Public-key Compression in M-SIDH

Kaizhan Lin[1], Jianming Lin[1], Shiping Cai[1], Weize Wang[1], and Chang-An Zhao✉[1,2]

[1] School of Mathematics, Sun Yat-sen University, Guangzhou 510275, P. R. China
{linkzh5,linjm28,caishp6,wangwz}@mail2.sysu.edu.cn
zhaochan3@mail.sysu.edu.cn
[2] Guangdong Key Laboratory of Information Security, Guangzhou 510006, P. R. China

Abstract. Recently, SIKE was broken by the Castryck-Decru attack in polynomial time. To avoid this attack, Fouotsa et al. proposed a SIDH-like scheme called M-SIDH, which hides the information of auxiliary points. The countermeasure also leads to huge parameter sizes, and correspondingly the public key size is relatively large.

In this paper, we propose compressed M-SIDH, which is reminiscent of compressed SIDH. Compared with SIDH, the isogeny degrees in M-SIDH consist of many factor primes, and thus most of the techniques used in compressed SIDH can not be applied into compressed M-SIDH directly. To overcome this issue, we apply several novel techniques to compress the public key of M-SIDH. We also show that our approach to compress the public key of M-SIDH is valid and prove that compressed M-SIDH is secure as long as M-SIDH is secure. In addition, we present new algorithms to accelerate the performance of public-key compression in M-SIDH.

We provide a proof-of-concept implementation of compressed M-SIDH in SageMath. Experimental results show that our approach fits well with compressed M-SIDH. It should be noted that most techniques proposed in this work could be also utilized into other SIDH-like protocols.

Keywords: M-SIDH · Post-quantum Cryptography · Public-key Compression · SIDH

1 Introduction

Since supersingular isogeny Diffie-Hellman (SIDH) [24] was proposed by Jao et al., isogeny-based cryptosystems are attractive in post-quantum cryptography. As the NIST [1] round 4 finalist, supersingular isogeny key encapsulation (SIKE) [4] is famous for its small public key size.

To make SIDH/SIKE more attractive, a large variety of works target public-key compression in SIDH/SIKE to reduce the public key size. Public-key compression in SIDH was first proposed by Azarderakhsh et al. [5]. The key was further compressed by Costello et al. [15]. There are three main procedures in public-key compression in SIDH: torsion basis generation, pairing computation and discrete logarithm computation. Zanon et al. [42] utilized several techniques to accelerate the implementation significantly. Later, Naehrig et al. [31] adapted the dual isogeny to speed up the performance of pairing computation, while Pereira et al. [33] extended the work of [42] and gave a fast method to generate binary torsion basis. However, most of the techniques require large storage for precomputation. An efficient method to compute discrete logarithms with smaller lookup tables was proposed in [22]. Lin et al. [25] improved the Miller evaluation, making the implementation faster with less storage. Several works [26,32] also managed to compress the key using other approaches.

Recently, Castryck and Decru [10] proposed an efficient attack to break SIDH and SIKE in polynomial time if the endomorphism ring of the starting curve is known. Maino et al. [28] gave a subexponential algorithm to attack SIDH with arbitrary starting curves. Inspired by these two works, Robert [37] presented a deterministic polynomial time attack on SIDH in all cases. The attacks also apply to Séta [17] and B-SIDH [14].

However, not all is lost. All the mentioned attacks entirely rely on the following information:

- the degree of the secret isogeny;
- the torsion point images.

Therefore, the attacks do not apply to a few SIDH-based schemes such the isogeny-based Proof of Knowledge proposed in [16]. Furthermore, one could construct new schemes by hiding either of the above information to avoid the attacks. Moriya managed to hide the degree of the secret isogenies and proposed a new SIDH-like scheme, while Fouosta proposed another scheme, called M-SIDH (Masked torsion points SIDH), to avoid the attacks by masking auxiliary points [30,18,19]. However, to satisfy the desired security, both of SIDH-like schemes require relatively large parameter sizes, resulting in larger public key size compared with that of SIDH. Since the new isogeny degrees consist of many factor primes, the approach to compress the public key of SIDH can not be directly extended to the case of M-SIDH and MD-SIDH. Therefore, how to compress the public key in M-SIDH and MD-SIDH is still an open problem.

In this paper, we give an approach to overcome this problem and propose several new techniques to compress the public key of M-SIDH, whose size is $6\log_2 p$ bits. We summarize our work as follows:

- We propose compressed M-SIDH to compress the public key of M-SIDH. Reminiscent of compressed SIDH/SIKE, our method to compress the key also involves torsion basis generation, pairing computation and discrete logarithm computation. We prove that the problem underlying compressed M-SIDH is the same as that of M-SIDH, and the key size is reduced from $6\log_2 p$ bits to $4\log_2 p$ bits.
- We propose several techniques to enhance the performance of compressed M-SIDH. Firstly, we propose a novel way to generate torsion basis. In particular, to determine whether two points could form a torsion basis we utilize

compressed pairings and Lucas sequences. Secondly, an efficient approach is proposed for discrete logarithm computation. Finally, we utilize the Chinese Remainder Theorem to further compress the public key, reducing the key size to around $3.5 \log_2 p$ bits.
- We give the first instantiation of compressed M-SIDH in SageMath. Experimental results verify the validity of our algorithms.

The rest of this paper is as follows. In Section 2 we recall the reduced Tate pairing, compressed pairings, Lucas sequences, M-SIDH and public-key compression in SIDH/SIKE. Section 3 sketches our approach to compress the public key of M-SIDH and prove that compressed M-SIDH is secure if M-SIDH is secure. In Section 4 we present several novel techniques to compress the public key of M-SIDH efficiently. Section 5 reports our implementation and we conclude in Section 6.

2 Preliminaries

In this section, we first introduce the reduced Tate pairings, compressed pairings and Lucas sequences. Next, we recall M-SIDH. Finally, we review several techniques used in public-key compression in SIDH/SIKE.

2.1 Reduced Tate pairings

Let E be an elliptic curve over the finite field $\mathbb{F}_q$, where q is a power of a prime p. Denote μ_n to be the cyclic group of order n in $\mathbb{F}_q^*$ with $n|q-1$, and $f_{n,R}$ to be a rational function on E satisfying $\mathrm{div}(f_{n,R}) = n(R) - n(\mathcal{O})$, where $R \in E(\mathbb{F}_q)[n]$ and $\mathcal{O}$ is the point at infinity. The reduced Tate pairing [20] is defined as:

$$e_n : E(\mathbb{F}_q)[n] \times E(\mathbb{F}_q)/nE(\mathbb{F}_q) \to \mu_n,$$
$$(R, S) \mapsto f_{n,R}(S)^{\frac{q-1}{n}}.$$

Similar with the Tate pairing [40], the reduced Tate pairing has the following properties:

- Bilinearity: $\forall R, R_1, R_2 \in E(\mathbb{F}_q)[n]$, $\forall S, S_1, S_2 \in E(\mathbb{F}_q)/nE(\mathbb{F}_q)$,
$$e_n(R, S_1 + S_2) = e_n(R, S_1) \cdot e_n(R, S_2),$$
$$e_n(R_1 + R_2, S) = e_n(R_1, S) \cdot e_n(R_2, S).$$
- Non-degeneracy: If $e_n(R, S) = 1$ for all $S \in E(\mathbb{F}_q)/nE(\mathbb{F}_q)$ then $R = \mathcal{O}$, and if $e_n(R, S) = 1$ for all $R \in E(\mathbb{F}_q)[n]$ then $S \in nE(\mathbb{F}_q)$.
- Compatibility with isogenies: Assume $\phi : E \to E'$ is a non-zero isogeny of degree m defined over $\mathbb{F}_q$. For $R \in E(\mathbb{F}_q)[n]$, $S \in E(\mathbb{F}_q)/nE(\mathbb{F}_q)$, $R' \in E'(\mathbb{F}_q)[n]$,
$$e_n(\phi(R), \phi(S)) = e_n(R, S)^m,$$
$$e_n(R', \phi(S)) = e_n(\hat{\phi}(R'), S).$$

2.2 Compressed pairings and Lucas sequences

Compressed pairings were first introduced by Scott et al. [38]. This kind of pairings reduces to the bandwidth of pairing values by taking the trace map. Assume that the elliptic curve is supersingular and it is defined over $\mathbb{F}_{p^2} = \mathbb{F}_p[i]/\langle i^2+1\rangle$ with $p \equiv 3 \bmod 4$[3]. In this case, computing the trace of the pairing value is more efficient than computing the pairing value itself.

The final exponentiation of pairings consists of a raising to the power of $p-1$ and the power of $(p+1)/n$. The former one is an easy part, but the latter requires relatively large computational resources. Thanks to Lucas sequences [36, Section 3.6.3], one could efficiently obtain $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\gamma^z)$ from $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\gamma)$ for any $\gamma \in \mu_{p+1}$ and $z = (z_0 z_1 \cdots z_t)_2 \in \mathbb{N}$, as shown in Algorithm 1. Therefore, this technique can improve the costly part of the final exponentiation.

Algorithm 1 LS: Lucas sequences

Require: $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\gamma)$ with $\gamma \in \mu_{p+1}$, $z = (z_0 z_1 \cdots z_t)_2 \in \mathbb{N}$;
Ensure: $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\gamma^z)$.
```
1: v0 ← 2, v1 ← tr_{F_{p^2}/F_p}(γ), tmp ← v1;
2: for each j ∈ {0, 1, ··· , t} do
3:     if z_j = 1 then
4:         v0 ← v0 · v1, v0 ← v0 − tmp, v1 ← v1^2, v1 ← v1 − 2;
5:     else
6:         v0 ← v0^2, v0 ← v0 − 2, v1 ← v0 · v1, v1 ← v1 − tmp;
7:     end if
8: end for
9: return v0.
```

Lucas sequences have potential to improve the exponentiation in the group μ_{p+1} as well. According to the observation in [38], for any element $\gamma = \gamma_1 + \gamma_2 \cdot i \in \mu_{p+1}$ and $z \in \mathbb{N}$,

$$(\gamma_1 + \gamma_2 \cdot i)^z = \frac{\mathrm{LS}(\gamma, z)}{2} + \frac{\gamma_1 \cdot \mathrm{LS}(\gamma, z) - \mathrm{LS}(\gamma, z-1)}{2\gamma_1^2 - 2} \cdot \gamma_2 \cdot i.$$

Note that when computing $\mathrm{LS}(\gamma,z-1)$, the explicit value of $\mathrm{LS}(\gamma,z)$ is also obtained. When the inverse operation is not costly (for instance one can adapt the binary GCD algorithm) and z is large, utilizing Lucas sequences will improve the performance significantly. The main idea is summarized in Algorithm 2.

[3] Indeed, the techniques proposed in this subsection also works when the elliptic curve is defined over $\mathbb{F}_{q^2}$, where q is a prime power.

Algorithm 2 ELS: Exponentiation using Lucas sequences

Require: $\gamma = \gamma_1 + \gamma_2 \cdot i \in \mu_{p+1}$, $z \in \mathbb{N}_+$;
Ensure: γ^z.
1: $tmp_1 \leftarrow \mathrm{LS}(\gamma, z)$, $tmp_2 \leftarrow \mathrm{LS}(\gamma, z-1)$;
//when computing LS(γ,$z-1$), LS(γ,z) is also obtained
2: $tmp_1 \leftarrow tmp_1/2$, $tmp_2 \leftarrow tmp_2/2$;
3: $tmp_2 \leftarrow \gamma_1 \cdot tmp_1 - tmp_2$, $tmp_2 \leftarrow tmp_2/(\gamma_1^2 - 1)$, $tmp_2 \leftarrow tmp_2 \cdot \gamma_2$;
4: **return** $tmp_1 + tmp_2 \cdot i$.

2.3 M-SIDH

Let $p = 4 \cdot f \cdot \ell_1 \cdot \ell_2 \cdots \ell_t - 1$, where the primes $\ell_1, \ell_2, \cdots, \ell_t$ are the first t odd primes and f is a small cofactor such that p is a prime. Denote $\ell_0 = 2$, $N_A = \ell_0 \cdot \ell_2 \cdots \ell_{t-1}$ and $N_B = \ell_1 \cdot \ell_3 \cdots \ell_t$. Define E_0 be a supersingular curve over $\mathbb{F}_{p^2}$ together with $E_0[N_A] = \langle P_A, Q_A \rangle$ and $E_0[N_B] = \langle P_B, Q_B \rangle$. Similar to the SIDH protocol, M-SIDH proceeds as follows:

- Key Generation: Alice chooses a random integer $s_A \in \mathbb{Z}/N_A\mathbb{Z}$ as her secret key. She computes the point $P_A + [s_A]Q_A$ and constructs the N_A-isogeny ϕ_A with kernel $\langle P_A + [s_A]Q_A \rangle$. Then she evaluates two torsion point images $\phi_A(P_B)$, $\phi_A(Q_B)$ and the image curve E_A. Finally, she transmits the tuple $(E_A, [a]\phi_A(P_B), [a]\phi_A(Q_B))$ to Bob, where $a \in \mu_2(N_B) = \{x \in \mathbb{Z}/N_B\mathbb{Z} | x^2 \equiv 1 \bmod N_B\}$. Similar to Alice, Bob selects a random integer $s_B \in \mathbb{Z}/N_B\mathbb{Z}$ to compute $P_B + [s_B]Q_B$ as the kernel generator of the N_B-isogeny ϕ_B. His public key is $(E_B, [b]\phi_B(P_A), [b]\phi_B(Q_A))$ with $b \in \mu_2(N_A) = \{x \in \mathbb{Z}/N_A\mathbb{Z} | x^2 \equiv 1 \bmod N_A\}$.
- Key Agreement: Alice begins her key agreement phase after receiving Bob's public key. She first checks whether $e_{N_A}([b]\phi_B(P_A), [b]\phi_B(Q_A))$ is equal to $e_{N_A}(P_A, Q_A)^{N_B}$, if not she aborts. Then she computes the point $[b]\phi_B(P_A) + [s_A]([b]\phi_B(Q_A))$ to construct the N_A-isogeny ϕ'_A with kernel $\langle \phi_B(P_A) + [s_A]\phi_B(Q_A) \rangle$ and regards the j-invariant $j(E_{BA})$ of the image curve as her shared key. Analogously, Bob checks whether $e_{N_B}([a]\phi_A(P_B), [a]\phi_A(Q_B))$ is equal to $e_{N_B}(P_B, Q_B)^{N_A}$, if not he aborts. He computes the image curve E_{AB} of the N_B-isogeny ϕ'_B and the shared key $j(E_{AB})$.

The security of M-SIDH relies on the hardness of Problem 1:

Problem 1. Let $N_A = \ell_0\ell_2 \cdots \ell_{t-1}$ and $N_B = \ell_1\ell_3 \cdots \ell_t$ be two smooth prime integers, and f be a small cofactor such that $p = N_A N_B f - 1$ is a prime, which $N_A \approx N_B$. Let $E_0/\mathbb{F}_{p^2}$ be a supersingular elliptic curve such that $\#E_0(\mathbb{F}_{p^2}) = (p+1)^2 = (N_A N_B f)^2$, set $E_0[N_A] = \langle P_A, Q_A \rangle$. Let $\phi : E_0 \to E_B$ be a uniformly random N_A-isogeny and let b be a uniformly random element of $\mu_2(N_A) = \{x \in \mathbb{Z}/N_A\mathbb{Z} | x^2 \equiv 1 \bmod N_A\}$.
Given E_0, P_A, Q_A, E_B, $[b]\phi_B(P_A)$, $[b]\phi_B(Q_A)$, compute ϕ_A.

2.4 Public-key compression in SIDH/SIKE

In this subsection, we briefly review the main techniques utilized in public-key compression in SIDH/SIKE. For simplicity, we only consider how to compress the key $(E_B, \phi_B(P_A), \phi_B(Q_A))$.

The main idea of public-key compression is to implement a deterministic pseudorandom number generator to generate a basis of the N_A-torsion group, and use this basis to linearly represent $\phi_B(P_A)$ and $\phi_B(Q_A)$, i.e.,

$$\begin{bmatrix} \phi_B(P_A) \\ \phi_B(Q_A) \end{bmatrix} = \begin{bmatrix} a_0 & b_0 \\ a_1 & b_1 \end{bmatrix} \begin{bmatrix} U_A \\ V_A \end{bmatrix}. \tag{1}$$

After revealing a_0, a_1, b_0 and b_1, Bob checks whether a_0 is invertible in $\mathbb{Z}/N_A\mathbb{Z}^\times$. If so, Bob sends $(E_B, 0, a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1)$ to Alice. Otherwise, the element b_0 must be invertible in $\mathbb{Z}/N_A\mathbb{Z}^\times$ and Bob transmits $(E_B, 1, b_0^{-1}a_0, b_0^{-1}a_1, b_0^{-1}b_1)$ instead.

Assume that $a_0 \in \mathbb{Z}/N_A\mathbb{Z}^\times$, while the other case is similar. After receiving Bob's public key, Alice could compute the kernel of the isogeny ϕ'_A [15]:

$$\begin{aligned}
\langle \phi_B(P_A) + [s_A]\phi_B(Q_A) \rangle &= \langle [a_0]U_A + [b_0]V_A + [s_A a_1]U_A + [s_A b_0]V_A \rangle \\
&= \langle U_A + [a_0^{-1}b_0]V_A + [s_A a_0^{-1}a_1]U_A + [s_A a_0^{-1}b_0]V_A \rangle \\
&= \langle [1 + s_A(a_0^{-1}a_1)]U_A + [(a_0^{-1}b_0) + s_A(a_0^{-1}b_0)]V_A \rangle.
\end{aligned}$$

Therefore, Alice could complete the key agreement phase without recovering $\phi_A(P_B)$ and $\phi_A(Q_B)$.

It remains how to obtain $a_0^{-1}b_0$, $a_0^{-1}a_1$ and $a_0^{-1}b_1$. Zanon et al. [42] proposed a new technique to speed up the performance. Since $\phi_B(P_A)$ and $\phi_B(Q_A)$ also form a basis of $E_B[N_A]$, they can also linearly represent U_A and V_A, i.e.,

$$\begin{bmatrix} U_A \\ V_A \end{bmatrix} = \begin{bmatrix} c_0 & d_0 \\ c_1 & d_1 \end{bmatrix} \begin{bmatrix} \phi_B(P_A) \\ \phi_B(Q_A) \end{bmatrix}. \tag{2}$$

It is easy to verify that

$$(a_0^{-1}b_0, a_0^{-1}a_1, a_0^{-1}b_1) = (-d_1^{-1}d_0/D, -d_1^{-1}c_1/D, d_1^{-1}c_0/D),$$

where $D = c_0d_1 - c_1d_0 \bmod N_A$. With the help of bilinear pairings,

$$\begin{aligned}
h_0 &= \mathrm{e}_{N_A}\left(\phi_B(P_A), \phi_B(Q_A)\right) = \mathrm{e}_{N_A}(P_A, Q_A)^{N_B}, \\
h_1 &= \mathrm{e}_{N_A}\left(\phi_B(P_A), U_A\right) = \mathrm{e}_{N_A}\left(\phi_B(P_A), c_0\phi_B(P_A) + d_0\phi_B(Q_A)\right) = h_0^{d_0}, \\
h_2 &= \mathrm{e}_{N_A}\left(\phi_B(P_A), V_A\right) = \mathrm{e}_{N_A}\left(\phi_B(P_A), c_1\phi_B(P_A) + d_1\phi_B(Q_A)\right) = h_0^{d_1}, \\
h_3 &= \mathrm{e}_{N_A}\left(\phi_B(Q_A), U_A\right) = \mathrm{e}_{N_A}\left(\phi_B(Q_A), c_0\phi_B(P_A) + d_0\phi_B(Q_A)\right) = h_0^{-c_0}, \\
h_4 &= \mathrm{e}_{N_A}\left(\phi_B(Q_A), V_A\right) = \mathrm{e}_{N_A}\left(\phi_B(Q_A), c_1\phi_B(P_A) + d_1\phi_B(Q_A)\right) = h_0^{-c_1}.
\end{aligned} \tag{3}$$

Note that h_0 only depends on public parameters. Therefore, one can recover c_0, c_1, d_0, d_1 by computing four discrete logarithms of h_1, h_2, h_3, h_4 to the base

h_0 efficiently with precomputed lookup tables [42,22,31,25]. Another approach is to compute only three discrete logarithms of h_1, h_3, h_4 (resp. h_2, h_3, h_4) to the base h_2 (resp. h_1) without precomputation as h_2 (resp. h_1) could not be computed in advance [26].

3 Public-key Compression in M-SIDH

In this section, we sketch our approach to compress the public key of M-SIDH and give Proposition 2 to show that compressed M-SIDH is secure if Problem 1 is hard.

3.1 Setup Modification

Different from the setup in M-SIDH, we make a minor modification of the parameter p, N_A and N_B for compressed M-SIDH. In our implementation, we set the parameter p as

$$p = 4 \cdot \ell_1 \cdot \ell_2 \cdots \ell_{t-1} \cdot \ell_t \cdot \ell_{t+1} - 1,$$

where $\ell_1, \ell_2, \cdots, \ell_t$ are the first t odd primes, while the prime ℓ_{t+1} is slightly larger than ℓ_t such that p is a prime. Correspondingly, define $N_A = \ell_1 \cdot \ell_3 \cdots \ell_t$ and $N_B = \ell_2 \cdot \ell_4 \cdots \ell_{t+1}$.

Clearly, this modification does not affect the hardness of Problem 1. The main reason why we modify the parameters is to compress the public key with the help of the reduced Tate pairing correctly. We will give a more detailed explanation in the following. Another advantage of applying the reduced Tate pairing is that the pairing computation would be more efficient compared to the case when using the Weil pairing [29].

3.2 Our approach to compress the key

Our approach to compress the public key of M-SIDH is reminiscent of public-key compression in SIDH/SIKE. Given a secret N_B-isogeny from E_0 to E_B, a sketch of our approach to compress the key is as follows:

1. Torsion basis generation: Generate $\{U_A, V_A\}$ such that $\langle U_A, V_A \rangle = E_B[N_A]$;
2. Pairing computation: Compute the following four pairings:

$$\begin{aligned} h_1 = e_{N_A}\left(\phi_B(P_A), U_A\right), \ h_2 = & e_{N_A}\left(\phi_B(P_A), V_A\right), \\ h_3 = e_{N_A}\left(\phi_B(Q_A), U_A\right), h_4 = & e_{N_A}\left(\phi_B(Q_A), V_A\right); \end{aligned} \tag{4}$$

3. Discrete logarithm computation: Compute discrete logarithms of h_i, $i = 1, 2, 3, 4$ to the base $h_0 = e_{N_A}(P_A, Q_A)^{N_B}$. Randomly select $b \in \mu_2(N_A)$ and then compute $s_i = b \cdot \log_{h_0}(h_i)$.

The public key is $(E_B, s_1, s_2, s_3, s_4)$. A question raised here is whether Equation (3) is correct in compressed M-SIDH when applying the reduced Tate pairing, because in general we do not have $e_{N_A}(P,P) = 1$ for any $P \in E_B(\mathbb{F}_{p^2})[N_A]$. Now we prove the following lemma to confirm that Equation (3) still holds in this situation.

Lemma 1. *Let E be a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with $p \equiv 3 \bmod 4$. Suppose that N is odd and it divides $p+1$. Then $e_N(P,P) = 1$ for any $P \in E(\mathbb{F}_{p^2})[N]$.*

Proof. Since isogeny graphs for supersingular elliptic curves have the Ramanujan property [34], there exists an isogeny $\psi : E \to E'$ of degree $2^\bullet$, where the elliptic curve $E' : y^2 = x^3 + x$ has j-invariant 1728. Since N is odd, we can deduce that $\psi(P)$ has order N for any $P \in E(\mathbb{F}_{p^2})[N]$. Therefore,

$$e_N(\psi(P), \psi(P)) = e_N(P,P)^{2^\bullet}.$$

This implies that $e_N(P,P) = 1$ for any $P \in E(\mathbb{F}_{p^2})[N]$ if and only if $e_N(P', P') = 1$ for any $P' \in E'(\mathbb{F}_{p^2})[N]$. In the following, we prove that $e_N(P', P') = 1$ for any $P' \in E'(\mathbb{F}_{p^2})[N]$.

From $E'(\mathbb{F}_p) \cong \mathbb{Z}/(p+1)\mathbb{Z}$, we can find a point $P_0 \in E'(\mathbb{F}_p)[N]$ of order N. Since the distortion map

$$\begin{aligned} \iota : E' &\to E', \\ (x,y) &\mapsto (-x, iy). \end{aligned}$$

is an isomorphism of E' such that P_0 and $\iota(P_0)$ are linearly independent. This implies that $\langle P_0, \iota(P_0) \rangle = E'(\mathbb{F}_{p^2})[N]$. Hence, for any P' there exist $r, s \in \mathbb{Z}/N\mathbb{Z}$ such that $P' = [r]P_0 + [s]\iota(P_0)$. As a consequence,

$$\begin{aligned} e_N(P', P') &= e_N([r]P_0 + [s]\iota(P_0), [r]P_0 + [s]\iota(P_0)) \\ &= e_N(P_0, P_0)^{r^2} e_N(P_0, \iota(P_0))^{rs} e_N(\iota(P_0), P_0)^{rs} e_N(\iota(P_0), \iota(P_0))^{s^2} \\ &= e_N(P_0, P_0)^{r^2} e_N(P_0, \iota(P_0))^{rs} e_N(P_0, \hat{\iota}(P_0))^{rs} e_N(P_0, P_0)^{\deg(\iota) s^2} \\ &= e_N(P_0, P_0)^{r^2 + \deg(\iota) s^2} e_N(P_0, \iota(P_0) + \hat{\iota}(P_0))^{rs}. \end{aligned}$$

Since the trace of ι is 0 and $\deg(\iota) = 1$, we have

$$e_N(P', P') = e_N(P_0, P_0)^{r^2+s^2} e_N(P_0, \mathcal{O})^{rs} = e_N(P_0, P_0)^{r^2+s^2}. \tag{5}$$

Note that $P_0 \in E'(\mathbb{F}_p)[N]$ and the final exponentiation is $(p-1) \cdot \frac{p+1}{N}$. Therefore, $e_N(P_0, P_0)$ is equal to 1. It follows from Equation (5) that $e_N(P', P') = 1$ for any $P' \in E'(\mathbb{F}_{p^2})[N]$, i.e., $e_N(P,P) = 1$ for any $P \in E(\mathbb{F}_{p^2})[N]$. This completes the proof. ■

From Lemma 1, it is easy to see that our method to compress the key is valid.

Proposition 1. *One can compress the public key by performing the above procedures.*

Remark 1. In the compressed SIDH protocol, it is impossible that none of h_i is a generator. However, it happens in compressed M-SIDH with small possibility. For example, in Equation (2) the prime ℓ_2 may divide c_0 and d_1, while ℓ_4 may divide d_0 and c_1. This is the reason why Bob needs to compute four discrete logarithms to the base h_0 instead of computing three discrete logarithms to one of h_i. In addition, it is possible that none of s_i is invertible in $\mathbb{Z}/N_A\mathbb{Z}$. Hence, we can not further compress the key by directly applying the technique proposed by Costello et al. [15, Section 6]. In Section 4.3, we will propose a method to overcome this issue, compressing the key size from $4\log_2 p$ bits to around $3.5\log_2 p$ bits.

Remark 2. As mentioned in Section 1, one could utilize dual isogenies to optimize pairing computation [31,25] in compressed SIDH. However, the dual isogeny construction in compressed M-SIDH is much more costly compared to that of compressed SIDH. According to our experiments, directly computing h_1, h_2, h_3 and h_4 in Equation (4) without the dual isogeny technique is more efficient. This is the reason why we do not utilize the dual isogeny technique.

In the following, we show that compressed M-SIDH is secure as long as Problem 1 is hard. Without loss of generality, now we only consider Bob's case, while the other case is similar. Obviously, from the compressed key one can deduce that

$$\begin{bmatrix} [b]\phi_B(P_A) \\ [b]\phi_B(Q_A) \end{bmatrix} = \frac{1}{D} \begin{bmatrix} s_2 & -s_1 \\ s_4 & -s_3 \end{bmatrix} \begin{bmatrix} U_A \\ V_A \end{bmatrix}. \tag{6}$$

where $D = s_1 s_4 - s_2 s_3 \bmod N_A$ and $b \in \mu_2(N_A)$ is unknown. Conversely, given the uncompressed key $(E_B, [b]\phi_B(P_A), [b]\phi_B(Q_A))$ where b is unknown, one can compress it by adapting the above procedures. Therefore, compressed M-SIDH is secure as long as M-SIDH is secure, i.e., Problem 1 is hard.

Proposition 2. *Compressed M-SIDH is secure if Problem 1 is hard.*

4 Optimizations on Compressed M-SIDH

It is easy to see that compressed M-SIDH saves two large scalar multiplications of length $\approx \sqrt{p}$ as M-SIDH does. However, it should be noted that the performance of compressed M-SIDH is still not as efficient as that of M-SIDH because of torsion basis generation, pairing computation and discrete logarithm computation. In this section we will optimize the performance of key compression to close the gap. As before, we only handle Bob's case and Alice could also adapt all the techniques to accelerate the performance.

4.1 Torsion basis generation

Since N_A and N_B are not the power of 2 and 3, torsion basis generation in compressed M-SIDH could not benefit from several techniques such as shared Elligator [42] and 3-descent of elliptic curves [15]. In this subsection we propose a new method to generate $\{U_A, V_A\}$ such that $\langle U_A, V_A \rangle = E_B[N_A]$, while torsion basis generation of the N_B-torsion group of E_A is similar. For simplicity, we abbreviate U_A and V_A to U and V, respectively.

Generating one of the torsion points is relatively easy: we can choose a point of order N_A and then set it as U. After U is successfully generated, we generate another point V such that $\langle U, V\rangle = E_B[N_A]$.

As for the first torsion point, a naive way is to sample a random point $R \in E_B(\mathbb{F}_{p^2})$, and then check whether the order of $[4N_B]R$ is N_A. Here we propose Algorithm 3 to generate U, which is more efficient than the naive approach. We also output $\{U_j | j \in I\}$ with $I = \{j|\ell_j \text{ divides } N_A\}$, which is useful for the generation of the second torsion point V.

Algorithm 3 GenerationU: generate a point of order N_A

Require: $E_B/\mathbb{F}_{p^2}$: a supersingular curve, $I : \{j|\ell_j \text{ divides } N_A\}$;
Ensure: A point $U \in E_B(\mathbb{F}_{p^2})$ of order N_A, $\{U_j|j \in I\}$.

Generate a point $R \in E_B(\mathbb{F}_{p^2})$ using Elligator;
$U \leftarrow [4N_B]R$;
$\{U_j\} \leftarrow$BCM(U, I); // Algorithm 4
$I_U \leftarrow \{j|U_j = \mathcal{O}\}$;
while $I_U \neq \emptyset$ **do**
    Generate a point $R \in E_B(\mathbb{F}_{p^2})$ using Elligator;
    $U' \leftarrow [4N_B]R$;
    $U' \leftarrow [\prod_{j\in I\setminus I_U} \ell_j]U'$;
    $\{U'_j\} \leftarrow$BCM(U', I_U); // Algorithm 4
    for each $j \in \{k|U'_k \neq \mathcal{O}\}$ **do**
        $U \leftarrow U + U'_j$, $U_j \leftarrow U'_j$;
    end for
    $I_U \leftarrow \{j|U'_j = \mathcal{O}\}$;
end while
return U, $\{U_j|j \in I\}$.

The main idea of Algorithm 3 is as follows:

Firstly, we randomly generate a point R using Elligator [8] and set $U = [4N_B]R$.

Next, we use Algorithm 4 to compute $U_j = [N_A/\ell_j]U$, where $j \in I = \{j|\ell_j \text{ divides } N_A\}$. It is easy to see that U_j is a point of order ℓ_j if ℓ_j divides the order of U. Otherwise, U_j is the point at infinity.

Denote $I_U = \{j|U_j = \mathcal{O}\}$. If I_U is not empty, we randomly sample another point R and compute $U' = [4N_B]R$. According to I_U, we compute $U'_j = [N_A/\ell_j]U'$ where $j \in I_U$. If U'_j is not the point at infinity, set $U = U + U'_j$. Finally, let $I_U = \{j|U'_j = \mathcal{O}\}$. We repeat the above progress to generate U' until I_U is empty. As a result, for each $j \in I$ we have $U_j \neq \mathcal{O}$. Therefore, U is a point of order N_A.

Remark 3. The approach to compute U_j is inspired by the public-key validation of CSIDH [11]. The authors check the key by generating a point and then check the order of the point using a divide-and-conquer approach [39]. Although this approach consumes slightly larger memory, it performs more efficient than directly computing each U_j.

Algorithm 4 BCM: Batch cofactor multiplication

Require: U: a point on $E_B[N_A]$, I_U: a subset of $I = \{j|\ell_j \text{ divides } N_A\}$;
Ensure: $\{U_k|k \in I_U\}$, where $U_k = [\prod_{j\in I_U\setminus\{k\}} \ell_j]U$.

$n' \leftarrow \#I_U$;
if $n' = 1$ **then**
  return $\{U\}$;
end if
$m' \leftarrow \lfloor n'/2 \rfloor$;
Divide I_U into two subsets I_1, I_2 such that $\#I_1 = n' - m'$ and $\#I_2 = m'$;
$L_1 \leftarrow \prod_{i\in I_2} \ell_i$, $L_2 \leftarrow \prod_{i\in I_1} \ell_i$;
$left \leftarrow [L_1]U$;
$right \leftarrow [L_2]U$;
$r_1 \leftarrow$BCM(left,I_1);
$r_2 \leftarrow$BCM(right,I_2);
return $r_1 \cup r_2$.

In the following we focus on how to generate another point V such that $\langle U, V\rangle = E_B[N_A]$. A naive approach is to generate V with respect to the above method, and then check if U and V can generate the N_A-torsion group. However, this method is not so practical because the success probability is relatively small. Here we present a more efficient method to generate V thanks to Proposition 3.

Proposition 3. *Assume that U is a point of order $N_A = \ell_1\ell_3\cdots\ell_t$ on E_B, and V a random point on $E_B(\mathbb{F}_{p^2})$. Let $I = \{j|\ell_j \text{ divides } N_A\}$, $U_k = [\prod_{j\in I\setminus\{k\}} \ell_j]U$. Denote by $ord(\gamma)$ the order of γ in μ_{N_A}. Then*

$$ord(e_{N_A}(U, V)) = \prod_{\substack{j\in I \\ e_{\ell_j}(U_j,V)\neq 1}} \ell_j. \tag{7}$$

In particular, $e_{N_A}(U, V)$ is a generator of μ_{N_A} if and only if $\langle U, V\rangle = E_B[N_A]$.

Proof. Let $s_k = \prod\limits_{j\in I\setminus\{k\}} \ell_j$ and $s'_k = s_k^{-1} \bmod \ell_k$. From $U_k = [\prod_{j\in I\setminus\{k\}} \ell_j]U$ we have $U = \sum_{k\in I}[s'_k]U_k$. Utilizing the bilinearity of the reduced Tate pairing,

$$\begin{aligned} &e_{N_A}(U, V) \\ =&e_{N_A}([s'_1]U_1, V) \cdot e_{N_A}([s'_3]U_3, V) \cdots e_{N_A}([s'_t]U_t, V) \\ =&e_{N_A}(U_1, V)^{s'_1} \cdot e_{N_A}(U_3, V)^{s'_3} \cdots e_{N_A}(U_t, V)^{s'_t}. \end{aligned} \tag{8}$$

From [21, Theorem IX.9], we have

$$e_{N_A}(U_k, V) = e_{\ell_k}(U_k, V).$$

Let $V_k = [\prod_{j\in I\setminus\{k\}} \ell_j]V$. Obviously, $e_{\ell_k}(U_k, V) = 1$ if and only if $e_{\ell_k}(U_k, V_k) = 1$.

In the following, we will prove that V_k and U_k are linearly dependent if and only if $e_{\ell_k}(U_k, V_k) = 1$, i.e., $e_{N_A}(U_k, V) = 1$.

We first assume that V_k and U_k are linearly dependent. Then we have

- $V_k = \mathcal{O}$, or
- $V_k \neq \mathcal{O}$, but $V_k \in \langle U_k\rangle$,

and *vice versa*. It follows from Lemma 1 that $e_{\ell_k}(U_k, V_k) = 1$. Conversely, if V_k and U_k are linearly independent, we can easily deduce that $e_{N_A}(U_k, V) \neq 1$ from the non-degeneracy of the reduced Tate pairing. In this case, $e_{N_A}(U_k, V)$ is a generator of the group μ_{ℓ_k}.

It is clear that $e_{N_A}(U_k, V) \neq 1$ if and only if $e_{N_A}(U_k, V)^{s'_k} \neq 1$. According to Equation (8), the order of $e_{N_A}(U, V)$ depends on the order of each $e_{N_A}(U_k, V)$:

$$ord\,(e_{N_A}(U, V)) = \prod_{k\in I} ord\left(e_{N_A}(U_k, V)^{s'_k}\right) = \prod_{k\in I} ord(e_{N_A}(U_k, V)).$$

If $e_{N_A}(U_k, V)$ is not equal to 1, then $e_{N_A}(U, V)$ has order ℓ_k. Otherwise, we know that ℓ_k does not divide the order of $e_{N_A}(U, V)$. Consequently, we have Equation (7).

If $e_{N_A}(U, V)$ is a generator of μ_{N_A}, for each k we have $e_{\ell_k}(U_k, V_k) \neq 1$, thus U_k and V_k are linearly independent. Hence, $\langle U_k, V_k\rangle = E_B[\ell_k]$ for each k. It should be noted that

$$E_B[N_A] \cong E_B[\ell_1] \oplus E_B[\ell_3] \oplus \cdots \oplus E_B[\ell_t]. \tag{9}$$

Therefore, $\langle U, V\rangle = E_B[N_A]$. Suppose that $\langle U, V\rangle = E_B[N_A]$, and now we are going to prove $e_{N_A}(U, V) \in \mu_{N_A}$ is of order N_A. Assume that ℓ_k does not divide the order of $e_{N_A}(U, V) \in \mu_{N_A}$. Then

$$e_{N_A}(U, V)^{N_A/\ell_k} = e_{N_A}([N_A/\ell_k]U, V) = e_{N_A}(U_k, V) = e_{\ell_k}(U_k, V_k) = 1.$$

This induces $\langle U_k, V_k\rangle \cong \mathbb{Z}/\ell_k\mathbb{Z}$. From Equation (9), $\{U, V\}$ is not the torsion basis of $E_B[N_A]$, which completes the proof. ■

Proposition 3 gives an approach to test whether two points could generate the torsion group $E_B[N_A]$ by checking the order of the pairing value in the group μ_{N_A}. One can randomly generate a point $V \in E_B(\mathbb{F}_{p^2})[N_A]$ using Elligator, and compute the order of $e_{N_A}(U, V)$ in μ_{N_A}. Then we have a subset $I_V = \{j_k | e_{\ell_{j_k}}(U_{j_k}, V) = 1\}$ of the set $I = \{j | \ell_j \text{ divides } N_A\}$. Similar to the method to generate the point U, we generate another point $V' \neq V$ and compute:

$$f' = e_{\prod_{j_k\in I_V} \ell_{j_k}}\Big(\sum_{j_k\in I_V} U_{j_k}, \prod_{j\in I\setminus I_V} [\ell_j]V'\Big). \tag{10}$$

After that, we check whether ℓ_{j_k} divides the order of $f' \in \mu_{N_A}$ for each $j_k \in I_V$. If so, set $V = V + V'_{j_k}$, where $V'_{j_k} = [N_A/\ell_{j_k}]V'$. We generate another new point V' and repeat the procedure until the set $I_V = \{j_k | f'_{j_k} = 1\}$ is empty. Finally, we have a point V such that $e_{N_A}(U, V)$ is a generator of μ_{N_A}, then $\langle U, V\rangle = E_B[N_A]$ according to Proposition 3.

It seems that once we would like to generate V, we need to randomly generate a point R on $E(\mathbb{F}_q)$ and then perform a large scalar multiplication $V = [4N_B]R$ such that $ord(V)|N_A$. Fortunately, this large scalar multiplication is not necessary when just computing $ord(e_{N_A}(U, V))$. It is obvious that $4N_B$ and N_A are coprime and therefore,

$$ord(e_{N_A}(U,V)) = ord\Big((e_{N_A}(U,R))^{4N_B}\Big) = ord(e_{N_A}(U,R)).$$

It confirms that we can just randomly generate a point $R \in E(\mathbb{F}_q)$ to compute $ord(e_{N_A}(U, V)) = ord(e_{N_A}(U, R))$. For the same reason we can save the scalar multiplication of V' in Equation (10) as well.

Checking the order of the pairing value is also a costly step. Indeed, the aim of the pairing computation is not to compute the precise pairing value but its order. Here we give a lemma, which allows us to compute compressed pairings to reach the goal.

Lemma 2. *If $\gamma \in \mu_{p+1} = \{x \in \mathbb{F}_p[i]/\langle i^2 + 1\rangle\}$ and $p \equiv 3 \bmod 4$, then $\gamma = 1$ if and only if $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\gamma) = 2$.*

Proof. The necessity is obvious. Now we show the sufficiency. Suppose that $\gamma = \gamma_1 + \gamma_2 \cdot i$. From $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\gamma) = 2$, we have $2\gamma_1 = 2$ and hence $\gamma_1 = 1$. Since $\gamma \in \mu_{p+1}$, $\gamma^{p+1} = \gamma_1^2 + \gamma_2^2 = 1$. It implies that $\gamma_2 = 0$. ■

Therefore, to check the order of the pairing value f', one can first compute $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(f')$, and then utilize Lucas sequences to obtain $tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}\left((f')^{N_A/\ell_k}\right)$ for each $k \in I_V$. Similar to Algorithm 4, we present Algorithm 5 to compute them efficiently.

After that, we check if each of them is equal to 2 or not. Thanks to Lemma 2, we can deduce whether $(f')^{N_A/\ell_k}$ is equal to 1, and thus its order could be determined.

In a nutshell, we present Algorithm 6 to generate V.

Remark 4. During the torsion basis generation, the first batch cofactor multiplication of U in Line 3 of Algorithm 3 and the first pairing computation in Line 2 of Algorithm 6 consume large computational resources. To eliminate these two expensive parts for Alice, Bob could send her the initial I_U (in Line 4 of Algorithm 3) and I_V (in Line 4 of Algorithm 6). They can be translated into two $(t + 1)/2$-bit strings. It would be a trade-off between the compressed key size and efficiency.

Algorithm 5 BCE: Batch cofactor exponentiation

Require: $f' \in tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}(\mu_{N_A})$, I_V: a subset of $I = \{j|\ell_j \text{ divides } N_A\}$;
Ensure: $\{f'_k|k \in I_V\}$, where $f'_k = tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}\left((f'_k)^{\prod_{j \in I_V \setminus \{k\}} \ell_j}\right)$.

```
n' ← #I_V;
if n' = 1 then
    return {f'};
end if
m' ← ⌊n'/2⌋;
Divide I_V into two subsets I_1, I_2 such that #I_1 = n' − m' and #I_2 = m';
L_1 ← ∏_{i∈I_2} ℓ_i, L_2 ← ∏_{i∈I_1} ℓ_i;
left ← LS(f', L_1);                                   // Algorithm 1
right ← LS(f', L_2);                                  // Algorithm 1
r_1 ← BCE(left, I_1);
r_2 ← BCE(right, I_2);
return r_1 ∪ r_2.
```

4.2 Discrete logarithm computation

Different from the case we handle in SIDH, one should compute discrete logarithms in the multiplicative group μ_{N_A}. Since N_A is smooth, one can use the Pohlig-Hellman algorithm [35] to simplify a discrete logarithm in μ_{N_A} to discrete logarithms in the groups μ_{ℓ_j} with $j \in I = \{j|\ell_j \text{ divides } N_A\}$, and finally use the Chinese Remainder Theorem to recombine.

Firstly, we should compute $h_i^{N_A/\ell_j}$ with $j \in I$ and $i = 1, 2, 3, 4$ using a divide-and-conquer approach. Note that this step could be also accelerated with the help of Lucas sequences [38, Section 3], as we proposed in Algorithm 7.

After that, for each $j \in I$ we compute the discrete logarithms of $h_i^{N_A/\ell_j}$ to the base $h_0^{N_A/\ell j}$, where $h_0 = e_{N_A}(P_A, Q_A)^{N_B}$. Since P_A and Q_A are fixed, all the values $h_0^{N_A/\ell j}$ can be precomputed to accelerate the performance. From Equation (2), it is clear that $d_i = \log_{h_0} h_i$, $c_i = -\log_{h_0} h_{i+2}$, $i = 0, 1$. For each $j \in I = \{j|\ell_j \text{ divides } N_A\}$, let $c_i^{(j)} = c_i \bmod \ell_j$, $d_i^{(j)} = d_i \bmod \ell_j$, $i = 0, 1$.

Finally, from $d_i^{(j)}$, $c_i^{(j)}$ with $j \in I$ we respectively recover $d_i = \log_{h_0} h_i$, $c_i = -\log_{h_0} h_{i+2}$, $i = 0, 1$. This step is fast with the help of the Chinese Remainder Theorem.

Algorithm 8 is the pseudocode summarizing our ideas to compute discrete logarithms.

4.3 Further compression

In this subsection we propose an approach to overcome the issue mentioned in Remark 1. The technique further reduces the public key size and simultaneously improve the performance of discrete logarithm computation. We also prove that the modification does not affect the security of compressed M-SIDH.

Algorithm 6 GenerationV: generate a point of order N_A such that $\langle U, V\rangle = E_B[N_A]$

Require: $E_B/\mathbb{F}_{p^2}$: a supersingular curve, $I : \{j|\ell_j \text{ divides } N_A\}$, U and $\{U_k\}$: output of Algorithm 3;

Ensure: A point $V \in E_B(\mathbb{F}_{p^2})$ of order N_A such that $\langle U, V\rangle = E_B[N_A]$.

Generate a point $V \in E_B(\mathbb{F}_{p^2})$ using Elligator;
$f' \leftarrow tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}\left(e_{N_A}(U, V)\right)$;
$\{f'_j\} \leftarrow$BCE(f', I); // Algorithm 5
$I_V \leftarrow \{j_k|f'_{j_k} = 2\}$;
while $I_V \neq \emptyset$ **do**
  Generate a point $V' \in E_B(\mathbb{F}_{p^2})$ using Elligator;
  $U' \leftarrow \sum_{j_k\in I_V} U_{j_k}$, $L \leftarrow \prod_{j_k\in I_V} \ell_{j_k}$;
  $f' \leftarrow tr_{\mathbb{F}_{p^2}/\mathbb{F}_p}\left(e_L(U', V')\right)$;
  $\{f'_{j_k}\} \leftarrow$BCE(f', I_V); // Algorithm 5
  if $f'_{j_k} \neq 2$ for some j_k **then**
    $V' \leftarrow [\prod_{j\in I\setminus I_V} \ell_j]V'$;
    $\{V'_{j_k}\} \leftarrow$BCM(V', I_V); // Algorithm 4
  end if
  for each $j_k \in \{j_k|f'_{j_k} \neq 2\}$ **do**
    $V \leftarrow V + V'_{j_k}$;
  end for
  $I_V \leftarrow \{j_k|f'_{j_k} = 2\}$;
end while
$V \leftarrow [2fN_B]V$;
return V.

As mentioned in Remark 1, none of s_i is invertible in $\mathbb{Z}/N_A\mathbb{Z}$ when none of h_i is a generator of μ_{p+1}. Nevertheless, from Equation (2) we have

$$\begin{bmatrix} U_j \\ V_j \end{bmatrix} = \begin{bmatrix} c_0^{(j)} & d_0^{(j)} \\ c_1^{(j)} & d_1^{(j)} \end{bmatrix} \begin{bmatrix} [N_A/\ell_j]\phi_B\left(P_A\right) \\ [N_A/\ell_j]\phi_B\left(Q_A\right) \end{bmatrix}. \tag{11}$$

where $c_i^{(j)} = c_i \bmod \ell_j$, $d_i^{(j)} = d_i \bmod \ell_j$, $i = 0, 1$ and $j \in I = \{j|\ell_j \text{ divides } N_A\}$. Note that $\langle U, V\rangle = \langle\phi_B(P_A), \phi_B(Q_A)\rangle = E_B[N_A]$ and ℓ_j is prime. Therefore, either $d_0^{(j)}$ or $d_1^{(j)}$ is invertible, i.e., either $h_1^{N_A/\ell_j}$ or $h_2^{N_A/\ell_j}$ is a generator of μ_{ℓ_j}. From this observation, we can compute the discrete logarithms as follows:

Firstly, compute $h_i^{N_A/\ell_j}$ with $j \in I$ and $i = 1, 2, 3, 4$ using a divide-and-conquer approach. This procedure is the same as that of the method presented in Section 4.2.

Secondly, for each $j \in I$ we check whether $h_1^{N_A/\ell_j}$ is the generator of μ_{ℓ_j}. Note that it is equivalent to check whether $h_1^{N_A/\ell_j}$ is equal to 1 since ℓ_j is a prime. If $h_1^{N_A/\ell_j}$ generates μ_{ℓ_j}, compute discrete logarithms of $h_2^{N_A/\ell_j}$, $h_3^{N_A/\ell_j}$, $h_4^{N_A/\ell_j}$ to the base $h_1^{N_A/\ell_j}$. Otherwise, we can deduce that $h_2^{N_A/\ell_j}$ is a generator and then compute discrete logarithms of $h_3^{N_A/\ell_j}$, $h_4^{N_A/\ell_j}$ to the base $h_2^{N_A/\ell_j}$. Suppose that

Algorithm 7 BCEA: Batch cofactor exponentiation in μ_{N_A}

Require: $h' \in \mu_{N_A}$, I': a subset of $I = \{j|\ell_j \text{ divides } N_A\}$;

Ensure: $\{h'_1, h'_2, \cdots, h'_{n'}\}$, where $h'_k = \left((f'_k)^{\prod_{j\in I'\setminus\{k\}} \ell_j}\right)$ and $n' = \#I'$.

if $n' = 1$ **then**
  return $\{h'\}$;
end if
$m' \leftarrow \lfloor n'/2 \rfloor$;
Divide I' into two subsets I_1, I_2 such that $\#I_1 = n' - m'$ and $\#I_2 = m'$;
$L_1 \leftarrow \prod_{i\in I_2} \ell_i$, $L_2 \leftarrow \prod_{i\in I_1} \ell_i$;
$left \leftarrow \text{ELS}(h, L_1)$; // Algorithm 2
$right \rightarrow \text{ELS}(h, L_2)$; // Algorithm 2
$r_1 \leftarrow$BCEA(left,I_1);
$r_2 \leftarrow$BCEA(right,I_2);
return $r_1 \cup r_2$.

Algorithm 8 Discrete logarithm computation

Require: : I: $\{j|\ell_j \text{ divides } N_A\}$; h_1, h_2, h_3, h_4: the values computed in Equation (4);

Ensure: : c_0, c_1, d_0, d_1: Integers in $\{0, 1, \cdots, N_A - 1\}$ such that $h_1 = h_0^{d_0}$, $h_2 = h_0^{d_1}$, $h_3 = h_0^{-c_0}$ and $h_4 = h_0^{-c_1}$.

for $k \in \{1, 2, 3, 4\}$ **do**
  $\{h_k^{(j)}\} \leftarrow \text{BCEA}(h_k, I)$; // Algorithm 7
end for
for $k \in \{1, 2\}$ **do**
  for each $j \in I$ **do**
    find $d_k^{(j)}$ such that $h_k^{(j)} = \left(h_0^{(j)}\right)^{d_k^{(j)}}$, find $c_k^{(j)}$ such that $h_{k+2}^{(j)} = \left(h_0^{(j)}\right)^{-c_k^{(j)}}$;
  end for
  Use the Chinese remainder theorem to compute $d_k \bmod N_A$ and $c_k \bmod N_A$ such that $d_k \equiv d_k^{(j)} \bmod \ell_j$ and $c_k \equiv c_k^{(j)} \bmod \ell_j$ with $j \in I$;
end for
return c_0, c_1, d_0, d_1.

$S_i^{(j)}$ $i = 1, 2, 3$ are the solutions and the label $label_j$ is used to mark whether $h_1^{N_A/\ell_j}$ is the generator. Hence, we have

$$\left(S_1^{(j)}, S_2^{(j)}, S_3^{(j)}, label_j\right) = \begin{cases} \left((d_0^{(j)})^{-1} d_1^{(j)}, -(d_0^{(j)})^{-1} c_0^{(j)}, -(d_0^{(j)})^{-1} c_1^{(j)}, 1\right), \text{ if } d_0^{(j)} \neq 0, \\ \left(1, -(d_1^{(j)})^{-1} c_0^{(j)}, -(d_1^{(j)})^{-1} c_1^{(j)}, 0\right), \text{ otherwise.} \end{cases} \tag{12}$$

Thanks to the Chinese Remainder Theorem, one could obtain $S_i \bmod N_A$ such that $S_i \equiv S_i^{(j)} \bmod \ell_j$ for each $j \in I$.

Using the above method, the public key is $(E_B, S_1, S_2, S_3, label)$, where

$$label = label_1 + label_3 \cdot 2 + \cdots + label_t \cdot 2^{(t-1)/2}. \tag{13}$$

The pesudocode to compute the discrete logarithms is proposed in Algorithm 9.

Algorithm 9 Another approach to compute discrete logarithms

Require: I: $\{j|\ell_j$ divides $N_A\}$; h_1, h_2, h_3, h_4: the values computed in Equation (4);
Ensure: : *label*: A $(t+1)/2$-bit integer defined in Equation (13); S_1, S_2, S_3: Integers in $\{0, 1, \cdots, N_A - 1\}$ defined as above, which satisfy Equation (12).

for $k \in \{1, 2, 3, 4\}$ **do**
  $\{h_k^{(j)}\} \leftarrow \mathrm{BCEA}(h_k, I)$; // Algorithm 7
end for
for each $j \in I$ **do**
  if $h_1^{(j)} \neq 1$ **then**
    for each $k \in \{1, 2, 3\}$ **do**
      find $S_k^{(j)}$ such that $h_{k+1}^{(j)} = (h_1^{(j)})^{S_k^{(j)}}$;
    end for
  else
    $S_1^{(j)} = 1$;
    for each $k \in \{2, 3\}$ **do**
      find $S_k^{(j)}$ such that $h_{k+1}^{(j)} = (h_2^{(j)})^{S_k^{(j)}}$;
    end for
  end if
end for
for each $k \in \{1, 2, 3\}$ **do**
  Use the Chinese remainder theorem to compute $S_k \bmod N_A$ such that $S_k \equiv S_k^{(j)} \bmod \ell_j$ with $j \in I$;
end for
$label \leftarrow \sum_{j \in I} label_j \cdot 2^{(j-1)/2}$;
return $S_1, S_2, S_3, label$.

It is obvious that Bob can compress the public key successfully, but a question raised here is how Alice generates a kernel generator G_A of the group $\langle \phi_B(P_A) + [sk_A]\phi_B(Q_A) \rangle = \langle [d_1 - c_1 \cdot sk_A]U + [-d_0 + c_0 \cdot sk_A]V \rangle$ according to $(E_B, S_1, S_2, S_3, label)$.

Using Algorithms 3 and 6, Alice obtains U and V. Besides, she could construct

$$S_4^{(j)} \equiv 1 \bmod \ell_j \text{ if } label_j = 1, \text{ or } S_4^{(j)} \equiv 0 \bmod \ell_j \text{ otherwise.} \tag{14}$$

Utilizing the Chinese Remainder Theorem, Alice could recover $S_4 \bmod N_A$ such that $S_4 \equiv S_4^{(j)} \bmod \ell_j$ from Equation (14). Let

$$G_A = [S_1 + S_3 \cdot sk_A]U - [S_4 + S_2 \cdot sk_A]V.$$

Now we show that G_A is a kernel generator of $\langle \phi_B(P_A) + [sk_A]\phi_B(Q_A) \rangle$. It is equivalent to show that for each $k \in I$,

$$\langle [N_A/\ell_k]G_A \rangle = \langle [d_1 - c_1 \cdot sk_A]U_k + [-d_0 + c_0 \cdot sk_A]V_k \rangle, \tag{15}$$

where $U_k = [N_A/\ell_k]U$ and $V_k = [N_A/\ell_k]V$. If $label_j = 1$, then $S_4 \equiv 1 \bmod \ell_j$ and hence

$$[N_A/\ell_k]G_A = [S_1 + S_3 \cdot sk_A]U_k - [1 + S_2 \cdot sk_A]V_k.$$

Note that

$$
\begin{aligned}
&[S_1 + S_3 \cdot sk_A]U_k - [1 + S_2 \cdot sk_A]V_k \\
=&[S_1^{(j)} + S_3^{(j)} \cdot sk_A]U_k - [S_1^{(j)} + S_2^{(j)} \cdot sk_A]V_k \\
=&[(d_0^{(j)})^{-1}d_1^{(j)} - (d_0^{(j)})^{-1}c_1^{(j)} \cdot sk_A]U_k - [1 - (d_0^{(j)})^{-1}c_0^{(j)} \cdot sk_A]V_k \\
=&[(d_0^{(j)})^{-1}] \cdot \left([d_1^{(j)} - c_1^{(j)} \cdot sk_A]U_k + [-d_0^{(j)} + c_0^{(j)} \cdot sk_A]V_k\right).
\end{aligned}
$$

In other words, we have

$$[N_A/\ell_k]G_A \in \langle [d_1 - c_1 \cdot sk_A]U_k + [-d_0 + c_0 \cdot sk_A]V_k \rangle$$

when $S_4^{(j)} = 1$. Similarly, we can deduce that $[N_A/\ell_k]G_A$ and $[d_1 - c_1 \cdot sk_A]U_k + [-d_0 + c_0 \cdot sk_A]V_k$ are linearly dependent when $S_4^{(j)} = 0$. Therefore, the point G_A satisfies Equation (15).

Proposition 4. *After applying Algorithm 9 and modifying the compressed key, one can still compress the public key or decompress the compressed key successfully.*

Now we show that the modification we propose in this subsection does not affect the security of compressed M-SIDH. From the compressed public key $(E_B, S_1, S_2, S_3, label)$, we can recover S_4 using the Chinese Remainder Theorem, thus we are able to compute

$$
\begin{aligned}
P'_A &= [S_1]U_A - [S_4]V_A = [b]\phi_B(P_A), \\
Q'_A &= [S_3]U_A - [S_2]V_A = [b]\phi_B(Q_A),
\end{aligned}
$$

where $b \in \mathbb{Z}/N_A\mathbb{Z}^\times$ satisfies

$$
\begin{cases}
bd_0^{(j)} \equiv 1 \bmod \ell_j, \text{if } label_j = 1, \\
bd_1^{(j)} \equiv 1 \bmod \ell_j, \text{if } label_j = 0.
\end{cases}
\tag{16}
$$

On the other hand, it is clear that one can also compress the public key to the compressed key successfully by applying the above procedures. Therefore, the problem underlying the security of compressed M-SIDH is Problem 2.

Problem 2. Let $N_A = \ell_0\ell_2 \cdots \ell_{t-1}$ and $N_B = \ell_1\ell_3 \cdots \ell_t$ be two smooth prime integers, and f be a small cofactor such that $p = N_A N_B f - 1$ is a prime, where $N_A \approx N_B$. Let $E_0/\mathbb{F}_{p^2}$ be a supersingular elliptic curve such that $\#E_0(\mathbb{F}_{p^2}) = (p+1)^2 = (N_A N_B f)^2$. Suppose that $E_0[N_A] = \langle P_A, Q_A \rangle$. Let $E_0 \to E_A$ be a uniformly random N_B-isogeny and let a be a uniformly random element of $\mathbb{Z}/N_A\mathbb{Z}^\times$.
Given E_0, P_A, Q_A, E_B, $[b]\phi_B(P_A)$ and $[b]\phi_B(Q_A)$, compute ϕ_A.

The main difference between Problem 1 and Problem 2 is that the former one has an additional restriction that $b \in \mu_2(N_A)$. It seems that Problem 1 is hard if Problem 2 is hard. Indeed, according to [19, Section 3.1], Problem 2 can be solved as long as Problem 1 is easy. Then we have Proposition 5:

Proposition 5. *After applying Algorithm 9 and modifying the public key, compressed M-SIDH is still secure whenever Problem 1 is hard.*

Compared to the former method in Section 4.2, the new method not only further compresses the key but performs better. The main reason is that the latter method saves at least one discrete logarithm in μ_{ℓ_j} for each $j \in I$. Furthermore, it saves considerable storage for precomputation since there is no need to compute discrete logarithms to the base h_0.

5 Implementation Results

In this section, we implement compressed M-SIDH in SageMath (version 9.5) [2] and give our experimental results.

Isogeny computation is the most expensive part of (compressed) M-SIDH. There are mainly two ways to construct the isogeny. One is the traditional Vélu's formula [41], and the other is a more efficient formula to construct the large degree isogeny [7]. We combine both of them to implement compressed M-SIDH. For small degree isogeny computations we use traditional Vélu's formula, and use the method proposed in [7] to compute the large degree isogeny.

Based on the code[1] from [7], we give a proof-of-concept implementation of compressed M-SIDH in SageMath. Our code is available at

`https://github.com/CompressedMSIDH/CompressedMSIDH`.

Table 1 reports the performance of the key generation phase. For discrete logarithm computation we apply the method proposed in Section 4.3.

Procedure	Alice	Bob
Isogeny Computation	304.67s	305.89s
Torsion Basis Generation	18.00s	18.81s
Pairing Computation	15.75s	15.66s
Discrete Logarithm Computation	5.68s	5.61s
Total Cost (the whole key generation phase)	344.10s	345.97s

Table 1. Experimental results of key generation of Alice in compressed M-SIDH for the NIST-1 level of security.

As shown in Table 1, isogeny computation dominates the cost of key generation. One may try to utilize several techniques proposed in the literature to speed up the compressed M-SIDH implementation. We adapt the technique proposed in [27, Section 5.2] to recover the image coefficient of the isogeny, which offers a significant speedup to isogeny computation. Besides, there are several works on the optimizations of CSIDH [11]. For example, the approach [13] to find an optimal strategy of CSIDH could be easily extended to the isogeny computation

[1] `https://velusqrt.isogeny.org/`

of M-SIDH. It is also possible to improve the performance by changing the permutation of the ℓ_j-isogeny computation [23]. The improvement of large degree isogeny computation is explored by [3].

Torsion basis generation and pairing computation are the efficiency bottlenecks of public-key compression in M-SIDH. The computational cost of discrete logarithm computation is approximately one third of that of torsion basis generation. We leave the exploration of the faster implementation of compressed M-SIDH for future work.

6 Conclusion

In this paper, we proposed compressed M-SIDH, reducing the public key size of M-SIDH from $6\log_2 p$ bits to around $3.5\log_2 p$ bits. We proved that compressed M-SIDH is secure as long as M-SIDH is secure. In addition, several novel techniques were proposed to accelerate the performance. It should be noted that some techniques proposed in this paper have potential to optimize other isogeny-based cryptosystems. For example, our method for torsion basis generation may improve finding full-torsion points in SQALE [12] and dCSIDH [9].

Very recently, Basso et al. [6] proposed new SIDH-like protocols called *binSIDH* and *terSIDH*. Our work could be also extended easily to these SIDH-like schemes. Although the implementation of SIDH-like schemes is not so efficient now because of the huge characteristic of the base field and expensive isogeny computation, we believe that compressed SIDH-like schemes could find their positions with further research.

References

1. The National Institute of Standards and Technology (NIST): Post-quantum cryptography standardization, `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization`
2. The Sage Developers: SageMath, the Sage Mathematics Software System (version 9.5) (2022), `https://sagemath.org`
3. Adj, G., Chi-Domínguez, J.J., Rodríguez-Henríquez, F.: Karatsuba-based square-root Vélu's formulas applied to two isogeny-based protocols. Journal of Cryptographic Engineering (Jul 2022)
4. Azarderakhsh, R., Campagna, M., Costello, C., De Feo, L., Hess, B., Hutchinson, A., Jalali, A., Jao, D., Karabina, K., Koziel, B., LaMacchia, B., Longa, P., Naehrig, M., Pereira, G., Renes, J., Soukharev, V., Urbanik, D.: Supersingular Isogeny Key Encapsulation (2020), `http://sike.org`
5. Azarderakhsh, R., Jao, D., Kalach, K., Koziel, B., Leonardi, C.: Key Compression for Isogeny-Based Cryptosystems. In: Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography. pp. 1–10 (2016)
6. Basso, A., Fouotsa, T.B.: New SIDH Countermeasures for a More Efficient Key Exchange. Cryptology ePrint Archive, Paper 2023/791 (2023), `https://eprint.iacr.org/2023/791`
7. Bernstein, D., Feo, L., Leroux, A., Smith, B.: Faster computation of isogenies of large prime degree. Open Book Series **4**, 39–55 (2020)

8. Bernstein, D.J., Hamburg, M., Krasnova, A., Lange, T.: Elligator: Elliptic-Curve Points Indistinguishable from Uniform Random Strings. In: Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security. p. 967980 (2013)
9. Campos, F., Chavez-Saab, J., Chi-Domínguez, J.J., Meyer, M., Reijnders, K., Rodríguez-Henríquez, F., Schwabe, P., Wiggers, T.: On the Practicality of Post-Quantum TLS Using Large-Parameter CSIDH. Cryptology ePrint Archive, Paper 2023/793 (2023), https://eprint.iacr.org/2023/793
10. Castryck, W., Decru, T.: An Efficient Key Recovery Attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 423–447. Springer Nature Switzerland, Cham (2023)
11. Castryck, W., Lange, T., Martindale, C., Panny, L., Renes, J.: CSIDH: An Efficient Post-Quantum Commutative Group Action. In: Peyrin, T., Galbraith, S. (eds.) Advances in Cryptology – ASIACRYPT 2018. pp. 395–427. Springer International Publishing, Cham (2018)
12. Chávez-Saab, J., Chi-Domínguez, J.J., Jaques, S., Rodríguez-Henríquez, F.: The SQALE of CSIDH: sublinear Vélu quantum-resistant isogeny action with low exponents. Journal of Cryptographic Engineering **12**(3), 349–368 (Sep 2022)
13. Chi-Domínguez, J.J., Rodríguez-Henríquez, F.: Optimal strategies for CSIDH. Advances in Mathematics of Communications **16**(2), 383–411 (2022)
14. Costello, C.: B-SIDH: Supersingular Isogeny Diffie-Hellman Using Twisted Torsion. In: Moriai, S., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2020. pp. 440–463. Springer International Publishing, Cham (2020)
15. 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.) Advances in Cryptology – EUROCRYPT 2017. pp. 679–706. Springer International Publishing, Cham (2017)
16. De Feo, L., Dobson, S., Galbraith, S.D., Zobernig, L.: SIDH Proof ofăKnowledge. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022. pp. 310–339. Springer Nature Switzerland, Cham (2022)
17. De Feo, L., Delpech de Saint Guilhem, C., Fouotsa, T.B., Kutas, P., Leroux, A., Petit, C., Silva, J., Wesolowski, B.: Séta: Supersingular Encryption fromăTorsion Attacks. In: Tibouchi, M., Wang, H. (eds.) Advances in Cryptology – ASIACRYPT 2021. pp. 249–278. Springer International Publishing, Cham (2021)
18. Fouotsa, T.B.: SIDH with masked torsion point images. Cryptology ePrint Archive, Paper 2022/1054 (2022), https://eprint.iacr.org/2022/1054
19. Fouotsa, T.B., Moriya, T., Petit, C.: M-SIDH and MD-SIDH: Countering SIDH Attacks by Masking Information. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 282–309. Springer Nature Switzerland, Cham (2023)
20. Frey, G., Rück, H.G.: A Remark Concerning M-Divisibility and the Discrete Logarithm in the Divisor Class Group of Curves. Math. Comput. **62**(206), 865874 (1994)
21. Galbraith, S.: Pairings, pp. 183–214. London Mathematical Society Lecture Note Series, Cambridge University Press (2005)
22. Hutchinson, A., Karabina, K., Pereira, G.: Memory Optimization Techniques for Computing Discrete Logarithms in Compressed SIKE. In: Cheon, J.H., Tillich, J.P. (eds.) Post-Quantum Cryptography. pp. 296–315. Springer International Publishing, Cham (2021)

23. Hutchinson, A., LeGrow, J., Koziel, B., Azarderakhsh, R.: Further Optimizations of CSIDH: A Systematic Approach to Efficient Strategies, Permutations, and Bound Vectors. In: Conti, M., Zhou, J., Casalicchio, E., Spognardi, A. (eds.) Applied Cryptography and Network Security. pp. 481–501. Springer International Publishing, Cham (2020)
24. Jao, D., De Feo, L.: Towards Quantum-Resistant Cryptosystems from Supersingular Elliptic Curve Isogenies. In: Yang, B.Y. (ed.) Post-Quantum Cryptography. pp. 19–34. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
25. Lin, K., Lin, J., Wang, W., Zhao, C.A.: Faster Public-key Compression of SIDH with Less Memory. Cryptology ePrint Archive, Paper 2021/992 (2021), `https://eprint.iacr.org/2021/992`
26. Lin, K., Wang, W., Wang, L., Zhao, C.A.: An Alternative Approach for Computing Discrete Logarithms in Compressed SIDH. Cryptology ePrint Archive, Paper 2021/1528 (2021), `https://eprint.iacr.org/2021/1528`
27. Lin, K., Wang, W., Xu, Z., Zhao, C.A.: A Faster Software Implementation of SQISign. Cryptology ePrint Archive, Paper 2023/753 (2023), `https://eprint.iacr.org/2023/753`
28. Maino, L., Martindale, C., Panny, L., Pope, G., Wesolowski, B.: A Direct Key Recovery Attack on SIDH. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 448–471. Springer Nature Switzerland, Cham (2023)
29. Miller, V.S.: The Weil Pairing, and Its Efficient Calculation. Journal of Cryptology **17**(4), 235–261 (Sep 2004)
30. Moriya, T.: Masked-degree SIDH. Cryptology ePrint Archive, Paper 2022/1019 (2022), `https://eprint.iacr.org/2022/1019`
31. Naehrig, M., Renes, J.: Dual Isogenies and Their Application to Public-Key Compression for Isogeny-Based Cryptography. In: Galbraith, S.D., Moriai, S. (eds.) Advances in Cryptology – ASIACRYPT 2019. pp. 243–272. Springer International Publishing, Cham (2019)
32. Pereira, G.C.C.F., Barreto, P.S.L.M.: Isogeny-Based Key Compression Without Pairings. In: Garay, J.A. (ed.) Public-Key Cryptography – PKC 2021. pp. 131–154. Springer International Publishing, Cham (2021)
33. Pereira, G.C.C.F., Doliskani, J., Jao, D.: x-only point addition formula and faster compressed SIKE. Journal of Cryptographic Engineering **11**, 57–69 (2021)
34. Pizer, A.K.: Ramanujan graphs and Hecke operators. Bulletin of the American Mathematical Society **23**(1), 127–137 (1990)
35. Pohlig, S., Hellman, M.: An Improved Algorithm for Computing Logarithms over GF(p) and Its Cryptographic Significance (Corresp.). IEEE Trans. Inf. Theor. **24**(1), 106110 (2006)
36. Richard Crandall, C.B.P.: Prime numbers: a computational perspective. Springer, 2nd ed edn. (2005)
37. Robert, D.: Breaking SIDH in Polynomial Time. In: Hazay, C., Stam, M. (eds.) Advances in Cryptology – EUROCRYPT 2023. pp. 472–503. Springer Nature Switzerland, Cham (2023)
38. Scott, M., Barreto, P.S.L.M.: Compressed Pairings. In: Franklin, M. (ed.) Advances in Cryptology – CRYPTO 2004. pp. 140–156. Springer Berlin Heidelberg, Berlin, Heidelberg (2004)
39. Sutherland, A.: Order computations in generic groups. PhD thesis, Massachusetts Institute of Technology (2007)
40. Tate, J.: WC-groups over p-adic fields. Exposé no. 156. In Années 1956/57 - 1957/58, exposés 137-168, volume 4 of Séminaire Bourbaki p. 265277 (1956-1958)

41. Vélu, J.: Isogénies entre courbes elliptiques. Comptes Rendus Hebdomadaires des Séances de l'Académie des Sciences, Série A **273**, 238–241 (1971)
42. 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 Transactions on Computers **68**(5), 688–701 (2019)