adjustment to be made to the pairing definition when employing this curve in the supersingular setting.

Implementing the Tate pairing on the curve $y^2 = cx^3 + 1$ requires the construction of the j-invariant zero curve $y^2 = x^3 + b$ where $b = c^2$ for $c \in \mathbb{F}_q$. All j-invariant zero curves have a special endomorphism ring and such curves have CM discriminant $D = 3$. In Construction 6.6 of [13], Freeman *et al.* extend on the results of Barreto *et al.* [5] and Brezing and Weng [11] to efficiently construct $D = 3$ curves for all values of k where $18 \nmid k$. Freeman *et al.* discuss

that this construction achieves the best ρ-value curve families for the majority of embedding degrees $k \leq 50$.

Our experiments showed that for most embedding degrees this method of construction will efficiently produce a curve of the desired form with the best ρ-value, however the extra condition we impose on the curve constant (being a quadratic residue) is restrictive. For instance, we were unable to obtain a $k = 8$ curve with b as a square using this construction. For $k = 12$, constructing the curve $y^2 = cx^3 + 1$ gives $\rho \approx 3/2$, which is significantly larger than what can be obtained for BN curves [11] where b is non-square, for which D is also 3 but which have the optimal ρ-value of $\rho = 1$.

Nevertheless, there is a wide range of useful embedding degrees that would welcome the speedups offered on the curve $y^2 = cx^3 + 1$. We present two pairing-friendly examples of the curve using Construction 6.6 of [13].

$k = 12,\ \rho \approx 3/2,\ c = 1,$
$q =$ `0x55555583E6AAB5415B22F364648CF7D4A1A9716C687F053\`
`39126A5FC2A09` (239 bits),
$r =$ `0x10000005D24000CB530E5C544B4E84E5B34F41BD1` (161 bits),
$t =$ `0x1000000174A` (41bits).

$k = 24,\ \rho \approx 5/4,\ c = 3,$
$q =$ `0x577380D96AF284FCF9200C2CC966EC756D86B4CBF2A3AAD\`
`3C1` (199 bits),
$r =$ `0x105121CA61CB6CAF9EF3A835A4442784FFF816AF1` (161 bits),
$t =$ `0x100A0F` (21 bits).

Supersingular curves. When the characteristic of the underlying field is $p \equiv 2 \mod 3$, the curve $y^2 = cx^3 + 1$ is supersingular with $k = 2$. We would usually define the symmetric pairing as $\hat{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $\hat{e}(P, Q) = e(P, \phi(Q))$ and ϕ is the distortion map $\phi(x, y) = (\xi x, y)$ for some non-trivial cube root of unity $\xi \in \mathbb{F}_{p^2}$. However, using the distortion map in this manner would not allow the use of the formulas derived in §4, since these formulas were derived under the assumption that it was the y-coordinate of the second argument in the pairing that was in the extension field. Instead, we follow Scott's technique [23] and define the supersingular pairing as $\tilde{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ where $\tilde{e}(P, Q) = e(P, \theta(Q))$ and θ is defined as $\theta(Q) = \phi(Q) - \pi_p(\phi(Q))$, where π_p is the p-power Frobenius endomorphism. For $Q = (x_Q, y_Q)$, we have that $\pi_p(\phi(Q)) = \pi_p(\xi x_Q, y_Q) = (\xi^2 x_Q, y_Q)$ so that $\theta(Q)$ becomes $\theta(x_Q, y_Q) = (\xi x_Q, y_Q) - (\xi^2 x_Q, y_Q)$. The map θ is an isomorphism from the base field subgroup to the trace zero subgroup, where the x-coordinates lie in the base field and the y-coordinates are in the extension field so that we can apply the formulas from §4 [23]. The inverse map from the trace zero subgroup to the base field subgroup is defined as $\theta^{-1}(Q) = \mathrm{Tr}(\phi(Q))$, where Tr is the trace map.

6 Comparison and conclusion

We have studied pairing computations on a non-standard Weierstrass curve of the form $y^2 = cx^3 + 1$. This is the most specific curve model studied so far since there are only 3 isomorphism classes of curves for this shape in the general case where $p \equiv 1 \bmod 3$. The main contribution of this paper is a faster computation of the Tate pairing on this special curve. Practical examples of such curves can be achieved using Construction 6.6 of [13]. There are many examples of embedding degrees for which this construction gives the best known ρ-value [13], however it remains an open question to find suitable curves of this form having ρ-values very close to 1 with practically interesting embedding degrees, e.g. $k = 8$.

The following table summarizes the advantage of employing this new curve in the Tate pairing by comparing our results with the fastest results achieved on other j-invariant zero curves documented prior to this work. The formulas given by Arène *et al.* [1] for j-invariant zero curves give an operation count that improves the operation count originally presented in [16], so we draw comparisons against these improved formulas below. We follow the trend of presenting the operation count for even k [18], since this is generally preferred in practice [4], [7]. We do not include the multiplications and squarings that take place in the extension field $\mathbb{F}_{q^k}$, since these are common to all operation counts (see lines 5 and 9 of Algorithm 1).

Tate pairing	DBL	mADD	ADD
Arène *et al.* [1]	$(k+3)\mathbf{m} + 8\mathbf{s}$	$(k+6)\mathbf{m} + 6\mathbf{s}$	$(k+12)\mathbf{m} + 5\mathbf{s}$
This work	$(k+3)\mathbf{m} + 5\mathbf{s}$	$(k+10)\mathbf{m} + 2\mathbf{s} + 1\mathbf{c}$	$(k+13)\mathbf{m} + 2\mathbf{s} + 1\mathbf{c}$

As k gets large in the Tate pairing, the overall speed up that is achieved through using the curve $y^2 = cx^3 + 1$ becomes less, since the more difficult operations in $\mathbb{F}_{q^k}$ consume more computation relative to those operations in the base field.

Lastly, we note that the EFD [8] reports $2\mathbf{m} + 5\mathbf{s}$ point doubling formulas in Jacobian coordinates for j-invariant zero curves. Therefore a protocol requiring scalar multiplications should use Jacobian coordinates and should only switch to our proposal when the pairing is being computed. This conversion comes at the cost of $2\mathbf{m} + 1\mathbf{s} + 1\mathbf{c}$ by taking $(X : Y : Z)$ in Jacobian coordinates to $(XZ : Y : cZ^3)$ in homogeneous projective coordinates on the curve $y^2 = cx^3 + 1$.

7 Acknowledgements

The authors wish to thank Tanja Lange and the anonymous referees for helpful comments and corrections.

References

[1] C. Arène, T. Lange, M. Naehrig, and C. Ritzenthaler. Faster pairing computation. Cryptology ePrint Archive, Report 2009/155, 2009. http://eprint.iacr.org/2009/155.

[2] P. S. L. M. Barreto, S. D. Galbraith, C. Ó' Héigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. Cryptology ePrint Archive, Report 2004/375, 2004. http://eprint.iacr.org/2004/375.

[3] P. S. L. M. Barreto, S. D. Galbraith, C. Ó' Héigeartaigh, and M. Scott. Efficient pairing computation on supersingular Abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, 2007.

[4] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO 2002*, volume 2442 of *LNCS*, pages 354–369. Springer, 2002.

[5] P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. In *Security in Communication Networks*, volume 2576 of *LNCS*, pages 257–267. Springer, 2003.

[6] P. S. L. M. Barreto, B. Lynn, and M. Scott. Efficient implementation of pairing-based cryptosystems. *Journal of Cryptology*, 17(4):321–334, 2004.

[7] P. S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In *SAC 2003*, volume 3006 of *LNCS*, pages 17–25. Springer, 2004.

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

[9] D. Boneh and M. K. Franklin. Identity-based encryption from the Weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[10] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. *Journal of Cryptology*, 17(4):297–319, 2004.

[11] F. Brezing and A. Weng. Elliptic curves suitable for pairing based cryptography. *Des. Codes Cryptography*, 37(1):133–141, 2005.

[12] 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.

[13] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372, 2006. http://eprint.iacr.org/2006/372.

[14] S. D. Galbraith. *Pairings*, volume 317 of *London Mathematics Society Lecture Note Series*, pages 183–213. Cambridge University Press, 2005.

[15] S. D. Galbraith and M. Scott. Exponentiation in pairing-friendly groups using homomorphisms. In *Pairing 2008*, volume 5209 of *LNCS*, pages 211–224. Springer, 2008.

[16] F. Hess, N. P. Smart, and F. Vercauteren. The Eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[17] A. Joux. A one round protocol for tripartite Diffie-Hellman. *Journal of Cryptology*, 17(4):263–276, 2004.

[18] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In *Cryptography and Coding*, volume 3796 of *LNCS*, pages 13–36. Springer, 2005.

[19] E. Lee, H.-S. Lee, and C.-M. Park. Efficient and generalized pairing computation on Abelian varieties. Cryptology ePrint Archive, Report 2008/040, 2008. `http://eprint.iacr.org/2008/040`.

[20] S. Matsuda, N. Kanayama, F. Hess, and E. Okamoto. Optimised versions of the Ate and twisted Ate pairings. In *Cryptography and Coding*, volume 4887 of *LNCS*, pages 302–312. Springer, 2007.

[21] V. S. Miller. The Weil pairing, and its efficient calculation. *Journal of Cryptology*, 17(4):235–261, 2004.

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

[23] M. Scott. Faster identity based encryption. *Electronics Letters*, 40(14):861–862, 2004.

[24] M. Scott. Faster pairings using an elliptic curve with an efficient endomorphism. In *INDOCRYPT 2005*, volume 3797 of *LNCS*, pages 258–269. Springer, 2005.

[25] F. Vercauteren. Optimal pairings. Cryptology ePrint Archive, Report 2008/096, 2008. `http://eprint.iacr.org/2008/096`.

[26] C.-A. Zhao, F. Zhang, and J. Huang. A note on the Ate pairing. Cryptology ePrint Archive, Report 2007/247, 2007. `http://eprint.iacr.org/2007/247`.

A Appendix

This Maple script verifies that (3) and (4) commute with the original point doubling formulas.

```
b:=c^2: W:=(x,y)->y^2-(x^3+a*x+b): #The short Weierstrass curve, W.
L:=(3*x1^2+a)/(2*y1): x3:=L^2-2*x1: y3:=L*(x1-x3)-y1: #Double on W.
mu:=(y1+3*c)/(2*y1): sigma:=(a-3*x1^2)/(2*y1)^2: #Double on W with new formulas.
delta:=(3*x1*(y1-3*c)*(y1+3*c)-a*(9*x1^2+a))/(2*y1)^3: #Double on W with new formulas.
x3new:=x1*(mu-mu^2)+ a*sigma: y3new:=(y1-c)*mu^3+a*delta-c: #Double on W with new formulas.
simplify(x3-x3new,[W(x1,y1)]); simplify(y3-y3new,[W(x1,y1)]); #Check.
```

B Appendix

This Maple script verifies that (5), (6), (7), and (8) commute with the original doubling and addition formulas.

```
Q:=(x,y)->y^2-(c*x^3+1): #The curve considered in this work, Q.
W:=(u,v)->v^2-(u^3+c^2): #The short Weierstrass curve, W.
QtoW:=(x,y)->c*x,(x,y)->c*y: #The map from Q to W.
WtoQ:=(u,v)->u/c,(u,v)->v/c: #The map from W to Q.
##Verify the correctness of point additon formulas.
u1,v1:=QtoW(x1,y1): u2,v2:=QtoW(x2,y2): #Map the points (x1,y1) and (x2,y2) on Q to W.
L:=(v1-v2)/(u1-u2): u3:=L^2-u1-u2: v3:=L*(u1-u3)-v1: #Add on W with the original formulas.
x3,y3:=WtoQ(u3,v3): #Map the sum (u3,v3) on W to Q.
simplify(W(u3,v3),[Q(x1,y1),Q(x2,y2)]); #Check.
Lnew:=(y1-y2)/(x1-x2): x3new:=c^(-1)*Lnew^2-x1-x2: y3new:=Lnew*(x1-x3)-y1: ##Add on Q.
simplify(x3-x3new,[Q(x1,y1),Q(x2,y2)]); simplify(y3-y3new,[Q(x1,y1),Q(x2,y2)]); #Check.
unassign('Lnew','L','u2','v2','u3','v3','x3','y3','x3new','y3new');
##Verify the correctness of point doubling formulas.
L:=3*u1^2/(2*v1): u3:=L^2-2*u1: v3:=L*(u1-u3)-v1: #Double on W with the original formulas.
x3,y3:=WtoQ(u3,v3): #Map the sum (u3,v3) on W to Q.
simplify(W(u3,v3),[Q(x1,y1)]); #Check.
mu:=(y1+3)/(2*y1): x3new:=x1*(mu-mu^2): y3new:=(y1-1)*mu^3-1: #Double on Q.
simplify(x3-x3new,[Q(x1,y1)]); simplify(y3-y3new,[Q(x1,y1)]); #Check.
```

C Appendix

This Maple script verifies the correctness of (9), (10), and (11).

```
Q:=(x,y)->y^2-(c*x^3+1): #The curve considered in this work, Q.
W:=(u,v)->v^2-(u^3+c^2): #The short Weierstrass curve, W.
QtoW:=(x,y)->c*x,(x,y)->c*y: #The maps from Q to W.
WtoQ:=(u,v)->u/c,(u,v)->v/c: #The maps from W to Q.
##Verify the correctness of the line formulas for addition.
u1,v1:=QtoW(x1,y1): u2,v2:=QtoW(x2,y2): uQ,vQ:=QtoW(xQ,yQ): ##(xi,yi) on Q to (ui,vi) on W.
L:=(v1-v2)/(u1-u2): l:=L*(u1-uQ)+vQ-v1: v:=uQ-(L^2-u1-u2): #Compute the addition-line on W.
Lnew:=(y1-y2)/(x1-x2): gadd:=c*(Lnew*(x2-xQ)-y2+yQ)/(c*(x1+x2+xQ)-Lnew^2): #New line on Q.
simplify(l/v-gadd,[Q(x1,y1),Q(x2,y2),Q(xQ,yQ)]); #Check.
##Verify the correctness of the line formulas for doubling.
L:=3*u1^2/(2*v1): l:=L*(u1-uQ)+vQ-v1: v:=uQ-(L^2-2*u1): #Compute the doubling-line on W.
gdbl:=2*c*y1*(x1-xQ)^2/(x1^2*(3*c*xQ)-y1^2+3+2*y1*yQ): #New line on Q.
simplify(l/v-gdbl,[Q(x1,y1),Q(xQ,yQ)]); #Check.
##Verify the correctness of the line formulas for the sum of negatives.
l:=uQ-u1: v:=1: #The vertical line on W.
gvert:=-c*(x1-xQ): #The new line on Q.
simplify(l/v-gvert,[Q(x1,y1),Q(x2,y2),Q(xQ,yQ)]); #Check.
```

D Appendix

This Maple script verifies the correctness of (14) and (15).

```
Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3): x1:=X1/Z1: y1:=Y1/Z1:
x3:=x1*(y1^2-9)/(2*y1)^2: y3:=(y1-1)*(y1+3)^3/(2*y1)^3-1:
Line:=x1^2*(3*c*xQ)-y1^2+3-2*y1*yQ:
```

```
##Point doubling formulas in homogenous projective coordinates.
X3:=2*X1*Y1*(Y1^2-9*Z1^2):
Y3:=(Y1-Z1)*(Y1+3*Z1)^3-8*Z1*Y1^3:
Z3:=(2*Y1*Z1)*(2*Y1)^2:
gDBL:=X1^2*(3*c*xQ)-Y1^2+3*Z1^2-2*Y1*Z1*yQ: #Line formulas.
simplify(x3-X3/Z3,[Q(X1,Y1,Z1)]); simplify(y3-Y3/Z3,[Q(X1,Y1,Z1)]); #Check.
factor(Line-gDBL/Z1^2); #Check.
```

This Maple script shows how to schedule operations for (14). The point doubling without line computation needs $4\mathbf{m} + 3\mathbf{s} + 0\mathbf{c}$.

```
Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
##Point doubling formulas with register allocations.
X3:=2*X1: X3:=X3*Y1: Z3:=3*Z1: t1:=Y1+Z3: t1:=t1^2: Y3:=Y1^2: Z3:=Z3^2: t2:=Y3-Z3:
t2:=3*t2: X3:=X3*t2: t2:=t2+Z3: t2:=t2+Z3: Z3:=Y3+Z3: Z3:=t1-Z3: t2:=t2+Z3: Z3:=Y3*Z3:
Z3:=4*Z3: Y3:=t1*t2: Y3:=Y3-Z3:
simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1)]); #Check.
```

This Maple script shows how to schedule operations for (14) and (15). Multiplication with `c1` or with y_Q counts as $(k/2)\mathbf{m}$. Assume that `c1` is precomputed. The point doubling with line computation needs $5\mathbf{m} + 5\mathbf{s}$ if $k = 2$ or more generally $(k + 3)\mathbf{m} + 5\mathbf{s}$ if k is even.

```
Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
Line:=X1^2*(3*c*xQ)-Y1^2+3*Z1^2-2*Y1*Z1*yQ:
c1:=3*c*xQ: #Precomputed value.
##Point doubling formulas and line computation with register allocations.
t1:=X1+Y1: t2:=Y1+Z1: t1:=t1^2: t2:=t2^2: X3:=X1^2: Y3:=Y1^2: Z3:=Z1^2: t1:=t1-X3:
t1:=t1-Y3: t2:=t2-Y3: t2:=t2-Z3: Z3:=3*Z3: t3:=Y3-Z3: gDBL:=X3*c1-t3-t2*yQ:
t3:=t3+t2: t4:=3*Z3: X3:=Y3-t4: X3:=t1*X3: t1:=3*t2: t2:=t1+t2:  Z3:=t2*Y3:
Y3:=Y3+t4: t1:=t1+Y3: Y3:=t3*t1: Y3:=Y3-Z3:
simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1)]); simplify(Line-gDBL); #Check.
```

This Maple script verifies the correctness of (16) and (17).

```
Q1:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3): x1:=X1/Z1: y1:=Y1/Z1: x2:=X2/Z2: y2:=Y2/Z2:
L:=(y1-y2)/(x1-x2): x3:=c^(-1)*L^2-x1-x2: y3:=L*(x1-x3)-y1:
Line:=(y1-y2)*(x2-xQ)-(x1-x2)*(y2-yQ):
##Point addition formulas in homogenous projective coordinates.
X3:=(X1*Z2-Z1*X2)*(Z1*Z2*(Y1*Z2-Z1*Y2)^2-c*(X1*Z2+Z1*X2)*(X1*Z2-Z1*X2)^2):
Y3:=(Y1*Z2-Z1*Y2)*(c*(2*X1*Z2+Z1*X2)*(X1*Z2-Z1*X2)^2-Z1*Z2*(Y1*Z2-Z1*Y2)^2) -
    c*Y1*Z2*(X1*Z2-Z1*X2)^3:
Z3:=c*Z1*Z2*(X1*Z2-Z1*X2)^3:
gADD:=(Y1*Z2-Z1*Y2)*(X2-xQ*Z2)-(X1*Z2-Z1*X2)*Y2+(X1*Z2-Z1*X2)*Z2*yQ: #Line formulas.
simplify(x3-X3/Z3,[Q1(X1,Y1,Z1),Q1(X2,Y2,Z2)]); #Check.
simplify(y3-Y3/Z3,[Q1(X1,Y1,Z1),Q1(X2,Y2,Z2)]); factor(Line-gADD/Z1/Z2^2); #Check.
```

This Maple script shows how to schedule operations for (16) and (17) with $Z_2 = 1$.

```
Z2:=1: Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
Line:=(Y1*Z2-Z1*Y2)*(X2-xQ*Z2)-(X1*Z2-Z1*X2)*(Y2-yQ*Z2):
c1:=X2-xQ: c2:=Y2-yQ: #Precomputed values.
##Point addition formulas and line computation with register allocations.
t1:=Z1*X2: t1:=X1-t1: t2:=Z1*Y2: t2:=Y1-t2: gADD:=c1*t2-t1*Y2+t1*yQ:
t3:=t1^2: t3:=c*t3: X3:=t3*X1: t3:=t1*t3: t4:=t2^2: t4:=t4*Z1: t4:=t3+t4:
t4:=t4-X3: t4:=t4-X3: X3:=X3-t4: t2:=t2*X3: Y3:=t3*Y1: Y3:=t2-Y3: X3:=t1*t4: Z3:=Z1*t3:
simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1),Q(X2,Y2,Z2)]); simplify(Line-gADD); #Check.
```

This Maple script shows how to schedule operations for (16) and (17).

```
Q:=(X,Y,Z)->Y^2*Z-(c*X^3+Z^3):
Line:=(Y1*Z2-Z1*Y2)*(X2-xQ*Z2)-(X1*Z2-Z1*X2)*(Y2-yQ*Z2):
c1:=X2-xQ*Z2: c2:=Y2-yQ*Z2: #Precomputed values.
##Point addition formulas and line computation with register allocations.
t1:=Z1*X2: X3:=X1*Z2: t1:=X3-t1: t2:=Z1*Y2: Y3:=Y1*Z2: t2:=Y3-t2:
gADD:=c1*t2-t1*Y2+t1*Z2*yQ:
Z3:=Z1*Z2: t3:=t1^2: t3:=c*t3: X3:=t3*X3: t3:=t1*t3: t4:=t2^2: t4:=t4*Z3: t4:=t3+t4:
t4:=t4-X3: t4:=t4-X3: X3:=X3-t4: t2:=t2*X3: Y3:=t3*Y3: Y3:=t2-Y3: X3:=t1*t4: Z3:=Z3*t3:
simplify(Q(X3,Y3,Z3),[Q(X1,Y1,Z1),Q(X2,Y2,Z2)]); simplify(Line-gADD); #Check.
```