Isogeny-based key compression without pairings

Geovandro C. C. F. Pereira[1] and Paulo S. L. M. Barreto[2]

[1] Institute for Quantum Computing, University of Waterloo; evolutionQ Inc.
geovandro.pereira@uwaterloo.ca
[2] University of Washington Tacoma
pbarreto@uw.edu

Abstract. SIDH/SIKE-style protocols benefit from key compression to minimize their bandwidth requirements, but proposed key compression mechanisms rely on computing bilinear pairings. Pairing computation is a notoriously expensive operation, and, unsurprisingly, it is typically one of the main efficiency bottlenecks in SIDH key compression, incurring processing time penalties that are only mitigated at the cost of trade-offs with precomputed tables. We address this issue by describing how to compress isogeny-based keys without pairings. As a bonus, we also substantially reduce the storage requirements of other operations involved in key compression.

1 Introduction

Supersingular Isogeny Diffie-Hellman (SIDH) public keys are, in an abstract sense, triples $(A, x_P, x_Q) \in \mathbb{F}_{p^2}^3$ where A specifies an elliptic curve in Montgomery form $E : y^2 = x^3 + Ax^2 + x$, and x_P, x_Q are the x-coordinates of points P, Q on E (which in turn are the images of fixed points on a certain starting curve through a private prime-power-degree isogeny).

Although such triples naively take $6 \lg p$ bits of storage or bandwidth, better representations are known. Since P, Q are chosen to be in torsion subgroups of form $E[\ell^m]$ with $m \lg \ell \approx (1/2) \lg p$, they can be represented on some conventional public ℓ^m-torsion basis (R, S) in E by four coefficients from $\mathbb{Z}/\ell^m\mathbb{Z}$, namely $P = [a]R + [b]S$ and $Q = [c]R + [d]S$, so that the public key becomes $(A, a, b, c, d) \in \mathbb{F}_{p^2} \times (\mathbb{Z}/\ell^m\mathbb{Z})^4$ and the key bandwidth decreases to about $2 \lg p + 4 \cdot (1/2) \lg p = 4 \lg p$ bits [2]. Besides, for SIDH-style protocols only the subgroup $\langle P, Q \rangle$ is actually relevant, not the points P, Q themselves. This redundancy can be exploited for further compression, since only three out of those four point coefficients above are needed to specify that subgroup. Thus, each key is essentially represented by a tuple $(A, a, b, c) \in \mathbb{F}_{p^2} \times (\mathbb{Z}/\ell^m\mathbb{Z})^3$ (plus one bit to identify which coefficient is omitted). This finally brings the key size down to $3.5 \lg p$ bits, nearly halving the naive requirements above [5]. At the time of writing, this appears to be the minimum attainable (and hence optimal) bandwidth per key.

This size reduction, however, incurs a high computational overhead. Existing research works [2,5,11,17,18] have achieved substantial improvements, but they

all rely on the computation of bilinear pairings to project (non-cyclic) elliptic discrete logarithm instances into the multiplicative group of units in $\mathbb{F}_{p^2}^*$, where variants of the Pohlig-Hellman algorithm [13] are effectively applied. Precomputed tables can substantially speed up pairing computations [11], but at the cost of increasing storage requirements, compounding with those incurred by the computation of discrete logarithms. More recently a work on reducing the storage component has been proposed [9].

Despite all improvements, pairing computation has consistently remained the efficiency bottleneck in all of those works (specially the case $\ell = 2$ per [11]).

Our contribution: We describe how to compute discrete logarithms as required to obtain the four a, b, c, d coefficients (which can then be reduced down to three in the usual fashion) via tailored ECDLP instances without resorting to pairings. Our proposed method relies almost entirely on arithmetic for elliptic curve points defined over $\mathbb{F}_p$ for the discrete logarithm computation itself. Arithmetic over $\mathbb{F}_{p^2}$ is only ever needed in evaluating very simple and efficient maps between curves and groups, and to complete partially computed logarithms. Precomputation trade-offs are possible in the same way as they are for discrete logarithms on $\mathbb{F}_{p^2}^*$, and if adopted, only the tables for a single, fixed, public group generator are needed for each value of ℓ. Moreover, we show how to halve the storage requirements of windowed computation of discrete logarithm when the index of the prime-power group order is not a multiple of the window size.

The remainder of this paper is organized as follows. In Section 2 we define the fundamental notions and notations our proposal is based upon. We recap the basic Pohlig-Hellman algorithm in Section 3, and then we discuss the improvements made possible by the adoption of strategy graphs and propose several associated techniques in Section 4. We introduce our methods to compute elliptic curve discrete logarithms efficiently as required for SIDH key compression in Section 5, and assess its performance in Section 6. Finally, we discuss the results and conclude in Section 7.

2 Preliminaries

Consider a prime $p = 2^{e_2} \cdot q^{e_q} - 1$ for some $e_2 \geq 2$, some $e_q > 0$ and some small odd prime q (typically $q = 3$). We will represent the finite field $\mathbb{F}_{p^2} \simeq \mathbb{F}_p[i]/\langle i^2+1\rangle$. Also, we will be mostly concerned with *supersingular* elliptic curves in Montgomery form $E : x^3 + Ax^2 + x$ with $A \in \mathbb{F}_{p^2}$ (or $A \in \mathbb{F}_p$ in a few cases). When required for clarity, the curve defined by a particular value of A will be denoted E_A.

Let $P, Q \in E[\ell^{e_\ell}]$ with $\ell \in \{2, q\}$. We say that P *is above* Q if $Q = [\ell^j]P$ for some $0 < j \leqslant e_\ell$.

The *trace map* on $E/\mathbb{F}_p$ is the linear map $\mathrm{tr} : E(\mathbb{F}_{p^2}) \to E(\mathbb{F}_p)$ defined as $\mathrm{tr}(P) := P + \Phi(P)$, where $\Phi : E(\mathbb{F}_{p^2}) \to E(\mathbb{F}_{p^2})$ is the Frobenius endomorphism, $\Phi(x, y) := (x^p, y^p)$. If $P \in E(\mathbb{F}_p)$, then $\mathrm{tr}(P) = [2]P$. Conversely, if $\mathrm{tr}(P) = [2]P$, then $P \in E(\mathbb{F}_p)$.

A *distortion map* is an endomorphism $\psi : E \to E$ such that $\psi(P) \notin \langle P \rangle$. Efficiently computable distortion maps are hard to come by for most curves [8], but the special curve E_0 is equipped with a simple one, namely, $\psi : E_0 \to E_0$ defined as $\psi(x, y) := (-x, iy)$ in the Weierstrass and Montgomery models, or $\psi(x, y) := (ix, 1/y)$ in the Edwards model. Notice that $\psi^{-1} = -\psi$, since $\psi(\psi(x, y)) = (x, -y) = -(x, y)$. Besides, if $P \in E(\mathbb{F}_p)$ then $\mathrm{tr}(\psi(P)) = \mathcal{O}$, as one can see by directly computing $\mathrm{tr}(\psi(x, y)) = \mathrm{tr}(-x, iy) = (-x, iy) + ((-x)^p, (iy)^p) = (-x, iy) + (-x, -iy) = (-x, iy) - (-x, iy) = \mathcal{O}$. For convenience we also define the isomorphism $\tilde{\psi} : E_A \to E_{-A}$ that maps $\tilde{\psi}(x, y) := (-x, iy)$ in the Weierstrass and Montgomery models, and $\tilde{\psi}(x, y) := (ix, 1/y)$ in the Edwards model.

An *isogeny* between two elliptic curves E and E' over a field $\mathbb{K}$ is a morphism $\varphi : E(\overline{\mathbb{K}}) \to E'(\overline{\mathbb{K}})$ specified by rational functions, $\varphi(x, y) = (r(x), s(x)y)$ for some $r, s \in \mathbb{K}[x]$, satisfying $\varphi(\mathcal{O}_E) = \mathcal{O}_{E'}$. Writing $r(x) = n(x)/d(x)$ for $n, d \in \mathbb{K}[x]$ with $\gcd(n, d) = 1$, the *degree* of φ is $\deg \varphi := \max\{\deg n, \deg d\}$. For short, if $\deg \varphi = g$ we say that φ is a g-isogeny. If $r(x)$ is non-constant, φ is said to be *separable*, in which case $\deg \varphi = \# \ker \varphi = \#\{P \in E \mid \varphi(P) = \mathcal{O}\}$. An isogeny is said to be of *prime power* if $\deg \varphi = \ell^m$ for some small prime ℓ.

We will make extensive use of the 2-isogeny $\varphi_0 : E_0 \to E_6$ with kernel $\langle (i, 0) \rangle$ and its dual $\widehat{\varphi}_0 : E_6 \to E_0$ with kernel $\langle (0, 0) \rangle$.

Supersingular isogeny-based cryptosystems build prime-power isogenies starting from a fixed curve. The starting curve originally adopted for SIDH protocols is E_0, but since all power-of-2 isogenies leaving E_0 must necessarily pass through the 2-isogenous curve E_6, a recent trend pioneered by the SIKE scheme [1] is to adopt E_6 as the starting curve instead.

An SIDH-style private key will thus be an isogeny $\varphi : E_6 \to E$ of degree λ^{e_λ}, and the corresponding public key is, formally, a triple $(E, P := \varphi(P_6), Q := \varphi(Q_6))$ where (P_6, Q_6) is a basis of $E_6[\ell^{e_\ell}]$, where $\lambda \in \{2, q\}$ and $\ell \in \{2, q\} \setminus \{\lambda\}$.

The basic idea of compressing public keys is to unambiguously specify a public basis (R, S) for $E[\ell^{e_\ell}]$ and obtaining coefficients $a, b, c, d \in \mathbb{Z}/\ell^{e_\ell}\mathbb{Z}$ such that $P = [a]R + [b]S$ and $Q = [c]R + [d]S$. Since the dual isogeny $\widehat{\varphi} : E \to E_6$ is readily available and efficiently applicable during key pair generation [11], this process can be equivalently stated on curve E_6 itself. Namely, defining $R_6 := \widehat{\varphi}(R)$ and $S_6 := \widehat{\varphi}(S)$, that decomposition is equivalent to $\widehat{\varphi}(P) = [\lambda^{e_\lambda}]P_6 = [a]\widehat{\varphi}(R) + [b]\widehat{\varphi}(S) = [a]R_6 + [b]S_6$ and $\widehat{\varphi}(Q) = [\lambda^{e_\lambda}]Q_6 = [c]\widehat{\varphi}(R) + [d]\widehat{\varphi}(S) = [c]R_6 + [d]S_6$. Thus, if one can find $\hat{a}, \hat{b}, \hat{c}, \hat{d} \in \mathbb{Z}/\ell^{e_\ell}\mathbb{Z}$ such that

$$\begin{aligned} P_6 &= [\hat{a}]R_6 + [\hat{b}]S_6, \\ Q_6 &= [\hat{c}]R_6 + [\hat{d}]S_6, \end{aligned} \tag{1}$$

the desired coefficients for the compressed public key might be obtained from these after multiplying them by the scale factor λ^{e_λ}. Yet, this is actually unnecessary, since those four coefficients would be further compressed into a triple $(b/a, c/a, d/a)$ or $(a/b, c/b, d/b)$ modulo ℓ^{e_ℓ} anyway, eliminating the need for any scale factor.

With this in mind, we henceforth assume that basis (R_6, S_6) has been prepared as outlined above, and we focus our attention to the specific task of finding $\hat{a}$, $\hat{b}$, $\hat{c}$, and $\hat{d}$ without any further reference to the private isogeny φ or its dual. Since we will also resort later to other curve models than Montgomery's in our algorithms, this assumption will also help keeping all computations contained in separate, well-delimited realms.

Being able to efficiently compute discrete logarithms is thus at the core of key compression methods. We next review the Pohlig-Hellman discrete logarithm algorithm and ways of making it as close to optimal as currently viable.

3 The Pohlig-Hellman algorithm

The Pohlig-Hellman method (Algorithm 3.1) to compute the discrete logarithm of $P = [d]G \in \langle G \rangle$, where $\langle G \rangle$ is a cyclic Abelian group of smooth order ℓ^m, requires solving equations of form:

$$[\ell^{m-1-k}]P_k = [d_k]L$$

for $k = 0, \ldots, m-1$, where $L := [\ell^{m-1}]G$ has order ℓ, $d_k \in \{0, \ldots, \ell-1\}$ is the ℓ-ary digit of d, $P_0 := P$, and P_{k+1} depends on P_k and d_k.

Algorithm 3.1 Pohlig-Hellman discrete logarithm algorithm

INPUT: cyclic Abelian group $\langle G \rangle$, challenge $P \in \langle G \rangle$.
OUTPUT: d: base-ℓ digits of $\log_G P$, i.e. $[d]G = P$.

1: $L \leftarrow [\ell^{m-1}]G$ ▷ NB: $[\ell]L = \mathcal{O}$
2: $d \leftarrow 0$, $P_0 \leftarrow P$
3: **for** $k \leftarrow 0$ **to** $m-1$ **do**
4: $\quad V_k \leftarrow [\ell^{m-1-k}]P_k$
5: $\quad$ find $d_k \in \{0, \ldots, \ell-1\}$ such that $V_k = [d_k]L$
6: $\quad d \leftarrow d + d_k\ell^k$, $P_{k+1} \leftarrow P_k - [d_k\ell^k]G$
7: **end for** ▷ NB: $[d]G = P$
8: **return** d

A simple improvement for Algorithm 3.1 would be to precompute a table $\mathsf{T}[k][c] := [c]([\ell^k]G)$ for all $0 \le k < m$ and $0 \le c < \ell$. This way, the condition on line 5, which essentially solves a discrete logarithm instance in the subgroup $\langle L \rangle$ of small order ℓ, would read $V_k = \mathsf{T}[m-1][d_k]$, and the second assignment on line 6 would read $P_{k+1} \leftarrow P_k - \mathsf{T}[k][d_k]$ instead.

The running time of this algorithm is dominated by line 4 (and line 6 in the naive approach without tables), leading to an overall $O(m^2)$ complexity as measured in basic group operations. Shoup [14, Chapter 11] shows that this approach is far from optimal, but although his RDL algorithm [14, Section 11.2.3] provides a framework targeting optimality, it lacks an effective strategy to attain it. A practical technique for that purpose has been proposed by Zanon *et al.* [18, Section 6] that uses so-called *strategy graphs*. We next recap that technique in Section 4, which was originally proposed in the context of finite field discrete

logarithms. We subsequently show (Sections 4.1 to 4.5) how to adapt it for the ECDLP case.

4 Strategy graphs

Let Δ be a graph with vertices $\{\Delta_{j,k} \mid j \geq 0, k \geq 0, j+k \leq m-1\}$ (that is, a triangular grid). Each vertex has either two downward outgoing edges, or no edges at all. Vertices $\Delta_{j,k}$ with $j+k < m-1$ are called internal vertices and have two weighted edges: a left-edge $\Delta_{j,k} \to \Delta_{j+1,k}$, and a right-edge $\Delta_{j,k} \to \Delta_{j,k+1}$. All left-edges are assigned weight $p > 0$ and all right-edges are assigned weight $q > 0$. Vertices $\Delta_{j,k}$ with $j+k = m-1$ are *leaves* since they have no outgoing edges. Vertex $\Delta_{0,0}$ is called the *root*.

A subgraph of Δ that contains a given vertex v, all leaves that can be reached from v and no vertex that cannot be reached from v is called a *strategy*. A *full* strategy is a strategy that contains the root.

Notice that a strategy is not required to contain all vertices reachable from v, and is thus not necessarily unique. This establishes the *optimal strategy problem*, that consists of finding a full strategy of minimum weight. This problem can be solved with the De Feo *et al.* $O(m^2)$ dynamic programming algorithm [7, Equation 5] or the Zanon *et al.* $O(m \lg m)$ method [18, Algorithm 6.2]. The latter is quoted in Appendix A.1 for reference.

Strategy graphs lend themselves to efficient prime-power-degree isogeny calculation [7], and can also model potential variants of the Pohlig-Hellman algorithm in a prime-power-order Abelian group [17,18]. Namely, an in-order traversal of a strategy spells out which operations are to be performed: left-edges correspond to multiplication by ℓ, and right-edges variously corresponds to applying ℓ-isogenies in the former case, or to erasing one base-ℓ digit from a discrete logarithm in the latter. Specifically in the solution of a discrete logarithm instance $P = [d]G$, these edge operations associate the value $[\ell^j](P - [d \bmod \ell^k]G)$ to vertex $\Delta_{j,k}$ for all j, k.

The reason this works is that, for those tasks, the overall calculation result (a prime-power-degree isogeny or a discrete logarithm) consists of a sequence of values computed at the leaves, and the value at each leaf depends only on the values computed at the leaves to its left. The original Pohlig-Hellman algorithm corresponds to a simple (non-optimal) traversal strategy whereby all $m(m+1)/2$ left-edges (i.e. multiplications by ℓ) are traversed top-down and left-to-right, while the only right-edges traversed (corresponding to the elimination of the least significant digits from the discrete logarithm being retrieved) are those on the rightmost diagonal of the strategy as a whole.

This idea is expressed in Algorithm 4.1, which is adapted from [17,18, Section 6]. Its complexity, measured by the number of edges traversed in a balanced strategy whereby the left-edge and right-edge costs are equal (and hence the number of leaves at each side of the strategy vertices is the same), is defined by the recurrence $T(m) = 2T(m/2) + m$ with $T(1) = 0$, yielding the overall cost $\Theta(m \lg m)$ which is very close to the minimal $\Theta(m \lg m / \lg \lg m)$

complexity [15]. The cost of an optimal albeit unbalanced strategy can be shown to lie asymptotically close to this [7, Equation 9].

Algorithm 4.1 Traverse$(V, j, k, z, \mathsf{path}, \mathsf{T})$

PURPOSE: retrieve d such that $[d]G = P$ for a cyclic Abelian group $\langle G \rangle$ of order ℓ^m.
INPUT: $V := [\ell^j](P - [d \bmod \ell^k]G)$: value associated to vertex $\Delta_{j,k}$;
j, k: coordinates of vertex $\Delta_{j,k}$;
z: number of leaves in the subtree rooted at $\Delta_{j,k}$;
path: traversal path (output of Algorithm A.1);
T: lookup table such that $\mathsf{T}[u][c] := [c]([\ell^u]G)$ for all $0 \leq u < m$ and $0 \leq c < \ell$.
OUTPUT: base-ℓ digits of d such that $[d]G = P$.
REMARK: the initial call is Traverse$(P, 0, 0, m, \mathsf{path}, \mathsf{T})$.

1: **if** $z > 1$ **then**
2: $t \leftarrow \mathsf{path}[z]$ ▷ z leaves: t to the left exp, $z - t$ to the right
3: $V' \leftarrow [\ell^{z-t}]V$ ▷ go left $(z - t)$ times
4: Traverse$(V', j + (z - t), k, t, \mathsf{path}, \mathsf{T})$
5: $V' \leftarrow V - \sum_{h=k}^{k+t-1} \mathsf{T}[j + h][d_h]$ ▷ go right t times
6: Traverse$(V', j, k + t, z - t, \mathsf{path}, \mathsf{T})$
7: **else** ▷ leaf
8: find $d_k \in \{0, \ldots, \ell - 1\}$ such that $V = \mathsf{T}[m - 1][d_k]$ ▷ recover the k-th digit d_k of the discrete logarithm from $V = [d_k]([\ell^{m-1}]G)$
9: **end if**

4.1 Choosing the curve model

Algorithm 4.1 makes extensive use of point multiplications by (powers of) ℓ (line 3) and also point additions/subtractions (line 5). This makes the Montgomery model less suited for this task.

For the most important practical cases of $\ell = 2$ and $\ell = 3$, the (twisted) Edwards model [3] seems to be the most adequate, specifically *inverted twisted Edwards coordinates* for $\ell = 2$ and *projective twisted Edwards coordinates* for $\ell = 3$.

In the former case, with the adoption of inverted twisted Edwards coordinates point addition costs $9\mathbf{m}+1\mathbf{s}+6\mathbf{a}$ (or $8\mathbf{m}+1\mathbf{s}+6\mathbf{a}$ if one of the points is affine), and point doubling costs $3\mathbf{m}+4\mathbf{s}+6\mathbf{a}$. In the latter case, projective twisted Edwards coordinates enable point addition at the cost $10\mathbf{m} + 1\mathbf{s} + 6\mathbf{a}$ (or $9\mathbf{m} + 1\mathbf{s} + 6\mathbf{a}$ if one of the points is affine), and point tripling [4], which is here more relevant than point doubling, costs $9\mathbf{m} + 3\mathbf{s} + 9\mathbf{a}$.

4.2 Handling the leaves

In a generalized, windowed implementation of Algorithm 4.1, that is, when expressing discrete logarithms in base ℓ^w for some $w \geqslant 1$ instead of simply sticking to base ℓ, finding digits $d_k \in \{0, \ldots, \ell^w - 1\}$ in line 8 depends on efficiently looking up the point V in a table containing the whole subgroup $\langle [\ell^{m-w}]G \rangle \subset E(\mathbb{F}_p)$,

which consists of ℓ^w elements. Since point V is typically available in projective coordinates as discussed in Section 4.1, lookup operations are not straightforward in the sense that hash table or binary search techniques are not available. This will incur extra costs that must be carefully minimized.

First, we observe that, since the whole ℓ^w-torsion is searched, lookups can be initially restricted to one of the coordinates, namely, the y-coordinate in the Edwards models, since for each such value there are no more than two points of form (x, y) and $(-x, y)$ in the torsion. This reduces the table size in half.

Second, in either projective twisted or inverted twisted coordinates, the point comparisons are made between points of form $[X : Y : Z]$ against points of form $[x' : y' : 1]$, and in both cases equality holds iff $Y = y' \cdot Z$ and $X = x' \cdot Z$, where the second comparison is performed exactly once after identifying the correct y'. Since there are $\lceil \ell^w/2 \rceil$ distinct values of y' and they are expected to occur uniformly, the expected number of $\mathbb{F}_p$ multiplications is $\lceil \ell^w/4 \rceil + 1$.

This cost limits the values of w that can be chosen in practice, but these coincide with the values that make table T itself too large for practical deployment anyway [17,18].

4.3 Windowing and signed digits

In order to reduce processing time, Algorithm 4.1 can be optimized to use a more general base ℓ^w to express the digits. This reduces the number of leaves of Δ from m to m/w. In this case, although the total cost of left edge traversals is unchanged (since each left traversal now involves computing w multiplications by ℓ), the cost of right traversals is greatly reduced as we can now remove larger digits (i.e. digits modulo ℓ^w) at the same cost of removing a single digit modulo ℓ. This computation time reduction comes with an associated storage overhead because although the number of rows in table $\mathsf{T}[k][c]$ is divided by w, the number of columns grows exponentially in w, i.e., $\mathsf{T}[k][c] := [c]([\ell^k]G)$ for all $0 \leq k < \lceil m/w \rceil$ and $0 \leq c < \ell^w$.

A straightforward storage reduction by a factor of two can be achieved by using the technique of signed digits [10, Section 14.7.1], which requires storing only elements $0 \leq c \leq \lfloor \ell^w/2 \rfloor$ and then merely deciding whether to add or subtract the corresponding multiple of G. Note that storing the column corresponding to $c = 0$ is not needed in practice because 1) removing such digit is equivalent to subtracting the identity and no lookup is indeed required (remember that the discrete logarithm computation is not required to be constant time), and 2) testing if a leaf element corresponds to $c = 0$ reduces to a simple identity test. In this case, the number of columns is reduced by half since removing a negative digit is equivalent to subtracting the opposite of the table entry corresponding to the positive digit. Such storage reduction first appeared in the SIKE Round 3 submission [1] and was introduced by [9] in the context of key compression and was deployed in the official SIKE Round 3 implementation submitted to NIST. The above set of optimizations (adapted to elliptic curve discrete logarithms) is summarized by Algorithm 4.2.

Algorithm 4.2 TraverseWSign$(V, j, k, z, \mathsf{path}, \mathsf{T}, w)$

PURPOSE: retrieve d such that $[d]G = P$ for a cyclic Abelian group $\langle G \rangle$ of order ℓ^m.
INPUT: $V := [L^j](P - [d \bmod L^k]G)$: value associated to vertex $\Delta_{j,k}$;
j, k: coordinates of vertex $\Delta_{j,k}$;
z: number of leaves in the subtree rooted at $\Delta_{j,k}$;
path: traversal path (output of Algorithm A.1);
T: lookup table such that $\mathsf{T}[u][c] := [c]([\ell^u]G)$ for all $0 \leq u < M := m/w$ and $1 \leq c \leq \lfloor L/2 \rfloor$.
w : window size such that $L := \ell^w$.
OUTPUT: base-L signed digits of d such that $[d]G = P$.
REMARK: the initial call is TraverseWSign$(P, 0, 0, M, \mathsf{path}, \mathsf{T}, w)$ and function $sign(\cdot) \in \{-, +\}$.

1: **if** $z > 1$ **then**
2: $t \leftarrow \mathsf{path}[z]$ ▷ z leaves: t to the left exp, $z - t$ to the right
3: $V' \leftarrow [L^{z-t}]V$ ▷ go left $(z - t)$ times
4: TraverseWSign$(V', j + (z - t), k, t, \mathsf{path}, \mathsf{T}, w)$
5: $V' \leftarrow V - \sum_{h=k \ s.t. \ d_h \neq 0}^{k+t-1} sign(d_h)\mathsf{T}[j + h][|d_h|]$ ▷ go right t times
6: TraverseWSign$(V', j, k + t, z - t, \mathsf{path}, \mathsf{T}, w)$
7: **else** ▷ leaf
8: find $d_k \in \{0, \pm 1, \ldots, \pm\lfloor L/2 \rfloor\}$ such that $V = \{\mathcal{O}, sign(d_k)\mathsf{T}[M - 1][|d_k|]\}$. ▷ recover the k-th digit d_k of the discrete logarithm from $V = [d_k]([\ell^{m-w}]G)$
9: **end if**

4.4 Halving storage requirements for windowed traversal

One can obtain the storage needed by Algorithm 4.2 by directly counting the number of $\mathbb{F}_p$ elements in table $T[u][c]$, i.e., $\#T = 2(m/w)\lfloor \ell^w/2 \rfloor$ elements.

Unfortunately, Algorithm 4.2 only works when w divides the exponent m, which is not the case in multiple SIKE parameters. For instance, in the ternary case ($\ell = 3$) of SIKEp434 and SIKEp751 we have $m = 137$ and $m = 239$, respectively, which are prime exponents and no w works. To circumvent this issue, authors in [18] suggested a modification to the traverse algorithm that requires using a second table of the same size of $T[u][c]$, imposing twice the storage.

We suggest a different approach that does not require an extra table and thus offers half of the storage of that proposed in [18] and adopted in subsequent works [11,12] including the official SIKE submission to NIST [1]. In order to achieve such reduction, assume that $m \equiv t \pmod{w}$ for any $t < w$, and write the discrete logarithm of P with respect to G as $d = q\ell^{m-t} + r$ with $0 \leq q < \ell^t$. Now, instead of invoking Algorithm 4.2 with the challenge P in the first argument we pass $[\ell^t]P$ which is a point whose order has index $m - t$ divisible by w. In this case, instead of recovering the full digit d, we get $r = d \pmod{\ell^{m-t}}$. Having r at hand, observe that the relation $P - [r]G = [q]([\ell^{m-t}]G)$ allows us to compare the point $P - [r]G$ (which can be efficiently computed using the technique described later in Section 4.5) against a very small precomputed table containing the points $[q]([\ell^{m-t}]G)$ for $0 \leq q < \ell^t$ in order to find the correct q. The original logarithm d can then be reconstructed with simple integer

arithmetic. Also note that although this approach introduces an apparently extra small table, it also reduces the number of rows in table $T[u][d]$ due to the smaller order of the discrete logarithm instance, i.e. ℓ^{m-t} instead of ℓ^m. The storage of our method amounts to the following number of $\mathbb{F}_p$ elements:

$$\#T = \begin{cases} 2\left(\frac{m}{w}\right)\lfloor \ell^w/2 \rfloor, & \text{if } t = 0 \\ 2\left(\frac{(m-t)}{w}\lfloor \ell^w/2 \rfloor + \ell^t\right), & \text{otherwise} \end{cases} \tag{2}$$

4.5 Shifted multiplications by scalar

Besides retrieving d such that $[d]G = P$, Algorithm 4.1 can be slightly modified to yield more useful information. Specifically, we will need points of form $[\lfloor d/\ell^\sigma \rfloor]G$ for certain values of $\sigma > 0$, and although this could be carried out from the recovered value of d via multiplication by a scalar (d right-shifted by σ positions in base ℓ), it would incur additional cost that can be mostly averted.

However, when traversing the rightmost diagonal of the strategy graph (characterized by $j = 0$), chunks of ℓ-ary digits d_h from d are erased at line 5 (and implicitly at line 8 as well) by subtracting $[d_h]([\ell^h]G)$ from V. Since the subtracted points overall sum up to the original point $[d]G$ itself, for $h \geqslant \sigma$ one could add chunks of points of form $[d_h]([\ell^{h-\sigma}]G)$ at the same locations along the rightmost diagonal, preserve their sum H, and subtract $[\ell^\sigma]H$ at those locations instead. Since the erasure takes place at the branching points of the strategy graph and there are roughly $\lg m$ (or $\lg(m/w)$ in a windowed base-ℓ^w implementation) such locations along any path from the root to the leaves (including the rightmost diagonal), the additional cost incurred by this is roughly $\sigma \lg m$ multiplications by ℓ (in practice we will have $\ell = 2$, meaning plain doublings) and $\lg m$ additions, far less than the $O(m)$ extra doublings and additions that would be incurred by a plain scalar multiplication.

The result (adapted for base-ℓ^w) is summarized in Algorithm 4.3, which performs the strategy graph traversal, and Algorithm 4.4, which computes the actual discrete logarithms.

5 Projecting elliptic discrete logarithms

As set forth in Equation 1, we are given two bases of ℓ^m-torsion on E_6, (R_6, S_6) and (P_6, Q_6), the latter being fixed. Our task is to compute the coefficients $\hat{a}, \hat{b}, \hat{c}, \hat{d} \in \mathbb{Z}/\ell^m\mathbb{Z}$ such that $P_6 = [\hat{a}]R_6 + [\hat{b}]S_6$ and $Q_6 = [\hat{c}]R_6 + [\hat{d}]S_6$. We will do this by projecting the elliptic curve discrete logarithms onto cyclic subgroups of points defined over $\mathbb{F}_p$, and to attain this goal, we will apply the trace map on carefully crafted torsion bases.

We will discuss the cases of odd ℓ and $\ell = 2$ separately, since the latter will require a somewhat different and more involved approach.

Algorithm 4.3 TraversePlus$(V, m, j, k, z, \mathsf{path}, \mathsf{T}, \sigma, w)$

PURPOSE: retrieve d such that $[d]G = P$ for a cyclic Abelian group $\langle G \rangle$ of order ℓ^m.
INPUT: $V := [L^j](P - [d \bmod L^k]G)$: value associated to vertex $\Delta_{j,k}$;
m: exponent of group order ℓ^m;
j, k: coordinates of vertex $\Delta_{j,k}$;
z: number of leaves in the subtree rooted at $\Delta_{j,k}$;
path: traversal path (output of Algorithm A.1);
T: lookup table such that $\mathsf{T}[u][c] := [c]([L^u]G)$ for all $0 \leq u < m/w$ and $1 \leq c \leq \lfloor L/2 \rfloor$;
σ: shift in the computation of $[\lfloor d/L^\sigma \rfloor]G$.
w: window size such that $L := \ell^w$.
OUTPUT: base-L digits of d such that $[d]G = P$, and point $H := [\lfloor d/L^\sigma \rfloor]G$.

if $z > 1$ **then**
  $t \leftarrow \mathsf{path}[z]$ ▷ z leaves: t to the left exp, $z - t$ to the right
  $V' \leftarrow [L^{z-t}]V$ ▷ go left $(z - t)$ times
  TraversePlus$(V', m, j + (z - t), k, t, \mathsf{path}, \mathsf{T}, \sigma, w)$
  $V' \leftarrow V$
  if $j = 0$ **then** ▷ rightmost diagonal
    $V' \leftarrow V' - \sum_{h=k \ s.t. d_h \neq 0}^{\min(\sigma-1, k+t-1)} sign(d_h)\mathsf{T}[h][|d_h|]$ ▷ σ least significant digits
    $V_H \leftarrow \sum_{h=\max(k,\sigma) \ s.t. d_h \neq 0}^{k+t-1} sign(d_h)\mathsf{T}[h - \sigma][|d_h|]$ ▷ all remaining digits
    if $V_H \neq \mathcal{O}$ **then**
      $V' \leftarrow V' - [L^\sigma]V_H$
      $H \leftarrow H + V_H$
    end if
  else ▷ internal diagonal
    $V' \leftarrow V - \sum_{h=k \ s.t. d_h \neq 0}^{k+t-1} sign(d_h)\mathsf{T}[j + h][|d_h|]$ ▷ go right t times
  end if
  TraversePlus$(V', m, j, k + t, z - t, \mathsf{path}, \mathsf{T}, \sigma, w)$
else ▷ leaf
  find $d_k \in \{0, \pm 1, \ldots, \pm\lfloor L/2 \rfloor\}$ such that $V = \{\mathcal{O}, sign(d_k)\mathsf{T}[m/w - 1][|d_k|]\}$
  if $j = 0$ and $k \geqslant \sigma$ and $d_k \neq 0$ **then**
    $H \leftarrow H + sign(d_k)\mathsf{T}[k - \sigma][|d_k|]$
  end if
end if

Algorithm 4.4 $\mathrm{Dlog}(P, m, \mathsf{path}, \mathsf{T}, \sigma, w)$

PURPOSE: retrieve d such that $[d]G = P$ for a cyclic Abelian group $\langle G \rangle$ of order ℓ^m, together with point $H := [\lfloor d/L^\sigma \rfloor]G$.
INPUT: P: point in $\langle G \rangle$;
m: exponent of group order ℓ^m;
path: traversal path (output of Algorithm A.1);
T: lookup table such that $\mathsf{T}[u][c] := [c]([L^u]G)$ for all $0 \leq u < m/w$ and $1 \leq c \leq \lfloor L/2 \rfloor$;
σ: shift in the computation of $[\lfloor d/L^\sigma \rfloor]G$.
w: window size such that $L := \ell^w$.
OUTPUT: base-L digits of d and point H.
REMARK: A call of form $d \leftarrow \mathrm{Dlog}(P, m, \mathsf{path}, \mathsf{T}, 0, w)$ is meant to disregard/discard H.

1: $d \leftarrow 0, \quad H \leftarrow \mathcal{O}$
2: $\mathrm{TraversePlus}(P, m, 0, 0, m, \mathsf{path}, \mathsf{T}, \sigma, w)$ ▷ NB: this modifies both d and H
3: **return** d, H

5.1 Handling odd ℓ

Let $R_0 := \widehat{\varphi}_0(R_6)$, $S_0 := \widehat{\varphi}_0(S_6)$, $P_0 := \widehat{\varphi}_0(P_6)$, and $Q_0 := \widehat{\varphi}_0(Q_6)$.

Let $G \in E_0(\mathbb{F}_p)$ be a point of order ℓ^m. Then G generates the whole ℓ^m-torsion in $E_0(\mathbb{F}_p)$. Since the trace of a point on E_0 is always in $E_0(\mathbb{F}_p)$, we can write

$$\begin{aligned} \mathrm{tr}(R_0) &= [\zeta_R]G, \quad \mathrm{tr}(\psi(R_0)) = [\xi_R]G, \\ \mathrm{tr}(S_0) &= [\zeta_S]G, \quad \mathrm{tr}(\psi(S_0)) = [\xi_S]G, \end{aligned}$$

for some $\zeta_R, \zeta_S, \xi_R, \xi_S \in \mathbb{Z}/\ell^m\mathbb{Z}$. These four discrete logarithms can be retrieved by applying Algorithm 4.4 (with $\sigma = 0$, since we do not need the point H) to $\langle G \rangle$.

Analogously, we can write

$$\begin{aligned} \mathrm{tr}(P_0) &= [\mu_P]G, \quad \mathrm{tr}(\psi(P_0)) = [\nu_P]G, \\ \mathrm{tr}(Q_0) &= [\mu_Q]G, \quad \mathrm{tr}(\psi(Q_0)) = [\nu_Q]G, \end{aligned}$$

for some $\mu_P, \mu_Q, \nu_P, \nu_Q \in \mathbb{Z}/\ell^m\mathbb{Z}$. Since points P_0 and Q_0 are fixed, these discrete logarithms can be precomputed (in contrast with the previous four logarithms, which must be computed on demand).

With the above notation, applying the dual isogeny $\widehat{\varphi}_0$ to the decomposition of (P_6, Q_6) in base (R_6, S_6) yields:

$$\begin{cases} P_6 = [\hat{a}]R_6 + [\hat{b}]S_6 \\ Q_6 = [\hat{c}]R_6 + [\hat{d}]S_6 \end{cases} \xRightarrow{\widehat{\varphi}_0} \begin{cases} P_0 = [\hat{a}]R_0 + [\hat{b}]S_0 \\ Q_0 = [\hat{c}]R_0 + [\hat{d}]S_0 \end{cases}$$

Applying the trace map to the system on the right and grouping scalar coefficients together yields:

$$\begin{cases} P_0 = [\hat{a}]R_0 + [\hat{b}]S_0 \\ Q_0 = [\hat{c}]R_0 + [\hat{d}]S_0 \end{cases} \xRightarrow{\mathrm{tr}} \begin{cases} \mu_P = \hat{a}\zeta_R + \hat{b}\zeta_S \\ \mu_Q = \hat{c}\zeta_R + \hat{d}\zeta_S \end{cases} \pmod{\ell^m}$$

Correspondingly, applying the distortion map and then the trace map to that system yields:

$$\begin{cases} P_0 = [\hat{a}]R_0 + [\hat{b}]S_0 \\ Q_0 = [\hat{c}]R_0 + [\hat{d}]S_0 \end{cases} \xRightarrow{\mathrm{tr} \circ \psi} \begin{cases} \nu_P = \hat{a}\xi_R + \hat{b}\xi_S \\ \nu_Q = \hat{c}\xi_R + \hat{d}\xi_S \end{cases} \pmod{\ell^m}$$

Overall we are left with two similar linear systems, one for $\hat{a}$ and $\hat{b}$, the other for $\hat{c}$ and $\hat{d}$:

$$M \begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} \mu_P \\ \nu_P \end{bmatrix}, \quad M \begin{bmatrix} \hat{c} \\ \hat{d} \end{bmatrix} = \begin{bmatrix} \mu_Q \\ \nu_Q \end{bmatrix} \pmod{\ell^m}$$

where

$$M := \begin{bmatrix} \zeta_R & \zeta_S \\ \xi_R & \xi_S \end{bmatrix} \pmod{\ell^m}$$

The formal solution of these systems is thus:

$$\begin{bmatrix} \hat{a} \\ \hat{b} \end{bmatrix} = M^{-1} \begin{bmatrix} \mu_P \\ \nu_P \end{bmatrix}, \quad \begin{bmatrix} \hat{c} \\ \hat{d} \end{bmatrix} = M^{-1} \begin{bmatrix} \mu_Q \\ \nu_Q \end{bmatrix} \pmod{\ell^m} \tag{3}$$

where

$$M^{-1} := \frac{1}{\zeta_R\xi_S - \zeta_S\xi_R} \begin{bmatrix} \xi_S & -\zeta_S \\ -\xi_R & \zeta_R \end{bmatrix} \pmod{\ell^m} \tag{4}$$

This solution, of course, requires $D := \zeta_R\xi_S - \zeta_S\xi_R$ to be invertible mod ℓ^m. This is the case for odd ℓ, for which Equations 3 and 4 completely solve the problem.

However, the same approach would not be complete for $\ell = 2$. For any $P_4 \in E_6[4]$, the composition of the 2-isogeny $\widehat{\varphi}_0$ followed by the trace maps pairs of points $(R_6, R_6 + P_4)$ on E_6 to the same point on E_0, and analogously for $(S_6, S_6 + P_4)$.

Efficiently solving this ambiguity requires a slightly different approach, which we will describe next. Indeed, we propose two solutions for this case: one that is slightly simpler conceptually, but requires tables twice as large for Algorithm 4.4, and another one that is considerably more involved in its one-time precomputation (in fact, it relies on the first method for that task), but minimizes the table space.

5.2 Handling $\ell = 2$, first solution

The ideal feature needed to use the trace map as a projection onto a cyclic subgroup of 2^m-torsion would be a basis $(\tilde{P}_6, \tilde{Q}_6)$ where $\mathrm{tr}(\tilde{Q}_6) = \mathcal{O}$. However, this would require a point of form $\tilde{Q}_6 = (-x_Q, iy_Q)$ with $x_Q, y_Q \in \mathbb{F}_p$, or equivalently $(x_Q, y_Q) \in E_{-6}(\mathbb{F}_p)[2^m]$ of *full* order, and no such point exists (the maximum order one can get over $\mathbb{F}_p$ is 2^{m-1} [6, Lemma 1]).

Hence the best possible scenario is a basis $(\tilde{P}_6, \tilde{Q}_6)$ where $\mathrm{tr}([2]\tilde{Q}_6) = \mathcal{O}$, since this is compatible with the order restriction above. The fundamental constraint on $\tilde{Q}_6$ is thus $[2]\tilde{Q}_6 = (-x_H, iy_H) = \tilde{\psi}(H)$ where $H := (x_H, y_H) \in E_{-6}(\mathbb{F}_p)[2^m]$

is any point of order 2^{m-1}. In fact, the computation of discrete logarithms in $\langle [2]\tilde{Q}_6 \rangle$ or a subgroup thereof can actually be carried out in $\langle H \rangle$ instead, with all arithmetic restricted to $\mathbb{F}_p$.

Besides, we want to keep the arithmetic carried out in the computation of discrete logarithms restricted as much as possible to $\mathbb{F}_p$. This constrains $\tilde{P}_6$ to being not only of full order and satisfying $\mathrm{tr}([2]\tilde{P}_6) \neq \mathcal{O}$, but also $[2]\tilde{P}_6 \in E_6(\mathbb{F}_p)$ even though $\tilde{P}_6 \in E_6(\mathbb{F}_{p^2}) \setminus E_6(\mathbb{F}_p)$. This is again compatible with the order restriction above.

With these constraints in mind, selecting a suitable basis $(\tilde{P}_6, \tilde{Q}_6)$ can be carried out as follows. Assume w.l.o.g. that basis (P_6, Q_6) is such that[3] Q_6 is above $(0,0)$, while P_6 is above a different point of order 2. Since we seek any point $\tilde{Q}_6$ of full order with $\mathrm{tr}([2]\tilde{Q}_6) = \mathcal{O}$, we are free to look for a point of form

$$\tilde{Q}_6 = P_6 - [\alpha]Q_6$$

for some α, since more general linear combinations of basis (P_6, Q_6) would merely yield multiples of such a point or a similar one with the roles of P_6 and Q_6 reversed. This specific form also ensures that $\tilde{Q}_6$ is a point of full order. Doubling and then taking the trace from both sides of this equation yields the constraint $\mathcal{O} = \mathrm{tr}([2]\tilde{Q}_6) = \mathrm{tr}([2]P_6) - [\alpha]\mathrm{tr}([2]Q_6)$, that is:

$$\mathrm{tr}([2]P_6) = [\alpha]\mathrm{tr}([2]Q_6)$$

from which $\alpha \pmod{2^{m-2}}$ can be uniquely determined, once and for all, by applying Algorithm 4.4 (with $\sigma = 0$) to $\mathrm{tr}([2]P_6) \in \langle \mathrm{tr}([2]Q_6) \rangle \subset E_6(\mathbb{F}_p)$, and the same value of α can be lifted to a valid solution mod 2^m.

From the above discussion it must hold that $[2]\tilde{Q}_6 = \tilde{\psi}(H)$ for $H \in E_{-6}(\mathbb{F}_p)[2^m]$ of order 2^{m-1}. Because the points from E_{-6} of order 4 above $(0,0)$ are $(1, \pm 2i)$ [7, Section 4.3.2], which are clearly not in $E_{-6}(\mathbb{F}_p)$, point H will necessarily be above one of the other points of order 2, and hence $\tilde{Q}_6$ will not be above $(0,0)$ in $E_6(\mathbb{F}_p)$. Since Q_6 is chosen above $(0,0)$, the pair $(Q_6, \tilde{Q}_6)$ constitutes a basis of 2^m-torsion.

We now seek $\tilde{P}_6$ satisfying $[2]\tilde{P}_6 \in E_6(\mathbb{F}_p)$ and $[2^{m-1}]\tilde{P}_6 = (0,0)$. These two conditions imply $\mathrm{tr}([2]\tilde{P}_6) = [4]\tilde{P}_6 \neq \mathcal{O}$ (for $m > 2$) and the pair $(\tilde{P}_6, \tilde{Q}_6)$ constitutes a basis for $E_6[2^m]$. Yet, finding $\tilde{P}_6$ satisfying these constraints by trial and error is unlikely to be viable.

To overcome this obstacle, we express $\tilde{P}_6$ in basis $(Q_6, \tilde{Q}_6)$ as

$$\tilde{P}_6 = [\beta]Q_6 + [\gamma]\tilde{Q}_6$$

with the requirement that $\mathrm{tr}([2]\tilde{P}_6) = [4]\tilde{P}_6 = [\beta]\mathrm{tr}([2]Q_6)$ be a point of 2^{m-2}-torsion in $E_6(\mathbb{F}_p)$. In other words, pick any point $Q'_6 \in E_6(\mathbb{F}_p)$ of order 2^{m-2} above $(0,0)$ such that $Q'_6 \in \langle \mathrm{tr}([2]Q_6) \rangle$, then simply choose $\tilde{P}_6$ so that $[4]\tilde{P}_6$ coincides with Q'_6. Now write

$$Q'_6 = [\beta]\mathrm{tr}([2]Q_6)$$

[3] The actual SIKE setting matches this convention.

and retrieve the discrete logarithm $\beta \pmod{2^{m-2}}$ to the same base as for α, again once and for all. Now observe that $[4]\tilde{P}_6 - [4\beta]Q_6 = Q'_6 - [4\beta]Q_6 = [\gamma]([4]\tilde{Q}_6)$, and hence

$$\tilde{\psi}(Q'_6 - [4\beta]Q_6) = [\gamma]\tilde{\psi}([4]\tilde{Q}_6)$$

which constitutes a discrete logarithm instance in $\langle\tilde{\psi}([4]\tilde{Q}_6)\rangle \subset E_{-6}(\mathbb{F}_p)$. Solving it reveals $\gamma \pmod{2^{m-2}}$, once and for all as usual.

The last equation also shows that, although β is only determined mod 2^{m-2}, only the value 4β is required to satisfy the constraints above, so that same value can be lifted and taken as representative mod 2^m. The same observation holds for γ. This yields our choice $\tilde{P}_6$ and completes the basis $(\tilde{P}_6, \tilde{Q}_6)$ we seek for $E_6[2^m]$. This basis selection process is summarized in Algorithm 5.1.

Algorithm 5.1 Selecting a basis $(\tilde{P}_6, \tilde{Q}_6)$

INPUT: path: traversal path (output of Algorithm A.1);
(P_6, Q_6): arbitrary basis for $E_6[2^m]$.
OUTPUT: basis $(\tilde{P}_6, \tilde{Q}_6)$ for $E_6[2^m]$ such that $[2]\tilde{P}_6 \in E_6(\mathbb{F}_p)$, $[2^{m-1}]\tilde{P}_6 = (0,0)$, and $[2]\tilde{Q}_6 = \tilde{\psi}(H)$ for some $H \in E_{-6}(\mathbb{F}_p)$ of order 2^{m-1}.

prepare discrete logarithm table T for $\langle \mathrm{tr}([2]Q_6)\rangle$
$\alpha \leftarrow \mathrm{Dlog}(\mathrm{tr}([2]P_6), m-2, \mathsf{path}, \mathsf{T}, 0, w)$ ▷ via Algorithm 4.4
$\tilde{Q}_6 \leftarrow P_6 - [\alpha]Q_6$
prepare discrete logarithm table $\tilde{\mathsf{T}}$ for $\langle\tilde{\psi}([4]\tilde{Q}_6)\rangle$
pick $Q'_6 \in E_6(\mathbb{F}_p)$ of order 2^{m-2} above $(0,0)$ ▷ hence $Q'_6 \in \langle \mathrm{tr}([2]Q_6)\rangle$
$\beta \leftarrow \mathrm{Dlog}(\mathrm{tr}(Q'_6), m-2, \mathsf{path}, \mathsf{T}, 0, w)$ ▷ via Algorithm 4.4
$\gamma \leftarrow \mathrm{Dlog}(\tilde{\psi}([2]Q'_6 - [4\beta]Q_6), m-2, \mathsf{path}, \tilde{\mathsf{T}}, 0, w)$ ▷ via Algorithm 4.4
$\tilde{P}_6 \leftarrow [\beta]Q_6 + [\gamma]\tilde{Q}_6$
return $(\tilde{P}_6, \tilde{Q}_6)$

Computing logarithms with basis $(\tilde{P}_6, \tilde{Q}_6)$:

Having found that special fixed basis, we reverse-decompose (R_6, S_6) as:

$$\begin{cases} R_6 = [a']\tilde{P}_6 + [b']\tilde{Q}_6 \\ S_6 = [c']\tilde{P}_6 + [d']\tilde{Q}_6 \end{cases}$$

Doubling and taking the trace from both sides yields:

$$\begin{cases} R_6 = [a']\tilde{P}_6 + [b']\tilde{Q}_6 \\ S_6 = [c']\tilde{P}_6 + [d']\tilde{Q}_6 \end{cases} \xRightarrow{\mathrm{tr}\circ[2]} \begin{cases} \mathrm{tr}([2]R_6) = [a']\mathrm{tr}([2]\tilde{P}_6) \\ \mathrm{tr}([2]S_6) = [c']\mathrm{tr}([2]\tilde{P}_6) \end{cases}$$

whereby $a', c' \pmod{2^{m-2}}$ are retrieved, and once they are known:

$$\begin{cases} R_6 = [a']\tilde{P}_6 + [b']\tilde{Q}_6 \\ S_6 = [c']\tilde{P}_6 + [d']\tilde{Q}_6 \end{cases} \xRightarrow{\tilde{\psi}\circ[4]} \begin{cases} \tilde{\psi}([4]R_6 - [4a']\tilde{P}_6) = [b']\tilde{\psi}([4]\tilde{Q}_6) \\ \tilde{\psi}([4]S_6 - [4c']\tilde{P}_6) = [d']\tilde{\psi}([4]\tilde{Q}_6) \end{cases}$$

whereby $b', d' \pmod{2^{m-2}}$ are similarly retrieved.

The computation of a' in $\langle \mathrm{tr}([2]\tilde{P}_6)\rangle$ by means of Algorithm 4.4 with $\sigma = 2$ yields, as a by-product, the point $P_a := [\lfloor a'/4 \rfloor]\mathrm{tr}([2]\tilde{P}_6) = [\lfloor a'/4 \rfloor]([4]\tilde{P}_6)$ which is almost the point $[4a']\tilde{P}_6$ needed for the computation of b', but not quite due to the loss of precision in $\lfloor a'/4 \rfloor$. To compensate for it, simply set $P_a \leftarrow P_a + [a' \bmod 4]\tilde{P}_6$, thereby ensuring that $[4]P_a = [4a']\tilde{P}_6$ at the cost of a single addition in $E_6(\mathbb{F}_{p^2})$ if the points $[u]\tilde{P}_6$ for $0 \leqslant u < 4$ are precomputed and stored in a small table. The same observations hold for the computation of d' from the by-product of c'.

So far we only recovered $a', b', c', d' \pmod{2^{m-2}}$, while we need these values mod 2^m instead. The complete values only differ by the retrieved ones by some amount in $D := \{k \cdot 2^{m-2} \mid 0 \leqslant k < 4\}$, that is, $R_6 - [a']\tilde{P}_6 - [b']\tilde{Q}_6 \in \{[u]\tilde{P}_6 + [v]\tilde{Q}_6 \mid u, v \in D\}$, and similarly for $S_6 - [c']\tilde{P}_6 - [d']\tilde{Q}_6$. Notice that the multiples of $\tilde{P}_6$ and $\tilde{Q}_6$ are essentially the by-products P_a, P_b, P_c, P_d of the computations of a', b', c', d' (points P_b and P_d must of course be mapped from E_{-6} back to E_6 via the $\tilde{\psi}^{-1} = -\tilde{\psi}$ map).

Thus, the missing terms u and v can be recovered from a lookup table L_6, incurring four point subtractions overall plus the search cost similar to that discussed in Section 4.2. Here, however, the set of points being searched always contains 16 points, so there are 9 distinct values of the y-coordinate in the twisted Edwards model, and the search for y incurs between 1 and 8 $\mathbb{F}_{p^2}$ multiplications, plus one more to determine the correct x for the retrieved y. In other words, the average search cost is 5.5 $\mathbb{F}_{p^2}$ multiplications, or about 16.5 $\mathbb{F}_p$ multiplications, and it never exceeds 27 $\mathbb{F}_p$ multiplications. Also, table L_6 only consists of 9 points, since the other ones are just the opposite of these.

This completely recovers the a', b', c', d' scalar factors, and a plain change of basis from $(\tilde{P}_6, \tilde{Q}_6)$ to (P_6, Q_6) yields $\hat{a}, \hat{b}, \hat{c}, \hat{d}$ as desired. This whole method is summarized in Algorithm 5.2.

Algorithm 5.2 Dlog6$(R_6, \tilde{P}_6, \tilde{Q}_6, \mathsf{T}_P, \mathsf{T}_Q, \mathsf{path}, \mathsf{L}_6, w)$

PURPOSE: retrieve $a', b' \pmod{2^m}$ such that $R_6 = [a']\tilde{P}_6 + [b']\tilde{Q}_6$.
INPUT: $R_6 \in E_6(\mathbb{F}_{p^2})$: point to express in basis $(\tilde{P}_6, \tilde{Q}_6)$;
$(\tilde{P}_6, \tilde{Q}_6)$: special basis for $E_6(\mathbb{F}_{p^2})[2^m]$ (output of Algorithm 5.1);
T_P, T_Q: lookup tables for $\mathrm{tr}([2]\tilde{P}_6)$ and $\tilde{\psi}([4]\tilde{Q}_6)$, respectively;
path: traversal path (output of Algorithm A.1);
L_6: lookup table for pairs $(u := j \cdot 2^{m-2}, v := k \cdot 2^{m-2})$ for $j, k \in \{0 \ldots 3\}$ with search key $[u]\tilde{P}_6 + [v]\tilde{Q}_6$ (see Section 4.2).
w : the window size.
OUTPUT: $a', b' \pmod{2^m}$ such that $R_6 = [a']\tilde{P}_6 + [b']\tilde{Q}_6$.

1: $a', P_a \leftarrow \mathrm{Dlog}(\mathrm{tr}([2]R_6), m-2, \mathsf{path}, \mathsf{T}_P, 2, w)$ ▷ via Algorithm 4.4
2: $P'_a \leftarrow P_a + [a' \bmod 4]\tilde{P}_6$
3: $b', P_b \leftarrow \mathrm{Dlog}(\tilde{\psi}([4](R_6 - P_a)), m-2, \mathsf{path}, \mathsf{T}_Q, 2, w)$ ▷ via Algorithm 4.4
4: $P'_b \leftarrow -\tilde{\psi}(P_b) + [b' \bmod 4]\tilde{Q}_6$
5: lookup $\delta_6 := R_6 - P'_a - P'_b$ in L_6 to retrieve (u, v) such that $[u]\tilde{P}_6 + [v]\tilde{Q}_6 = \delta_6$.
6: **return** $a' + u \pmod{2^m}$, $b' + v \pmod{2^m}$

This method requires computing logarithms in $\langle \mathrm{tr}([2]\tilde{P}_6)\rangle \subset E_6(\mathbb{F}_p)$ and $\tilde{\psi}([4]\tilde{Q}_6) \subset E_{-6}(\mathbb{F}_p)$. Therefore, two distinct tables would be required for Algorithm 4.4. We show next how to avoid this, thereby restricting all logarithm computations to a single subgroup over $\mathbb{F}_p$ with a somewhat more sophisticated method.

5.3 Handling $\ell = 2$, second solution

Our second solution involves mapping the discrete logarithm instances to curve E_0, where an efficient distortion map is available and enables using a single table for a subgroup of $E_0(\mathbb{F}_p)$ for all of those instances. However, a few obstacles must be overcome.

Namely, in this setting Algorithm 4.4 can only determine logarithms mod 2^{m-4} or mod 2^{m-3}, since the composition of the 2-isogeny between E_0 and E_6 with its dual introduces an extra factor 2 where one has already been placed by construction or indirectly by the trace map, thereby mapping to a point of incomplete order. This is slightly worse than our first solution, but more importantly, completing the values must still be done in E_6 where a', b', c', d' are properly defined, and the strategy graph traversal will no longer yield scalar multiples of points in E_6. Besides, if a change of basis is required to test the partial logarithms, it is likely to incur further point multiplications by scalars, introducing even more computational overhead.

We address these issues, coping with the incompleteness of computed discrete logarithms while avoiding multiplications by large scalars, by carefully picking a basis (P_0, Q_0) for $E_0[2^m]$ and a matching basis (P_6^*, Q_6^*) for $E_6[2^m]$ satisfying several constraints:

- P_0 must be a point of full order above $(i, 0)$, the generator of the kernel of the 2-isogeny φ_0, that is, $[2^{m-1}]P_0 = (i, 0)$, so that $\varphi_0(P_0)$ is a point of order exactly 2^{m-1} (NB: P_0 cannot be taken from $E_0(\mathbb{F}_p)$ since the only point of order 2 in $E_0(\mathbb{F}_p)$ is $(0, 0)$);
- $Q_0 := [2]P_0 - \mathrm{tr}(P_0)$, which is a point of trace zero, must satisfy $Q_0 = -\psi(\mathrm{tr}(P_0))$ which lies above $(0, 0)$ (NB: this specific choice is what enables using a single table to compute logarithms);
- P_6^* must be such that $[2]P_6^* = \varphi_0(P_0)$, so that $\varphi_0(\langle P_0\rangle) \subset \langle P_6^*\rangle$ (NB: it is not possible for P_6^* to be defined as $\varphi_0(P_0)$ itself because P_0 is chosen above a point in the kernel of the isogeny, and hence its image will not be a point of full order);
- $Q_6^* := \varphi_0(Q_0)$ (NB: since Q_0 is not above the generator of the kernel of φ_0, Q_6^* is a point of full order 2^m).

While points Q_0 and Q_6^* are trivially determined, finding suitable points P_0 and P_6^* is not trivial. We now show how to find these. Notice that the following computations are only required once, that is, all of these points are fixed and can be precomputed.

Finding P_0: Pick any point $P_2 \in E_0[2^m]$ of full order above $(i, 0)$. Then $Q_2 := \psi(\mathrm{tr}(P_2)) \in E_0[2^m]$ is a linearly independent point of full order and trace zero. We can express $\mathrm{tr}(P_2)$ itself in basis (P_2, Q_2) as $\mathrm{tr}(P_2) = [u]P_2 + [v]Q_2$ for some $u, v \in \mathbb{Z}/2^m\mathbb{Z}$. Taking the trace from both sides of this equation yields $[2]\mathrm{tr}(P_2) = [u]\mathrm{tr}(P_2)$, from which we retrieve $u = 2 \pmod{2^m}$. Besides, applying the distortion map to that same equation and regrouping terms yields $\psi([2]P_2 - \mathrm{tr}(P_2)) = [v]\mathrm{tr}(P_2)$, from which we retrieve the discrete logarithm $v \pmod{2^m}$. Notice that v must be odd, since $\psi([2]P_2 - \mathrm{tr}(P_2))$ is a point of full order.

We now seek a point P_0 such that $[2]P_0 - \mathrm{tr}(P_0) = -\psi(\mathrm{tr}(P_0))$. Write $P_0 = [\alpha]P_2 + [\beta]Q_2$ for some α, β, and notice that $\mathrm{tr}(P_0) = [\alpha]\mathrm{tr}(P_2) = [2\alpha]P_2 + [v\alpha]Q_2$. Then $[2]P_0 - \mathrm{tr}(P_0) = [2\alpha]P_2 + [2\beta]Q_2 - [2\alpha]P_2 - [v\alpha]Q_2 = [2\beta - v\alpha]Q_2$, while $-\psi(\mathrm{tr}(P_0)) = -\psi([\alpha]\mathrm{tr}(P_2)) = -[\alpha]Q_2$, and hence $[2]P_0 - \mathrm{tr}(P_0) = -\psi(\mathrm{tr}(P_0)) \Rightarrow [2\beta - v\alpha]Q_2 = -[\alpha]Q_2 \Rightarrow [2\beta - v\alpha + \alpha]Q_2 = \mathcal{O}$, that is, $2\beta - v\alpha + \alpha = 0 \pmod{2^m}$ or simply $\beta = \alpha(v-1)/2 \pmod{2^m}$, where the division $(v-1)/2$ is exact because v is odd. Since any point P_0 satisfying this constraint is equally suitable, we simply take $\alpha = 1$ and $\beta = (v-1)/2$, that is, we choose $P_0 := P_2 + [(v-1)/2]Q_2$. This process is summarized in Algorithm 5.3.

Algorithm 5.3 Selecting a basis (P_0, Q_0)

INPUT: path: traversal path (output of Algorithm A.1).
OUTPUT: basis (P_0, Q_0) for $E_0[2^m]$ such that $P_0 \in E_0[2^m]$ is above $(i, 0)$ and $Q_0 := [2]P_0 - \mathrm{tr}(P_0) = -\psi(\mathrm{tr}(P_0))$.
REMARK: This algorithm is only used once, for precomputation.

1: pick $P_2 \in E_0(\mathbb{F}_{p^2})$ of order 2^m above $(i, 0)$
2: prepare discrete logarithm table T_2 for $\langle \mathrm{tr}(P_2) \rangle$
3: $v \leftarrow \mathrm{Dlog}(\psi([2]P_2 - \mathrm{tr}(P_2)), \mathsf{T}_2, m, \mathsf{path}, 0, w)$ ▷ via Algorithm 4.4
4: $P_0 \leftarrow P_2 + [\lfloor (v-1)/2 \rfloor]\psi(\mathrm{tr}(P_2))$
5: $Q_0 \leftarrow -\psi(\mathrm{tr}(P_0))$
6: **return** (P_0, Q_0)

Finding P_6^*: Find a basis $(\tilde{P}_6, \tilde{Q}_6)$ as prescribed in Section 5.2 (Algorithm 5.1). Since all that is required from P_6^* is that $[2]P_6^* = \varphi_0(P_0)$, we can write $\varphi_0(P_0) = [u']\tilde{P}_6 + [v']\tilde{Q}_6$, solve the discrete logarithms for u' and v' (which must be both even since $\varphi_0(P_0)$ is a point of order 2^{m-1}), and then choose $P_6^* := [u'/2]\tilde{P}_6 + [v'/2]\tilde{Q}_6$. This process is summarized in Algorithm 5.4.

Computing logarithms with bases (P_0, Q_0) and (P_6^*, Q_6^*):

Algorithm 5.5 computes the decomposition of a point on $E_0[2^m]$ in the special basis (P_0, Q_0) precomputed by Algorithm 5.3, and also points $P_a^* = [\lfloor a^*/4 \rfloor]P_0$, $P_b^* = [\lfloor b^*/4 \rfloor]\mathrm{tr}(P_0)$.

Analogously, algorithm 5.6 computes the decomposition of a point on $E_6[2^m]$ in the special basis (P_6^*, Q_6^*) precomputed by Algorithm 5.4. It does so by mapping the given discrete logarithm instance $R_6 \in E_6$ to a related instance $R_0 \in E_0$

Algorithm 5.4 Selecting a basis (P_6^*, Q_6^*)

INPUT: (P_0, Q_0): special basis for $E_0[2^m]$ (output of Algorithm 5.3);
$(\tilde{P}_6, \tilde{Q}_6)$: special basis for $E_6(\mathbb{F}_{p^2})[2^m]$ (output of Algorithm 5.1);
T_P, T_Q: lookup tables for $\mathrm{tr}([2]\tilde{P}_6)$ and $\tilde{\psi}([4]\tilde{Q}_6)$, respectively;
path: traversal path (output of Algorithm A.1);
L_6: lookup table for pairs $(u := j \cdot 2^{m-2}, v := k \cdot 2^{m-2})$ for $j, k \in \{0 \dots 3\}$ with search key $[u]\tilde{P}_6 + [v]\tilde{Q}_6$.
OUTPUT: basis (P_6^*, Q_6^*) for $E_6[2^m]$ such that $[2]P_6^* = \varphi_0(P_0)$ and $Q_6^* = \varphi(Q_0)$.
REMARK: This algorithm is only used once, for precomputation.

1: $T_6 \leftarrow \varphi_0(P_0)$
2: $u, v \leftarrow \mathrm{Dlog6}(T_6, \tilde{P}_6, \tilde{Q}_6, \mathsf{T}_P, \mathsf{T}_Q, \mathsf{path}, \mathsf{L}_6, w)$ ▷ via Algorithm 5.2
3: $P_6^* \leftarrow [\lfloor u/2 \rfloor]\tilde{P}_6 + [\lfloor v/2 \rfloor]\tilde{Q}_6$
4: $Q_6^* \leftarrow \varphi(Q_0)$
5: **return** (P_6^*, Q_6^*)

Algorithm 5.5 $\mathrm{Dlog0}(R_0, P_0, \mathsf{T}_0, \mathsf{path}, w)$

PURPOSE: retrieve $a^*, b^* \pmod{2^m}$ such that $R_0 = [a^*]P_0 + [b^*]Q_0$.
INPUT: $R_0 \in E_0(\mathbb{F}_{p^2})$: point to express in basis (P_0, Q_0);
$P_0 \in E_0[2^m]$: point of full order above $(i, 0)$ with $[2]P_0 - \mathrm{tr}(P_0) = -\psi(\mathrm{tr}(P_0))$ (output of Algorithm 5.3);
T_0: discrete logarithm lookup table for $\langle \mathrm{tr}(P_0) \rangle$;
path: traversal path (output of Algorithm A.1);
w : the window size.
OUTPUT: $a^*, b^* \pmod{2^m}$ such that $R_0 = [a^*]P_0 + [b^*]Q_0$ where $Q_0 = -\psi(\mathrm{tr}(P_0))$.

1: $a^*, P_a \leftarrow \mathrm{Dlog}(\mathrm{tr}(R_0), \mathsf{T}_0, m, \mathsf{path}, 3, w)$ ▷ via Algorithm 4.4
2: $P'_a \leftarrow [4]P_a + [(\lfloor a^*/2 \rfloor) \bmod 4]\mathsf{T}_0[0][1]$
3: $P'_a \leftarrow P'_a - \psi(P'_a)$
4: $P_a^* \leftarrow P_a - \psi(P_a) + [(\lfloor a^*/4 \rfloor) \bmod 2]P_0$
5: $b^*, P_b^* \leftarrow \mathrm{Dlog}(\psi(R_0 - P'_a), \mathsf{T}_0, m, \mathsf{path}, 2, w)$ ▷ via Algorithm 4.4
6: **return** a^*, b^*, P_a^*, P_b^*

via the dual isogeny $\widehat{\varphi}_0$, and then decomposing this instance in the special basis (P_0, Q_0), which is precomputed by Algorithm 5.3. The careful choice of this basis enables restricting the computations to a single subgroup of $E_0(\mathbb{F}_p)$ of full order. The obtained decomposition is finally mapped back from E_0 to E_6 via the isogeny φ_0. The isogeny evaluations only allow for a partial recovery of the desired logarithms, but the complete solution can now be retrieved at low cost by a lookup in a small table L_6^* of 9 entries, analogous to the L_6 table used in the first solution. The search overhead is the same as for the first solution for $\ell = 2$, namely, about $16.5\,\mathbb{F}_p$ multiplications on average, and no more than $27\,\mathbb{F}_p$ multiplications in any case.

Algorithm 5.6 Dlog2$(R_6, P_0, \mathsf{T}_0, \mathsf{path}, \mathsf{L}_6^*)$

PURPOSE: retrieve $a^*, b^* \pmod{2^m}$ such that $R_6 = [a^*]P_6^* + [b^*]Q_6^*$.
INPUT: $R_6 \in E_6(\mathbb{F}_{p^2})$: point to express in basis (P_6^*, Q_6^*);
$P_0 \in E_0[2^m]$: point of full order above $(i, 0)$ with $[2]P_0 - \mathrm{tr}(P_0) = -\psi(\mathrm{tr}(P_0))$ (output of Algorithm 5.3);
T_0: discrete logarithm lookup table for $\langle \mathrm{tr}(P_0) \rangle$;
path: traversal path (output of Algorithm A.1);
L_6^*: lookup table for pairs $(u := h + j \cdot 2^{m-1}, v := k \cdot 2^{m-2})$ for $h, j \in \{0, 1\}$ and $k \in \{0 \ldots 3\}$ with search key $[u]P_6^* + [v]Q_6^*$.
OUTPUT: $a^*, b^* \pmod{2^m}$ such that $R_6 = [a^*]P_6^* + [b^*]Q_6^*$.

1: $R_0 \leftarrow [2]\widehat{\varphi}_0(R_6)$
2: $a^*, b^*, P_a^*, P_b^* \leftarrow$ Dlog0$(R_0, P_0, \mathsf{T}_0, \mathsf{path}, w)$ ▷ via Algorithm 5.5
3: $a^* \leftarrow \lfloor a^*/4 \rfloor$, $b^* \leftarrow \lfloor b^*/4 \rfloor$
4: lookup $\delta_0 := R_6 - \widehat{\varphi}_0(P_a^*) + \widehat{\varphi}_0(\psi(P_b^*))$ in L_6^* to retrieve (u, v) such that $\delta_0 = [u]P_6^* + [v]Q_6^*$
5: **return** $2a^* + u \pmod{2^m}$, $b^* + v \pmod{2^m}$

6 Experimental results

Table 1 lists the average cost to decompose one point from $E_6(\mathbb{F}_{p^2})$ in basis $(\tilde{P}_6, \tilde{Q}_6)$, when Algorithm 5.2 is set to retrieve discrete logarithm digits in base 2, base 2^3, base 2^4 and base 2^6 (that is, with windows of size $w = 1$, $w = 3$, $w = 4$, and $w = 6$, respectively) for the official SIKE parameters. Fluctuations occur because, since a constant-time implementation is hardly needed for operations involving purely public information, one can omit dummy operations like point additions or multiplications by zero. Results are averaged over 1000 random discrete logarithm instances.

Table 2 lists the corresponding costs for Algorithm 5.6. We see that the greater complexity of this method has a detectable effect on its cost, but it is quite modest compared to Algorithm 5.2: 0.8%–1.3% for $w = 1$, 2.3%–4.0% for $w = 3$, and 4.7%–8.4% for $w = 6$.

Table 3 lists the costs for Algorithm 4.4 applied to $\ell = 3$ for its practical importance. Only values $w = 1$, $w = 3$, and $w = 4$ are listed; larger values would increase the table size without substantially improving processing effi-

Table 1: Average cost of Algorithm 5.2 (in $\mathbb{F}_p$ multiplications) and (two) tables size (in $\#\mathbb{F}_{p^2}$ elements) to compute $a', b' \pmod{2^m}$ such that $R_6 = [a']\tilde{P}_6 + [b']\tilde{Q}_6$ for a random $R_6 \in E_6(\mathbb{F}_{p^2})[2^m]$.

setting	$w=1$		$w=3$		$w=4$		$w=6$	
	cost	size	cost	size	cost	size	cost	size
SIKEp434	21148	428	13005	570	11420	852	10507	2256
SIKEp503	24901	496	15603	660	13792	992	12690	2628
SIKEp610	31464	606	19955	808	17530	1208	16003	3208
SIKEp751	39712	740	25051	986	21962	1476	20112	3920

Table 2: Average cost of Algorithm 5.6 (in $\mathbb{F}_p$ multiplications) and table size (in $\#\mathbb{F}_{p^2}$ elements) to compute $a^*, b^* \pmod{2^m}$ such that $R_6 = [a^*]P_6^* + [b^*]Q^*$ for a random $R_6 \in E_6(\mathbb{F}_{p^2})[2^m]$.

setting	$w=1$		$w=3$		$w=4$		$w=6$	
	cost	size	cost	size	cost	size	cost	size
SIKEp434	21420	216	13528	288	12027	432	11393	1152
SIKEp503	25194	250	16173	334	14435	500	13559	1328
SIKEp610	31781	305	20514	408	18187	610	16861	1632
SIKEp751	40035	372	25639	496	22632	744	21057	1984

ciency (indeed, if w is too large we expect the overhead to exceed the gains anyway). However, in this case the costs are reported to decompose the whole basis (P_6, Q_6) in basis (R_6, Q_6), not just for one point, given the subtle difference between the methods for even and odd ℓ.

Table 3: Average cost of Algorithm 4.4 (in $\mathbb{F}_p$ multiplications) to compute $\hat{a}, \hat{b}, \hat{c}, \hat{d} \pmod{3^n}$ such that $P_6 = [\hat{a}]R_6 + [\hat{b}]S_6$ and $Q_6 = [\hat{c}]R_6 + [\hat{d}]S_6$ for $P_6, Q_6 \in E_6(\mathbb{F}_{p^2})[3^n]$.

setting	$w=1$		$w=3$		$w=4$	
	cost	size	cost	size	cost	size
SIKEp434	37073	137	21504	594	20702	1363
SIKEp503	43949	159	26333	689	24418	1587
SIKEp610	55003	192	33193	832	30489	1920
SIKEp751	71936	239	44191	1036	39888	2387

Direct comparisons with methods like [11] are hard, since we count basic operations in the underlying field $\mathbb{F}_p$ and developed our implementation in Magma, while that work only lists clock cycles for a C/assembly implementation.

Yet, one can make reasonably precise estimates of the joint cost of computing pairings and discrete logarithms with those techniques. When estimating the multiplications incurred *without* precomputed pairing tables, we assume the costs of the pairing algorithms from [18] which appear to be the most efficient currently available.

Results are summarized on Table 4. In all cases we list the results adopting $w = 3$ for the 2^m-torsion discrete logarithms, and $w = 4$ for the 3^n-torsion, to match the implementation in [11] and ensure the discrete logarithm tables take the same space[4].

Table 4: Average $\mathbb{F}_p$ multiplication counts of joint pairing and discrete logarithm computation from [11] and our pairingless method, for SIKEp751 ($m = 372$, $n = 239$).

torsion	[11] w/ precomp	[11] no precomp	ours
2^m	33052	56253	45264
3^n	33144	65180	44191

Finally, we compare the storage requirements of our proposals, as measured equivalently either in $E(\mathbb{F}_p)$ points or in $\mathbb{F}_{p^2}$ elements, with prior techniques.

In general, Algorithm 4.1 and its variants (Algorithm 4.2 and 4.3) require tables of sizes given by Equation 2. Thus, for instance, in the case of SIKEp751 this means $8 \cdot \lceil 372/4 \rceil = 744$ elements (points over $E(\mathbb{F}_p)$) for the 2^m-torsion and $w = 4$, and $13 \cdot (239 - 2)/3 + 3^2 = 1036$ elements for the 3^n-torsion with $w = 3$.

By contrast, both [11] and [18], which do not use the techniques we describe in Sections 4.3 and 4.4, need up to *four* times as much space: $2^w \lceil m/w \rceil$ (resp. $3^w \lceil n/w \rceil$) elements if $w \mid m$ (resp. $w \mid n$), and twice as much for two separate sets of tables for each torsion if $w \nmid m$ (resp. $w \nmid n$). Thus, at the time of writing the official SIKEp751 implementation, which builds upon those two research works, takes $16 \cdot 372/4 = 1488$ elements for the 2^m-torsion and $w = 4$ (so $w \mid m$), and $2 \cdot 27 \cdot \lceil 239/3 \rceil = 4320$ elements for the 3^n-torsion with $w = 3$ (so $w \nmid m$).

Besides, in general the T_{P_2} and T_{Q_2} precomputed pairing tables as defined in [11, Section 5.3] require $6(m - 1)$ $\mathbb{F}_p$ elements altogether, while the T_P table as defined in [11, Section 5.4] requires space equivalent to that of $3(n - 1) + 2$ $\mathbb{F}_{p^2}$ elements (albeit in the form of individual $\mathbb{F}_p$ elements). For instance, for SIKEp751 this means $(372-1)\cdot 3 = 1113$ $\mathbb{F}_{p^2}$ elements for T_{P_2} and T_{Q_2} together, and $(239 - 1) \cdot 6 + 4 = 1432$ $\mathbb{F}_p$ elements for table T_P, equivalent to 716 $\mathbb{F}_{p^2}$ elements. Our techniques require none of that. This is summarized on Table 5. The storage requirements of our technique are less than 29% of the state of the art for the 2^m-torsion, and about 21% for the 3^n-torsion.

Table 5: Storage requirements, measured in $E(\mathbb{F}_p)$ points or equivalently in $\mathbb{F}_{p^2}$ elements for SIKEp751 ($m = 372$, $n = 239$).

torsion	[11] and [18] dlogT ($\mathbb{F}_{p^2}$) + pairingT ($\mathbb{F}_{p^2}$)	ours dlogT ($E(\mathbb{F}_p)$)	ratio (%)
2^m	$1488 + 1113 = 2601$	744	28.6
3^n	$4320 + 716 = 5036$	1036	20.6

[4] For the binary torsion, our methods require an extra table, L_6 or L_6^*, containing just 9 points over $\mathbb{F}_{p^2}$, a small fraction of the space required for the other tables.

7 Discussion and conclusion

Apart from initialization, both the method for odd ℓ in Section 5.1 and the second method for $\ell = 2$ in Section 5.3 require each a single table for all calls to Algorithm 4.4, that is, they are carried out in the same torsion group over $\mathbb{F}_p$.

As a consequence, those constructions require essentially the *same* table space as needed for discrete logarithms in $\mathbb{F}_{p^2}^*$, but *no* tables as required to speedup the computation of pairings as suggested in [11], since we avoid pairings altogether. Trade-offs between table size and processing speed are possible, particularly the windowed approach discussed in [18, Section 6.1].

We remark that solving two instances of the discrete logarithm in a subgroup of $E_0(\mathbb{F}_p)$ is computationally less expensive than solving a single instance in a subgroup of $E_0(\mathbb{F}_{p^2})$, given that the relative cost of the arithmetic over those fields is essentially the only difference between the two scenarios. This shows that adapting Teske's algorithm [16] to a strategy graph-based approach, thereby retrieving both u and v at once while keeping the number of group operations potentially the same as that of Algorithm 4.4, would incur not only an already larger computational cost due to the contrast between $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ arithmetic, but the precomputed tables themselves would have to be *quadratically* larger to cope with computing pairs of digits at once.

In this context, Sutherland's algorithm [15] extends Teske's approach and promises to be asymptotically faster, but it is far more involved in the way it retrieves the digits of a discrete logarithm. It is unclear how that method could avoid the larger cost of $\mathbb{F}_{p^2}$ arithmetic, nor how large the underlying group would have to be for the asymptotic speedup to overcome the corresponding overhead, nor even whether precomputed tables could be made any smaller than what Teske's method would require. For these reasons, neither of these two approaches seems practical for the task at hand.

We have thus described methods to compute discrete logarithms in the elliptic curve torsion groups of the starting curves in SIDH-style cryptosystems, as required to compress the corresponding public keys. Our methods do not rely on bilinear pairings, yet their efficiency is comparable to the best available pairing-based methods while vastly reducing the storage space needed for pairing computations. The table storage needed for discrete logarithm computation is essentially the same required for discrete logarithms in the $\mathbb{F}_{p^2}$ finite field over which the curves are defined, the excess space being constant and very small, limited to just a few extra points.

Acknowledgements This work is supported in part by NSERC, CryptoWorks21, Canada First Research Excellence Fund, Public Works and Government Services Canada.

References

1. R. Azarderakhsh, M. Campagna, C. Costello, L. DeFeo, B. Hess, A. Jalali, D. Jao, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, G. Pereira, J. Renes, V. Soukharev,

and D. Urbanik. *Supersingular Isogeny Key Encapsulation*. SIKE Team, https://sike.org/, 2020.
2. 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.
3. D. J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards curves. In *Progress in Cryptology – Africacrypt 2008*, number 5023 in Lecture Notes in Computer Science, pages 389–405, Casablanca, Morocco, 2008. Springer.
4. C. Chuengsatiansup. *Optimizing curve-based cryptography*. PhD thesis, Technische Universiteit Eindhoven, March 2017.
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. C. Costello, P. Longa, and M. Naehrig. Efficient algorithms for supersingular isogeny Diffie-Hellman. In *Advances in Cryptology – Crypto 2016*, number 9814 in Lecture Notes in Computer Science, pages 572–601, Santa Barbara (CA), USA, 2016. Springer.
7. 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.
8. S. D. Galbraith and V. Rotger. Easy decision-Diffie-Hellman groups. *LMS Journal of Computation and Mathematics*, 7:201–218, 2004.
9. A. Hutchinson, K. Karabina, and G. Pereira. Memory Optimization Techniques for Computing Discrete Logarithms in Compressed SIKE. Cryptology ePrint Archive, Report 2021/368, 2020. http://eprint.iacr.org/2021/368.
10. A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, USA, 1999.
11. M. Naehrig and J. Renes. Dual isogenies and their application to public-key compression for isogeny-based cryptography. In S. D. Galbraith and S. Moriai, editors, *Advances in Cryptology – Asiacrypt 2019*, volume 11922 of *Lecture Notes in Computer Science*, pages 243–272. Springer, 2019.
12. G. Pereira, J. Doliskani, and D. Jao. x-only point addition formula and faster compressed SIKE. *Journal of Cryptographic Engineering*, pages 1–13, 2020.
13. S. C. Pohlig and M. E. Hellman. An improved algorithm for computing logarithms over $GF(p)$ and its cryptographic significance. *IEEE Transactions on Information Theory*, 24(1):106–110, 1978.
14. V. Shoup. *A computational introduction to number theory and algebra*. Cambridge University Press, 2005.
15. A. V. Sutherland. Structure computation and discrete logarithms in finite Abelian p-groups. *Mathematics of Computation*, 80:477–500, 2011.
16. E. Teske. The Pohlig-Hellman method generalized for group structure computation. *Journal of Symbolic Computation*, 27(6):521–534, 1999.
17. G. H. M. Zanon, M. A. Simplicio Jr, G. C. C. F. Pereira, J. Doliskani, and P. S. L. M. Barreto. Faster isogeny-based compressed key agreement. In *International Conference on Post-Quantum Cryptography – PQCrypto 2018*, number 10786 in Lecture Notes in Computer Science, pages 248–268, Fort Lauderdale, Florida, US, 2018. Springer.
18. G. H. M. Zanon, M. A. Simplicio Jr, G. C. C. F. Pereira, J. Doliskani, and P. S. L. M. Barreto. Faster key compression for isogeny-based cryptosystems. *IEEE Transactions on Computers*, 68(5):688–701, 2018.

A The OptPath algorithm

Algorithm A.1 OptPath(p, q, e): optimal subtree traversal path

INPUT: p, q: left and right edge traversal cost; e: number of leaves of Δ.
OUTPUT: $\mathsf{path}[1 \ldots e]$: array of indices specifying an optimal traversal path.

```
▷ Define C[1...e] as an array of costs.
C[1] ← 0,   path[1] ← 0
for k ← 2 to e do
    j ← 1,   z ← k − 1
    while j < z do
        m ← j + ⌊(z − j)/2⌋,   w ← m + 1
        t1 ← C[m] + C[k − m] + (k − m) · p + m · q
        t2 ← C[w] + C[k − w] + (k − w) · p + w · q
        if t1 ≤ t2 then
            z ← m
        else
            j ← w
        end if
    end while
    C[k] ← C[j] + C[k − j] + (k − j) · p + j · q,   path[k] ← j
end for
return path
```