Dual Isogenies and Their Application to Public-Key Compression for Isogeny-Based Cryptography

Michael Naehrig[1(✉)] and Joost Renes[2]

[1] Microsoft Research, Redmond, WA, USA
mnaehrig@microsoft.com
[2] Digital Security Group, Radboud University, Nijmegen, The Netherlands
j.renes@cs.ru.nl

Abstract. The isogeny-based protocols SIDH and SIKE have received much attention for being post-quantum key agreement candidates that retain relatively small keys. A recent line of work has proposed and further improved compression of public keys, leading to the inclusion of public-key compression in the SIKE proposal for Round 2 of the NIST Post-Quantum Cryptography Standardization effort. We show how to employ the *dual* isogeny to significantly increase performance of compression techniques, reducing their overhead from 160–182% to 77–86% for Alice's key generation and from 98–104% to 59–61% for Bob's across different SIDH parameter sets. For SIKE, we reduce the overhead of (1) key generation from 140–153% to 61–74%, (2) key encapsulation from 67–90% to 38–57%, and (3) decapsulation from 59–65% to 34–39%. This is mostly achieved by speeding up the pairing computations, which has until now been the main bottleneck, but we also improve (deterministic) basis generation.

Keywords: Post-Quantum Cryptography · Public-key compression · Supersingular elliptic curves · Dual isogenies · Reduced Tate pairings

1 Introduction

Isogeny-based protocols are an alternative to the more mainstream proposals for post-quantum key agreement, such as lattice-based or code-based schemes. Beyond their reliance on a different type of hard computational problem, the main distinguishing characteristics of isogeny-based key exchange and key encapsulation schemes are their small keys and low communication costs. This is for example seen in the supersingular-isogeny Diffie–Hellman (SIDH) scheme first proposed by Jao and De Feo [12,19], its IND-CCA secure key encapsulation variant SIKE [17] and the more recent CSIDH proposal of Castryck, Lange, Martindale, Panny and Renes [5].

J. Renes—Partially supported by the Technology Foundation STW (project 13499 – TYPHOON & ASPASIA), from the Dutch government.

S. D. Galbraith and S. Moriai (Eds.): ASIACRYPT 2019, LNCS 11922, pp. 243–272, 2019.
https://doi.org/10.1007/978-3-030-34621-8_9

The supersingular isogeny key encapsulation scheme SIKE—one of the 17 key encapsulation mechanisms that advanced to the second round of the NIST standardization process for post-quantum cryptography [24]—has the advantage of small public keys and ciphertexts, with sizes in the low hundreds of bytes. For example, the Round 1 submission of SIKE [18] supports public keys of 378 bytes and ciphertexts of 402 bytes at NIST security level 1. In comparison, at the same security level the lattice-based scheme Saber [11] uses 672 byte public keys and 736 byte ciphertexts, while Kyber [4] uses 800 byte public keys and 736 byte ciphertexts. However, such compact keys and ciphertexts in SIKE are contrasted by comparatively high latencies. The recent CSIDH protocol, a non-interactive key exchange scheme and a potential candidate as a drop-in replacement for the standard Diffie-Hellman key exchange, exhibits these characteristics in an even more extreme fashion. It supports even smaller keys, with significantly larger runtimes.

In a similar fashion, techniques for public-key compression for the SIDH protocol also amplify these characteristics. They allow to reduce the communication bandwidth further, but come at the cost of a large computational overhead compared to the uncompressed variant of SIDH. The same techniques apply to SIKE and reduce its public key and ciphertext sizes. Compression has been included in the Round 2 submission of SIKE [17] and, along with the introduction of new parameter sets, has enabled public keys of merely 196 bytes and ciphertexts of only 209 bytes for NIST level 1. We propose several techniques to reduce the large computational overhead, making the use of public-key compression significantly more interesting for practitioners.

Public-Key Compression for SIDH. Let ℓ, m be distinct primes and e_ℓ, e_m be strictly positive integers such that $p = \ell^{e_\ell} \cdot m^{e_m} - 1$ is prime. Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$, and let $\phi_\ell : E \to E/\langle R \rangle$ be an isogeny of degree ℓ^{e_ℓ} such that $\ker \phi_\ell = \langle R \rangle$ for some point $R \in E[\ell^{e_\ell}]$. Similarly, let $\phi_m : E \to E/\langle S \rangle$ be an isogeny of degree m^{e_m} such that $\ker \phi_m = \langle S \rangle$ for some point $S \in E[m^{e_m}]$. Fix parameters P_ℓ, Q_ℓ such that $\langle P_\ell, Q_\ell \rangle = E[\ell^{e_\ell}]$ and P_m, Q_m such that $\langle P_m, Q_m \rangle = E[m^{e_m}]$. The public keys for SIDH are encoded by the triples of x-coordinates of the form

$$\left[x_{\phi_\ell(P_m)}, x_{\phi_\ell(Q_m)}, x_{\phi_\ell(P_m - Q_m)}\right] , \left[x_{\phi_m(P_\ell)}, x_{\phi_m(Q_\ell)}, x_{\phi_m(P_\ell - Q_\ell)}\right] ,$$

which can be represented with $6 \log_2 p$ bits each.

It is possible to further reduce the size of the public key to $4 \log_2 p$, as first proposed by Azarderakhsh, Jao, Kalach, Koziel, and Leonardi [1], by deterministically generating points U_m, V_m such that $\langle U_m, V_m \rangle = (E/\langle R \rangle)[m^{e_m}]$ and computing $a_0, b_0, a_1, b_1 \in \mathbb{Z}/m^{e_m}\mathbb{Z}$ such that

$$\begin{bmatrix} \phi_\ell(P_m) \\ \phi_\ell(Q_m) \end{bmatrix} = \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} \begin{bmatrix} U_m \\ V_m \end{bmatrix} .$$

Although initially believed to be orders of magnitude more expensive than the original isogeny computation, the work of Costello, Jao, Longa, Naehrig,

Renes, and Urbanik [7] significantly reduced the cost to a factor 2.4 slowdown, while simultaneously compressing the public keys to $\frac{7}{2}\log_2 p$ bits. Further computational improvements have since been made by Zanon, Simplicio, Pereira, Doliskani and Barreto [30]. An interesting observation by Zanon et al. [30, §2] is that one can equivalently compute $c_0, d_0, c_1, d_1 \in \mathbb{Z}/m^{e_m}\mathbb{Z}$ such that

$$\begin{bmatrix} U_m \\ V_m \end{bmatrix} = \begin{bmatrix} c_0 & d_0 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} \phi_\ell(P_m) \\ \phi_\ell(Q_m) \end{bmatrix}, \quad \text{i. e.} \quad \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} = \begin{bmatrix} c_0 & d_0 \\ c_1 & d_1 \end{bmatrix}^{-1},$$

which they refer to as *reverse* basis decomposition. Its main upside is that the pairing value

$$g_0 := \tau_{m^{e_m}}(\phi_\ell(P_m), \phi_\ell(Q_m)) = \tau_{m^{e_m}}(P_m, Q_m)^{\ell^{e_\ell}},$$

where $\tau_{m^{e_m}}$ is the order-m^{e_m} *reduced Tate pairing* [20], solely depends on public parameters and can therefore be precomputed. The computation of the full compression algorithm is then typically divided into three stages.

1. (*Basis generation*) Compute the basis U_m, V_m.
2. (*Pairing computation*) Compute the pairings

$$\begin{aligned} g_1 &= \tau_{m^{e_m}}(\phi_\ell(P_m), U_m), & g_2 &= \tau_{m^{e_m}}(\phi_\ell(P_m), V_m), \\ g_3 &= \tau_{m^{e_m}}(\phi_\ell(Q_m), U_m), & g_4 &= \tau_{m^{e_m}}(\phi_\ell(Q_m), V_m). \end{aligned} \tag{1}$$

3. (*Discrete logarithm computation*) Find $c_0, d_0, c_1, d_1 \in \mathbb{Z}/m^{e_m}\mathbb{Z}$ such that

$$g_1 = g_0^{d_0}, \qquad g_2 = g_0^{d_1}, \qquad g_3 = g_0^{-c_0}, \qquad g_4 = g_0^{-c_1}.$$

The first stage is easy for $m = 2$, in which case an *entangled* basis can be computed at extremely low cost [30, §3]. The case of $m = 3$ is more complicated; the work of Costello et al. [7] proposes to use techniques related to explicit 3-descent by Schaefer and Stoll [27] to generate points in $(E/\langle R\rangle) \setminus [3](E/\langle R\rangle)$, while Zanon et al. [30, §3] find that naïve generation of such points via cofactor multiplication yields better results. We observe that the difficulty of applying the 3-descent techniques seems to lie in the fact that there are no known non-trivial 3-torsion points, requiring initial cofactor multiplications to find such points.

The most costly part of the algorithm is the second phase, in which 4 simultaneous pairings are computed. Although optimizations can be made by observing that inputs are shared and by choosing an optimal curve model $E/\langle R\rangle$, the large cost remains.

Finally 4 (simultaneous) discrete logarithms need to be computed, which is feasible due to the smoothness of the group order of points on the elliptic curve. This can be done relatively cheaply with very little requirements on memory [7, §5], and can be sped up through the use of precomputed tables [30, §6].

Contributions. We propose several improvements that together significantly reduce the computational overhead imposed on SIDH and SIKE by public-key

compression. The main idea behind our optimizations is to utilize the fact that the pairing values in Eq. (1) satisfy

$$g_1 = \tau_{m^{e_m}}(P_m, \widehat{\phi}_\ell(U_m)), \qquad g_2 = \tau_{m^{e_m}}(P_m, \widehat{\phi}_\ell(V_m)),$$
$$g_3 = \tau_{m^{e_m}}(Q_m, \widehat{\phi}_\ell(U_m)), \qquad g_4 = \tau_{m^{e_m}}(Q_m, \widehat{\phi}_\ell(V_m)),$$

where $\widehat{\phi}_\ell$ denotes the (unique) *dual isogeny* of ϕ_ℓ.

Our first contribution (see Sect. 3) is to propose explicit efficient formulas for computing duals of isogenies of degree 2, 4 and prime $\ell > 2$ between Montgomery curves, as proposed by Costello and Hisil [6] and Renes [25]. Through the use of optimal strategies, these can be used to compute dual isogenies of degrees ℓ^{e_ℓ} for any prime ℓ. However, the crucial observation is that the efficiency of evaluating $\widehat{\phi}_\ell$ increases significantly by re-using values that are obtained during the computation of ϕ_ℓ. In Proposition 3 we describe the kernels of all ℓ-isogenies appearing in the decomposition of $\widehat{\phi}_\ell$, and we give details on the computation and the values to store in Sects. 3.1 and 3.2. Having an efficient way to compute the dual, we gain the flexibility to apply the following idea.

Instead of evaluating ϕ_ℓ on the basis points P_m and Q_m, we pull back the deterministically generated basis points U_m and V_m through $\widehat{\phi}_\ell$ from $E/\langle R\rangle$ to E. This has several advantages, as the starting curve is a fixed system parameter and does not change throughout multiple executions of the protocol. As such, the starting curve can be chosen to have special properties, leading to more efficient operations. For example, precomputing $E(\mathbb{F}_{p^2})[3]$ leads to more efficient basis generation (see Sect. 4.3), while E being defined over $\mathbb{F}_p$ and having the basis points P_m and Q_m (almost) defined over $\mathbb{F}_p$ has significant advantage for the pairing computation (see Sect. 5). More specifically, we summarize our contributions as follows.

- The main contribution is the proposal to use the dual isogeny to pull back computations from $E/\langle R\rangle$ to E. For this purpose we show how to decompose $\widehat{\phi}_\ell$ as a sequence of ℓ-isogenies and how to evaluate them with very little overhead.
- We show how to utilize the dual isogeny in the basis generation phase. First, we adapt the entangled basis construction to an x-only setting. More importantly, we show that pulling back order and independence checking to E gives new interest to 3-descent methods. We analyze these methods in more detail, proving a strong relation to the reduced Tate pairing. Using this connection, we can reduce the cofactor scalar multiplications on E to exponentiations in $\mathbb{F}_{p^2}$ (i. e. pairings), significantly reducing the cost.
- We address the main bottleneck of public-key compression, namely the pairing computation. In the case of $\ell = 2$ we pull back the pairing to the $A = 0$ curve, on which a distortion basis for the m^{e_m}-torsion is available which greatly simplifies the pairing computation. For $\ell > 2$ and $m = 2$ we can pull back to the $A = 6$ curve, for which we can find basis points P_2 and Q_2 such that $[2]P_2$ is $\mathbb{F}_p$-rational and $[2]Q_2 = (x, iy)$, where $x, y \in \mathbb{F}_p$. This constrains many of the field operations to $\mathbb{F}_p$.

– As the m^{e_m}-torsion is a fixed parameter, we propose to use *affine* Weierstrass coordinates for the pairings and to precompute all Miller line functions. This leads to line functions that are very simple to evaluate, at the cost of a precomputed table. However, these tables are only several hundreds of kilobytes large and significantly smaller than those (already) used for discrete logarithms. Therefore, the memory overhead is small.

We have implemented[1] our techniques on top of the C library provided in the Round 2 submission package of SIKE, and compared our implementation to the uncompressed and compressed versions of SIKE as submitted to NIST across all parameter sets `SIKEpXXX`, where `XXX` $\in \{434, 503, 610, 751\}$. Our results show that public-key compression for SIDH can be implemented with an induced overhead of 77–86% (resp. 59–61%), compared to the previously best 160–182% (resp. 98–104%) of [17,30] for Alice (resp. Bob) across the different parameter sets (see Table 4a). Moreover, the compression techniques for SIKE induce an overhead of 61–74% (was 140–153%) for key generation, of 38–57% (was 67–90%) for encapsulation and 34–39% for decapsulation (was 59–65%) for the different parameter sets (see Table 4b). Finally, our results show that we speed up the pairing phase by a factor at least 2.97 for $\ell = 2$ and a factor at least 2.70 for $\ell = 3$, while also increasing efficiency of basis generation and decompression for $\ell = 2$. (see Table 2).

Remark 1. As the implementation focuses on $\{\ell, m\} = \{2, 3\}$, which seems to be the optimal parametrization for SIKE, our descriptions often also make this assumption for the sake of simplicity. Everything that is described in this work naturally generalizes to other primes. In that case it should be noted that $\ell = 2$ often exhibits special behavior (e. g. the existence of an entangled basis, or a special case for isogeny formulas [25, Proposition 1]) so we treat it separately, but our contributions work perfectly well by selecting m to be an arbitrary odd prime. Of course, one is also free to choose both ℓ and m to be odd primes without any (theoretical) problems.

Remark 2. The techniques that we describe rely on being able to evaluate the dual isogeny $\widehat{\phi_\ell}$ on a torsion basis $(E/\langle R\rangle)[m^{e_m}]$, which is equivalent to being able to evaluate ϕ_ℓ on $E[m^{e_m}]$. That is, given a point $U \in (E/\langle R\rangle)[m^{e_m}]$ we could (efficiently) solve the two-dimensional discrete logarithm $U = [a]\phi_\ell(P_m) + [b]\phi_\ell(Q_m)$ for some $a, b \in \mathbb{Z}/m^{e_m}\mathbb{Z}$, from which it follows that $\widehat{\phi_\ell}(U) = [\ell^{e_\ell} a]P_m + [\ell^{e_\ell} b]Q_m$. Thus, computing on the points $\widehat{\phi_\ell}(U)$ and $\widehat{\phi_\ell}(V)$ leaks no more information about the secret key than computing on $\phi_\ell(P)$ and $\phi_\ell(Q)$ does.

On the other hand, the evaluation of the dual isogeny itself does rely on secret data, while the intermediate points that are stored are also sensitive (as is the case with the evaluation of ϕ_ℓ). We simply apply the same protections to the dual evaluation as are applied to ϕ_ℓ, which in the implementation of SIKE just means that all algorithms are constant-time (see Sect. 3).

[1] The implementation is available as part of the SIDH Library v3.2, https://github.com/microsoft/PQCrypto-SIDH.

2 Preliminaries and Notation

We begin by recalling the basic theory, to remind the reader of typical notions and to establish notation for the rest of this work. As we already discussed public-key compression techniques related to SIDH and SIKE in Sect. 1, we omit those details here.

Elliptic Curves. Let $p > 3$ be prime. An elliptic curve E defined over a field k of characteristic p is a smooth projective curve of genus 1 with specified base point $\mathcal{O}_E$. Although typically defined by the Weierstrass model [29, §III.1], we shall always assume E to be described by the (less general) Montgomery form [22]

$$E : y^2 = x^3 + Ax^2 + x$$

for some $A \in k$ such that $A^2 \neq 4$, and may write E_A to emphasize the curve coefficient. As is the case for the Weierstrass model, the base point $\mathcal{O}_E$ is the unique point at infinity. The points on E form an abelian group with neutral element $\mathcal{O}_E$, and for any $m \in \mathbb{Z}$ we let $[m]P = P + \ldots + P$ be the sum of m copies of P (and a negation if m is negative). For any such non-zero $m \in \mathbb{Z}$, we let

$$E[m] = \{P \in E \mid [m]P = \mathcal{O}_E\}$$

be the m-torsion subgroup and say that E is *supersingular* whenever $\#E[p] = 1$. In that case we have $j(E) \in \mathbb{F}_{p^2}$ [29, Theorem V.3.1], so that E is isomorphic to a curve defined over $\mathbb{F}_{p^2}$. The number of isomorphism classes of supersingular elliptic curves over $\overline{k}$ of characteristic p is seen to be exactly $\lfloor p/12 \rfloor + \varepsilon_p$ [14, Theorem 9.11.11], where

$$\varepsilon_p = \begin{cases} 0 & \text{if } p \equiv 1 \bmod 12\,, \\ 1 & \text{if } p \equiv 5, 7 \bmod 12\,, \\ 2 & \text{if } p \equiv 11 \bmod 12\,. \end{cases}$$

Indeed, in this work we only concern ourselves with Montgomery curves defined over $\mathbb{F}_{p^2}$ for some (large) prime p.

Isogenies and Their Duals. Given any two elliptic curves E and $\overline{E}$ defined over k, an *isogeny* $\phi : E \to \overline{E}$ is a non-constant morphism such that $\phi(\mathcal{O}_E) = \mathcal{O}_{\overline{E}}$. It induces a field embedding $\phi^* : k(\overline{E}) \to k(E)$, and we say that ϕ is *separable* whenever the finite [29, Theorem II.2.4(a)] field extension $k(E)/\phi^* k(\overline{E})$ is separable, in which case we define $\deg \phi = [k(E) : \phi^* k(\overline{E})]$. The map $\phi \mapsto \ker \phi$ defines a correspondence between separable isogenies defined over k emanating from E and subgroups of E that are invariant under the action of $\mathrm{Gal}(\overline{k}/k)$, up to post-composition with an isomorphism [14, Theorem 9.6.19]. Given any isogeny ϕ defined over k, there exists a unique isogeny $\widehat{\phi}$ defined over k of the same degree as ϕ such that $\phi\widehat{\phi} = \widehat{\phi}\phi = [\deg \phi]$. The isogeny $\widehat{\phi}$ is called the *dual isogeny* of ϕ [29, Theorem III.6.1].

Reduced Tate Pairing. Now let $k = \mathbb{F}_q$ be a finite field containing the (cyclic) group of m-th roots of unity μ_m. We denote by

$$\tau_m : E(k)[m] \times E(k)/mE(k) \to \mu_m$$

the *reduced Tate pairing* [20] of order m defined by $\tau_m(S,T) = f_{m,S}(T)^{(q-1)/m}$, where $f_{m,S}$ is a rational function with divisor $m(S) - m(\mathcal{O})$. Interestingly, we have

$$\tau_m(\phi(S), T') = \tau_m(S, \widehat{\phi}(T'))$$

for any isogeny $\phi : E \to E'$ and points $S \in E(k)[m]$, $T' \in E'(k)/mE'(k)$ [3, Theorem IX.9]. Although not generally true, in the cases of our interest we shall always have $E(\mathbb{F}_q)/mE(\mathbb{F}_q) \cong E(\mathbb{F}_q)[m]$, and have the additional property that $\tau_m(S,T) = \tau_m(T,S)^{-1}$ for any $S, T \in E(\mathbb{F}_q)[m]$.

SIDH and SIKE. First we consider the SIDH protocol, proposed in 2011 by Jao and De Feo [19]. Let ℓ, m be distinct primes and e_ℓ, e_m be strictly positive integers such that $p = \ell^{e_\ell} \cdot m^{e_m} - 1$ is prime. Let $E/\mathbb{F}_{p^2}$ be a supersingular elliptic curve such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$, and let $\phi_\ell : E \to E/\langle R \rangle$ be an isogeny of degree ℓ^{e_ℓ} such that $\ker \phi_\ell = \langle R \rangle$ for some point $R \in E[\ell^{e_\ell}]$. Similarly, let $\phi_m : E \to E/\langle S \rangle$ be an isogeny of degree m^{e_m} such that $\ker \phi_m = \langle S \rangle$ for some point $S \in E[m^{e_m}]$. The shared secret is then (derived from) $j(E/\langle R, S \rangle)$. Notably, this is not feasibly computable from R and $E/\langle S \rangle$ or from S and $E/\langle R \rangle$ respectively.

Instead, we fix public parameters $P_\ell, Q_\ell \in E[\ell^{e_\ell}]$ and $P_m, Q_m \in E[m^{e_m}]$ and derive the points R, S from secret keys $s_0, s_1 \in \mathbb{Z}/\ell^{e_\ell}\mathbb{Z}$ and $t_0, t_1 \in \mathbb{Z}/m^{e_m}\mathbb{Z}$ such that

$$R = [s_0]P_\ell + [s_1]Q_\ell\,, \quad S = [t_0]P_m + [t_1]Q_m$$

have the desired order. That is, not both s_0 and s_1 (resp. t_0 and t_1) are divisible by ℓ (resp. m). The (naïve) public keys are then $[E/\langle R \rangle, \phi_\ell(P_m), \phi_\ell(Q_m)]$ and $[E/\langle S \rangle, \phi_m(P_\ell), \phi_m(Q_\ell)]$, observing that

$$\begin{aligned} & \left(E/\langle R \rangle\right) / \langle [t_0]\phi_\ell(P_m) + [t_1]\phi_\ell(Q_m) \rangle \\ \cong\ & E/\langle R, S \rangle \\ \cong\ & \left(E/\langle S \rangle\right) / \langle [s_0]\phi_m(P_\ell) + [s_1]\phi_m(Q_\ell) \rangle\,. \end{aligned}$$

It was noted by Costello, Longa and Naehrig [8, §6] that the public keys can be encoded (up to *simultaneous* sign) by the triples of x-coordinates

$$\left[x_{\phi_\ell(P_m)}, x_{\phi_\ell(Q_m)}, x_{\phi_\ell(P_m - Q_m)}\right]\,, \left[x_{\phi_m(P_\ell)}, x_{\phi_m(Q_\ell)}, x_{\phi_m(P_\ell - Q_\ell)}\right]\,,$$

which can be represented with $6 \log_2 p$ bits each.

Unfortunately, the SIDH key exchange scheme combined with static keys is insecure as the result of an active adaptive attack by Galbraith, Petit, Shani and Ti [15]. Consequently, one must resort to using ephemeral public keys. Alternatively, one can apply standard protocol transformations [13,16] to turn the

IND-CPA key exchange into an IND-CCA key encapsulation mechanism. The resulting scheme is referred to as SIKE [17] and is currently part of the NIST Post-Quantum Cryptography Standardization effort [24]. Although we refer to [17] for more detail on the submission, we remark that the secret keys are chosen such that $s_0 = 1$ and $t_0 = 1$, simplifying some of the treatment.

Field Operations. We denote by $\mathbf{M}$ and $\mathbf{S}$ the cost of an $\mathbb{F}_{p^2}$ field multiplication and squaring respectively, and by $\mathbf{A}$ a field addition or subtraction (which are therefore assumed to have the same cost). We denote by $\mathbf{E}$ the cost of a square root in $\mathbb{F}_{p^2}$. Similarly, we write $\mathbf{m}$ and $\mathbf{s}$ for the cost of an $\mathbb{F}_p$ field multiplication and squaring respectively, while $\mathbf{a}$ denotes an addition or subtraction in $\mathbb{F}_p$. Reflecting the properties of the SIKE implementation, we use $\mathbf{M} = 3\mathbf{m} + 5\mathbf{a}$ and $\mathbf{S} = 2\mathbf{m} + 3\mathbf{a}$.

3 Evaluating Dual Isogenies

In this section we consider the computation of the dual isogeny in the context of SIDH and SIKE. That is, we look towards the case where E is a Montgomery curve defined over some field k with $\mathrm{char}(k) \neq 2$ and $\phi_\ell : E \to E/\langle R \rangle$ a separable isogeny of degree ℓ^{e_ℓ} with kernel $\langle R \rangle$ for some point $R \in E[\ell^{e_\ell}]$. In addition, we could let $p = \ell^{e_\ell} \cdot m^{e_m} - 1$ be a prime, and let E be a supersingular elliptic curve defined over $k = \mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Then R lies in $E(\mathbb{F}_{p^2})$ and, as a result, all arithmetic can be performed over $\mathbb{F}_{p^2}$. The latter is merely a computational advantage and not necessary for the statements below.

The first step to computing $\widehat{\phi}_\ell$ is finding its kernel. For this we note that $\widehat{\phi}_\ell \phi_\ell = [\ell^{e_\ell}]$, hence $\ker(\widehat{\phi}_\ell \phi_\ell) = E[\ell^{e_\ell}] \cong \langle R, S \rangle$ for some point $S \in E[\ell^{e_\ell}]$ of order ℓ^{e_ℓ}. From $\ker \phi_\ell = \langle R \rangle$ it is then immediate that $\ker \widehat{\phi}_\ell = \langle \phi_\ell(S) \rangle$. However, in cryptographic contexts the degree of ϕ_ℓ is too large for ϕ_ℓ to be computed directly, while the same is true for $\widehat{\phi}_\ell$. Instead, ϕ_ℓ is decomposed as

$$\phi_\ell = \phi_\ell^{(e_\ell - 1)} \circ \cdots \circ \phi_\ell^{(0)}$$

as a sequence of ℓ-isogenies. We begin by showing (see Proposition 3) how $\widehat{\phi}_\ell$ can be decomposed in a similar fashion, and describe the kernel of all intermediate ℓ-isogenies.

Proposition 3. *Let E be an elliptic curve defined over a field k and let $\phi_\ell : E \to E/\langle R \rangle$ be an isogeny of degree ℓ^{e_ℓ} with kernel $\langle R \rangle$ for some point $R \in E[\ell^{e_\ell}]$. Let $\phi_\ell = \phi_\ell^{(e_\ell - 1)} \circ \cdots \circ \phi_\ell^{(0)}$, where $\ker \phi_\ell^{(0)} = \langle [\ell^{e_\ell - 1}]R \rangle$ and $\ker \phi_\ell^{(i)} = \langle [\ell^{e_\ell - 1 - i}](\phi_\ell^{(i-1)} \circ \cdots \circ \phi_\ell^{(0)})(R) \rangle$ for $i = 1, \ldots, e_\ell - 1$. Then*

$$\widehat{\phi}_\ell = \widehat{\phi}_\ell^{(0)} \circ \cdots \circ \widehat{\phi}_\ell^{(e_\ell - 1)}, \quad \text{with} \quad \ker \widehat{\phi}_\ell^{(i)} = \langle (\phi_\ell^{(i)} \circ \cdots \circ \phi_\ell^{(0)})([\ell^{e_\ell - 1}]S) \rangle$$

for $i = 0, \ldots, e_\ell - 1$ and any $S \in E[\ell^{e_\ell}]$ such that $\langle R, S \rangle = E[\ell^{e_\ell}]$.

Proof. The first part follows by uniqueness of the dual isogeny, and since

$$\begin{aligned}
&\widehat{\phi}_\ell^{(0)} \circ \cdots \circ \widehat{\phi}_\ell^{(e_\ell-1)} \circ \phi_\ell^{(e_\ell-1)} \circ \cdots \circ \phi_\ell^{(0)} \\
&= \widehat{\phi}_\ell^{(0)} \circ \cdots \circ \widehat{\phi}_\ell^{(e_\ell-2)} \circ \phi_\ell^{(e_\ell-2)} \circ \cdots \circ \phi_\ell^{(0)} \circ [\ell] \\
&\quad \vdots \\
&= [\ell^{e_\ell}] .
\end{aligned}$$

Now observe that $E[\ell] = \langle [\ell^{e_\ell-1}]R, [\ell^{e_\ell-1}]S \rangle$, so by using a similar argument as above it follows that $\ker \widehat{\phi}_\ell^{(0)} = \phi_\ell^{(0)}([\ell^{e_\ell-1}]S)$. As any linear relation between the $\ell^{e_\ell-1}$-torsion points $\phi_\ell^{(0)}(R)$ and $\phi_\ell^{(0)}([2]S)$ leads to one between R and S, they form a basis for $(E/\langle [\ell^{e_\ell-1}]R \rangle)[\ell^{e_\ell-1}]$. The statement then follows by proceeding via induction on e_ℓ. □

It is now clear how we can evaluate $\widehat{\phi}_\ell$. We select an arbitrary point S, linearly independent of the kernel point R, and during the computation of ϕ_ℓ we evaluate and store the intermediate evaluations of $[\ell^{e_\ell-1}]S$. These determine the kernels of the ℓ-isogenies appearing in the decomposition of $\widehat{\phi}_\ell$, so it remains to show how to compute the dual of an ℓ-isogeny (i. e. the case $e_\ell = 1$). This of course strongly depends on the choice for ϕ_ℓ, for which we restrict to the parameters of SIKE. That is, we assume E and $E/\langle R \rangle$ to be Montgomery curves and consider the cases where $\ell > 2$ (Sect. 3.1) and where $\ell = 2$ (Sects. 3.2 and 3.3) separately.

Remark 4. This is especially easy in the case of SIKE, where $R = P_\ell + [s_1]Q_\ell$ for some $s_1 \in \mathbb{Z}/\ell^{e_\ell}\mathbb{Z}$. In that case we simply select $S = Q_\ell$ and store intermediate evaluations of $[\ell^{e_\ell-1}]Q_\ell$.

We note that we can write $\phi_\ell = (f_\ell(x), cyf'_\ell(x))$ for some rational function $f_\ell(x)$ in $k(x)$ [14, Theorem 9.7.5] and some $c \in k^*$, where $f'_\ell(x)$ is the formal derivative $df_\ell(x)/dx$ of $f_\ell(x)$. Therefore, the isogeny ϕ_ℓ is determined by $f_\ell(x)$ up to a possible twisting of the y-coordinate by varying c. As the only monomial containing y in Montgomery form is y^2, which has coefficient 1, it follows that $f_\ell(x)$ determines ϕ_ℓ up to composition by $[\pm 1]$. Similarly, the dual $\widehat{\phi}_\ell = (\widehat{f}_\ell(x), \widehat{c}y\widehat{f}'_\ell(x))$ is determined by $\widehat{f}_\ell(x)$ up to composition $[\pm 1]$. As it suffices for SIDH to compute ϕ_ℓ up to sign, and for our purposes it suffices to compute $\widehat{\phi}_\ell$ up to sign, in what follows we focus on the description of the function $\widehat{f}_\ell$.

3.1 The Case $\ell > 2$

First we consider the case where $\ell > 2$ (which turns out to be the simplest) and let R be a point of order ℓ (i. e. $e_\ell = 1$). In that case, the isogeny $\phi_\ell = (f_\ell(x), cyf'_\ell(x))$ with

$$f_\ell(x) = x \cdot \prod_{T \in \langle R \rangle \setminus \{\mathcal{O}\}} \frac{xx_T - 1}{x - x_T} \tag{2}$$

is an ℓ-isogeny with kernel $\langle R \rangle$, see [6, Theorem 1]. The case $\ell = 3$ is used for computations in the SIKE proposal [17] and in our implementation, but since the more general case follows analogously we also treat it here.

Proposition 5. *Let $E : y^2 = x^3 + Ax^2 + x$ be a Montgomery curve defined over a field k with $\mathrm{char}(k) \neq 2$. Let R and S be two linearly independent points of (prime) order ℓ and let $\phi_\ell = (f_\ell(x), cyf'_\ell(x))$ with*

$$f_\ell(x) = x \cdot \prod_{T \in \langle R \rangle \setminus \{\mathcal{O}\}} \frac{xx_T - 1}{x - x_T}$$

be an ℓ-isogeny of Montgomery curves with dual $\widehat{\phi}_\ell = (\widehat{f}_\ell(x), \widehat{c}y\widehat{f}'_\ell(x))$. Then

$$\widehat{f}_\ell(x) = x \cdot \prod_{T \in \langle \phi_\ell(S) \rangle \setminus \{\mathcal{O}\}} \frac{xx_T - 1}{x - x_T}.$$

Proof. Let $\overline{\phi}_\ell = (\overline{f_\ell}(x), \overline{c}y\overline{f'_\ell})$ be an isogeny with

$$\overline{f_\ell}(x) = x \cdot \prod_{T \in \langle \phi_\ell(S) \rangle \setminus \{\mathcal{O}\}} \frac{xx_T - 1}{x - x_T}.$$

It is clear that $\phi_\ell(S)$ is a point of order ℓ on $E/\langle R \rangle$, so applying [6, Theorem 1] to $\phi_\ell(S)$ shows that $\overline{\phi}_\ell$ is indeed an isogeny such that $\ker \overline{\phi}_\ell \phi_\ell = E[\ell]$. As the kernels are equal, $\overline{\phi}_\ell$ is equal to $\widehat{\phi}_\ell$ up to post-composition with an isomorphism. We finish the proof by showing that the only possible isomorphisms are $[\pm 1]$.

For that purpose we consider the point $(1, \sqrt{A+2})$ of order 4 on E, which satisfies $[\ell](1, \sqrt{A+2}) = (1, \pm\sqrt{A+2})$ depending on the value of $\ell \bmod 4$. No matter which is the case, it follows that $[\ell] = \widehat{\phi}_\ell \phi_\ell$ fixes x-coordinate 1 or, in other words, that $\widehat{f}_\ell f_\ell(1) = 1$. Similarly, considering the point $(0, 0)$ of order 2 shows that $\widehat{f}_\ell f_\ell(0) = 0$.

Now note that indeed $\overline{f_\ell} f_\ell(1) = 1$ and $\overline{f_\ell} f_\ell(0) = 0$, so that any isomorphism post-composed with $\overline{\phi_\ell}$ to obtain $\widehat{\phi}_\ell$ must act as the identity on the x-coordinates 0 and 1. By [29, Proposition III.3.1(b)], the only such isomorphisms are $[\pm 1]$. Therefore, $\overline{\phi}_\ell = [\pm 1]\widehat{\phi}_\ell$ and the result follows. □

Interestingly, Proposition 5 shows that we can compute duals of ℓ-isogenies using the exact same formulas for the isogeny ϕ_ℓ itself. In the case of $\ell = 3$ this (i. e. its projectivized version) can be computed at the cost of $4\mathbf{M} + 2\mathbf{S} + 4\mathbf{A}$ for each first evaluation, and $4\mathbf{M} + 2\mathbf{S} + 2\mathbf{A}$ for each subsequent evaluation [6, Appendix A].

3.2 The Case of 4-Isogenies

Now assume that $\ell = 2$ and that the point R has order 4 (i. e. $e_2 = 2$) such that $[2]R \neq (0, 0)$. Again, the isogeny $\phi_2 = (f_2(x), cyf'_2(x))$ can be described by an

equation of the form (2), see [25, Proposition 1]. If S is any other point of order 4 linearly independent from R, i. e. $E[4] = \langle R, S\rangle$, then again $\ker \widehat{\phi}_2 = \langle \phi_2(S)\rangle$. However, in contrast to the case of $\ell > 2$, applying the formulas from the SIKE proposal [17] (which are essentially those from [25, Proposition 1]) leads to a point $\phi_2(S)$ such that $x_{\phi_2(S)} = 1$ and $[2]\phi_2(S) = (0, 0)$. As a result, the dual isogeny can not be described by the formulas of [25, Proposition 1].

Instead, the original work of De Feo–Jao–Plût [12, Eqn. (18)–(21)] describes formulas for a 4-isogeny whose kernel is generated by a point with x-coordinate 1. Unfortunately, unlike before, there is no reason that this isogeny has the correct co-domain. As such, we post-compose with an appropriate isomorphism. One option for such an isomorphism is given in [12, Eqn. (15)], but it is described through the knowledge of a point of order 2. As such a point is not readily (or cheaply) available in our context, one needs to compute a (typically expensive) doubling. We show that this is much cheaper due to the assumption that R has order 4. We summarize this in Proposition 6.

Proposition 6. *Let $E : y^2 = x^3 + Ax^2 + x$ be a Montgomery curve defined over a field k with $\mathrm{char}(k) \neq 2$. Let R be a point of order 4 such that $[2]R \neq (0, 0)$, and let $\phi_2 = (f_2(x), cyf_2'(x)) : E \to E/\langle R\rangle : y^2 = x^3 + \widehat{A}x^2 + x$ with*

$$f_2(x) = x \cdot \prod_{T \in \langle R\rangle \setminus \{\mathcal{O}\}} \frac{xx_T - 1}{x - x_T}$$

be a 4-isogeny of Montgomery curves with dual $\widehat{\phi}_2 = (\widehat{f}_2(x), \widehat{c}y\widehat{f}_2'(x))$. Then

$$\widehat{f}_2(x) = \frac{(x_R^2 - 1)\overline{X} + (x_R^2 + 1)\overline{Z}}{2x_R\overline{Z}},$$

where

$$\overline{X} = (x + 1)^2\left((x + 1)^2 - 4(1 - \widehat{A}_{24})x\right), \quad \overline{Z} = 4(1 - \widehat{A}_{24})x(x - 1)^2,$$

and $\widehat{A}_{24} = (\widehat{A} + 2)/4$.

Proof. Let $S = (1, \sqrt{A + 2})$ be a point on E, which has order 4 and is linearly independent from R. As a result, the kernel of the dual of ϕ_2 is generated by $\phi_2(S)$. As $f_2(1) = 1$, the kernel of $\widehat{\phi}_2$ is generated by a point with x-coordinate equal to 1.

The map $\phi_4(x)$ (not to be confused with ϕ_2) computed from [12, Eqn. (18)–(21)] as the concatenation of the maps

$$(x, y) \mapsto \left(\frac{(x - 1)^2}{x}, y\left(1 - \frac{1}{x^2}\right)\right)$$

followed by

$$(x, y) \mapsto \left(\frac{1}{2 - \widehat{A}}\frac{(x + 4)(x + \widehat{A} + 2)}{x}, \frac{y}{2 - \widehat{A}}\left(1 - \frac{4(2 + \widehat{A})}{x^2}\right)\right)$$

is seen to be an isogeny of degree 4 such that the generator of its kernel has x-coordinate 1, and satisfies $\phi_4(x) = \overline{X}/\overline{Z}$. Thus ϕ_4 on $E/\langle R\rangle$ determines an isogeny equal to $\widehat{\phi}_2$ up to post-composition by an isomorphism. As $\ker\widehat{\phi}_4$ is generated by a point of x-coordinate equal to 1 [12, §4.3.2], while $x_R \neq 1$, it follows that $\widehat{\phi}_4 \neq \phi_2$. Taking duals on both sides, we find that $\phi_4 \neq \widehat{\phi}_2$ [29, Theorem III.6.2]. Instead, the isomorphism ψ^{-1}, where

$$\psi : (x, y) \mapsto \left(\frac{x - x_{[2]R}}{x_R - x_{[2]R}}, \frac{y}{x_R - x_{[2]R}}\right)$$

is the map described in [12, Eqn. (15)], maps the kernel of $\widehat{\phi}_4$ to $\langle R\rangle$ and we conclude that $\widehat{\phi}_2 = \psi^{-1}\phi_4$.

At first glance the map ψ requires the use of (the x-coordinate of) $[2]R$, which is generally costly to compute. We show that this simplifies due to R being a point of order 4. Writing $\psi^{-1} = (h(x), yh'(x))$, we note that $h(1) = x_R$ and $h(0) = x_{[2]R}$. Also, let $T = \psi(0, 0)$ be a point of order 2 such that $(-1, \sqrt{A-2}) = (1, \sqrt{A+2}) + T$ for an appropriate choice of square roots. Then

$$\psi^{-1}(-1, \sqrt{A-2}) = \psi^{-1}(1, \sqrt{A+2}) + (0, 0)\,,$$

implying that $h(-1) = 1/x_R$. Again, by [29, Proposition III.3.1(b)] we have $h(x) = ax + b$ for some $a, b \in k$, for which the above restrictions imply that $b + a = x_R$, $b - a = 1/x_R$ and $b = x_{[2]R}$. It follows that $x_{[2]R} = (x_R + 1/x_R)/2$ and thus

$$h(x) = (x_R - x_{[2]R})x + x_{[2]R} = \frac{(x_R^2 - 1)x + x_R^2 + 1}{2x_R}\,,$$

completing the proof. □

Projectivizing and writing $x_R = X_R/Z_R$, $x = X/Z$ and $\widehat{A}_{24} = \widehat{a}_{24}/\widehat{c}_{24}$, we can compute $\widehat{f}_2(x)$ as follows. First we compute the coefficients $[K_0, K_1, K_2] = [X_R^2 - Z_R^2, X_R^2 + Z_R^2, 2X_RZ_R]$ through the sequence of operations

$$T_0 \leftarrow X_R^2\,,\ T_1 \leftarrow Z_R^2\,,\ K_0 \leftarrow T_0 - T_1\,,\ K_1 \leftarrow T_0 + T_1\,,$$
$$K_2 \leftarrow X_R + Z_R\,,\ K_2 \leftarrow K_2^2\,,\ K_2 \leftarrow K_2 - K_1\,,$$

that can be computed at a cost of $3\mathbf{S} + 4\mathbf{A}$. We note that these operations are independent of x and can therefore be shared among multiple evaluations of $\widehat{f}_2(x)$ at distinct points. Moreover, in the context of SIDH and SIKE such an evaluation is always preceded by an evaluation of f_2 in which X_R^2, X_Z^2 and $X_R + Z_R$ are computed. Storing those intermediate values reduces the cost to $1\mathbf{S}+3\mathbf{A}$. We then complete the computation of $\widehat{f}_2(x) = X'/Z'$ via the operations

$$T_0 \leftarrow X + Z\,,\ T_1 \leftarrow X - Z\,,\ T_0 \leftarrow T_0^2\,,\ T_1 \leftarrow T_1^2\,,\ T_2 \leftarrow T_0 - T_1\,,$$
$$T_3 \leftarrow \widehat{c}_{24} - \widehat{a}_{24}\,,\ T_3 \leftarrow T_2 \cdot T_3\,,\ T_2 \leftarrow \widehat{c}_{24} \cdot T_0\,, T_2 \leftarrow T_2 - T_3\,,\ \overline{X} \leftarrow T_2 \cdot T_0\,,$$
$$\overline{Z} \leftarrow T_3 \cdot T_1\,,\ X' \leftarrow K_0 \cdot \overline{X}\,,\ T_0 \leftarrow K_1 \cdot \overline{Z}\,,\ X' \leftarrow X' + T_0\,,\ Z' \leftarrow K_2 \cdot \overline{Z}\,,$$

at a cost of $7\mathbf{M}+2\mathbf{S}+6\mathbf{A}$. Summarizing, assuming having stored the intermediate values $[\widehat{a}_{24}, \widehat{c}_{24}, X_R^2, Z_R^2, 2X_RZ_R]$, the first evaluation of $\widehat{f}_2(x)$ can be performed at a cost of $7\mathbf{M} + 3\mathbf{S} + 9\mathbf{A}$. Any subsequent evaluation can be computed at a cost of $7\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$. For comparison, the evaluation of f_2 in SIKE currently has a cost of $6\mathbf{M} + 2\mathbf{S} + 6\mathbf{A}$. Hence, although the dual is more expensive than the original 4-isogeny, the difference is small.

3.3 The Case of 2-Isogenies

Finally consider $\ell = 2$ and assume that $R \neq (0,0)$ is a point of order 2. We note that 2-isogenies are only employed in the SIKE proposal whenever $e_\ell \not\equiv 0 \bmod 4$, and in that case only a single one is computed. Therefore its cost is negligible to the overall cost of the isogeny. The 2-isogeny is computed as in [25, Proposition 2], and we refer to Proposition 7 for the computation of its dual.

Proposition 7. *Let $E : y^2 = x^3 + Ax^2 + x$ be a Montgomery curve defined over a field k with $\mathrm{char}(k) \neq 2$. Let $R \neq (0,0)$ be a point of order 2 and let $\phi_2 = (f_2(x), cyf_2'(x))$ with*

$$f_2(x) = x \cdot \frac{xx_R - 1}{x - x_R}$$

be a 2-isogeny of Montgomery curves with dual $\widehat{\phi}_2 = (\widehat{f}_2(x), \widehat{c}y\widehat{f}_2'(x))$. Then

$$\widehat{f}_2(x) = \frac{(x+1)^2}{4x_Rx}.$$

Proof. First we note that $\ker \widehat{\phi}_2 = \langle (0,0) \rangle$ by [25, Corollary 1]. An isogeny with such a kernel can be computed by composing the maps

$$(x, y) \mapsto \left(\frac{(x-1)^2}{x}, y\Big(1 - \frac{1}{x^2}\Big) \right)$$

from [12, Eqn. (19)] followed by the map

$$(x, y) \mapsto \left(\frac{x + \widehat{A} + 2}{\sqrt{\widehat{A}^2 - 4}}, \frac{y}{\sqrt{\widehat{A}^2 - 4}} \right),$$

as observed in [25, Remark 6], where $\widehat{A} = 2(1 - 2x_R^2)$ [25, Proposition 2]. After twisting the y-coordinate, this lands on the curve defined by the equation

$$y^2 = x^3 - \frac{2\widehat{A}}{\sqrt{\widehat{A}^2 - 4}}x^2 + x$$

whose dual is again generated by $(0,0)$. Finally, we post-compose with an isomorphism $\psi(x, y) = (h(x) = ax + b, yh'(x))$. As noted earlier, using the fact that taking duals acts as an involution implies that $h(0) = x_R$ and thus $b = x_R$.

Writing out the curve equation for $\psi(x, y)$ and noting that the coefficient of x^2 is A shows that

$$a = -\sqrt{\widehat{A}^2 - 4}\,(3x_R + A)\,/\,(2\widehat{A})\,.$$

Composing all these maps leads to the result, for which we omit the details as they are straightforward yet tedious. □

Letting $x = X/Z$ and $x_R = X_R/Z_R$, the following sequence of operations

$$T_0 \leftarrow X + Z\,,\ T_0 \leftarrow T_0^2\,,\ X' \leftarrow Z_R \cdot T_0\,,\ T_1 \leftarrow X - Z\,,$$
$$T_1 \leftarrow T_1^2\,,\ T_1 \leftarrow T_0 - T_1\,,\ Z' \leftarrow X_R \cdot T_1\,,$$

computes $\widehat{f}_2(x) = X'/Z'$ at a cost of $2\mathbf{M} + 2\mathbf{S} + 3\mathbf{A}$.

4 Generation of Torsion Bases

As usual we let $p = \ell^{e_\ell} \cdot m^{e_m} - 1$ be a prime, and let $E : y^2 = x^3 + Ax^2 + x$ be a supersingular elliptic curve defined over $k = \mathbb{F}_{p^2}$ such that $\#E(\mathbb{F}_{p^2}) = (p+1)^2$. Again, we let $\phi_\ell : E \to E/\langle R\rangle$ be a separable isogeny of degree ℓ^{e_ℓ} with kernel $\langle R\rangle$ for some point $R \in E[\ell^{e_\ell}]$. The aim of this section is to describe how to compute $\widehat{\phi}_\ell(U_m)$ and $\widehat{\phi}_\ell(V_m)$ for some deterministically generated basis points U_m and V_m such that $(E/\langle R\rangle)[m^{e_m}] = \langle U_m, V_m\rangle$. This is (naïvely) done in a few steps.[2]

1. Deterministically generate a first point $U \in E/\langle R\rangle$.
2. Repeat 1–2 until $U_m = [\ell^{e_\ell}]U$ has order m^{e_m}.
3. Deterministically generate a second point $V \in E/\langle R\rangle$.
4. Repeat 3–4 until $V_m = [\ell^{e_\ell}]V$ has order m^{e_m} and $(E/\langle R\rangle)[m^{e_m}] = \langle U_m, V_m\rangle$.
5. Compute $\widehat{\phi}_\ell(U_m)$ and $\widehat{\phi}_\ell(V_m)$.

For $\ell = 3$ we do not deviate much from this, yet we remark that it is not necessary to generate the full points U_2 and V_2. Instead, since the dual isogeny computes only on x-coordinates, it suffices to compute x_{U_2} and x_{V_2}. In fact, it is even enough to only obtain x_U and x_V, as the cofactor multiplications naturally factor out during the pairing and discrete logarithm phase [30, §3.1]. However, for the pairing to remain consistent we need to also deterministically compute x_{U-V} (without recovering y_U and y_V). We show how this can be done in Sect. 4.1 and how this applies to the entangled basis generation of [30, §3] in Sect. 4.2.

In the case of $\ell = 2$ we do take an alternative approach. The difference is that checking the order of U and V has to be done through cofactor multiplications $[3^{e_3-1}2^{e_2}]U$ and $[3^{e_3-1}2^{e_2}]V$, both of which are very costly. We propose generating the basis in the following way, recalling that the dual isogeny is defined as $\widehat{\phi}_2 = (\widehat{f}_2(x), \widehat{c}y\widehat{f}_2')$.

[2] Note that when considering ϕ_ℓ of degree ℓ^{e_ℓ}, we generate a basis of the m^{e_m}-torsion.

1. Deterministically generate x_U for a point $U \in E/\langle R \rangle$.
2. Compute $\widehat{f}_2(x_U)$ and recover $\widehat{\phi}_2(U)$.
3. Repeat 1–3 until $[2^{e_2}]\widehat{\phi}_2(U)$ has order 3^{e_3}.
4. Deterministically generate x_V for a point $V \in E/\langle R \rangle$.
5. Compute $\widehat{f}_2(x_V)$ and recover $\widehat{\phi}_2(V)$.
6. Repeat 4–6 until $E[3^{e_3}] = \langle [2^{e_2}]\widehat{\phi}_2(U), [2^{e_2}]\widehat{\phi}_2(V) \rangle$.
7. Deterministically generate x_{U-V} and compute $\widehat{f}_2(x_{U-V})$.
8. Modify signs of $\widehat{\phi}_2(U)$, $\widehat{\phi}_2(V)$ so that $x_{\widehat{\phi}_2(U)-\widehat{\phi}_2(V)} = \widehat{f}_2(x_{U-V})$.

We can then obtain $\widehat{\phi}_2(U_3) = [2^{e_2}]\widehat{\phi}_2(U)$ and $\widehat{\phi}_2(V_3) = [2^{e_2}]\widehat{\phi}_2(V)$, but we show in Sect. 4.3 that this is never explicitly necessary. This presents some trade-offs, which we briefly discuss. Firstly, we note that more computation is wasted when a point of the wrong order is generated (i. e. in step 3) or when it is not independent (i. e. in step 6). That is, the evaluation of $\widehat{f}_2$ would unfortunately have been done for nothing. However, since $E[3] \cong \mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$, we expect the points to have full order with probability 8/9 and to be independent with probability 3/4. Thus on average we require to perform steps 1–3 only 9/8 times and step 4–6 only 3/2 times.

Moreover, we observe that we can check the order of $\widehat{\phi}_2(U)$ and $\widehat{\phi}_2(V)$ on E as opposed to $E/\langle R \rangle$. The main advantage is that E is a fixed public parameter, whereas $E/\langle R \rangle$ varies per choice of R. This allows pre-computation on E, and in particular the generation of 3-torsion points to apply the 3-descent methods of Schaefer and Stoll [27]. We further analyze this in Sect. 4.3 and show how this leads to improved performance.

4.1 Deterministically Generating X-Coordinates

The generation of the points U (and similarly for V) is done in two steps. First, one uses the Elligator 2 map [2] to generate an x-coordinate x_U in $\mathbb{F}_{p^2}$, after which the y-coordinate can be recovered (which is guaranteed to lie in $\mathbb{F}_{p^2}$). Therefore, the generation of the two points U and V requires performing two square roots in $\mathbb{F}_{p^2}$ (although only a single one is needed for the entangled basis, see Sect. 4.2). Evaluating the abscissa as well as the ordinate of $\widehat{\phi}_\ell$ on U and V is also very costly. We show how this can be done much more efficiently.

Instead, we take the approach of SIDH and only ever evaluate $\widehat{f}_\ell$ (i. e. the abscissa of $\widehat{\phi}_\ell$) on U and V, and thus never require their y-coordinates. As usual, we also evaluate $\widehat{f}_\ell$ at their difference (i. e. in step 8) for the computation to remain consistent. This leaves us with the problems of deterministically computing x_{U-V}, and consistently recovering the signs of $\widehat{\phi}_\ell(U)$ and $\widehat{\phi}_\ell(V)$ from $x_{\widehat{\phi}_\ell(U)}$, $x_{\widehat{\phi}_\ell(V)}$, and $x_{\widehat{\phi}_\ell(U-V)}$.

For the generation of x_{U-V} we refer to the techniques applied in the qDSA [26] signature scheme of Renes and Smith. More specifically, in [26, Proposition 3] it is shown that $a \cdot x_{U-V}^2 - 2b \cdot x_{U-V} + c = 0$, where

$$a = (x_U - x_V)^2 \,, \quad c = (x_U x_V - 1)^2 \,,$$
$$b = (x_U x_V + 1)\,(x_U + x_V) + 2\widehat{A} x_U x_V \,,$$

and where $\widehat{A}$ is the Montgomery curve coefficient of $E/\langle R\rangle$. It follows that

$$x_{U-V} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a},$$

allowing to (projectively) compute x_{U-V} at a cost of $1\mathbf{E} + 6\mathbf{M} + 5\mathbf{S} + 15\mathbf{A}$. This is made deterministic by fixing the choice for $\sqrt{b^2 - 4ac}$ in $\mathbb{F}_{p^2}$. As we evaluate $\widehat{f}_\ell$ projectively, there is no need for an inversion to obtain an affine representation. Notably, the computation above does not affect decompression, which uses the points in an x-only three-point ladder.

For the recovery of $\widehat{\phi}_\ell(U)$ and $\widehat{\phi}_\ell(V)$ we refer to [22, §10.3.1]. Writing

$$\widehat{\phi}_\ell(U) = (x_1, y_1), \quad \widehat{\phi}_\ell(V) = (x_2, y_2), \quad \widehat{\phi}_\ell(U - V) = (x_3, y_3),$$

we have

$$x_3 = \frac{(x_2y_1 + x_1y_2)^2}{x_1x_2(x_1 - x_2)^2} = \frac{x_2^2y_1^2 + x_1^2y_2^2 + 2x_1x_2y_1y_2}{x_1x_2(x_1 - x_2)^2}. \tag{3}$$

Using the curve equation for $\widehat{\phi}_\ell(V)$, a simple reorganization shows that

$$y_2 = \frac{x_1x_2x_3(x_1 - x_2)^2 - x_2^2y_1^2 + x_1^2(x_2^3 + Ax_2^2 + x_2)}{2x_1x_2y_1}. \tag{4}$$

Therefore, it suffices to compute y_1 at the cost of a single square root, after which y_2 is determined. Note that this only recovers $\widehat{\phi}_\ell(U)$ and $\widehat{\phi}_\ell(V)$ up to *simultaneous* sign, determined by the choice of y-coordinate for $\widehat{\phi}_\ell(U)$. As we are only interested in subgroups generated by linear combinations of these two points, this is not an issue. If we only want to verify that our choices of signs are consistent, it suffices to check that Eq. (3) holds. This is what we use in step 8 above, and in Sect. 4.3.

4.2 *X*-only Entangled Basis Generation for $\ell = 3$

The work of Sect. 4.1 becomes especially simple in the case of $\ell = 3$, where U and V are generated as an *entangled* basis [30, §3]. That is, $U = (x_1, y_1)$, where $x_1 = -\widehat{A}/(1 + t^2)$ is a quadratic non-residue and $t \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$ such that $t^2 \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$, and $V = (x_2, y_2)$ where $x_2 = -x_1 - \widehat{A}$ and $y_2 = t \cdot y_1$ [30, Theorem 1]. Writing $U - V = (x_3, y_3)$, we have

$$x_3 = \frac{(y_1 + y_2)^2}{(x_1 - x_2)^2} - \widehat{A} - x_1 - x_2 = \frac{(y_1 + y_2)^2}{(x_1 - x_2)^2} = \frac{(x_1^3 + \widehat{A}x_1^2 + x_1)(1 + t)^2}{(x_1 - x_2)^2},$$

see [22, §10.3.1] again. As done by Zanon et al. [30], we fix $u_0 = 1 + i$ and run over $t = u_0 \cdot r$ for $r = 1, 2, \ldots$ until we succeed. Building tables (r, v) for $r = 1, 2, \ldots$ and $v = 1/(1 + ur^2)$ of quadratic and non-quadratic residues, we can select $x_1 = -\widehat{A}v$, after which the values of x_2 and x_3 can be computed as above. We note that this does not require the computation of y_1, but merely requires checking whether $x_1^3 + \widehat{A}x_1^2 + x_1$ is a square (which has a lower cost).

Having generated x_U, x_V and x_{U-V}, we evaluate the values $\widehat{f}_3(x_U)$, $\widehat{f}_3(x_V)$ and $\widehat{f}_3(x_{U-V})$. After a square root computation to recover $\widehat{\phi}_3(U)$, we use Eqn. (4) to (consistently) recover $\widehat{\phi}_3(V)$.

4.3 Basis Generation with the Reduced Tate Pairing for $\ell = 2$

The situation is more complex for $\ell = 2$, for which there is no (known) analogue of an entangled basis. Instead, checking the order of U and V is done through cofactor multiplications $[3^{e_3-1}2^{e_2}]U$ and $[3^{e_3-1}2^{e_2}]V$. For that purpose, we revisit the 3-descent techniques of Schaefer and Stoll [27].

More precisely, let $T = (x_T, y_T) \in E/\langle R\rangle$ be a point of order 3, necessarily $\mathbb{F}_{p^2}$-rational, and let $g_T(x, y) = y - (\lambda x + \mu)$ be the function defining the tangent line at T. Then Costello et al. [7, §3.3] observe that $U \in [3](E/\langle R\rangle)$ if and only if $g_T(U)$ is a cube in $\mathbb{F}_{p^2}$ for all non-trivial 3-torsion points $T \in (E/\langle R\rangle)[3]$ (and similarly for $g_T(V)$). This method is more complicated due to the fact that 3-torsion points are not readily available on $E/\langle R\rangle$. As such, Costello et al. [7] first find a point of order 3 (and potentially immediately find U), and only afterwards apply the 3-descent techniques. Moreover, since only a single 3-torsion point is found (as opposed to all of $(E/\langle R\rangle)[3]$), a slightly weaker check is performed. It was shown by Zanon et al. [30, §4] that this does not lead to better results than naïve cofactor multiplications.

Explicit 3-Descent and the Reduced Tate Pairing. We begin our analysis by relating the 3-descent techniques to the reduced Tate pairing $\tau_{3^{e_3}}$. That is, we note that for any $T \in (E/\langle R\rangle)[3]$ we have that $g_T(U)$ is a cube in $\mathbb{F}_{p^2}$ if and only if $g_T(U)^{(p^2-1)/3} = 1$. We observe that

$$g_T(U)^{(p^2-1)/3} = \tau_{3^{e_3}}(T, U)\,,$$

which is easily seen by observing that the only non-trivial Miller line function is the first one, which equals $g_T(x, y)$. By properties of the Tate pairing, it follows that $\tau_{3^{e_3}}(T, U) = 1$ if and only if $[3^{e_3-1}2^{e_2}]U \in \langle T\rangle$. In particular, if $U \in [3](E/\langle R\rangle)$, then $[3^{e_3-1}2^{e_2}]U = \mathcal{O}$. Thus all pairings are trivial and we recover the statement from Costello et al. (i. e. $g_T(U)$ is a cube for all 3-torsion points T).

As $\#(E/\langle R\rangle)[3] = 9$, this still leaves many pairings to be computed to test whether $U \in [3](E/\langle R\rangle)$. We can simplify the treatment by fixing a basis for $(E/\langle R\rangle)[3^{e_3}]$. Let $\overline{S}, \overline{T} \in E/\langle R\rangle$ such that $(E/\langle R\rangle)[3^{e_3}] = \langle \overline{S}, \overline{T}\rangle$, and let $S = [3^{e_3-1}]\overline{S}$ and $T = [3^{e_3-1}]\overline{T}$ form a basis for $(E/\langle R\rangle)[3]$. Then it follows by bilinearity of $\tau_{3^{e_3}}$ that

$$U \in [3](E/\langle R\rangle) \iff \tau_{3^{e_3}}(S, U) = \tau_{3^{e_3}}(T, U) = 1\,,$$

leaving only 2 pairings to be computed. Although this is a good start, we can do a lot more.

For that purpose, we define $h_0 = \tau_{3^{e_3}}(T, \overline{S})$ and note that $h_0 = \tau_{3^{e_3}}(S, \overline{T})^{-1}$. Then there exist (unique) $a, b \in \mathbb{Z}/3^{e_3}\mathbb{Z}$ such that $[2^{e_2}]U = [a]\overline{S} + [b]\overline{T}$, while

$$\tau_{3^{e_3}}(S, U)^{2^{e_2}} = h_0^{-b}\,, \quad \tau_{3^{e_3}}(T, U)^{2^{e_2}} = h_0^{a}\,.$$

As h_0 has order 3 and 2^{e_2} is invertible modulo 3, these discrete logarithms can easily be solved to retrieve $a, b \bmod 3$. Hence, we can compute $[3^{e_3-1}2^{e_2}]U = [a \bmod 3]S + [b \bmod 3]T$ at the cost of a single point addition (or, by simply selecting it from a pre-computed table). In practice we can ignore the factor $2^{e_2} \bmod 3$, since it only changes a and b up to a simultaneous factor, while it is enough to compute any generator of $\langle [3^{e_3-1}2^{e_2}]U \rangle$ as opposed to finding $[3^{e_3-1}2^{e_2}]U$ itself.

We can repeat the above by (deterministically) generating $U \in E/\langle R \rangle$ until not both $a = 0$ and $b = 0$, in which case $U \in E/\langle R \rangle \setminus [3](E/\langle R \rangle)$. Once that is done, we repeatedly (and deterministically) generate V until

$$\tau_{3^{e_3}}([3^{e_3-1}2^{e_2}]U, V) \neq 1\,,$$

which implies that $[3^{e_3-1}2^{e_2}]V \notin \langle [3^{e_3-1}2^{e_2}]U \rangle$, and in turn that $(E/\langle R \rangle)[3^{e_3}] = \langle [2^{e_2}]U, [2^{e_2}]V \rangle$. Under the assumption that $\overline{S}$, $\overline{T}$, S, T and h_0 are all precomputed, the cost of generating U is determined by the cost of the 2 pairings, while the generation of V requires a single pairing. As before, the first needs to be repeated 9/8 times on average, while the latter (cheaper) step needs to be repeated 3/2 times on average.

The main drawback of this method is that $\overline{S}$, $\overline{T}$ form a basis of $(E/\langle R \rangle)[3^{e_3}]$, so to compute a basis we assume to already know one. In fact we do *not* know such a basis on $E/\langle R \rangle$, seemingly making this much less interesting (which is the exact problem that Costello et al. [7] faced). However, by evaluating $\widehat{\phi}_2$ on U and V *before* checking that they are independent (i. e. multiply to independent 3^{e_3}-torsion points) and have the right order, we can apply the above to the public parameter E where we *can* precompute as much as we want. This allows us to check the independence and orders of $\widehat{\phi}_2(U)$ and $\widehat{\phi}_2(V)$ on E much more efficiently than via naïve cofactor multiplication. For completeness, we provide a full description of the method.

1. Deterministically generate x_U for a point $U \in E/\langle R \rangle$.
2. Compute $\widehat{f}_2(x_U)$ and recover $\widehat{\phi}_2(U)$.
3. For $h_0 := \tau_{3^{e_3}}([3^{e_3-1}]P_3, Q_3)$, compute $a, b \in \mathbb{Z}/3\mathbb{Z}$ such that

$$\tau_{3^{e_3}}([3^{e_3-1}]P_3, \widehat{\phi}_2(U)) = h_0^{b}\,, \quad \tau_{3^{e_3}}([3^{e_3-1}]Q_3, \widehat{\phi}_2(U)) = h_0^{-a}\,,$$

   and repeat 1–3 until not both a and b are zero.
4. Deterministically generate x_V for a point $V \in E/\langle R \rangle$.
5. Compute $\widehat{f}_2(x_V)$ and recover $\widehat{\phi}_2(V)$.
6. Repeat 4–6 until $\tau_{3^{e_3}}([a \cdot 3^{e_3-1}]P_3 + [b \cdot 3^{e_3-1}]Q_3, \widehat{\phi}_2(V)) \neq 1$.
7. Deterministically generate x_{U-V} and compute $\widehat{f}_2(x_{U-V})$.
8. Modify signs of $\widehat{\phi}_2(U)$, $\widehat{\phi}_2(V)$ so that $x_{\widehat{\phi}_2(U)-\widehat{\phi}_2(V)} = \widehat{f}_2(x_{U-V})$.

As both P_3 and Q_3 are public parameters, the points $[3^{e_3-1}]P_3$ and $[3^{e_3-1}]Q_3$ can be precomputed and the above sequence of steps does not involve any scalar multiplication on E or $E/\langle R\rangle$. Although the improvement is obvious by comparing the number of required field operations, we simply confirm the feasibility of our approach through our implementation in Table 1, leading to a speedup of about 17% across the different parameter sets. Note that by including the `iso` operation, we also count the overhead generated by evaluating $\widehat{\phi}_2$ as opposed to ϕ_2. That is, Table 1 shows that checking independence and orders on E not only makes up for this slowdown, but even leads to a speedup. This showcases the utility of the methods even before we arrive at the main optimization (i. e. the pairings, see Sect. 5).

Table 1. Performance benchmarks (rounded to 10^3 cycles) on a 3.4 GHz Intel Core i7-6700 (Skylake) processor, for the isogeny (`iso`) + basis generation (`basis`) operation for $\ell = 2$. The columns labeled `comp` denote the results from SIKE, and the columns labeled `dual` denote our results. Cycle counts are averaged over 10 000 iterations.

	p434		p503		p610		p751	
	comp	dual	comp	dual	comp	dual	comp	dual
iso + basis	9 649	**7 921**	13 332	**11 039**	24 238	**20 269**	37 294	**30 922**

Remark 8. The points U and V (resp. $\widehat{\phi}_2(U)$ and $\widehat{\phi}_2(V)$) that are generated are not a basis for the 3^{e_3}-torsion, as they do not have order 3^{e_3}. Instead, we should use the points $U_3 = [2^{e_2}]U$ and $V_3 = [2^{e_2}]V$ (resp. $[2^{e_2}]\widehat{\phi}_2(U_3)$ and $[2^{e_2}]\widehat{\phi}_2(V_3)$), and by doing so would generate the exact same basis as in the SIKE proposal [17]. However, as noted by Zanon et al. [30, §3.1] in the context of the entangled basis for $\ell = 3$, the cofactors 2^{e_2} naturally factor out during the pairing and discrete logarithm phase and thus do not need to be performed explicitly.

Remark 9. For simplicity the focus of this section is limited to the SIKE parameters where $\ell = 2$ and $m = 3$. However, at no point is any restriction on m made (except not being equal to ℓ), so the above works equally well for any other odd prime m.

5 Pairing Computation

We now turn to the pairing, which is the phase of the compression algorithm on which the use of the dual isogeny has the largest effect. Recall that the reason for computing pairings of the images $\phi_\ell(P_m)$ and $\phi_\ell(Q_m)$ with respect to the deterministically generated basis points U_m and V_m is that in this way, we can transfer the discrete logarithm problems that yield the basis decomposition to the finite field $\mathbb{F}_{p^2}$. They are then solved in the multiplicative group $\mu_{m^{e_m}}$ of m^{e_m}-th roots of unity instead of in the elliptic curve group on the co-domain.

This is more efficient, even including the pairing computation, than solving the discrete logarithm problems on the elliptic curve because field operations are much more efficient and it is possible to precompute large tables of powers of a fixed basis in $\mu_{m^{e_m}}$, as described in [30]. Still, the pairings constitute the main bottleneck of the compression and we discuss how to significantly reduce their computational cost.

5.1 Pulling Back Pairing Arguments

First, recall that we fix a generator g_0 of the group of m^{e_m}-th roots of unity (and the base for the discrete logarithms) as

$$g_0 := \tau_{m^{e_m}}(\phi_\ell(P_m), \phi_\ell(Q_m)) = \tau_{m^{e_m}}(P_m, Q_m)^{\ell^{e_\ell}}.$$

As noted in [30], g_0 can be precomputed via the latter pairing, which only depends on system parameters. We aim to find $c_0, d_0, c_1, d_1 \in \mathbb{Z}/m^{e_m}\mathbb{Z}$ such that

$$g_1 = g_0^{d_0}\,, \qquad g_2 = g_0^{d_1}\,, \qquad g_3 = g_0^{-c_0}\,, \qquad g_4 = g_0^{-c_1}\,,$$

where the g_i are computed as the four pairing values

$$\begin{aligned} g_1 &= \tau_{m^{e_m}}(\phi_\ell(P_m), U_m)\,, & g_2 &= \tau_{m^{e_m}}(\phi_\ell(P_m), V_m)\,, \\ g_3 &= \tau_{m^{e_m}}(\phi_\ell(Q_m), U_m)\,, & g_4 &= \tau_{m^{e_m}}(\phi_\ell(Q_m), V_m)\,. \end{aligned}$$

Utilizing the dual isogeny $\widehat{\phi}_\ell$ and the torsion basis generation algorithms from the previous section, we compute the pairings with the points $\widehat{\phi}_\ell(U_m)$ and $\widehat{\phi}_\ell(V_m)$ on E instead. That is, the Tate pairing satisfies the property $\tau_m(\phi(S), T) = \tau_m(S, \widehat{\phi}(T))$ as stated in Sect. 2, so that the g_i can be computed as

$$\begin{aligned} g_1 &= \tau_{m^{e_m}}(P_m, \widehat{\phi}_\ell(U_m))\,, & g_2 &= \tau_{m^{e_m}}(P_m, \widehat{\phi}_\ell(V_m))\,, \\ g_3 &= \tau_{m^{e_m}}(Q_m, \widehat{\phi}_\ell(U_m))\,, & g_4 &= \tau_{m^{e_m}}(Q_m, \widehat{\phi}_\ell(V_m))\,. \end{aligned}$$

This has the great advantage that the first arguments of all pairings are now fixed torsion basis points on the starting curve.

To see why this is useful, we consider Miller's algorithm [21] for computing pairings, which consists of a loop that carries out a scalar multiplication in a double-and-add fashion of the first pairing argument. On the way, it evaluates and accumulates corresponding line functions at the second argument via a square-and-multiply approach. It was first noted by Scott [28] and further discussed by Costello and Stebila [10] that all information depending on the fixed first argument can be precomputed and stored in a lookup table. This includes all required multiples of the first argument as well as the coefficients of the corresponding line functions. The online phase of the Miller loop consequently only needs to evaluate line functions at the second argument and accumulate them. In particular, this setting thus favors the use of affine coordinates because all inversions for computing the line slopes for point doublings and additions are done as a precomputation and the line functions take a very simple form for affine coordinates. We return to this in Sects. 5.3 and 5.4.

5.2 Special Curves and Torsion Bases for SIKE

From now on we restrict the discussion to the specific setting of SIKE. In particular, we make use of the special starting curve with $A = 6$ that is used in the SIKE proposal.

Let $\ell = 2$ and $m = 3$. Then we are concerned with computing the Tate pairing $\tau_{3^{e_3}}$ with either P_3 or Q_3 as the first argument. This is a special case since there exists a 2-isogeny $\chi : E_0 \to E_6$, while the endomorphism ring of E_0 contains the distortion map $\psi : (x, y) \mapsto (-x, iy)$. As such, there exists a point $P \in E_0(\mathbb{F}_p)[3^{e_3}]$ (any such non-trivial point suffices) such that $E_0[3^{e_3}] = \langle P, \psi(P)\rangle$, i. e. there exists a *distortion basis*, and we set up P_3 and Q_3 such that $P_3 = \chi(P)$ and $Q_3 = \chi\psi(P)$. Finally, by duality of the Tate pairing, we observe that

$$g_1 = \tau_{m^{e_m}}(P, \widehat{\chi}\widehat{\phi}_\ell(U_m)), \qquad g_3 = \tau_{m^{e_m}}(\psi(P), \widehat{\chi}\widehat{\phi}_\ell(U_m)),$$
$$g_2 = \tau_{m^{e_m}}(P, \widehat{\chi}\widehat{\phi}_\ell(V_m)), \qquad g_4 = \tau_{m^{e_m}}(\psi(P), \widehat{\chi}\widehat{\phi}_\ell(V_m)).$$

Hence, by applying an extra (dual of a) 2-isogeny (see Sect. 3.3) we can assume the first arguments to compose a distortion basis. The choice of this basis does not matter, and we simply set $P = [2^{e_2}](x_0, y_0)$ where $x_0 \in \mathbb{F}_p$ is the smallest (positive) integer such that P has order 3^{e_3}. As E_0 is in short Weierstrass form, we can immediately compute with affine Weierstrass coordinates.

The situation is slightly different for $\ell = 3$ and $m = 2$. It is not immediately obvious how to map to E_0 since it is *not* 3-isogenous to E_6. Also, even if we could, it is not possible to pick a distortion basis for $E_0(\mathbb{F}_{p^2})[2^{e_2}]$ according to [8, Lemma 1]. Instead, we map to the Weierstrass curve $E_{a,b} : y^2 = x^3 + ax + b$ over $\mathbb{F}_p$ where $a = -11$ and $b = 14$, which is isomorphic to E_6 via the isomorphism $E_6 \to E_{a,b} : (x, y) \mapsto (x + 2, y)$. Since $E_6(\mathbb{F}_p)[2^{e_2}] \cong \mathbb{Z}/2^{e_2-1}\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z}$, the best we can do is to pick $P_2 \in E_{a,b}(\mathbb{F}_{p^2})[2^{e_2}]$ such that $[2]P_2 \in E_{a,b}(\mathbb{F}_p)$. The second basis point $Q_2 \in E_{a,b}(\mathbb{F}_{p^2})$ can be chosen such that $[2]Q_2 = (x, iy)$, where $x, y \in \mathbb{F}_p$ [9, §3.1].

By setting up the curves and torsion bases this way, the pairings in both the 2^{e_2}- and the 3^{e_3}-torsion groups can be improved by making use of the fact that all operations depending on the first argument are essentially operations in $\mathbb{F}_p$. Furthermore, the distortion basis for the 3^{e_3}-torsion group ensures that pairings with first argument Q_3 can use the same pre-computed table as those with first argument P_3. We explain how this works in detail for both cases.

5.3 Precomputation and the Miller Loop for $\ell = 3$

For $\ell = 3$ we compute order-2^{e_2} pairings of the form $\tau_{2^{e_2}}(P, U)$, meaning that the Miller loop consists of only doubling steps. Recall that for any point $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_{p^2})$ with $y_1 \neq 0$ its double is given by $[2]P = (x_2, y_2)$, where $x_2 = \lambda_1^2 - 2x_1$, $y_2 = \lambda_1(x_1 - x_2) - y_1$ and $\lambda_1 = (3x_1^2 + a)/(2y_1)$. A Miller doubling step with running point P and pairing value $f \in \mathbb{F}_{p^2}$ then updates f

by computing $f \leftarrow f^2 \cdot g/v$, where the tangent g and vertical line v, evaluated at the second pairing argument $U = (x_U, y_U) \in E_{a,b}(\mathbb{F}_{p^2})$, are given as

$$g = \lambda_1(x_U - x_1) + y_1 - y_U, \;\; v = x_U - x_2\,.$$

Hence, we can precompute all doublings of the first pairing argument, and store the point coefficients and the slopes used in the doubling formulas. We obtain two tables in the specific setting using the basis points P_2 and Q_2 fixed above as follows.

For P_2 we simply create the table where

$$T_{P_2}[j] = [x_{j+1}, y_{j+1}, \lambda_j]\;, \quad \text{for } j = 0, \ldots, e_2 - 2\,,$$

denoting $(x_j, y_j, \lambda_j) = (x_{[2^j]P_2}, y_{[2^j]P_2}, (3x_j^2 + a)/(2y_j))$. Since P_2 has order 2^{e_2} and we only carry out $e_2 - 1$ doublings, all doubling operations are well-defined. Note that by the choice of P_2 all point coordinates x_{j+1} and y_{j+1} are in $\mathbb{F}_p$, as are the slopes computed from them, except for the first slope $\lambda_1 \in \mathbb{F}_{p^2} \setminus \mathbb{F}_p$. Therefore, there are exactly $e_2 - 1$ triples and hence $3 \cdot (e_2 - 1)$ field elements in the table. There is an additional $\mathbb{F}_p$ element due to the first slope being an $\mathbb{F}_{p^2}$ element, but the last triple contains a point of order 2 which has y-coordinate 0 and does not have to be stored, keeping the overall element count at $3 \cdot (e_2 - 1)$.

The precomputed table for the point Q_2 is computed similarly. The only difference is that the multiples of Q_2 have the form (x, iy) with $x, y \in \mathbb{F}_p$ instead of being fully defined over $\mathbb{F}_p$. Since all multiples have this form, we can just store x and y and take care of the factor i when computing the line functions in the online phase of the algorithm. The same holds for the slope, as $(3x^2 + a)/(2iy) = -i \cdot (3x^2 + a)/(2y)$. Thus the table T_{Q_2} is defined analogously as

$$T_{Q_2}[j] = [w_{j+1}, z_{j+1}, \kappa_j]\;, \quad \text{for } j = 0, \ldots, e_2 - 2\,,$$

writing $(w_j, z_j, \kappa_j) = (x_{[2^j]Q_2}, y_{[2^j]Q_2}, (3w_j^2 + a)/(2z_j))$. Again, this table consists of $e_2 - 1$ triples of $\mathbb{F}_p$ elements, except for the first slope, which is in $\mathbb{F}_{p^2}$ and the last y-coordinate, which is 0. So the table stores $3 \cdot (e_2 - 1)$ elements in $\mathbb{F}_p$, and the total table size for storing the precomputed values needed to compute the four $\tau_{2^{e_2}}$ pairings is $6 \cdot (e_2 - 1)$ $\mathbb{F}_p$-elements.

The first Miller iteration for each of the two functions can be computed using $2 \cdot (3\mathbf{M} + \mathbf{S} + 8\mathbf{a}) \approx 2 \cdot (11\mathbf{m} + 26\mathbf{a})$, after which a single Miller iteration can be computed in $2 \cdot (2\mathbf{M} + 1\mathbf{S} + 2\mathbf{m} + 6\mathbf{a}) \approx 2 \cdot (10\mathbf{m} + 19\mathbf{a})$. For all four pairings (we assume $\mathbf{m} \approx \mathbf{s}$), this amounts to $40\mathbf{m} + 76\mathbf{a}$ for each Miller iteration except the first (which is slightly more expensive). For comparison, Zanon et al. [30] state the cost $55\mathbf{m} + 126\mathbf{a}$ for only two pairings, or $110\mathbf{m} + 252\mathbf{a}$ for all four. For a complete description of the algorithms we refer to the full version of the paper [23, Algorithms 1–2].

5.4 Precomputation and the Miller Loop for $\ell = 2$

For $\ell = 2$ we compute order-3^{e_3} pairings of the form $\tau_{3^{e_3}}(P, U)$, meaning that the Miller loop consists of only tripling steps instead. Again, for any point $P = (x_1, y_1) \in E_{a,b}(\mathbb{F}_{p^2})$ with $y_1 \neq 0$ its double is given by $[2]P = (x_2, y_2)$ as before, where $\lambda_1 = 3x_1^2/(2y_1)$ (note that here $a = 0$). If $x_2 \neq x_1$, its triple $[3]P = (x_3, y_3)$ is given by $x_3 = \lambda_2^2 - x_2 - x_1$, $y_3 = \lambda_2(x_1 - x_3) - y_1$ and $\lambda_2 = (y_2 - y_1)/(x_2 - x_1)$. A Miller tripling step with running point P and pairing value $f \in \mathbb{F}_{p^2}$ then updates f by computing $f \leftarrow f^3 \cdot g/v$, where g and v are now quadratic functions evaluated at $U = (x_U, y_U)$ given by

$$g = (\lambda_1(x_U - x_1) + y_1 - y_U)(\lambda_2(x_U - x_1) + y_1 - y_U)\,,$$
$$v = (x_U - x_2)(x_U - x_3)\,.$$

To compute g, we can precompute λ_1, λ_2, $n_1 = y_1 - \lambda_1 x_1$ and $n_2 = y_1 - \lambda_2 x_1$, so that $g = (\lambda_1 x_U + n_1 - y_U)(\lambda_2 x_U + n_2 - y_U)$. As for the function v, we expand it to $v = x_U^2 - (x_2 + x_3)x_U + x_2 x_3$. Now we precompute $x_{2p3} = x_2 + x_3$ and $x_{23} = x_2 x_3$ and on input of U at the beginning of the loop also $x_{U,2} = x_U^2$. Then $v = x_{U,2} + x_{23} - x_{2p3} x_U$.

Now let $P \in E_0(\mathbb{F}_p)$ be the point of order 3^{e_3} such that $\{P, \psi(P)\}$ is the distortion basis of $E_0[3^{e_3}]$. We denote

$$(x_2^{(j)}, y_2^{(j)}) = [2 \cdot 3^j]P\,, \quad (x_3^{(j)}, y_3^{(j)}) = [3^{j+1}]P\,, \quad \text{for } j = 0, \ldots e_3 - 2\,,$$

and define $x_3^{(-1)} = x_1$ and $y_3^{(-1)} = y_1$. Then we define the table T_P by

$$T_P[j] = \left[\lambda_1^{(j)}, \lambda_2^{(j)}, n_1^{(j)}, n_2^{(j)}, x_{2p3}^{(j)}, x_{23}^{(j)}\right]\,,$$

where

$$\lambda_1^{(j)} = 3(x_3^{(j-1)})^2/(2y_3^{(j-1)})\,, \qquad \lambda_2^{(j)} = (y_2^{(j)} - y_3^{(j-1)})/(x_2^{(j)} - x_3^{(j-1)})\,,$$
$$n_1^{(j)} = y_3^{(j-1)} - \lambda_1 x_3^{(j-1)}\,, \qquad n_2^{(j)} = y_3^{(j-1)} - \lambda_2 x_3^{(j-1)}\,,$$
$$x_{2p3}^{(j)} = x_2^{(j)} + x_3^{(j)}\,, \qquad x_{23}^{(j)} = x_2^{(j)} \cdot x_3^{(j)}\,.$$

For the last iteration of the Miller loop we append the four extra values $x_3^{(e_3-2)}$, $y_3^{(e_3-2)}$, $\lambda_1^{(e_3-1)}$ and $x_2^{(e_3-1)}$. The second point in the distortion basis has the form $\psi(P) = (x_1, iy_1)$. This means that the functions g and v for pairings with $\psi(P)$ as the first argument are

$$g = (-i\lambda_1 x_U + in_1 - y_U)(-i\lambda_2 x_U + in_2 - y_U)\,,$$
$$v = x_{U,2} - x_{23} + x_{2p3} x_U\,.$$

As a result, the same precomputed values can be used for those pairings without changes. The different signs and factors of i can be adjusted in the online phase of the pairing. The overall table size for all four $\tau_{3^{e_3}}$ pairings is thus $6 \cdot (e_3 - 1) + 4$ elements in $\mathbb{F}_p$.

A single Miller iteration can be computed in $2 \cdot (8\mathbf{M} + 2\mathbf{s} + 6\mathbf{m} + 18\mathbf{a}) \approx 68\mathbf{m} + 128\mathbf{a}$ for all four pairings (we assume $\mathbf{m} \approx \mathbf{s}$). For comparison, Zanon et al. [30] list $104\mathbf{m} + 2\mathbf{s} + 266\mathbf{a}$ for only two pairings. For a complete description of the algorithm we refer to the full version of the paper [23, Algorithm 3].

The Final Exponentiation. The final exponentiation raises all four pairing values to the power $(p^2 - 1)/m^{e_m}$. This is done as usual and as described in [7]. It is split up into the easy part, i.e. the power $p - 1$, which is computed by one application of the Frobenius endomorphism and one inversion per pairing value. Here, inversions are pushed down to the subfield $\mathbb{F}_p$ and shared using Montgomery's inversion sharing trick. The hard part of the final exponentiation is raising to the power $(p + 1)/m^{e_m}$. As $p + 1 = \ell^{e_\ell} \cdot m^{e_m}$, this is performed through a sequence of e_ℓ cyclotomic powerings by ℓ (e. g. squarings for $\ell = 2$ and cubings for $\ell = 3$).

Remark 10. We obtain significant speedups during the pairing computation as the use of the dual isogeny allows us to fix the first pairing arguments as system parameters, which benefits us for two reasons. Firstly, it allows us to pick the basis points P_m and Q_m of special form, either chosen as a distortion basis for $m = 3$ or as a basis such that the coefficients of (multiples of) $[2]P_m$ are in $\mathbb{F}_p$ and the coefficients of (multiples of) Q_m are of the form (x, iy) for $x, y \in \mathbb{F}_p$ for $m = 2$. In this case, most point doublings, triplings and line functions can be computed with arithmetic in $\mathbb{F}_p$ instead of $\mathbb{F}_{p^2}$. When using a distortion basis, all four pairings (as opposed to only two) share many of these computations.

Secondly, having fixed system parameters enables large precomputations. Although it leads to very significant speedups, it does have an impact on the memory usage. If, instead, one chooses to not use precomputation to keep the memory footprint of the implementation small, the special characteristics of the bases still lead to a reasonable speedup. Simply sharing operations across all four pairings for $m = 3$ and replacing general $\mathbb{F}_{p^2}$ operations by subfield operations can be implemented in the pairing algorithms as they are described by Zanon et al. in [30, §5] using extended Jacobian coordinates and by moving back to the starting curve. Operation counts predict savings of roughly 30% for the four Tate pairings of order 2^{e_2} and about 40% for the Tate pairings of order 3^{e_3}.

6 Implementation Results

We have added all of our techniques to the software library that is part of the SIKE proposal [17], so consider the set of primes $p \in \{\texttt{p434}, \texttt{p503}, \texttt{p610}, \texttt{p751}\}$, where

$$\texttt{p434} = 2^{216} \cdot 3^{137} - 1\,, \qquad \texttt{p503} = 2^{250} \cdot 3^{159} - 1\,,$$
$$\texttt{p610} = 2^{305} \cdot 3^{192} - 1\,, \qquad \texttt{p751} = 2^{372} \cdot 3^{239} - 1\,,$$

targeting the different security levels specified by NIST. The software is compiled with clang version 6.0.1 with the `-O` flag, and benchmarked on a 3.4 GHz

Intel Core i7-6700 Skylake processor running Ubuntu version 16.04.3 LTS with TurboBoost turned off. This is the exact same setting that was used for the performance numbers of SIKE Round 2 [18, Table 2.1]. Although we rederive their cycle counts for fairness of comparison, there is indeed a negligible difference (see Table 4).

We distinguish between functions in the SIKE library *without* the use of public-key compression techniques (`SIKEpXXX`), functions in the SIKE library *with* the use of public-key compression (`SIKEpXXX_comp`), and the functions used in our software (`SIKEpXXX_dual`). We begin by comparing the functions related to public-key compression to those in the SIKE library in Table 2, showing that we significantly improve the functions that currently bottleneck the computation, and analyze where the remaining bottlenecks are. We consider the impact on the key generation and exchange functions in the IND-CPA secure SIDH protocol in Table 4a and look at the impact on SIKE in Table 4b.

6.1 Cycle Counts for Compression Functions

In this section we discuss the performance of several functions as they are used in public-key compression. For the results we refer to Table 2.

`iso.` This function takes a secret key as input, and computes the isogeny ϕ_ℓ to obtain the co-domain curve $E/\langle R\rangle$ and potentially the images of basis points. The original compression techniques evaluate ϕ_ℓ at three (x-coordinates of) points, while we do not need to evaluate any points for $\ell = 2$. This leads to a speedup of 18–19% for $\ell = 2$. For $\ell = 3$ we evaluate ϕ_3 at $[2^{e_2-1}]Q_2$ to obtain the intermediate kernels for the dual, leading to a speedup of only 10–11%.

`basis.` This function starts where `iso` left off, and outputs U_m and V_m or $\widehat{\phi}_\ell(U_m)$ and $\widehat{\phi}_\ell(V_m)$, respectively. For $\ell = 2$ we apply the techniques described in Sect. 4.3, leading also to a speedup of 13–15%. For $\ell = 3$ the basis generation does not change significantly (as described in Sect. 4.2), while there is the added overhead of applying $\widehat{\phi}_\ell$. This leads to a slowdown for this function of 151–186%. However, since basis generation for $\ell = 3$ contributed only 4% of the total cost, this is much less bad in absolute terms.

`pair.` We see that the pairing computation significantly bottlenecked compression for $\ell = 3$, while also being the most expensive operation for $\ell = 2$. Applying the results from Sect. 5 leads to a speedup of at least 66% for $\ell = 2$, and a speedup of 62–63% for $\ell = 3$. This has an impressive impact on the efficiency of the full algorithm.

`dlog.` We have not made any changes to the discrete logarithm computations.

`decomp.` Decompression is slightly sped up due to simplifications to x-only basis generation in Sect. 4.1 and due to the avoidance of cofactor multiplications of the basis points. We obtain a 14–15% speedup for $\ell = 2$, and a 6–7% speedup for $\ell = 3$.

As a result of the improvements, we note that the pairing phase is no longer a bottleneck for public-key compression. For $\ell = 2$ it is actually significantly

cheaper than basis generation, while for $\ell = 3$ it is only moderately more expensive than the basis generation and discrete logarithm phases.

Table 2. Performance benchmarks (rounded to 10^3 cycles) on a 3.4 GHz Intel Core i7-6700 (Skylake) processor, for the compression operations: co-domain generation (`iso`), basis generation (`basis`), pairing computation (`pair`), discrete logarithm computation (`dlog`) and decompression (`decomp`). Cycle counts are averaged over 10 000 iterations.

	ℓ	iso	basis	pair	dlog	decomp
SIKEp434_comp	2	5 811	3 838	5 821	923	2 549
SIKEp434_dual	2	**4 690**	**3 231**	**1 954**	**923**	**1 910**
SIKEp434_comp	3	6 464	598	4 921	1 222	1 890
SIKEp434_dual	3	**5 750**	**1 618**	**1 821**	**1 223**	**1 741**
SIKEp503_comp	2	8 141	5 191	8 033	1 556	3 513
SIKEp503_dual	2	**6 594**	**4 445**	**2 676**	**1 554**	**2 613**
SIKEp503_comp	3	9 015	844	6 716	1 532	2 551
SIKEp503_dual	3	**7 992**	**2 219**	**2 486**	**1 535**	**2 380**
SIKEp610_comp	2	15 430	8 808	13 458	2 351	5 868
SIKEp610_dual	2	**12 778**	**7 491**	**4 525**	**2 349**	**4 403**
SIKEp610_comp	3	15 490	1 340	11 365	2 685	4 365
SIKEp610_dual	3	**13 747**	**3 750**	**4 214**	**2 685**	**4 039**
SIKEp751_comp	2	23 133	14 161	21 908	3 529	9 434
SIKEp751_dual	2	**18 898**	**12 024**	**7 348**	**3 528**	**7 135**
SIKEp751_comp	3	26 133	2 125	18 224	5 030	6 914
SIKEp751_dual	3	**23 316**	**6 081**	**6 727**	**5 055**	**6 489**

6.2 Impact on SIDH and SIKE

Finally, we summarize the impact of improved public-key compression when included in a cryptographic protocol. The schemes that are of interest for this purpose are the passively secure SIDH protocol, and its actively secure variant SIKE. Although one can of course argue about the best metric for comparison, we believe the most interesting from an implementers perspective is the overhead that is caused by including public-key compression. This gives a relatively clear idea of the loss of efficiency that is to be paid for a reduction of the size of the public keys.

In Table 4a we see that, across different SIDH parameter sets, for key generation the overhead is reduced from 160–182% to 77–86% for $\ell = 2$ and from 98–104% to 59–61% for $\ell = 3$, respectively. The overhead for the key exchange phase is reduced by about 10–13% in both cases. For SIKE (see Table 4b), we reduce the overhead of (1) key generation from 140–153% to 61–74%, (2) key encapsulation from 67–90% to 38–57%, and (3) decapsulation from 59–65% to 34–39%. Following the SIKE specification [17, Table 2.1], we also provide the

impact on the "total" cost, i. e. on the sum of the costs of encapsulation and decapsulation. This reduces from 62–83% to an overhead of 36–48% across the different parameter sets.

Memory Constraints. Having remarked on the memory usage before, we provide some more detail here. The first notable consequence of our techniques is that we need to build a table containing the kernels of all intermediate ℓ-isogenies appearing in the decomposition of $\widehat{\phi}_\ell$. For $\ell = 2$, and assuming that e_2 is even for simplicity (if not the difference is very minor), we compute a sequence of $e_2/2$ 4-isogenies. For each such isogeny we store 5 elements (see Sect. 3.2), resulting in a table of $5 \cdot e_2/2$ $\mathbb{F}_{p^2}$-elements or simply $5 \cdot e_2$ elements of $\mathbb{F}_p$. For $\ell = 3$, for each 3-isogeny we simply store a generator of the kernel of its dual, requiring 2 elements of $\mathbb{F}_{p^2}$. Hence we store a table containing $4 \cdot e_3$ $\mathbb{F}_p$-elements. Note that these are not precomputed, but need to be temporarily stored on the stack. However, recall from Sect. 5 that we do precompute a table of $6 \cdot (e_2 - 1)$ elements in $\mathbb{F}_p$ for $\ell = 3$ and $6 \cdot (e_3 - 1) + 4)$ for $\ell = 2$ to compute the pairings, i. e. in contrast to the intermediate kernel information, pairing tables are precomputed public parameters.

To aid the discrete logarithm computation, Zanon et al. [30] introduced the use of large precomputed tables. For some fixed window w_3, the discrete logarithms for $\ell = 2$ use a table containing $e_3/w_3 \cdot 3^{w_3}$ or $2 \cdot \lceil e_3/w_3 \rceil \cdot 3^{w_3}$ elements in $\mathbb{F}_{p^2}$ when $w_3 \mid e_3$ or $w_3 \nmid e_3$, respectively. Similarly, the discrete logarithms for $\ell = 3$ use a table of size $e_2/w_2 \cdot 2^{w_2}$ resp. $\lceil e_2/w_2 \rceil \cdot 2^{w_2+1}$ when $w_2 \mid e_2$ resp. $w_2 \nmid e_2$, for some window size w_2. Though small windows of course lead to relatively small tables, for SIKE we always have $4 \leq w \leq 6$ and the current SIKE submission contains very large tables for the discrete logarithms. We summarize memory requirements in terms of the number of field elements in $\mathbb{F}_p$ for the different parameters in Table 3.

Table 3. Required memory in $\mathbb{F}_p$-elements for storing intermediate information used for computing the dual isogeny (`iso`) and for the precomputed tables for the pairing (`pair`) and discrete logarithm computation (`dlog`).

	ℓ	iso	pair	dlog	ℓ	iso	pair	dlog
SIKEp434_dual	2	1 080	820	26 244	3	548	1 290	1 728
SIKEp503_dual		1 250	952	30 132		636	1 494	3 200
SIKEp610_dual		1 525	1 150	46 656		768	1 824	3 904
SIKEp751_dual		1 860	1 432	45 684		956	2 226	2 976

Section 2.3 of the SIKE specification points out that due to the large tables, the current compression method cannot be used in a straightforward manner on constrained devices. Our methods add another possibility for a time-memory trade-off. As pointed out earlier in Remark 10, the choice of special bases

Table 4. Performance benchmarks (rounded to 10^3 cycles) on a 3.4 GHz Intel Core i7-6700 (Skylake) processor. Cycle counts are averaged over 10 000 iterations. The label `oh` denotes the cpu overhead over the corresponding uncompressed version of the function.

(a) The SIDH operations: public key generation ($\texttt{isogen}_2$ and $\texttt{isogen}_3$) and key exchange ($\texttt{isoex}_2$ and $\texttt{isoex}_3$).

	$\texttt{isogen}_2$		$\texttt{isoex}_2$		$\texttt{isogen}_3$		$\texttt{isoex}_3$	
	cyc	oh	cyc	oh	cyc	oh	cyc	oh
SIKEp434	5 821	–	4 726	–	6 469	–	5 467	–
SIKEp434_comp	16 397	182%	5 425	15%	13 208	104%	6 825	25%
SIKEp434_dual	**10 836**	**86%**	**5 298**	**12%**	**10 412**	**61%**	**6 192**	**13%**
SIKEp503	8 154	–	6 745	–	9 002	–	7 623	–
SIKEp503_comp	22 931	181%	7 582	12%	18 107	101%	9 466	24%
SIKEp503_dual	**15 310**	**88%**	**7 422**	**10%**	**14 270**	**59%**	**8 651**	**13%**
SIKEp610	15 438	–	12 881	–	15 464	–	13 282	–
SIKEp610_comp	40 097	160%	14 458	12%	31 031	101%	16 251	22%
SIKEp610_dual	**27 270**	**77%**	**14 170**	**10%**	**24 527**	**59%**	**14 796**	**11%**
SIKEp751	23 229	–	18 961	–	26 024	–	22 255	–
SIKEp751_comp	62 998	171%	21 517	13%	51 443	98%	27 257	22%
SIKEp751_dual	**41 778**	**80%**	**21 104**	**11%**	**41 298**	**59%**	**24 952**	**12%**

(b) The SIKE operations: public key generation (`KeyGen`), encapsulation (`Encaps`), and decapsulation (`Decaps`).

	Size (B)		KeyGen		Encaps		Decaps	
	pk	ct	cyc	oh	cyc	oh	cyc	oh
SIKEp434	330	346	6 482	–	10 563	–	11 290	–
SIKEp434_comp	196	209	16 397	153 %	20 056	90%	18 622	65%
SIKEp434_dual	196	209	**10 849**	**67%**	**16 600**	**57%**	**15 682**	**39%**
SIKEp503	378	402	9 043	–	14 950	–	15 749	–
SIKEp503_comp	224	248	23 066	155%	27 665	85%	25 646	63%
SIKEp503_dual	224	248	**15 294**	**69%**	**22 875**	**53%**	**21 841**	**39%**
SIKEp610	462	486	15 651	–	28 346	–	28 603	–
SIKEp610_comp	273	297	40 078	156%	47 279	67%	45 536	59%
SIKEp610_dual	273	297	**27 277**	**74%**	**39 238**	**38%**	**38 371**	**34%**
SIKEp751	564	596	26 064	–	42 102	–	45 361	–
SIKEp751_comp	331	363	62 663	140%	78 895	87%	72 924	61%
SIKEp751_dual	331	363	**41 909**	**61%**	**66 096**	**57%**	**62 337**	**37%**

already improves the performance even without precomputation. The precomputed tables can be adjusted in size linearly, where computation of required values can be moved to the online phase. Given that the main bottleneck in both [7] and [30] is clearly the pairing phase, it might be worthwhile to use memory for the pairing tables instead of the discrete logarithm tables and find a more space efficient trade-off than the one currently deployed in the SIKE submission.

Acknowledgements. We thank the anonymous reviewers for their detailed remarks and Paulo S.L.M. Barreto for valuable feedback to improve the paper.

References

1. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key compression for isogeny-based cryptosystems. In: AsiaPKC 2016, pp. 1–10. ACM (2016)
2. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: elliptic-curve points indistinguishable from uniform random strings. In: ACM SIGSAC 2013, pp. 967–980. ACM (2013)
3. Blake, I., Seroussi, G., Smart, N., Cassels, J.W.S.: Advances in Elliptic Curve Cryptography. Cambridge University Press, Cambridge (2005)
4. Bos, J.W., et al.: CRYSTALS - Kyber: a CCA-secure module-lattice-based KEM. In: EuroS&P 2018, pp. 353–367. IEEE (2018)
5. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: an efficient post-quantum commutative group action. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018. LNCS, vol. 11274, pp. 395–427. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-03332-3_15
6. Costello, C., Hisil, H.: A simple and compact algorithm for SIDH with arbitrary degree isogenies. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 303–329. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_11
7. Costello, C., Jao, D., Longa, P., Naehrig, M., Renes, J., Urbanik, D.: Efficient compression of SIDH public keys. In: Coron, J.-S., Nielsen, J.B. (eds.) EUROCRYPT 2017. LNCS, vol. 10210, pp. 679–706. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-56620-7_24
8. Costello, C., Longa, P., Naehrig, M.: Efficient algorithms for supersingular isogeny Diffie-Hellman. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9814, pp. 572–601. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53018-4_21
9. Costello, C., Longa, P., Naehrig, M., Renes, J., Virdia, F.: Improved classical cryptanalysis of the computational supersingular isogeny problem. Cryptology ePrint Archive, Report 2019/298 (2019). https://eprint.iacr.org/2019/298
10. Costello, C., Stebila, D.: Fixed argument pairings. In: Abdalla, M., Barreto, P.S.L.M. (eds.) LATINCRYPT 2010. LNCS, vol. 6212, pp. 92–108. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14712-8_6
11. D'Anvers, J.-P., Karmakar, A., Sinha Roy, S., Vercauteren, F.: Saber: module-LWR based key exchange, CPA-secure encryption and CCA-secure KEM. In: Joux, A., Nitaj, A., Rachidi, T. (eds.) AFRICACRYPT 2018. LNCS, vol. 10831, pp. 282–305. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89339-6_16

12. De Feo, L., Jao, D., Plût, J.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. J. Math. Cryptol. **8**, 209–247 (2014)
13. Fujisaki, E., Okamoto, T.: Secure integration of asymmetric and symmetric encryption schemes. In: Wiener, M. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 537–554. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48405-1_34
14. Galbraith, S.D.: Mathematics of Public Key Cryptography. Cambridge University Press, Cambridge (2012)
15. Galbraith, S.D., Petit, C., Shani, B., Ti, Y.B.: On the security of supersingular isogeny cryptosystems. In: Cheon, J.H., Takagi, T. (eds.) ASIACRYPT 2016. LNCS, vol. 10031, pp. 63–91. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53887-6_3
16. Hofheinz, D., Hövelmanns, K., Kiltz, E.: A modular analysis of the Fujisaki-Okamoto transformation. In: Kalai, Y., Reyzin, L. (eds.) TCC 2017. LNCS, vol. 10677, pp. 341–371. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70500-2_12
17. Jao, D., et al.: SIKE (2019). Submission to round 2 of [24]. http://sike.org
18. Jao, D., et al.: SIKE (2016). Submission to [24]. http://sike.org
19. Jao, D., De Feo, L.: Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In: Yang, B.-Y. (ed.) PQCrypto 2011. LNCS, vol. 7071, pp. 19–34. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25405-5_2
20. Lichtenbaum, S.: Duality theorems for curves over P-adic fields. Inventiones Mathematicae **7**, 120–136 (1969)
21. Miller, V.S.: The Weil pairing, and its efficient calculation. J. Cryptol. **17**(4), 235–261 (2004)
22. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. Math. Comput. **48**(177), 243–264 (1987)
23. Naehrig, M., Renes, J.: Dual isogenies and their application to public-key compression for isogeny-based cryptography. Cryptology ePrint Archive, Report 2019/499 (2019). https://eprint.iacr.org/2019/499
24. National Institute of Standards and Technology: Post-quantum cryptography standardization, December 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization
25. Renes, J.: Computing isogenies between Montgomery curves using the action of (0, 0). In: Lange, T., Steinwandt, R. (eds.) PQCrypto 2018. LNCS, vol. 10786, pp. 229–247. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-79063-3_11
26. Renes, J., Smith, B.: qDSA: small and secure digital signatures with curve-based Diffie–Hellman key pairs. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10625, pp. 273–302. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70697-9_10
27. Schaefer, E., Stoll, M.: How to do a p-descent on an elliptic curve. Trans. Am. Math. Soc. **356**(3), 1209–1231 (2004)
28. Scott, M.: Implementing cryptographic pairings. In: Takagi, T., et al. (eds.) Pairing 2007, pp. 177–196. Springer, Heidelberg (2007)
29. Silverman, J.H.: The Arithmetic of Elliptic Curves. GTM, vol. 106. Springer, New York (2009). https://doi.org/10.1007/978-0-387-09494-6
30. Zanon, G.H.M., Simplicio, M.A., Pereira, G.C.C.F., Doliskani, J., Barreto, P.S.L.M.: Faster key compression for isogeny-based cryptosystems. IEEE Trans. Comput. **68**(5), 688–701 (2019)