Faster isogeny-based compressed key agreement

Gustavo H. M. Zanon[1], Marcos A. Simplicio Jr[1],
Geovandro C. C. F. Pereira[2], Javad Doliskani[2], and
Paulo S. L. M. Barreto[3]

[1] Escola Politécnica, University of São Paulo
{gzanon,msimplicio}@larc.usp.br
[2] Institute for Quantum Computing, University of Waterloo
{geovandro.pereira,javad.doliskani}@uwaterloo.ca
[3] University of Washington Tacoma
pbarreto@uw.edu

Abstract. Supersingular isogeny-based cryptography is one of the more recent families of post-quantum proposals. An interesting feature is the comparatively low bandwidth occupation in key agreement protocols, which stems from the possibility of key compression. However, compression and decompression introduce a significant overhead to the overall processing cost despite recent progress. In this paper we address the main processing bottlenecks involved in key compression and decompression, and suggest substantial improvements for each of them. Some of our techniques may have an independent interest for other, more conventional areas of elliptic curve cryptography as well.

1 Introduction

In the Supersingular Isogeny Diffie-Hellman (SIDH) protocol [9], the two parties need to exchange a representation of their public keys each consisting of an elliptic curve E together with two points P, Q on E. The curve E is supersingular and is defined over an extension field $\mathbb{F}_{p^2}$ for a prime of the form $p = \ell_A^m \ell_B^n - 1$ where ℓ_A, ℓ_B are small primes, usually equal to 2 and 3, respectively. Originally, this exchange was done using triples of the form (E, x_P, x_Q) where $E : y^2 = x^3 + ax + b$ and x_P, x_Q are the abscissas of P and Q. Some extra bits were also needed to recover the correct y-coordinates. Thus, the public keys are transfered using essentially the four elements $a, b, x_P, x_Q \in \mathbb{F}_{p^2}$ which require $8 \log p$ bits.

A different representation of the SIDH public keys was proposed by [1] that reduced the size to $4 \log p$ bits. The idea was to first represent the curve E using its j-invariant, which is an element of $\mathbb{F}_{p^2}$, rather than the coefficients a, b. This way E is represented using $2 \log p$ bits. The isomorphism class of an elliptic curve is uniquely determined by its j-invariant. Second, since the points P, Q are always in the torsion subgroups $E[\ell_A^m]$ or $E[\ell_B^n]$, they can be represented using elements of $\mathbb{Z}_t \oplus \mathbb{Z}_t$ where either $t = \ell_A^m$ or $t = \ell_B^n$. Since the parameters are such that $\ell_A^m \approx \ell_B^n$, a pair $(t_1, t_2) \in \mathbb{Z}_t \oplus \mathbb{Z}_t$ is represented using $2 \log p$ bits. This reduction of size of the public keys, however, comes with a rather high computational overhead. The conversion between the coefficients a, b of a curve E and its j-invariant is done efficiently; the expensive part is the conversion between elements of $\mathbb{Z}_t \oplus \mathbb{Z}_t$ and the points P, Q. As reported in [1], the compression

phase for each party was slower than a full round of uncompressed key exchange by a factor of more than 10 times.

Costello *et al.* [5] further improved the key compression scheme by reducing the public key sizes to $3.5 \log p$ bits and decreasing the runtime by almost an order of magnitude. With this improvement, the key compression phase for each party is as fast as one full round of uncompressed key exchange. This certainly motivates the idea of including the compression and decompression phases as default parts of SIDH. However, compared to the currently deployed (classical) schemes, the compression/decompression runtime is not acceptable.

Our contributions. We propose new algorithms that further decrease the runtime of SIDH compression and decompression. In contrast to the previous works which have deployed "known" algorithms to optimize the performance of key compression, some of the algorithms presented here are new and of broader interest than isogeny-based crypto. A summary of the improvements are as follows.

- Constructing torsion basis. Assuming the usual parameters $\ell_A = 2, \ell_B = 3$, we improve basis generation for both $E[2^m]$ and $E[3^n]$. To generate a basis for the 2^m-torsion, we propose an algorithm dubbed ***entangled basis*** generation. This algorithm is around 15.9× faster than the usual basis generation presented in [5]. For the 3^n-torsion, we observed that the naive algorithm is more efficient (both in theory and practice) than the explicit 3-descent of [12] used by Costello *et al.* [5].
- Computing discrete logarithms. Inspired by the ***optimal strategy*** method of [6] to compute smooth degree isogenies, we propose an algorithm to compute discrete logs in the group μ_{ℓ^n} where ℓ is a small prime. For a window of size $w = 6$, our algorithm is 3.9× and 4.6× faster than the algorithm used by [5] for the groups $\mu_{2^{372}}$ and $\mu_{3^{239}}$ respectively.
- Pairing computation. We exploit the special shapes of pairs of points generated as *entangled bases* and the existence of a subfield dismissed by [5] to optimize the Tate pairing. We achieve a speedup of 1.4× for the pairing phase over the algorithms used by [5] for both binary and ternary pairings.
- Other improvements. We introduce a *reverse basis decomposition*, which combined with the previous improvements, allows for further optimizations of compression and decompression. For example each party only needs to compute 4 pairings rather than 5. Also, two expensive cofactor multiplications by 3^n are saved during Bob's compression, and one cofactor multiplication by 3^n is saved during Alice's decompression.

We have implemented the above improvements on top of (the then-latest version of) the Microsoft SIDH library [10]. The library is designed for the specific prime $p = 2^{372}3^{239} - 1$ of size $\log p = 751$ bits. Our software can be found at https://github.com/geovandro/PQCrypto-SIDH.

1.1 Notations and conventions

For simplicity, we assume that finite field arithmetic is carried out in a base field $\mathbb{F}_p$ and its quadratic extension $\mathbb{F}_{p^2}$ for a prime p of form $p := 2^m \cdot 3^n - 1$ for some $m > 2$

and $n > 1$, so that $p \equiv 3 \pmod 4$. The quadratic extension $\mathbb{F}_{p^2}/\mathbb{F}_p$ is represented as $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2 + 1\rangle$, and arithmetic closely mimics that of the complex numbers.

All curves are represented using the Montgomery model unless otherwise specified. We follow the convention of using subscripts A and B for Alice and Bob, respectively. For example, the secret isogeny ϕ_A is computed by Alice and her public parameters are denoted by the points P_A, Q_A and the curve E_A. Similarly, Bob's isogeny is denoted by ϕ_B, and his public parameters are P_B, Q_B, E_B.

We denote by $\mathbf{i}$, $\mathbf{c}$, $\mathbf{m}$, $\mathbf{s}$, and $\mathbf{a}$ the costs of inverting, cubing, multiplying, squaring, and adding/subtracting/shifting in $\mathbb{F}_p$, respectively, and by $\mathbf{I}$, $\mathbf{C}$, $\mathbf{M}$, $\mathbf{S}$, and $\mathbf{A}$ the costs of the corresponding operations in $\mathbb{F}_{p^2}$. We disregard the cost of changing a sign (for instance, when handling the conjugate of a field element). The costs of the $\mathbb{F}_{p^2}$ operations relative to the costs of operations in $\mathbb{F}_p$ can be approximated by $1\mathbf{I} = 1\mathbf{i}+2\mathbf{m}+2\mathbf{s}+1\mathbf{a}$, $1\mathbf{C} = 2\mathbf{m}+2\mathbf{s}+6\mathbf{a}$, $1\mathbf{M} = 3\mathbf{m}+5\mathbf{a}$, $1\mathbf{S} = 2\mathbf{m}+3\mathbf{a}$, and $1\mathbf{A} = 2\mathbf{a}$, by using the finite-field analogues to well-known Viète multiple-angle trigonometric identities [15, Formulas 5.68 and 5.69].

2 Reverse basis decomposition

In this section, we use *reverse basis decomposition* to speed up both Alice and Bob's key compression by saving one pairing computation. Later in Section 3.1 we show that when combined with an entangled basis generation, this technique will allow for avoiding two cofactor multiplications by 3^n in Bob's key compression and one in Alice's key decompression. We prove our results from an Alice's point of view. The proofs for Bob are similar.

The main previous idea to achieve key compression [1,5] is the following: instead of transmitting points $\phi_A(P_B), \phi_A(Q_B) \in E_A[3^n]$, which are represented by two abscissas in $\mathbb{F}_{p^2}$ and consume $4\log p$ bits, Alice computes a canonical basis $R_1, R_2 \in E_A[3^n]$ and expresses the expanded public key in that basis as $\phi_A(P_B) = a_0R_1 + b_0R_2$ and $\phi_A(Q_B) = a_1R_1 + b_1R_2$. In matrix notation,

$$\begin{bmatrix}\phi_A(P_B)\\ \phi_A(Q_B)\end{bmatrix} = \begin{bmatrix}a_0 & b_0\\ a_1 & b_1\end{bmatrix}\begin{bmatrix}R_1\\ R_2\end{bmatrix}. \tag{1}$$

This representation consists of four smaller integers $(a_0, b_0, a_1, b_1) \in (\mathbb{Z}/3^n\mathbb{Z})^4$ of total size $2\log p$ bits as suggested in [1]. This was improved in [5] by transmitting only the triple $(a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1) \in (\mathbb{Z}/3^n\mathbb{Z})^3$ or $(b_0^{-1}a_0, b_0^{-1}a_1, b_0^{-1}b_1) \in (\mathbb{Z}/3^n\mathbb{Z})^3$ depending on whether a_0 or b_0 is invertible. Therefore, only $3/2\log p$, plus one bit indicating the invertibility of a_0 or b_0 modulo 3^n, is needed. In the above mentioned techniques, the coefficients a_0, b_0, a_1, b_1 can be computed using five Tate pairings given by

$$\begin{aligned}
g_0 &= e_{3^n}(R_1, R_2)\\
g_1 &= e_{3^n}(R_1, \phi_A(P_B)) = e_{3^n}(R_1, a_0R_1 + b_0R_2) = g_0^{b_0}\\
g_2 &= e_{3^n}(R_1, \phi_A(Q_B)) = e_{3^n}(R_1, a_1R_1 + b_1R_2) = g_0^{b_1}\\
g_3 &= e_{3^n}(R_2, \phi_A(P_B)) = e_{3^n}(R_2, a_0R_1 + b_0R_2) = g_0^{-a_0}\\
g_4 &= e_{3^n}(R_2, \phi_A(Q_B)) = e_{3^n}(R_2, a_1R_1 + b_1R_2) = g_0^{-a_1}.
\end{aligned} \tag{2}$$

From this, Alice can recover a_0, b_0, a_1, and b_1 by solving discrete logs in a multiplicative subgroup of smooth order 3^n using the Pohlig-Hellman algorithm.

Now since $\phi_A(P_B)$ and $\phi_A(Q_B)$ also form a basis for $E_A[3^n]$, we see that the coefficient matrix in (1) is invertible modulo 3^n. So, we can write

$$\begin{bmatrix} R_1 \\ R_2 \end{bmatrix} = \begin{bmatrix} c_0 & d_0 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} \phi_A(P_B) \\ \phi_A(Q_B) \end{bmatrix} \tag{3}$$

by inverting the matrix in (1). Changing the roles of the bases $\{R_1, R_2\}$ and $\{\phi_A(P_B), \phi_A(Q_B)\}$ in (2) we get

$$\begin{aligned} h_0 &= e_{3^n}(\phi_A(P_B), \phi_A(Q_B)) \\ h_1 &= e_{3^n}(\phi_A(P_B), R_1) = e_{3^n}(\phi_A(P_B), c_0\phi_A(P_B) + d_0\phi_A(Q_B)) = h_0^{d_0} \\ h_2 &= e_{3^n}(\phi_A(P_B), R_2) = e_{3^n}(\phi_A(P_B), c_1\phi_A(P_B) + d_1\phi_A(Q_B)) = h_0^{d_1} \\ h_3 &= e_{3^n}(\phi_A(Q_B), R_1) = e_{3^n}(\phi_A(Q_B), c_0\phi_A(P_B) + d_0\phi_A(Q_B)) = h_0^{-c_0} \\ h_4 &= e_{3^n}(\phi_A(Q_B), R_2) = e_{3^n}(\phi_A(Q_B), c_1\phi_A(P_B) + d_1\phi_A(Q_B)) = h_0^{-c_1}. \end{aligned} \tag{4}$$

The first pairing in (4) is computed as $h_0 = e_{3^n}(P_B, \hat{\phi}_A \circ \phi_A(Q_B)) = e_{3^n}(P_B, [\deg \phi_A]Q_B) = e_{3^n}(P_B, Q_B)^{2^m}$, which only depends on the public parameters P_B, Q_B and m. Therefore, it can be computed once and for all and be included in the public parameters. In particular, only the pairings h_1, h_2, h_3 and h_4 need to be computed at runtime. The discrete logs are computed as before using Pohlig-Hellman, yielding $c_0 = -\log_{h_0} h_3, d_0 = \log_{h_0} h_1, c_1 = -\log_{h_0} h_4$ and $d_1 = \log_{h_0} h_2$. Next, Alice inverts the computed coefficients matrix of (3) to obtain the coefficient matrix of (1). Explicitly,

$$\begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} = \frac{1}{D} \begin{bmatrix} d_1 & -d_0 \\ -c_1 & c_0 \end{bmatrix}$$

where $D = c_0d_1 - c_1d_0$. Notice that the extra inversion of D^{-1} does not need to be carried out when using the technique in [5]. More precisely, since at least one of d_0 and d_1, say d_1, is invertible modulo 3^n, Alice transmits the tuple

$$\begin{aligned} (a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1) &= (-d_1^{-1}DD^{-1}d_0, -d_1^{-1}DD^{-1}c_1, d_1^{-1}DD^{-1}c_0) \\ &= (-d_1^{-1}d_0, -d_1^{-1}c_1, d_1^{-1}c_0) \end{aligned}$$

which is independent of D.

3 Entangled basis generation

We now introduce a technique to create a complete basis of the 2^m-torsion from a single (albeit specific) point of order 2^m. In other words, the cost involved is essentially that of creating a generator for a single subgroup of order 2^m in $E[2^m]$: a generator for the linearly independent subgroup becomes immediately available almost for free. Consequently, the linear independence test consisting of two scalar multiplications by 2^{m-1} can be avoided. This is akin to distortion maps even though none is typically available for the curves involved in SIDH. We call the resulting bases "entangled" by analogy with the quantum phenomenon whereby the properties of one entity are entirely determined by the properties of another entity despite their separation[4].

[4] We stress, however, that here the naming is purely analogous: there is no quantum process involved in the construction.

In order to build an entangled basis $\langle P, Q\rangle = E[2^m]$ for $E : y^2 = x^3 + Ax^2 + x$, we somewhat "subvert" the original Elligator 2 formulas [3] under a different motivation than encoding points to random strings: obtaining two linearly independent points on E at once. Herein the value $t := ur^2$, for $u \in \mathbb{F}_q$ and $r \in \mathbb{F}_p^*$ will be a square rather than a non-square. The new construction is proved in Theorem 1.

Theorem 1. *Given a Montgomery elliptic curve $E_A(\mathbb{F}_q) : y^2 = x(x^2 + Ax + 1)$ where $p = 2^m \cdot 3^n - 1$ and $A \neq 0$, let $t \in \mathbb{F}_q$ be a field element such that $t^2 \in \mathbb{F}_q \setminus \mathbb{F}_p$ and let $x_1 := -A/(1 + t^2)$ be a quadratic non-residue that defines the abscissa of a point $P_1 \in E_A(\mathbb{F}_q)$. Then $x(P_2) := -x_1 - A$ defines the abscissa of a point $P_2 \in E_A(\mathbb{F}_q)$ such that $\langle [h]P_1, [h]P_2\rangle = E_A[2^m]$, where $h := 3^n$ is the cofactor of the 2^m-torsion group.*

Proof. Since $x(P_2) = t^2 x_1$, both abscissas are quadratic non-residues and by [8, Chapter 1 (§4), Theorem 4.1] the two points $P_1 = (x_1, y_1)$, $P_2 = (t^2 x_1, t y_1)$, with $x_1 + t^2 x_1 + A = 0$, are not in $[2]E_A$. So the points $[h]P_1$ and $[h]P_2$ are full 2^m-torsion points. To prove that $h \cdot P_1, h \cdot P_2$ generate $E_A[2^m]$ we have to prove that $[h \cdot 2^{m-1}](P_1 - P_2) \neq 0$, or equivalently that $(u, v) = P_1 + (-P_2) \notin [2]E_A$.

By the addition law [14, Algorithm 2.3] on E_A we get

$$\begin{aligned}
\lambda &= \frac{y_2 - y_1}{x_2 - x_1} = \frac{-ty_1 - y_1}{t^2 x_1 - x_1} = \frac{-(t+1)y_1}{(t^2-1)x_1} = \frac{-y_1}{(t-1)x_1},\\
\mu &= \frac{y_1 x_2 - y_2 x_1}{x_2 - x_1} = \frac{t(t+1)y_1 x_1}{(t+1)(t-1)x_1} = \frac{y_1}{(t-1)x_1} t x_1 = -\lambda t x_1,\\
u &= \lambda^2 - A - x_1 - x_2 = \lambda^2,\\
v &= -\lambda u - \mu = -\lambda u - (-\lambda t x_1) = -\lambda(u - t x_1).
\end{aligned}$$

From the above equalities we see that $v^2 = \lambda^2(u - tx_1)^2 = u(u^2 + Au + 1)$ and hence $u^2 + Au + 1 = (u - tx_1)^2$. Let $w := u - tx_1 = \sqrt{u^2 + Au + 1}$. Then $1 - (u - w)^2 = 1 - t^2 x_1^2 = x_1^2 + Ax_1 + 1$, which is a quadratic non-residue because x_1 is itself a quadratic non-residue while their product is obviously a square, $x_1(x_1^2 + Ax_1 + 1) = y_1^2$. A straightforward calculation shows that $(1 - (u + w)^2)(1 - (u - w)^2) = u^2(A^2 - 4)$. But $A^2 - 4$ is a quadratic residue since E_A has the full 2-torsion over $\mathbb{F}_{p^2}$. Therefore, both $(u \pm w)^2 - 1$ have the same quadratic residuosity, that is, they are both quadratic non-residues by the above.

Now[5] assume by contradiction that $P_1 - P_2 \in [2]E_A$, i.e. that there is a point $(x, y) \in E_A(\mathbb{F}_q)$ such that $[2](x, y) = (u, v)$. From the doubling formula on E_A we get

$$u = \frac{(x^2 - 1)^2}{4x(x^2 + Ax + 1)}.$$

From this we get a quartic equation $(x^2 - 1)^2 - 4ux(x^2 + Ax + 1) = 0$. Since $x \neq 0$, we can divide both sides by x^2 and rearrange some terms to get

$$\left(x + \frac{1}{x}\right)^2 - 4u\left(x + \frac{1}{x}\right) - 4(Au + 1) = 0.$$

[5] This part closely follows the idea behind [8, Chapter 1 (§4), Theorem 4.1].

From this we obtain

$$x + \frac{1}{x} = \frac{4u \pm \sqrt{16(u^2 + Au + 1)}}{2} = \frac{4u \pm 4w}{2} = 2(u \pm w).$$

In turn, from this we get $x^2 - 2(u \pm w)x + 1 = 0$. Again since $x \in \mathbb{F}_q$, the discriminant $4(u \pm w)^2 - 4$, and hence at least one of the $(u \pm w)^2 - 1$ must be a quadratic residue. But this contradicts the earlier observation that $(u \pm w)^2 - 1$ are both quadratic non-residues. Therefore $P_1 - P_2 \notin [2]E$, yielding the claim that $\langle [h]P_1, [h]P_2 \rangle = E_A[2^m]$. □

In practice, one can efficiently implement entangled basis generation as follows. Let $u_0 \in \mathbb{F}_q \backslash \mathbb{F}_p$ such that $u := u_0^2 \in \mathbb{F}_q \backslash \mathbb{F}_p$, e.g. $u_0 = 1 + i$ and $u = 2i$. Define two separate tables of pairs (r, v) with $v := 1/(1 + ur^2)$:

- table T_1 contains pairs (r, v) in which v is quadratic non-residue,
- table T_2 contains pairs (r, v) in which v is quadratic residues.

Performing one quadraticity test on A, only once per curve, and restricting table lookup to the table of opposite quadraticity ensures that $x := -Av$ is a non-square. Repeating quadraticity tests to ensure that a corresponding y exists, and completing one square root extraction in $\mathbb{F}_q$ to obtain y, one gets 2 points whose orders are multiples of 2^m at once. This is detailed in Algorithm 3.1.

Let us compare the number of operations required by the entangled basis algorithm with the plain basis generation algorithm used in Costello *et al.* [5].

Entangled basis: testing the quadraticity of A takes $(m + n + 1)\mathbf{s} + n\mathbf{m}$. The main loop runs twice on average at a cost $2(m + n + 1)\mathbf{s} + (2n + 22)\mathbf{m}$. The last stage is to complete a square root and costs $(m + n - 1)\mathbf{s} + (n + 1)\mathbf{m} + 1\mathbf{i}$. The total cost of the algorithm is then

$$(4m + 4n + 2)\mathbf{s} + (4n + 23)\mathbf{m} + 1\mathbf{i}.$$

Plain basis: To get the abscissa of a point on the curve takes $(2n+22)\mathbf{m}+2(m+n+1)\mathbf{s}$. Clearing the cofactor 3^n requires n point tripling at a cost $32n\mathbf{m}$. We also need to compute $m - 1$ point doubling for linear independence check that is required in the next steps. So obtaining the first basis point costs $(34n+16m+6)\mathbf{m}+2(m+n+1)\mathbf{s}$. The second basis point is obtained exactly the same way, except we also need a linear independence check. This is done in loop that runs twice on average. The expected cost of obtaining the second point is then twice the cost of obtaining the first point including the the $m - 1$ doubling step. The last stage of the algorithm is to recover the y coordinates of the points which costs $(4m + 4n)\mathbf{s} + (4n + 36)\mathbf{m} + 2\mathbf{i}$. Adding all these, the total cost of the algorithm is

$$(10m + 10n + 6)\mathbf{s} + (48m + 106n + 54)\mathbf{m} + 2\mathbf{i}.$$

For the values $m = 372$ and $n = 239$, and assuming $\mathbf{s} = 0.8\mathbf{m}$ and $\mathbf{i} = 100\mathbf{m}$, we get the performance ratio of 15.92.

Algorithm 3.1 Entangled basis generation for $E[2^m](\mathbb{F}_q) : y^2 = x^3 + Ax^2 + x$

INPUT: $A = a + bi \in \mathbb{F}_q$; $u_0 \in \mathbb{F}_q : u = u_0^2 \in \mathbb{F}_q \backslash \mathbb{F}_p$; tables T_1, T_2 of pairs $(r \in \mathbb{F}_p, v = 1/(1+ur^2) \in \mathbb{F}_q)$ of QNR and QR.
OUTPUT: $\{S_1, S_2\}$ such that $\langle [3^n]S_1, [3^n]S_2 \rangle = E[2^m](\mathbb{F}_q)$.

$z \leftarrow a^2 + b^2, \quad s \leftarrow z^{(p+1)/4}$
$T \leftarrow (s^2 \stackrel{?}{=} z)\ T_1 : T_2$ // select proper table by testing quadraticity of A
repeat
  lookup next entry (r, v) from T
  $x \leftarrow -A \cdot v$ // NB: x nonsquare
  $t \leftarrow x \cdot (x^2 + A \cdot x + 1)$ // test quadraticity of $t = c + di$
  $z \leftarrow c^2 + d^2, \quad s \leftarrow z^{(p+1)/4}$
until $s^2 = z$ // compute $y \leftarrow \sqrt{x^3 + A \cdot x^2 + x}$
$z \leftarrow (c + s)/2, \quad \alpha \leftarrow z^{(p+1)/4}, \quad \beta \leftarrow d \cdot (2\alpha)^{-1}$
$y \leftarrow (\alpha^2 \stackrel{?}{=} z)\ \alpha + \beta i : -\beta + \alpha i$ // compute basis
return $S_1 \leftarrow (x, y),\ S_2 \leftarrow (ur^2 x, u_0 r y)$

3.1 Avoiding cofactor multiplication

Combining reverse basis decomposition and entangled basis generation enables us to further avoid two scalar multiplications by the large cofactor 3^n during Bob's public key compression, and one during Alice's decompression. First notice that Algorithm 3.1 already incorporates the mentioned optimization, i.e. the output points S_1 and S_2 satisfy $(R_1, R_2) := ([3^n]S_1, [3^n]S_2)$ such that $\langle R_1, R_2 \rangle = E[2^m]$. This is only possible because in reverse basis decomposition the Tate pairings h_i take the points S_i in their second argument which does not need to be necessarily cofactor-reduced. In this case, for $R_1 = c_0\phi_B(P_A) + d_0\phi_B(Q_A)$ and $R_2 = c_1\phi_B(P_A) + d_1\phi_B(Q_A)$, the respective pairing computations are

$$\begin{aligned}
k_0 &= e_{2^m}(\phi_B(P_A), \phi_B(Q_A)) \\
k_1 &= e_{2^m}(\phi_B(P_A), S_1) = e_{2^m}(\phi_B(P_A), [3^{-n}]R_1) = k_0^{3^{-n}d_0} \\
k_2 &= e_{2^m}(\phi_B(P_A), S_2) = e_{2^m}(\phi_B(P_A), [3^{-n}]R_2) = k_0^{3^{-n}d_1} \\
k_3 &= e_{2^m}(\phi_B(Q_A), S_1) = e_{2^m}(\phi_B(Q_A), [3^{-n}]R_1) = k_0^{-3^{-n}c_0} \\
k_4 &= e_{2^m}(\phi_B(Q_A), S_2) = e_{2^m}(\phi_B(Q_A), [3^{-n}]R_2) = h_0^{-3^{-n}c_1}.
\end{aligned}$$

Thus, the discrete logarithms are the desired ones up to a factor 3^{-n}, and given by $\hat{c}_0 = -\log_{k_0} k_3 = 3^{-n}c_0$, $\hat{d}_0 = \log_{k_0} k_1 = 3^{-n}d_0$, $\hat{c}_1 = -\log_{k_0} k_4 = 3^{-n}c_1$, and $\hat{d}_1 = \log_{k_0} k_2 = 3^{-n}d_1$. Notice that $3^{-n} \bmod 2^m$ must be odd which implies that $\hat{c}_0$ or $\hat{d}_0$ is invertible if and only if c_0 or d_0 is invertible. Similar to the situation in Section 2, when using the compression with only 3 coefficients as in [5] Bob transmits exactly the original coefficients: assuming $\hat{c}_0$ is invertible, then

$$\begin{aligned}
(\hat{c}_0^{-1}\hat{d}_0, \hat{c}_0^{-1}\hat{c}_1, \hat{c}_0^{-1}\hat{d}_1) &= (c_0^{-1}3^n3^{-n}d_0, c_0^{-1}3^n3^{-n}c_1, c_0^{-1}3^n3^{-n}d_1) \\
&= (c_0^{-1}d_0, c_0^{-1}c_1, c_0^{-1}d_1)
\end{aligned}$$

The derivation when d_0 is invertible is analogous.

To decompress Bob's public key, Alice needs to perform a single cofactor multiplication by 3^n as follows. Assume that a_0 is invertible modulo 2^m so that Alice re-

ceives the triple $(a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1)$. She needs to compute the kernel $\ker(\phi_{AB}) = \langle \phi_B(P_A) + sk_A \cdot \phi_B(Q_A) \rangle$ which can be written as

$$\langle a_0R_1 + b_0R_2 + sk_A \cdot (a_1R_1 + b_1R_2) \rangle = \langle (a_0 + sk_Aa_1)R_1 + (b_0 + sk_Ab_1)R_2 \rangle$$

As noted in [5], one computes $\ker(\phi_{AB})$ as $a_0^{-1} \ker(\phi_{AB}) = \langle (1+sk_Aa_0^{-1}a_1)R_1 + (a_0^{-1}b_0 + sk_Aa_0^{-1}b_1)R_2 \rangle$, which can be done with one scalar multiplication and one point addition by writing $\ker(\phi_{AB}) = \langle R_1 + (1 + sk_Aa_0^{-1}a_1)^{-1}(a_0^{-1}b_0 + sk_Aa_0^{-1}b_1)R_2 \rangle$. Now if Alice uses Algorithm 3.1, she obtains an entangled basis $\{S_1, S_2\}$ such that $(R_1, R_2) = ([3^n]S_1, [3^n]S_2)$. She can then compute $T = \langle S_1 + (1+sk_Aa_0^{-1}a_1)^{-1}(a_0^{-1}b_0 + sk_Aa_0^{-1}b_1)S_2$ first and then recover the correct kernel $\ker(\phi_{AB}) = \langle [3^n]T \rangle$ by performing one cofactor scalar multiplication.

4 Pairing computation

The pairing computation techniques by Costello *et al.* [5] are based on curves in a variant of the Montgomery model, with projective coordinates (X^2, XZ, Z^2, YZ), which turned out to be the best setting among several models they assessed. We will argue that the older and today less favored short Weierstraß model leads to more efficient pairing algorithms. For convenience, we extend Jacobian coordinates $[X : Y : Z]$ with a fourth component, $[X : Y : Z : T]$ with $T = Z^2$.

Interestingly, Costello *et al.* dismiss the technique of denominator elimination [2] and keep numerators and denominators separate during pairing evaluation. We point out, however, that pairing values are defined over $\mathbb{F}_{p^2}$ and the inverse of a field element $a+bi$ is $(a-bi)/(a^2+b^2)$. Hence, rather than keeping a separate denominator $a+bi$ one can simply and immediately multiply the pairing value by the conjugate $a-bi$ instead; the result only differs from the original one by a denominator consisting of the norm $a^2 + b^2 \in \mathbb{F}_p$, and this denominator does get eliminated by the final exponentiation in the reduced Tate pairing computation. This leaves the cost of pairing computation unchanged, but it simplifies the implementation as it entirely does away with separate numerators and denominators.

Let $r \geq 0$ be the pairing order. For embedding degree $k = 2$, $r \mid \Phi_2(p) = p+1 = 2^m \cdot 3^n$, and by construction r is always either 2^m or 3^n. We will be interested in computing reduced Tate pairings of order r, whose first argument must have that order as well. In the case of compressed SIDH keys, pairings of the following forms are computed together (recall that a fifth pairing $e_0 := e_r(P, Q) = e_r(P_0, Q_0)^{\deg \phi}$ is readily available through precomputation):

$$e_1 := e_r(P, R_1),\ e_2 := e_r(P, R_2),\ e_3 := e_r(Q, R_1),\ e_4 := e_r(Q, R_2),$$

where the first two pairings share the same first argument P, and next two pairings share the same first argument Q.

From now on, we will split the discussion into two cases: binary-order pairings, $r = 2^m$, and ternary-order pairings, $r = 3^n$.

4.1 Binary-order pairings

The computation of the reduced Tate pairing $e_r(P, Q)$ of order $r = 2^m$ proceeds as described in Algorithm 4.1, which requires doubling a point $V \in E(\mathbb{F}_{p^2})$. The doubling

formulas in Jacobian coordinates have a single exception, that occurs when the point being doubled has order 2. That is, when $y = 0$, since the angular coefficient of the tangent to the curve at that point becomes undefined. That exception, however, can only occur deterministically in the scenario contemplated here, namely at the last step of the Miller loop; since by definition the first pairing argument is always a point of order 2^m, chosen by the very entity that is computing the pairing.

Besides, the difference in running time reveals no private information, since the pairing arguments are either already public for being part of a conventional torsion basis, or else are about to be made public for being part of a public key.

Algorithm 4.1 Tate2(P, Q): basic reduced Tate pairing of order $r = 2^m$:

INPUT: points P, Q.
OUTPUT: $e_r(P, Q)$.

1: $f \leftarrow 1, \quad V \leftarrow P$
2: **for** $i \leftarrow 0$ **to** $m - 1$ **do**
3: $\quad f \leftarrow f^2 \cdot g_{V,V}(Q)/g_{[2]V}(Q), \quad V \leftarrow [2]V$
4: **end for**
5: **return** $e_r(P, Q) \leftarrow f^{(p^2-1)/r}$

The most efficient doubling method known for Jacobian coordinates is due to Bernstein and Lange [4]. Let $V = [X : Y : Z : T]$ and $2V = [X' : Y' : Z' : T']$ in the extended coordinate system defined above. Then

$$\begin{array}{lll} X_2 \leftarrow X^2; & Y_2 \leftarrow Y^2; & Y_4 \leftarrow Y_2^2; \\ S \leftarrow 2((X + Y_2)^2 - X_2 - Y_4); & M \leftarrow 3X_2 + A \cdot T^2; & X' \leftarrow M^2 - 2S; \\ Y' \leftarrow M \cdot (S - X') - 8Y_4; & Z' \leftarrow (Y + Z)^2 - Y_2 - T; & T' \leftarrow (Z')^2; \end{array}$$

The cost is $2\mathbf{M} + 8\mathbf{S} + 15\mathbf{A} = 2(3\mathbf{m} + 5\mathbf{a}) + 8(2\mathbf{m} + 3\mathbf{a}) + 15(2\mathbf{a}) = 22\mathbf{m} + 64\mathbf{a}$. The curve coefficient A typically lacks any structure that might enable optimizations, and hence incurs full multiplication cost.

This algorithm yields the values $\lambda_N := M$ and $\lambda_D := Z' = 2YZ$, besides X, X', T, T', and Y_2. These values are useful in the calculation of a function equivalent to $g_{V,V}(Q)/g_{[2]V}(Q)$, namely $[M \cdot (T \cdot x - X) + W - L \cdot y] \cdot R \cdot (T' \cdot x - X')^*$ when $V \neq O$ and $[2]V \neq O$ (i.e. $Z \neq 0$ and $Z' \neq 0$), $(T \cdot x - X) \cdot T^*$ when $V = -V \neq O$ (i.e. $Z \neq 0$ and $Z' = 0$), or simply 1 when $Z = O$. where $L := Z' \cdot T$, $R := Z' \cdot T^*$, $W \leftarrow 2Y_2$. Denominators in the base field ($|Z^2 \cdot (T' \cdot x - X')|^2 \in \mathbb{F}_p$ in the first case, $|Z^2|^2 \in \mathbb{F}_p$ in the second case) are eliminated.

Since the scenario where the pairing computations take place only involve bases of the 2^m-torsion group, and hence points of full order 2^m, the exceptional formula is indeed never invoked until the end of Miller's loop. The difference in processing time is irrelevant for security here, since the computations only involve information that is meant to be public.

However, one can further optimize the computation of (a function equivalent to) $g_{V,V}(Q)/g_{[2]V}(Q)$. First, the expression $(T' \cdot x - X')$ that occurs at a certain step will play the role of $(T \cdot x - X)$ at the next step, so one can simply store it from one step to the next and thus save $1\mathbf{M}$. Second, the R factor above is irrelevant to the pairing value. Let Z_j denote the computed z-coordinate of $[2^j]P$, let $T_j := Z_j^2$ and let

$R_j := Z_{j+1} \cdot T_j^* = Z_{j+1} \cdot (Z_j^*)^2$ denote the contribution of the R factor above at the j-th step in Miller's loop for $0 \leq j < m-1$, with $R_{m-1} := 1$, rather than 0 as the general expression would yield, for convenience. One can show by induction that the contribution of all R factors to the pairing value before the final exponentiation is

$$\hat{R} := \prod_{j=0}^{m-1} R_j^{2^{m-1-j}} = \prod_{j=0}^{m-1} \left(Z_{j+1} \cdot T_j^*\right)^{2^{m-1-j}},$$

which can be rearranged as

$$\hat{R} = (T_0^*)^{2^{m-1}} Z_{m-1}^2 \prod_{j=1}^{m-2} \left(Z_j^{2^{m-j}} (T_j^*)^{2^{m-1-j}}\right) = (Z_0^*)^{2^m} T_{m-1} \prod_{j=1}^{m-2} \left(Z_j Z_j^*\right)^{2^{m-j}}.$$

But the final exponentiation will erase the first factor as $((Z_0^*)^{2^m})^{(p^2-1)/2^m} = (Z_0^*)^{(p^2-1)} = 1$, and also the last product above, which only involves norms in $\mathbb{F}_p$. Hence the actual contribution is simply $\hat{R}^{(p^2-1)/2^m} = T_{m-1}^{(p^2-1)/2^m}$, but the line function at the last step of Miller's loop contributes a factor $(T \cdot x - X) \cdot T^*$ to the pairing value before the final exponentiation, so $\hat{R}$ could be incorporated there as $(T \cdot x - X) \cdot T_{m-1}^* \cdot T_{m-1} \sim T \cdot x - X$. This simplified formula can be used instead without making any reference at all to the R factors. Consequently, initializing $h \leftarrow T \cdot x - X$ before Miller's loop at a cost of $1\mathbf{M}$ per pairing, the line function value g can be evaluated as

$$g \leftarrow M \cdot h + W - L \cdot y;\ h \leftarrow T' \cdot x - X';\ g \leftarrow g \cdot h^*$$

at a cost of $4\mathbf{M} + 3\mathbf{A} = 12\mathbf{m} + 26\mathbf{a}$ per step of Miller's loop. The cost of computing $L \leftarrow Z' \cdot T$ alone is $1\mathbf{M} = 3\mathbf{m} + 5\mathbf{a}$ and that of computing W is $1\mathbf{A} = 2\mathbf{a}$. This completes the line function construction.

The updating of f at each step as $f \leftarrow f^2 \cdot g_{V,V}(Q)/g_{[2]V}(Q)$ incurs a cost $2\mathbf{m} + 3\mathbf{a}$ to compute the complex square f^2 plus $3\mathbf{m} + 5\mathbf{a}$ to compute $f^2 \cdot g_{V,V}(Q)/g_{[2]V}(Q)$ from f^2 and $g_{V,V}(Q)/g_{[2]V}(Q)$, totaling $5\mathbf{m} + 8\mathbf{a}$. Therefore, the proposed variant has the following overall cost per step:

- (shared) cost of point doubling and line function construction: $22\mathbf{m} + 64\mathbf{a} + 3\mathbf{m} + 7\mathbf{a} = 25\mathbf{m} + 71\mathbf{a}$;
- (individual) cost of line function evaluation and accumulation: $12\mathbf{m} + 26\mathbf{a} + 5\mathbf{m} + 8\mathbf{a} = 17\mathbf{m} + 34\mathbf{a}$.

In summary, in our method the total cost for t parallel pairings that share the same first argument is $25\mathbf{m} + 71\mathbf{a} + t \cdot (17\mathbf{m} + 34\mathbf{a})$ per step. This means $42\mathbf{m} + 105\mathbf{a}$ for an individual pairing, and $59\mathbf{m} + 139\mathbf{a}$ for two parallel pairings that share the same first argument.

By comparison, the Costello *et al.* [5] technique has the following costs:

- (shared) cost of point doubling and line function construction: $9\mathbf{M} + 5\mathbf{S} + 1\mathbf{s} + 7\mathbf{a} = 37\mathbf{m} + 1\mathbf{s} + 67\mathbf{a}$;
- (individual) cost of line function evaluation and accumulation: $5\mathbf{M} + 2\mathbf{S} + 2\mathbf{s} + 1\mathbf{a} = 19\mathbf{m} + 2\mathbf{s} + 32\mathbf{a}$;

In summary, in their method the total cost for t parallel pairings that share the same first argument is $37\mathbf{m}+1\mathbf{s}+67\mathbf{a}+t\cdot(19\mathbf{m}+2\mathbf{s}+32\mathbf{a})$ per step. This means $56\mathbf{m}+3\mathbf{s}+99\mathbf{a}$ for an individual pairing, and $75\mathbf{m}+5\mathbf{s}+131\mathbf{a}$ for two parallel pairings that share the same first argument. We see that, for the case of interest here, which is two parallel pairings, our technique costs a fraction $\approx 59.0/79.0 \approx 74.7\%$ of the Costello *et al.* method, assuming $1\mathbf{s} \approx 0.8\mathbf{m}$ and essentially ignoring $\mathbf{a}$.

Pairings on an entangled basis. If two pairings $e(P, R_1)$, $e(P, R_2)$ sharing the same first argument P are computed on an entangled basis $R_1 = (x_1, y_1)$, $R_2 = (x_2, y_2)$ with $x_2 = t^2 \cdot x_1$, $y_2 = t \cdot y_1$, one can slightly improve the line function evaluation and accumulation. Because further multiplication by carefully chosen t or t^2 given the values of $T' \cdot x_1$ or $L \cdot y_1$ is less expensive than the full multiplications $T' \cdot x_2$ or $L \cdot y_2$ for generic (x_2, y_2).

Specifically, for $t = (1+i)r$ and $t^2 = 2ir^2$ with some small $r \in \mathbb{F}_p$, the cost of a dedicated implementation of parallel entangled pairings drops by $2\mathbf{m}+10\mathbf{a}$, thus becoming only $57\mathbf{m}+129\mathbf{a}$, or about 72.2% of the cost of the Costello *et al.* method. The performance improvements brought about by the techniques we proposed are summarized on Table 1. Our proposed variant of the parallel reduced Tate pairing is shown in full detail as Algorithm A.1 in the Appendix.

Table 1: Cost of the binary Miller loop, assuming $\mathbf{s} \approx 0.8\mathbf{m}$ and ignoring $\mathbf{a}$.

t	Costello *et al.*	ours	ratio
1	$56\mathbf{m}+3\mathbf{s}+99\mathbf{a}$	$42\mathbf{m}+105\mathbf{a}$	$42.0/58.4 \approx 0.719$
2	$75\mathbf{m}+5\mathbf{s}+131\mathbf{a}$	$59\mathbf{m}+139\mathbf{a}$	$59.0/79.0 \approx 0.747$
$2^\dagger$	$75\mathbf{m}+5\mathbf{s}+131\mathbf{a}$	$57\mathbf{m}+129\mathbf{a}$	$57.0/79.0 \approx 0.722$

†Parallel entangled basis

4.2 Ternary-order pairings

The computation of the reduced Tate pairing $e_r(P, Q)$ of order $r = 3^n$ proceeds as described in Algorithm 4.2. Again, the tripling formulas in Jacobian coordinates have an exception when $y = 0$, but this can be handled in a similar fashion to the binary case. The difference in running time reveals no private information for the same reason, namely only public data is involved in the pairing computations.

Algorithm 4.2 Tate3(P, Q): basic reduced Tate pairing of order $r = 3^n$:

INPUT: points P, Q.
OUTPUT: $e_r(P, Q)$.

1: $f \leftarrow 1, \quad V \leftarrow P$
2: **for** $i \leftarrow 0$ **to** $n-1$ **do**
3: $\quad f \leftarrow f^3 \cdot g_{V,V}(Q) \cdot g_{V,[2]V}(Q)/g_{[2]V}(Q)/g_{[3]V}(Q), \quad V \leftarrow [3]V$
4: **end for**
5: **return** $e_r(P, Q) \leftarrow f^{(p^2-1)/r}$

The most efficient tripling algorithm known for Jacobian coordinates is due to Bernstein and Lange [4]. Let $V = [X : Y : Z : T]$ and $[3]V = [X' : Y' : Z' : T']$ in the extended coordinate system as before. Then:

$$\begin{array}{lll}
X_2 \leftarrow X^2; & Y_2 \leftarrow Y^2; & Y_4 \leftarrow Y_2^2; \\
T_2 \leftarrow T^2; & M \leftarrow 3X_2 + a \cdot T_2; & M_2 \leftarrow M^2; \\
D \leftarrow (X + Y_2)^2 - X_2 - Y_4; & F \leftarrow 6D - M_2; & F_2 \leftarrow F^2; \\
W \leftarrow 2Y_2; & W' \leftarrow 2W & S \leftarrow 16Y_4; \\
U \leftarrow (M + F)^2 - M_2 - F_2 - S; & U' \leftarrow S - U; & \\
X' \leftarrow 4(X \cdot F_2 - W' \cdot U); & Y' \leftarrow 8Y \cdot (U \cdot U' - F \cdot F_2); & \\
Z' \leftarrow (Z + F)^2 - T - F_2; & T' \leftarrow (Z')^2; &
\end{array}$$

The cost is $6\mathbf{M} + 10\mathbf{S} + 25\mathbf{A} = 6(3\mathbf{m} + 5\mathbf{a}) + 10(2\mathbf{m} + 3\mathbf{a}) + 25(2\mathbf{a}) = 38\mathbf{m} + 110\mathbf{a}$. The pairing computation requires constructing and evaluating a parabola function equivalent to $g_{V,V}(Q) \cdot g_{V,[2]V}(Q)/g_{[2]V}(Q)/g_{[3]V}(Q)$ at each step together with point tripling. Assuming initially $[3]V \neq O$ and reusing variables available from point tripling, this can be carried out as:

$$\begin{array}{lll}
L \leftarrow ((Y + Z)^2 - Y_2 - T) \cdot T; & F' \leftarrow 2F & R \leftarrow F \cdot T^*; \\
d \leftarrow W - L \cdot y; & h \leftarrow T \cdot x - X; & h_3 \leftarrow T' \cdot x - X';
\end{array}$$

and then the parabola value is $g \leftarrow (M \cdot h + d) \cdot (U' \cdot h + F' \cdot d) \cdot (W' \cdot h + F)^* \cdot R \cdot h_3^*$.

However, one can further optimize the computation of (a function equivalent to) the above parabola in a similar fashion to what was done for the binary case. First, the expression $(T' \cdot x - X')$ that occurs at a certain step will play the role of $(T \cdot x - X)$ at the next step, so one can simply store it from one step to the next and thus save $1\mathbf{M}$. Second, the above R factor is again irrelevant to the pairing value. Let Z_j denote the computed z-coordinate of $[3^j]P$, and let $T_j := Z_j^2$, and let $R_j := 2|Z_j|^2 \cdot R_j = (2F_j \cdot Z_j) \cdot (Z_j^* \cdot T_j^*) = Z_{j+1} \cdot (Z_j^*)^3$ denote a contribution equivalent to that of the R factor above at the j-th step in Miller's loop for $0 \leq j < n - 1$, with $R_{n-1} := 1$ for convenience. One can show by induction and term rearrangement that the contribution of all R factors to the pairing value before the final exponentiation is

$$\hat{R} := (Z_0^*)^{3^n} \cdot Z_{n-1}^3 \cdot \prod_{j=1}^{n-2} \left(Z_j \cdot Z_j^*\right)^{3^{n-j}}.$$

Therefore, the actual contribution after the final exponentiation is simply $\hat{R}^{(p^2-1)/3^n} = (Z_{n-1}^3)^{(p^2-1)/3^n}$, but the parabola function at the last step of Miller's loop contributes a factor $((M \cdot h + d) \cdot L^*)^3$ to the pairing value before the final exponentiation, so $\hat{R}$ could be incorporated to that expression as $(M \cdot h + d)^3 \cdot (L^*)^3 \cdot Z_{n-1}^3 = ((M \cdot h + d) \cdot Y^*)^3 \cdot (2Z_{n-1}^* \cdot Z_{n-1})^3 \sim ((M \cdot h + d) \cdot Y^*)^3$. This simplified formula can be used instead without making any reference at all to the R factors. Consequently, the parabola function construction can be completed by computing only L and F' as above at a cost $1\mathbf{M} + 1\mathbf{S} + 3\mathbf{A} + 1\mathbf{A} = 5\mathbf{m} + 16\mathbf{a}$. After initializing $h \leftarrow T \cdot x - X$ before Miller's loop at a cost of $1\mathbf{M}$ per pairing, the parabola function value g can be evaluated as

$$\begin{array}{ll}
d \leftarrow W - L \cdot y; & g \leftarrow (M \cdot h + d) \cdot (U' \cdot h + F' \cdot d) \cdot (W' \cdot h + F); \\
h \leftarrow T' \cdot x - X'; & g \leftarrow g \cdot h^*;
\end{array}$$

at a cost $9\mathbf{M}+5\mathbf{A} = 27\mathbf{m}+55\mathbf{a}$ per step of Miller's loop. The updating of f at each step as $f \leftarrow f^3 \cdot g_{V,V}(Q) \cdot g_{V,2V}(Q)/g_{2V}(Q)/g_{3V}(Q)$ incurs a cost $2\mathbf{m}+2\mathbf{s}+6\mathbf{a}$ to compute the complex cube f^3, plus $3\mathbf{m}+5\mathbf{a}$ to compute $f^3 \cdot g_{V,V}(Q) \cdot g_{V,2V}(Q)/g_{2V}(Q)/g_{3V}(Q)$ from f^3 and $g_{V,V}(Q) \cdot g_{V,2V}(Q)/g_{2V}(Q)/g_{3V}(Q)$, totaling $5\mathbf{m}+2\mathbf{s}+11\mathbf{a}$. Therefore, the proposed variant has the following overall cost per step:

- (shared) cost of point tripling and parabola function construction: $38\mathbf{m}+110\mathbf{a}+5\mathbf{m}+16\mathbf{a} = 43\mathbf{m}+126\mathbf{a}$;
- (individual) cost of parabola function evaluation and accumulation: $27\mathbf{m}+55\mathbf{a}+5\mathbf{m}+2\mathbf{s}+11\mathbf{a} = 32\mathbf{m}+2\mathbf{s}+66\mathbf{a}$.

In summary, in our method the total cost for t parallel pairings that share the same first argument is $43\mathbf{m}+126\mathbf{a}+t\cdot(32\mathbf{m}+2\mathbf{s}+66\mathbf{a})$ per step. This means $75\mathbf{m}+2\mathbf{s}+192\mathbf{a}$ for an individual pairing, and $107\mathbf{m}+4\mathbf{s}+258\mathbf{a}$ for two parallel pairings that share the same first argument.

By comparison, the Costello *et al.* [5] technique has the following costs:

- (shared) cost of point tripling and construction of the parabola functions: $19\mathbf{M}+6\mathbf{S}+6\mathbf{s}+15\mathbf{a} = 69\mathbf{m}+6\mathbf{s}+128\mathbf{a}$;
- (individual) cost of evaluating the parabola functions and accumulating the results: $10\mathbf{M}+2\mathbf{S}+4\mathbf{a} = 34\mathbf{m}+60\mathbf{a}$;

In summary, in their method the total cost for t parallel pairings that share the same first argument is $69\mathbf{m}+6\mathbf{s}+128\mathbf{a}+t\cdot(34\mathbf{m}+60\mathbf{a})$ per step. This means $103\mathbf{m}+6\mathbf{s}+188\mathbf{a}+60\mathbf{a}$ for an individual pairing, and $137\mathbf{m}+6\mathbf{s}+248\mathbf{a}$ for two parallel pairings that share the same first argument.

For the case of interest here, which is two parallel pairings, our technique costs a fraction $\approx 110.2/141.8 \approx 77.7\%$ of the Costello *et al.* method, assuming $1\mathbf{s} \approx 0.8\mathbf{m}$ and essentially ignoring $\mathbf{a}$ as before. In contrast with the binary case, no effective entangled basis construction is known for the ternary case, and a mere repetition of the same technique is unlikely to improve performance, so we refrain from extending the discussion here.

The performance improvements brought about by the techniques we propose are summarized on Table 2. Our proposed variant of the parallel reduced Tate pairing is shown in full detail in the Appendix as Algorithm A.2.

Table 2: Cost of the ternary Miller loop, assuming $\mathbf{s} \approx 0.8\mathbf{m}$ and ignoring $\mathbf{a}$

t	Costello *et al.*	ours	ratio
1	$103\mathbf{m}+6\mathbf{s}+188\mathbf{a}$	$75\mathbf{m}+2\mathbf{s}+192\mathbf{a}$	$76.6/107.8 \approx 0.711$
2	$137\mathbf{m}+6\mathbf{s}+248\mathbf{a}$	$107\mathbf{m}+4\mathbf{s}+258\mathbf{a}$	$110.2/141.8 \approx 0.777$

5 Discrete logarithm computation

Let $\mu_{\ell^e} \subset \mathbb{F}_{p^2}$ be the set of ℓ^e-th root of unity in $\mathbb{F}_{p^2}$, i.e. $\mu_{\ell^e} := \{v \in \mathbb{F}_{p^2} \mid v^{\ell^e} = 1\}$. Inverting in μ_{ℓ^e} is a mere conjugation, $(a+bi)^{-1} = a-bi$ since the norm is 1. The

Pohlig-Hellman method (Algorithm 5.1) to compute the discrete logarithm of $c \in \mu_{\ell^e}$ requires solving an equation of the form:

$$r_k^{\ell^{e-1-k}} = s^{d_k}$$

where $s = g^{\ell^{e-1}}$ has order ℓ and, for $k = 0, \ldots, e-1$, $d_k \in \{0, \ldots, \ell-1\}$ is an ℓ-ary digit, $r_0 = c$, and r_{k+1} depends on r_k and d_k.

Algorithm 5.1 Basic Pohlig-Hellman discrete logarithm algorithm

INPUT: generator $g \in \mu_{\ell^e}$, challenge $c \in \mu_{\ell^e}$.
OUTPUT: $d := \log_g c$, i.e. $g^d = c$.

1: $s \leftarrow g^{\ell^{e-1}}$ // NB: $s^\ell = 1$
2: $d \leftarrow 0$, $r_0 \leftarrow c$
3: **for** $k \leftarrow 0$ **to** $e-1$ **do**
4: $\quad v_k \leftarrow r_k^{\ell^{e-1-k}}$
5: $\quad$ find $d_k \in \{0, \ldots, \ell-1\}$ such that $v_k = s^{d_k}$
6: $\quad d \leftarrow d + d_k \ell^k$, $r_{k+1} \leftarrow r_k \cdot g^{-\ell^k d_k}$
7: **end for** // NB: $g^d = c$
8: **return** d

Assuming that $g^{-\ell^k}$ is precomputed and stored for all k as a by-product of the computation of s, the naive strategy to obtain the discrete logarithm requires repeatedly computing the exponential $r_k^{\ell^{e-1-k}}$ at the cost of $e-1-k$ raisings to the ℓ, then solving a small discrete logarithm instance in a subgroup of order ℓ to get one ℓ-ary digit, then clearing that digit in the exponent of r_k at a cost not exceeding ℓ multiplications to obtain r_{k+1}. The overall cost is thus $O(e^2)$.

It turns out that this strategy is far from optimal, as pointed out by Shoup [13, Chapter 11]. The crucial task is to obtain the sequence $r_0^{\ell^{e-1}}, r_1^{\ell^{e-2}}, r_2^{\ell^{e-3}}, \ldots, r_{e-1}^{\ell^0}$ in order, since each r_k depends on the previous one. We can visualize this task using a directed acyclic graph Δ strikingly similar to De Feo *et al.*'s T_n graph, which they call a "discrete equilateral triangle", that models the construction of smooth-degree isogenies [6, Section 4.2.2].

In our case, the set of vertices is $\{\Delta_{jk} \mid j + k \leq e-1\}$ where $\Delta_{jk} := r_k^{\ell^j}$. Each vertex has either two downward outgoing edges, or no edges at all. Vertices Δ_{jk} with $j + k > e-1$ have two edges: a left edge $\Delta_{jk} \to \Delta_{j+1,k}$ that models raising the source vertex to the ℓ-th power to yield the destination vertex, $r_k^{\ell^{j+1}} \leftarrow (r_k^{\ell^j})^\ell$, and a right edge $\Delta_{jk} \to \Delta_{j,k+1}$ that models clearing the $(j+k)$-th digit in the exponent of the source vertex, $r_{k+1}^{\ell^j} \leftarrow r_k^{\ell^j} \cdot g^{-\ell^{(j+k)} d_k}$. Vertices Δ_{jk} with $j + k = e-1$ are leaves since they have no outgoing edges.

De Feo *et al.* [6, Equation 5] describe an $O(e^2)$ dynamic programming algorithm that computes the cost of an optimal subtree of Δ with root at Δ_{00} and covering all leaves. If the cost of traversing a left edge is p, the cost of traversing a right edge is q, and the cost of an optimal subtree of k edges is $C_{p,q}(k)$, their algorithm is based on the relations $C_{p,q}(1) = 0$ and $C_{p,q}(k) = \min_{1 \leq j < k} (C_{p,q}(j) + C_{p,q}(k-j) + (k-j)p + jq)$ for $k > 1$.

The naive dynamic programming approach is to store the values of $C_{p,q}(k)$ for $k = 1 \ldots e$, invoking the above relation $k-1$ times at each step to find the corresponding

minimum, for a total $e(e-1)/2$ invocations, hence the $O(e^2)$ cost. However, because $C_{p,q}(k)$ has no local minimum other than the single global minimum (or two adjacent, equivalent copies of the global minimum at worst), one can find that minimum with a variant of binary search that compares two consecutive values near the middle of the search interval $[1 \dots k-1]$ and then halves that interval. This yields the $O(e \log e)$ Algorithm 5.2, which computes $C_{p,q}(k)$ and the structure of the optimal traversal strategy by storing the values of j above that attain the minimum at each step.

Algorithm 5.2 OptPath(p, q, e): optimal subtree traversal path

INPUT: p, q: left and right edge traversal cost; e: number of leaves of Δ.
OUTPUT: P: optimal traversal path

Define $C[1 \dots e]$ as an array of costs and $P[1 \dots e]$ as an array of indices.
$C[1] \leftarrow 0, \quad P[1] \leftarrow 0$
for $k \leftarrow 2$ **to** e **do**
    $j \leftarrow 1, \quad z \leftarrow k-1$
    while $j < z$ **do**
        $m \leftarrow j + \lfloor (z-j)/2 \rfloor, \quad u \leftarrow m+1$
        $t_1 \leftarrow C[m] + C[k-m] + (k-m) \cdot p + m \cdot q$
        $t_2 \leftarrow C[u] + C[k-u] + (k-u) \cdot p + u \cdot q$
        if $t_1 \leq t_2$ **then**
            $z \leftarrow m$
        else
            $j \leftarrow u$
        end if
    end while
    $C[k] \leftarrow C[j] + C[k-j] + (k-j) \cdot p + j \cdot q, \quad P[k] \leftarrow j$
end for
return P

5.1 Discrete logarithm computation cost

The cost of an optimal strategy depends on the individual costs of traversing a left edge and a right edge. We now show that, because of our proposed reversed base decomposition technique, the total cost of discrete logarithm computation is drastically reduced. A left edge traversal represents the computation $r_k^{\ell^{j+1}} \leftarrow (r_k^{\ell^j})^\ell$ at a cost $w\mathbf{S} \approx 2w\mathbf{m}$ in the binary case and $w\mathbf{C} = w(2\mathbf{m} + 1\mathbf{s}) \approx 2.8w\mathbf{m}$ in the ternary case, with windows of size w.

A right edge traversal represents the computation $r_{k+1}^{\ell^j} \leftarrow r_k^{\ell^j} \cdot g^{-\ell^{(j+k)} d_k}$, which can be performed via table lookup $r_{k+1}^{\ell^j} \leftarrow r_k^{\ell^j} \cdot T[j+k][d_k]$ where $T[u][d] := g^{-\ell^u \cdot d}$. Since $j + k \leq e - 1$, the table size is $(e/w) \cdot \ell^w$ field elements. However, no more than a single multiplication is incurred regardless of ℓ, e, or w, namely, $1\mathbf{M} \approx 3\mathbf{m}$. When w is very small, avoiding the multiplication for $d_k = 1$ noticeably reduces the running time and requires fewer table entries. Moreover, the table is *fixed* with the reverse basis decomposition technique, because $g = e(P_B, Q_B)^{\deg \phi_A}$, or $g = e(P_A, Q_A)^{\deg \phi_B}$, thus incurring no table building cost at running time for each newly generated key. Even the simple discrete logarithm instances at the leaves only incur $O(\ell)$ lookups on the same table, since $s^{d_k} = T[e-1][d_k]^*$.

Table 3: Discrete logarithm computation costs (assuming $\mathbf{s} \approx 0.8\mathbf{m}$)

group	Costello *et al.* [5]	ours, $w = 1$ (ratio)	ours, $w = 3$ (ratio)	ours, $w = 6$ (ratio)
$\mu_{2^{372}}$	8271.6**m**	4958.4**m** (0.60)	3127.9**m** (0.39)	2103.7**m** (0.25)
$\mu_{3^{239}}$	7999.2**m**	4507.6**m** (0.56)	2638.1**m** (0.33)	1739.8**m** (0.22)

Algorithm 5.3 summarizes the proposed technique, combining Shoup's RDL algoritm [13, Section 11.2.3] with the optimal divide-and-conquer strategy of De Feo *et al.* and the efficient table lookup enabled by reverse basis decomposition.

Algorithm 5.3 Traverse(r, j, k, z, P, T, d)

INPUT: r: value of root vertex Δ_{jk}, i.e. $r := r_k^{\ell^j}$; j, k: coordinates of root vertex Δ_{jk}; z: number of leaves in subtree rooted at Δ_{jk}; P: traversal path; T: lookup table.
OUTPUT: d: digits (base ℓ) of $\log_g r_0$.
REMARK: initial call is Traverse$(r_0, 0, 0, e, P, T, d)$.

if $z > 1$ **then**
  $t \leftarrow P[z]$ // z leaves: t to the left exp, $z - t$ to the right
  $r' \leftarrow r^{\ell^{z-t}}$ // go left $(z - t)$ times
  Traverse$(r', j + (z - t), k, t, P, T)$
  $r' \leftarrow r \cdot \prod_{h=k}^{k+t-1} T[j + h][d_h]$ // go right t times
  Traverse$(r', j, k + t, z - t, P, T)$
else // leaf
  find $t \in \{0, \ldots, \ell - 1\}$ such that $r = T[e - 1][t]^*$
  $d_k \leftarrow t$ // recover k-th digit d_k of the discrete logarithm from $r = s^{d_k}$
end if

The resulting improvements are substantial. For discrete logs in $\mu_{2^{372}}$, the optimal cost is $\approx 4958.4\mathbf{m}$ with windows of size $w = 1$, $\approx 3127.9\mathbf{m}$ with windows of size $w = 3$, and $\approx 2103.7\mathbf{m}$ with windows of size $w = 6$. For discrete logs in $\mu_{3^{239}}$, the optimal cost is $\approx 4507.6\mathbf{m}$ with windows of size $w = 1$, $\approx 2638.1\mathbf{m}$ with windows of size $w = 3$, and $\approx 1739.8\mathbf{m}$ with windows of size $w = 6$.

Tradeoffs are also possible. Instead of being a matrix of size $(e/w) \cdot \ell^w$, the lookup table could be restricted to a single array $T_1[u] := g^{-\ell^u}$ of (e/w) entries, by computing $T_1[u]^d = g^{-\ell^u \cdot d}$ on demand using an optimal multiplication chain for cyclotomic exponentiation. For instance, discrete logs in $\mu_{2^{372}}$ with windows size $w = 3$ would require a table of size 124 at an average cost $\approx$ 4453.9**m**. For comparison, the best results reported in [5, Section 5] are $5320\mathbf{m} + 3349\mathbf{s} \approx 8271.6\mathbf{m}$ for discrete logs in $\mu_{2^{372}}$ and $5320\mathbf{m} + 3349\mathbf{s} \approx 7999.2\mathbf{m}$ for discrete logs in $\mu_{3^{239}}$, both with windows of size $w = 3$, which is optimal in that technique; increasing the window size actually causes a cost increase.

Table 3 summarizes the gains our technique makes possible and compares them against the results from Costello *et al.*, in terms of both the raw number of multiplications in the base field and the ratio between our results and theirs. We recall that no side-channel security concern arises from this technique, since all information involved in the processing is public.

6 Point tripling on Montgomery curves

Multiplication by 3^n, be it as a cofactor in the case of the 2^m torsion or as a tool to test linear independence in the 3^n torsion, is a computationally expensive operation. We describe in Algorithm 6.1 an improved method for point tripling on Montgomery curves that, though modest, directly addresses this bottleneck.

Algorithm 6.1 Improved tripling on the Montgomery curve $By^2 = x^3 + Ax^2 + x$

INPUT: $P = (x, z)$: Montgomery curve point in xz representation.
OUTPUT: $[3]P = (x', z')$.

1: $t_1 \leftarrow x^2$; $t_2 \leftarrow z^2$;
2: $t_3 \leftarrow t_1 + t_2$
3: $t_4 \leftarrow (A/2) \cdot ((x + z)^2 - t_3) + t_3$ // NB: $A/2$ can be precomputed for a given curve
4: $t_3 \leftarrow (t_1 - t_2)^2$;
5: $t_1 \leftarrow (4t_1 \cdot t_4 - t_3)^2$; $t_2 \leftarrow (4t_2 \cdot t_4 - t_3)^2$;
6: $x' \leftarrow x \cdot t_2$; $z' \leftarrow z \cdot t_1$;
7: **return** (x', z')

The cost of our tripling is $5\mathbf{M}+6\mathbf{S}+9\mathbf{A}$ (or one less multiplication in scenarios where the curve coefficient A can be carefully chosen and fixed) with 4 ancillary variables, counting each left shift as an addition. It is less expensive than the previously best tripling algorithm in the literature, which only attains $6\mathbf{M} + 5\mathbf{S} + 7\mathbf{A}$ with 8 ancillary variables [11, Appendix B]. Note that this tripling algorithm can be employed in the key (de)compression operations since they do not require the curve coefficient A to be in projective form. The projective version is only required in the computation of 3^n-isogenies where field inversions can be avoided if projective form is adopted. That is the case of the tripling formula by Faz *et al.* [7], which costs $7\mathbf{M} + 5\mathbf{S} + 9\mathbf{A}$.

7 Implementation and experimental results

Our improved key compression and decompression techniques are implemented on top of the SIDH C library [10] so that a full-fledge key-exchange is available. We left the previous (de)compression functions in the new version so that one can replicate the experiments and comparisons.

Since we only process public information (compression and decompression of public keys), side-channel attacks are not an issue, and faster non-isochronous algorithms like extended Euclidean algorithm have been adopted.

Table 4: Benchmarks in cycles on an Intel Core i5 clocked at 2.9 GHz (clang compiler with `-O3` flag, and $\mathbf{s} = \mathbf{m}$ in this implementation).

	2^m-torsion ($w = 2$)			3^n-torsion ($w = 1$)		
operations	SIDH v2.0 [5]	ours	ratio	SIDH v2.0 [5]	ours	ratio
basis generation	24497344	1690452	**14.49**	20632876	17930437	**1.15**
discrete log.	6206319	2776568	**2.24**	4710245	3069234	**1.53**
four pairings	33853114	25755714	**1.31**	39970384	30763841	**1.30**
compression	78952537	38755681	**2.04**	78919488	61768917	**1.28**
decompression	30057506	9990949	**3.01**	25809348	23667913	**1.09**

The initial public curve is the usual Montgomery curve $E_0 : y^2 = x^3 + x$ defined over $\mathbb{F}_{p^2}$ where $p = 2^{372}3^{239} - 1$. It is worth mentioning that before applying our (de)compression techniques, the SIDH v2.0 library was first modified to perform Alice's key generation with both points P_A and Q_A defined over the extension $E_0(\mathbb{F}_q)\backslash E_0(\mathbb{F}_p)$ instead of defining P_A in the base field as suggested in [5] . The approach in [5] starts with point $P_A = (x, y) \in E_0(\mathbb{F}_p)$ over the base field and then applies the distortion map τ to get a linearly independent point $Q_A = \tau(P_A) = (-x, iy)$ lying on the trace zero group. This optimization cannot be combined with our techniques because using distortion maps on binary torsions only gives a basis $\langle P_A, \tau(P_A)\rangle = E_0[2^{m-1}]$ of a smaller group of order $2^{2(m-1)}$, and in this case the images of P_A and $Q_A = \tau(P_A)$ under Bob's isogeny consequently generate a smaller torsion as well, i.e., $\langle \phi_B(P_A), \phi_B(Q_A)\rangle = E_B[2^{m-1}]$. In particular, the reverse basis decomposition technique combined with entangled basis would not work since an entangled basis generates the full 2^m-torsion, and this basis cannot be converted to a basis of a smaller torsion, i.e., the change of basis matrix in Equation 3 would not exist. Therefore, we selected the new points

$$P_A := 3^{239} \cdot (5 + i, \sqrt{(5 + i)^3 + 5 + i}) \in E_0(\mathbb{F}_q)\backslash E_0(\mathbb{F}_p)$$

and $Q_A := \tau(P_A) \in E_0(\mathbb{F}_q)\backslash E_0(\mathbb{F}_p)$. Points P_B and Q_B are the ones in [5] since for ℓ^n torsions with ℓ odd, distortion maps do generate the full group $E_0[\ell^n]$ and P_B can be kept over the base field. For the discrete logarithms we set $w = 2$ for the binary case and $w = 1$ for the ternary one. Table 4 summarizes our experimental results with respect to the previous state-of-the-art implementation.

8 Conclusion

In this paper we proposed a range of new algorithms and techniques to speedup the supersingular-isogeny Diffie-Hellman. For example, in the 2^m-torsion using $w = 2$ for the discrete logarithms, the key compression is about $2\times$ faster than SIDH library and decompression achieves a factor of $3\times$, while the basis generation itself is nearly $14.5\times$ faster. The main bottleneck now, by far, is the pairing phase taking about 25.8M cycles against 1.7M for basis generation and 2.8M for the discrete logarithm phase. It is worthwhile to point out that the techniques of entangled basis generation and the optimal strategy applied to solve smooth-order discrete logarithms not only set up new

speed records for those tasks, but might find new applications in different contexts in cryptography. We leave the possibility of extending the new entangled basis generation technique to non-binary torsions as an open problem.

References

1. R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel, and C. Leonardi. Key compression for isogeny-based cryptosystems. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pages 1–10. ACM, 2016.
2. P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *Advances in Cryptology – Crypto 2002*, number 2442 in Lecture Notes in Computer Science, pages 354–368, Santa Barbara (CA), USA, 2002. Springer.
3. D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange. Elligator: Elliptic-curve points indistinguishable from uniform random strings. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 967–980. ACM, 2013.
4. D. J. Bernstein and T. Lange. Analysis and optimization of elliptic-curve single-scalar multiplication. In *Finite Fields and Applications: Proceedings of Fq8*, number 461 in Contemporary Mathematics, pages 1–18, Providence (RI), USA, 2008. American Mathematical Society.
5. C. Costello, D. Jao, P. Longa, M. Naehrig, J. Renes, and D. Urbanik. Efficient compression of SIDH public keys. In *Advances in Cryptology – Eurocrypt 2017*, number 10210 in Lecture Notes in Computer Science, pages 679–706, Paris, France, 2017. Springer.
6. L. De Feo, D. Jao, and J. Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.
7. A. Faz-Hernández, J. López, E. Ochoa-Jiménez, and F. Rodríguez-Henríquez. A faster software implementation of the supersingular isogeny Diffie-Hellman key exchange protocol. Cryptology ePrint Archive, Report 2017/1015, 2017.
8. D. Husemöller. *Elliptic Curves*, volume 111 of *Graduate Texts in Mathematics*. Springer, New York, USA, 2nd edition, 2004.
9. D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *Post-Quantum Cryptography – PQCrypto 2011*, number 7071 in Lecture Notes in Computer Science, pages 19–34, Taipei, Taiwan, 2011. Springer.
10. MS SIDH team. SIDH v2.0, 2017. https://github.com/Microsoft/PQCrypto-SIDH.
11. S. R. S. Rao. Three dimensional Montgomery ladder, differential point tripling on Montgomery curves and point quintupling on Weierstrass and Edwards curves. In *Progress in Cryptology – AfricaCrypt 2016*, number 9646 in Lecture Notes in Computer Science, pages 84–106, Fes, Morocco, 2016. Springer.
12. E. Schaefer and M. Stoll. How to do a p-descent on an elliptic curve. *Transactions of the American Mathematical Society*, 356(3):1209–1231, 2004.
13. V. Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2005.
14. J. H. Silverman. *The Arithmetic of Elliptic Curves*, volume 106 of *Graduate Texts in Mathematics*. Springer, New York, USA, 2nd edition, 2009.
15. M. R. Spiegel and J. Liu. *Mathematical Handbook of Formulas and Tables*. Schaum's Outline Series. McGraw-Hill, New York, USA, 2nd edition, 1999.

A Pairing algorithms

Algorithm A.1: Tate2$(P, [Q_j], m)$: reduced Tate pairing of order $r = 2^m$

INPUT: Curve $E : y^2 = x^3 + ax + b$
– Point $P = [X_P : Y_P : Z_P]$ on E of order 2^m
– t points $Q_j = [X_{Q_j} : Y_{Q_j} : Z_{Q_j}]$ on E, $Z_{Q_j} \in \{0, 1\}$
OUTPUT: List of t values $e_{2^m}(P, Q_j)$

$X \leftarrow X_P$; $Y \leftarrow Y_P$; $Z \leftarrow Z_P$; $T \leftarrow Z^2$
▷ NB: the following operations are in $\mathbb{F}_{p^2}$
for $j \leftarrow 0$ **to** $t - 1$ **do**
  $f_j \leftarrow 1$; $h_j \leftarrow T \cdot X_{Q_j} - X$;
end for
for $k \leftarrow 0$ **to** $m - 1$ **do**
▷ *point doubling and line function construction:*
  $X_2 \leftarrow X^2$; $Y_2 \leftarrow Y^2$; $Y_4 \leftarrow Y_2^2$
  $M \leftarrow 3X_2 + a \cdot T^2$
  $S \leftarrow 2((X + Y_2)^2 - X_2 - Y_4)$
  $X' \leftarrow M^2 - 2S$
  $Y' \leftarrow M \cdot (S - X') - 8Y_4$
  $Z' \leftarrow (Y + Z)^2 - Y_2 - T$;
  $T' \leftarrow (Z')^2$; $L \leftarrow Z' \cdot T$; $W \leftarrow 2Y_2$
  if $Z' = 0$ **then** // *exception for points in* $[2]E$
    $X' \leftarrow 0$; $Y' \leftarrow 1$
  end if
▷ *line function evaluation and accumulation:*
  for $j \leftarrow 0$ **to** $t - 1$ **do**
    if $Z' \neq 0$ **then**
      $g \leftarrow M \cdot h_j + W - L \cdot Y_{Q_j}$
      $h_j \leftarrow T' \cdot X_{Q_j} - X'$
      $g \leftarrow g \cdot h_j^*$
    else // *exception for points in* $[2]E$
      $g \leftarrow h_j$;
    end if
    $f_j \leftarrow f_j^2$; $f_j \leftarrow f_j \cdot g$;
  end for
  $X \leftarrow X'$; $Y \leftarrow Y'$;
  $Z \leftarrow Z'$; $T \leftarrow T'$;
end for
return $[(Z_{Q_j} \stackrel{?}{\neq} 0)\, f_j^{(p^2-1)/r} : 1 \mid j = 0 \ldots t - 1]$

Algorithm A.2: Tate3$(P, [Q_j], n)$: reduced Tate pairing of order $r = 3^n$

INPUT: Curve $E : y^2 = x^3 + ax + b$
– Point $P = [X_P : Y_P : Z_P]$ on E of order 3^n
– t points $Q_j = [X_{Q_j} : Y_{Q_j} : Z_{Q_j}]$ on E, $Z_{Q_j} \in \{0, 1\}$
OUTPUT: List of t values $e_{3^n}(P, Q_j)$

$X \leftarrow X_P$; $Y \leftarrow Y_P$; $Z \leftarrow Z_P$; $T \leftarrow Z^2$;
▷ NB: the following operations are in $\mathbb{F}_{p^2}$
for $j \leftarrow 0$ **to** $t - 1$ **do**
  $f_j \leftarrow 1$; $h_j \leftarrow T \cdot X_{Q_j} - X$;
end for
for $k \leftarrow 0$ **to** $n - 1$ **do**
▷ *point tripling and parabola function construction:*
  $X_2 \leftarrow X^2$; $Y_2 \leftarrow Y^2$; $Y_4 \leftarrow Y_2^2$; $T_2 \leftarrow T^2$;
  $M \leftarrow 3X_2 + a \cdot T_2$; $M_2 \leftarrow M^2$
  $D \leftarrow (X + Y_2)^2 - X_2 - Y_4$;
  $F \leftarrow 6D - M_2$; $F_2 \leftarrow F^2$
  $W \leftarrow 2Y_2$; $W' \leftarrow 2W$; $S \leftarrow 16Y_4$
  $U \leftarrow (M + F)^2 - M_2 - F_2 - S$; $U' \leftarrow S - U$
  $X' \leftarrow 4(X \cdot F_2 - W' \cdot U)$
  $Y' \leftarrow 8Y \cdot (U \cdot U' - F \cdot F_2)$
  $Z' \leftarrow (Z + F)^2 - T - F_2$; $T' \leftarrow (Z')^2$
  $L \leftarrow ((Y + Z)^2 - Y_2 - T) \cdot T$; $F' \leftarrow 2F$;
  if $Z' = 0$ **then** // *exception for points in* $[3]E$
    $X' \leftarrow 0$; $Y' \leftarrow 1$
  end if
▷ *parabola function evaluation and accumulation:*
  for $j \leftarrow 0$ **to** $t - 1$ **do**
    $d \leftarrow W - L \cdot Y_{Q_j}$;
    if $Z' \neq 0$ **then**
      $g \leftarrow (M \cdot h + d)(U' \cdot h + F' \cdot d)(W' \cdot h + F)^*$
      $h \leftarrow T' \cdot X_{Q_j} - X'$; $g \leftarrow g \cdot h^*$
    else // *exception for points in* $[3]E$
      $g \leftarrow (M \cdot h + d) \cdot Y^*$
    end if
    $f \leftarrow f^3$; $f \leftarrow f \cdot g$
  end for
  $X \leftarrow X'$; $Y \leftarrow Y'$; $Z \leftarrow Z'$; $T \leftarrow T'$
end for
return $[(Z_{Q_j} \stackrel{?}{\neq} 0)\, f_j^{(p^2-1)/r} : 1 \mid j = 0 \ldots t - 1]$