PQDH: A Quantum-Safe Replacement for Diffie-Hellman based on SIDH

Vladimir Soukharev[1] and Basil Hess[2]

[1] InfoSec Global, Toronto, Ontario, Canada
Vladimir.Soukharev@infosecglobal.com
[2] InfoSec Global, Zurich, Switzerland
Basil.Hess@infosecglobal.com

Abstract. We present a post-quantum key agreement scheme that does not require distinguishing between the initiator and the responder. This scheme is based on elliptic curve isogenies and can be viewed as a variant of the well-known SIDH protocol. Then, we present an efficient countermeasure against a side-channel attack that applies to both static and ephemeral versions of SIDH and our scheme. Finally, we show how to obtain an isogeny-based password-authenticated key exchange protocol based on our scheme by applying a construction shown in [16]. Security and computational complexities summaries are also presented.

Keywords: isogenies, key agreement, side-channel attack, countermeasure, password-authenticated key exchange, post-quantum cryptography, elliptic curves, cryptography, authentication

1 Introduction

Recently, NIST has started the first post-quantum cryptography standardization process [12]. Cryptographers around the world have submitted in total 82 proposals, of which 69 were accepted for the first round, then 26 were selected for the second round. The research and cryptanalysis in this area have been drawing more and more researchers from academia and industry. One goal of the competition is to find a quantum-safe alternative to today's widely used key agreement schemes: DH and ECDH. There are many different candidates in the competition, but all of them have one property in common - they differ from DH-like schemes. The quantum-safe candidates for key establishment are called Key Encapsulation Mechanisms (KEM), where one party generates a shared secret, encrypts it and transmits it to the other party. In such a case a user of the KEM needs to know who initiates the exchange and who responds to it. A portion of those KEMs are based on key agreement schemes, which all still have the property that the initiator and the responder must be distinguished. In DH-like schemes, both parties contribute to the shared secret and from the user perspective, there is no difference between the initiator and the responder. This might seem like a small difference, however, from an adoption and migration point of view, it will cause difficulties.

Among the NIST candidates, there is one that resembles DH-like schemes more than the others: SIKE [10]. SIKE is a KEM, which is based on SIDH [9, 7] where both parties contribute to the shared secret in an equal manner. However, even though SIDH is the scheme that resembles DH-type schemes, it still requires to distinguish between the initiator and the responder. In this paper, we improve upon SIDH and propose one of the first post-quantum key agreement schemes, PQDH (Post-Quantum Diffie-Hellman), that behaves exactly as DH-type schemes and thus allows an easier migration. The only other known scheme that has this property is CSIDH (commutative SIDH) [5], which is also based on supersingular isogenies, but conceptually different from SIDH.

Furthermore, we look at other use cases of key agreement schemes in the industry where the NIST candidates do not offer a direct solution. Specifically, we looked at password-authenticated key exchange (PAKE) schemes which are heavily used in the payment industry. We demonstrate how PQDH can be used to construct a post-quantum PAKE scheme by using the construction methods presented in [16].

As we move towards such systems, we need to be especially careful with regards to side-channel attacks. Although side-channel attacks are a general concern, in payment systems they are even a bigger threat. Recently, a fault attack [17] has been published applicable to isogeny-based schemes, like SIDH and PQDH. In this paper, we demonstrate an efficient countermeasure against these fault attacks, which can be applied not only to PQDH but to all isogeny-based schemes.

The paper is structured as follows. In Section 2, we highlight our main contributions. In Section 3, we provide an overview of isogeny-based cryptography. In Section 4 we present PQDH followed by the side-channel attack countermeasures in Section 5. Post-quantum PAKE based on PQDH is presented in Section 6. The evaluation results are shown in Section 7, and we conclude in Section 8. A list of explicit algorithms can be found in Appendix A.

2 Main Contributions

In this work, we present our three main contributions to this field. First, we demonstrate an improved evolution of the SIDH scheme, which we named PQDH (Post-Quantum Diffie-Hellman) and which can be used as a direct drop-in replacement for the conventional Diffie-Hellman-type schemes. Second, we contribute an efficient countermeasure against a side-channel fault attack on isogeny-based schemes, which applies to both static and ephemeral versions. Finally, we present a PAKE (password-authenticated key exchange) scheme based on PQDH.

2.1 Post-Quantum Diffie-Hellman

SIDH is one of the most promising candidates for a quantum-resistant key agreement scheme. We focus on the ephemeral version as the most frequently used in practice, but it is possible to extend the results to the static version. We have found that although the SIDH scheme resembles the flow of a Diffie-Hellman protocol, it lacks compatibility with the existing IT infrastructure, which is a characteristic shared among all post-quantum key agreement schemes submitted to the NIST PQC standardization. The lack of such compatibility occurs because the computational actions of the initiator and the responder in a communication session differ from each other, making it problematic to use SIDH with the existing cryptographic APIs, as we need to constantly distinguish between the two types of users. One option would be to send extra information about the basis used or the user type. However, this is not an option as we still would need to change the APIs and users are still performing different actions. We found a way to overcome the obstacle of differentiating between the initiator and the responder and will present a scheme, named PQDH, which is fully based on SIDH, but works with the existing Diffie-Hellman APIs.

2.2 Fault Attack Countermeasure

A cryptographic scheme may be secure from the theoretical point of view, however, there is more to consider to ensure full protection. The attacker could be listening, monitoring, and/or capturing radio-frequency waves, electricity consumption, or other emissions or behaviour data of the computational device performing cryptographic operations. The obtained information and its analysis can be used to recover part of or the entire secret key. These kinds of attacks are known as side-channel attacks. There are different categories of such attacks, including simple side-channel attacks, differential side-channel attacks, fault attacks, and others. If a protocol is vulnerable to such attacks, then either software or hardware countermeasures need to be implemented. Given that we are living in an era where software is expected to work on multiple platforms, the software countermeasures are the preferred option to provide the desired flexibility.

Fault attacks are categorized as active attacks and occur when the attacker either modifies the input data for cryptographic computations or inserts some faults on purpose. The attacker can then make use of the resulting computation with faults to recover some private information.

For isogeny-based schemes, one possible attack is a fault attack that enables one to compute an isogeny value for a random point. The attack is unique as it is currently the only known side-channel and fault attack specific to isogeny-based cryptosystems that could be applied to both static and ephemeral versions of the schemes, while all the other known attacks apply only to static versions of the schemes. Thus, switching to an ephemeral version will not help to prevent this attack. Hence, a countermeasure for this attack is needed in any case. The attack has been discovered by Ti [17].

In general, a secret isogeny can be computed and return the resulting values only for the other party's basis points. Otherwise, if we know the isogeny value for any other points, which lie outside the corresponding torsion subgroup, we can, with high probability, recover the secret isogeny. The attacker tries to get the party A to compute their isogeny

ϕ_A for points from $E[\ell_A^{e_A}]$. This is done by providing a random point instead of a proper point from $E[\ell_B^{e_B}]$. The random point can be decomposed with respect to all the basis, and the $E[\ell_A^{e_A}]$ related point can be isolated, using scalar multiplication of order $f \cdot \ell_B^{e_B}$. Having obtained this information, with high probability, the secret isogeny can be recovered. As here, the parties A and B are generic, we can symmetrically switch all A's and B's in our description.

2.3 Password-Authenticated Key Exchange

Password-Authenticated Key Exchange (PAKE) is a protocol where parties establish a common cryptographic key based on the knowledge of the common password. In practice, it is a key agreement with an added password. The password should be computationally infeasible to guess. The password is also used to mask the public key information sent over an insecure channel.

The first PAKE protocol was designed by Bellovin and Merritt in 1992 [1]. One of its main advantages is allowing the usage of low-entropy passwords. Today, many protocols of this type exist. Most of them are based on either a multiplicative group $\mathbb{Z}_p^*$ or on a group of points of elliptic curves. There are only lattice-based and isogeny-based schemes that are post-quantum PAKE-type protocols [8, 19] and [16]. The first isogeny-based PAKE protocol was proposed by Taraskin et al. [16], which is based on SIDH. We propose a PAKE protocol that builds on top of PQDH, using the same construction as the one by Taraskin et al. Given that PQDH truly resembles Diffie-Hellman-type schemes, building other protocols from it makes them less complex. The main advantage of our protocol is that, once again, both parties are performing the same actions and we do not need to distinguish between the initiator and the responder. It is roughly two times slower, hence a trade-off between the speed and convenience of use in today's infrastructure.

2.4 Applications

As a key agreement scheme, PQDH has many possible applications. Today, most commonly used key agreements are Diffie-Hellman-type schemes based either on a multiplicative group of integers modulo a prime or on an elliptic curve group (ECDH). The exposure to side-channel attacks leads to the need for countermeasures implemented in an efficient way to avoid performance trade-offs.

One of the most natural applications of PAKE is establishing a common session secret key for mutual authentication between a smartcard and a terminal that provides PIN entry. Another highly important application is using it in the IEEE 802.11 WLAN standard. There are several other applications of PAKE, including its usage in client-server networks.

3 Background

In this section, we provide a brief overview of post-quantum elliptic-curve cryptography, also often referred to as "isogeny-based cryptography".

3.1 Isogenies

We provide a brief background review on isogenies between elliptic curves. For further details on the mathematical foundations of isogenies, we refer the reader to [9, 14].

Given two elliptic curves E_1 and E_2 defined over finite field $\mathbb{F}_q$ of size q, an *isogeny* ϕ is an algebraic morphism from E_1 to E_2 of the form

$$\phi(x, y) = \frac{f_1(x, y)}{g_1(x, y)}, \frac{f_2(x, y)}{g_2(x, y)}\Bigg)\Bigg(,$$

such that $\phi(\infty) = \infty$ (here f_1, f_2, g_1, g_2 are polynomials in two variables, and ∞ denotes the identity element on an elliptic curve). Isogeny is an algebraic morphism which is a group homomorphism. The degree of a given isogeny ϕ, $\deg(\phi)$, is its degree as an algebraic morphism.

Given an isogeny $\phi\colon E_1 \to E_2$ of degree n, there exists a related isogeny $\hat{\phi}\colon E_2 \to E_1$ of the same degree, which is called the *dual isogeny*. Two elliptic curves are called *isogenous* if there exists an isogeny between them.

Let n be any natural number; we define $E[n]$ to be the n-torsion subgroup of E, namely

$$E[n] = \{P \in E(\bar{\mathbb{F}}_q) : nP = \infty\}.$$

Thus, $E[n]$ is the kernel of the multiplication by n map over the algebraic closure $\bar{\mathbb{F}}_q$ of $\mathbb{F}_q$. It is important to note that the group $E[n]$ is isomorphic to $(\mathbb{Z}/n\mathbb{Z})^2$ as a group whenever n and q are relatively prime [14].

We define $\mathrm{End}(E)$ to be the *endomorphism ring*, which is a set of all isogenies from an elliptic curve E to itself, defined over the algebraic closure $\bar{\mathbb{F}}_q$ of $\mathbb{F}_q$. $\mathrm{End}(E)$ is a ring under the operations of pointwise addition and functional composition. If $\dim_{\mathbb{Z}}(\mathrm{End}(E)) = 2$, then we say that the curve E is *ordinary*; otherwise, $\dim_{\mathbb{Z}}(\mathrm{End}(E)) = 4$, in which case we say that the curve E is *supersingular*. An important property relating isogenies and types of curves is that two isogenous curves are either both ordinary or both supersingular. All elliptic curves used in this work are supersingular. The isogeny $\phi\colon E_1 \to E_2$ is *separable* if the extension $\mathbb{F}_q(E_1)/\phi^*(\mathbb{F}_q(E_2))$ is separable. We will only consider separable isogenies. The size of the kernel of that isogeny is equal to the degree of that isogeny [14, III.4.10(c)].

Up to isomorphism, an isogeny is uniquely defined by the kernel. Various methods, complementing each other, for computing and evaluating isogenies are given in [4, 9, 11, 18]. We use the isogenies whose kernels are cyclic groups. Knowledge of the kernel, or any single generator of the kernel, allows us to perform an efficient evaluation of the isogeny (up to isomorphism). Conversely, the ability to evaluate the isogeny via a black box allows for efficient determination of the kernel. Thus, in our application, the following are equivalent:

- knowledge of the isogeny,
- knowledge of the kernel,
- knowledge of any generator of the kernel.

3.2 Isogeny-Based Key Agreement

The term ECC (elliptic-curve cryptography) typically refers to cryptographic primitives and protocols whose security is based on the hardness of the discrete logarithm problem on elliptic curves. This hardness assumption is invalid against quantum computers [13]. Hence, traditional elliptic-curve cryptography is not a viable foundation for constructing quantum-resistant cryptosystems. As a result, alternative elliptic-curve cryptosystems based on hardness assumptions other than discrete logarithms have been proposed for use in settings where quantum resistance is desired. One early proposal by Stolbunov [15], based on isogenies between ordinary elliptic curves, was subsequently shown by Childs, Jao, and Soukharev [6] to offer only subexponential difficulty against quantum computers. The algorithm has recently been further improved by Bonnetain [2].

In response to these developments, Jao, Plût and De Feo [9] proposed a new collection of quantum-resistant public-key cryptographic protocols for entity authentication, key exchange, and public-key cryptography, based on the difficulty of computing isogenies between supersingular elliptic curves. We review here the most fundamental protocol in the collection - key exchange protocol, which forms the main building block for our proposed schemes.

Fix a prime p of the form

$$p = \ell_A^{e_A} \ell_B^{e_B} \cdot f \pm 1,$$

where ℓ_A and ℓ_B are small primes, e_A and e_B are positive integers, and f is some (typically very small) cofactor. Then, fix a supersingular curve E defined over $\mathbb{F}_{p^2}$, and bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ which generate $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ respectively, so that $\langle P_A, Q_A\rangle = E[\ell_A^{e_A}]$ and $\langle P_B, Q_B\rangle = E[\ell_B^{e_B}]$. Alice chooses two random elements $m_A, n_A \in_R \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$, not both divisible by ℓ_A, and computes an isogeny $\phi_A\colon E \to E_A$ with kernel $K_A := \langle [m_A]P_A + [n_A]Q_A\rangle$. Alice also computes the points $\{\phi_A(P_B), \phi_A(Q_B)\} \subset E_A(\mathbb{F}_{p^2})$ obtained by applying her secret isogeny ϕ_A to the basis $\{P_B, Q_B\}$ for $E[\ell_B^{e_B}]$, which are called auxiliary points, and sends these points to Bob together with E_A. Similarly, Bob selects random elements $m_B, n_B \in_R \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$, not both divisible by ℓ_B, and computes an isogeny $\phi_B\colon E \to E_B$ having kernel $K_B := \langle [m_B]P_B + [n_B]Q_B\rangle$, along with the auxiliary points $\{\phi_B(P_A), \phi_B(Q_A)\}$. Upon receipt of E_B and $\phi_B(P_A), \phi_B(Q_A) \in E_B(\mathbb{F}_{p^2})$ from Bob, Alice computes an isogeny $\phi'_A\colon E_B \to E_{AB}$ having kernel equal to $\langle [m_A]\phi_B(P_A) + [n_A]\phi_B(Q_A)\rangle$; Bob proceeds symmetrically. Alice and Bob can then use the common j-invariant of

$$E_{AB} = \phi'_B(\phi_A(E)) = \phi'_A(\phi_B(E)) = E/\langle [m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B\rangle$$

$\mathcal{A}$		$\mathcal{B}$
Input: A, B, sID		**Input:** B
$m_A, n_A \in_R \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$		$m_B, n_B \in_R \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$
$\phi_A : E \to E_A =$		$\phi_B : E \to E_B =$
$E/\langle [m_A]P_A + [n_A]Q_A \rangle$		$E/\langle [m_B]P_B + [n_B]Q_B \rangle$
	$\xrightarrow{A,\text{sID},\phi_A(P_B),\phi_A(Q_B),E_A}$	
	$\xleftarrow{B,\text{sID},\phi_B(P_A),\phi_B(Q_A),E_B}$	
$E_{AB} :=$		$E_{BA} :=$
$E_B/_{\langle [m_A]\phi_B(P_A)+[n_A]\phi_B(Q_A)\rangle}$		$E_A/_{\langle [m_B]\phi_A(P_B)+[n_B]\phi_A(Q_B)\rangle}$
Output: $j(E_{AB})$, sID		**Output:** $j(E_{BA})$, sID

Fig. 1: Key Agreement protocol using isogenies on supersingular curves.

to form a secret shared key.

The protocol is presented in Figure 1. We denote by A and B the identifiers of Alice and Bob, and use sID to denote the unique session identifier.

It is important to note that Alice and Bob operate on different bases, one set for the initiator and a different one for the responder. As a result, besides bases, different spaces are selected for scalars. Hence, one must always distinguish between the two roles, otherwise, the protocol will fail.

Remark 1. Alice's auxiliary points $\{\phi_A(P_B), \phi_A(Q_B)\}$ allow Bob, or any eavesdropper, to compute Alice's isogeny ϕ_A on any point in $E[\ell_B^{e_B}]$. This ability is necessary for the scheme to function since Bob needs to compute $\phi_A(K_B)$ as part of the scheme. However, Alice must never disclose $\phi_A(P_A)$ or $\phi_A(Q_A)$, or more generally any information that allows an adversary to evaluate ϕ_A on $E[\ell_A^{e_A}]$, since disclosing this information would allow the adversary to solve a system of discrete logarithms in $E[\ell_A^{e_A}]$, which are easy since $E[\ell_A^{e_A}]$ has a smooth order, to recover K_A. The same applies to Bob. The side-channel attack presented in [17] tries to force the user to compute their isogeny on their basis points.

Remark 2. The textbook version of the SIDH protocol assumes that the secret scalars m and n, which are not both divisible by ℓ, where basis points P, Q are of order ℓ^e. In practice, following the explanations in Section 4.2.1 of [9] and Section 4 of [7], we may assume $m = 1$ and compute the kernel point as $P + [n]Q$.

3.3 CSIDH

The only other post-quantum scheme, besides the one presented in this paper, that achieves the property that there is no need to distinguish between the initiator and the responder is CSIDH (commutative SIDH) [5]. This scheme has similar principles as SIDH but works over $\mathbb{F}_p$, which means that the resulting $\mathbb{F}_p$-endomorphism ring is commutative. It is an interesting solution. One thing to note is that due to its commutativity, the same quantum subexponential attack applies as for the ordinary curves [6]. The authors have shown that the proposed scheme provides 64 quantum bit security. Recently, the scheme was further studied in [3], where the authors show that it provides 35 bit of quantum security. Hence, the scheme is worth looking into and continuing the development, but one must be careful with security parameters and expect less efficiency, but at the same time a "fully commutative" scheme.

4 PQDH - Post-Quantum Diffie-Hellman

The operations performed by the initiator and the responder differ in the original SIDH protocol. This can be inconvenient, as we want the key generation and key derivation steps to be identical on both sides to provide a true drop-in replacement for ephemeral Diffie-Hellman-like protocols used today. In this section, we propose a protocol PQDH (Post-Quantum Diffie-Hellman), the improved version of SIDH which resolves the initiator-responder difference issue.

4.1 Our Protocol

The main idea of our protocol is to generate own (*private*, *public*) key pair using both sets of bases. Then each user matches them with the opposite parameters of the other user and computes two common values, which are then combined in a commutative manner.

Key Generation: Each party performs the following steps:

1. Randomly select $n_A \in \mathbb{Z}_{\ell_A^{e_A}}$ and $n_B \in \mathbb{Z}_{\ell_B^{e_B}}$.
2. Compute $K_A = P_A + [n_A]Q_A$.
3. Compute $K_B = P_B + [n_B]Q_B$.
4. Obtain E_A using the kernel $\langle K_A \rangle$ for the isogeny $\phi_A : E \to E_A = E/\langle K_A \rangle$.
5. Obtain E_B using the kernel $\langle K_B \rangle$ for the isogeny $\phi_B : E \to E_B = E/\langle K_B \rangle$.
6. Compute the images of the values P_B and Q_B under ϕ_A, namely $\phi_A(P_B)$ and $\phi_A(Q_B)$.
7. Compute the images of the values P_A and Q_A under ϕ_B, namely $\phi_B(P_A)$ and $\phi_B(Q_A)$.

The private key is: $\{n_A, n_B\}$. The public key is: $\{E_A, E_B, \phi_A(P_B), \phi_A(Q_B), \phi_B(P_A), \phi_B(Q_A)\}$. The party either sends or publishes eathe public key. In practice, since we are in concentrating on the ephemeral version, we work in the context of sending public key.

Key Derivation: After obtaining the other party's public key, $\{E_1, E_2, P_{00}, P_{01}, P_{10}, P_{11}\}$, the user performs the following steps to derive the common key:

1. Using their own private key value n_A and the other user's auxiliary points P_{10}, P_{11} computes $K_{A2} = P_{10} + [n_A]P_{11}$.
2. Using their own private key value n_B and the other user's auxiliary points P_{00}, P_{01} computes $K_{B1} = P_{00} + [n_B]P_{01}$.
3. Using K_{A2} as the generator point for the kernel computes $E_{A2} = E_2/\langle K_{A2} \rangle$.
4. Using K_{B1} as the generator point for the kernel computes $E_{B1} = E_1/\langle K_{B1} \rangle$.
5. Computes the j-invariants of the resulting curves: $j_1 = j(E_{A2})$ and $j_2 = j(E_{B1})$.
6. Combines j_1 and j_2 in a commutative manner to obtain j
7. Obtains a common key $k = KDF(j)$.

Remark 3. For the step where the user combines j_1 and j_2 to obtain j, the method would be predefined in the protocol. The two most practical approaches are either to add them as field elements or to XOR them as binary elements, component by component.

Diagram We present the diagram of the protocol.

$$\begin{tikzcd}
E_{A'B'} & E_{B'} \arrow[l, dashed] \arrow[r, dotted] & E_{AB'} \\
E_{A'} \arrow[u, dashed] \arrow[d, dotted] & E \arrow[l, dashed] \arrow[r, dotted] \arrow[u, dashed] \arrow[d, dotted] & E_A \arrow[u, dashed] \arrow[d, dotted] \\
E_{A'B} & E_B \arrow[l, dashed] \arrow[r, dotted] & E_{AB}
\end{tikzcd}$$

Fig. 2: PQDH Diagram

In the diagram, the densely dotted edges represent the secret isogenies of one party and the loosely dashed edges represent the secret isogenies of the other party. The red curves are the common derived secret used for the common secret key. The yellow edges and curves are possible to compute but are not computed or used.

Complexity The computational complexity of this protocol is about double compared to SIDH, as each user is simultaneously operating on both bases. The same is true for the overhead, we have exactly a double amount of data sent in comparison to SIDH. The private key size is also doubled.

For the global parameters nothing is changed; they are the same as in the SIDH case. This property allows users to choose between SIDH or PQDH, operating on the same set of parameters.

Security The security of the scheme relies on the same assumptions as SIDH, which is defined and discussed in [9]. We can see that breaking the PQDH scheme is equivalent to breaking two instances of SIDH to get both j-invariants. On the other hand, we are combining the two j-invariants, which causes dependencies between the final key and both of them, and hence we might not get quite the double effort to break the scheme. This means that the security of the scheme is about the same as the one of SIDH. In fact, its security level is slightly higher than that of SIDH, but less than two times higher compared to SIDH. (Note: by *two times*, we mean the complexity of attack, i.e., which is equivalent to one bit of security.) Thus, we claim that the PQDH security level is at least equivalent to that of SIDH with the same parameters for both classical and post-quantum security.

5 Efficient Countermeasure

The only known side-channel attack that applies to the ephemeral version of SIDH and PQDH is the fault attack described in Section 2.2. A direct countermeasure to this attack is to check the order of the other party's basis points. Order-checking computation, although polynomial in running time, is expensive and could cost 100 percent running time per point. Given that there are two basis pointss, we could have a cost of 200 percent. In this section, we propose a more efficient approach to providing such countermeasure.

5.1 Efficient Approach for SIDH

In the case of the SIDH protocol, we assume that we obtain or receive the other party's basis points P and Q whose *expected* order is ℓ^e.
Let ϕ be the current party's secret isogeny, S be either P or $\phi(P)$, and T be either Q or $\phi(Q)$. Note: this choice must match the choice for S.

Remark 4. In this case, we recommend choosing $\phi(P), \phi(Q)$ as values of S and T, respectively. This would also prevent a fault-injection attack that could happen during the course of the computation.

Before we proceed, we prove the following claim.

Claim. Let $S, T \in E(\mathbb{F}_{p^2})$ be non-identity points, where E is supersingular and $p = \ell^e \cdot \ell'^{e'} \cdot f \pm 1$ (where ℓ and ℓ' are small primes). If $S + T \in E[\ell^e]$, then either both S and T are in $E[\ell^e]$ or both are in $E \setminus E[\ell^e]$.

Proof. We can express $S = S_1 + S_2$, such that $S_1 \in E[\ell^e]$ and $S_2 \in E[\ell'^{e'} f]$. Similarly, we can express $T = T_1 + T_2$, such that $T_1 \in E[\ell^e]$ and $T_2 \in E[\ell'^{e'} f]$. Given the fact that $S + T \in E[\ell^e]$ and so are S_1 and T_1,

$$\begin{aligned}\infty &= [\ell^e](S + T) = [\ell^e](S_1 + S_2 + T_1 + T_2) \\ &= [\ell^e]S_1 + [\ell^e]S_2 + [\ell^e]T_1 + [\ell^e]T_2 \\ &= [\ell^e]S_2 + [\ell^e]T_2 \\ &= [\ell^e](S_2 + T_2).\end{aligned}$$

It follows that

$$[\ell^e](S_2 + T_2) = \infty,$$

which means that $S_2 + T_2 \in E[\ell^e]$. At the same time, $S_2 + T_2 \in E[\ell'^{e'} f]$. Since $E[\ell^e] \cap E[\ell'^{e'} f] = \{\infty\}$, we obtain that

$$S_2 + T_2 = \infty.$$

Hence, either $S_2 = T_2 = \infty$, meaning that $S, T \in E[\ell^e]$ or $S_2 = -T_2 \neq \infty$, meaning that $S, T \in E \setminus E[\ell^e]$. ■

The above claim is a standard group theory result. It shows that if the sum (or in fact a linear combination) of the two points is checked, then we only need to check one of the two points in the case that their sum verifies. This is an alternative to checking each point separately, as in the case of someone trying to forge the basis points, we will be able to catch that faster, as most likely $S + T$ provided will not verify if there is a forgery. Another application of this approach is when using Montgomery curves, we could already be provided with $S, T, S - T$, so that we can choose to verify $S - T$ and either of S or T, which provides a speed-up as well.

Perform the following:

1. Compute $R = S + T$.
2. Compute $O = [\ell^e]R$.
3. If the resulting value of O is the point at infinity (∞), continue, otherwise abort the session.
4. Compute $O_S = [\ell^e]S$.
5. If the resulting value of O_S is the point at infinity (∞), continue, otherwise abort the session.

Compared to the general countermeasure approach (i.e. computing the orders of points), the proposed one is expected to be about 10 times faster, depending on the curve format. We avoid expensive computations of a point-order finding algorithm and, instead, perform one special scalar multiplication. In practice, the value of ℓ is either 2 or 3. Given that our scalar is in the form of ℓ^e, we only need to compute DOUBLE or TRIPLE elliptic curve arithmetic operation e times. The results in Section 7.1 show that the cost of this countermeasure is approximately 22%-24%.

Step 2 of our approach is ensuring that $R \in E[\ell^e]$. It is computationally more efficient than finding the exact order. At the same time, this suffices for our purposes, as the attack only works when we have some point in $E \setminus E[\ell^e]$ that is of order other than ℓ^i for $i \in \{0, 1, \ldots, e\}$.

5.2 Efficient Approach for PQDH

In this section, we will present the countermeasure against the described fault attack for the PQDH protocol.

We first describe how to properly perform a check of the order of the basis points used for the kernel that is being used to compute the isogeny and the image curve.

Assume that the basis points used for computation of our isogeny are P, Q. The expected order of each point is ℓ^e. Let (m, n) be the private key (where $m = 1$ in practice). Hence, the kernel point is $K = [m]P + [n]Q$. Our first observation is that we only need to make sure that K is of order ℓ^e instead of just P and Q, as n is random each time and hence chances of picking P', Q' of different order, which would still result K of the correct order are equivalent to guessing the actual private key. Also, we enforce that K is not just an element of $E[\ell^e]$ to ensure that our resulting isogeny is of full-span degree.

Throughout the execution of computing and evaluating our isogeny with kernel K, we observe that we are computing and obtaining values of the following format $[\ell^i]\phi_j(K)$, where ϕ_j is and isogeny of degree ℓ^j for some i's and j's in $\{0, \ldots, e-1\}$.

We also note that $[\ell^i]\phi_j(K) = \phi_j([\ell^i]K)$. We choose such available value with the highest value of $i + j$ available. In practice, since we use isogenies with $\ell = 2, 3$, we could be either one or two steps away from $i + j = e$. Namely, we will be able to find $i + j = e - 2$ or $e - 1$, depending whether we are using a multiplication-based, an isogeny-based, or an optimal strategy. These three strategies are described in detail in [9, 7]

If we are using a multiplication-based or an isogeny-based strategy, then we know that in the process of the isogeny computation, we will obtain $R = [\ell^i]\phi_j(K)$ such that $i + j = e - 1$. In this case, we perform the following:

1. Check that $R \neq \infty$, otherwise, abort.
2. Compute $F = [\ell]R$.
3. If $F = \infty$, return ‘valid’, otherwise, abort.

For step 2, if $\ell = 2$, perform a DOUBLE operation; if $\ell = 3$, perform a TRIPLE operation.

When we are using the optimal strategy, we might obtain $R = [\ell^i]\phi_j(K)$ such that $i + j = e - 2$. In this case, we perform the following:

1. Compute $F_1 = [\ell]R$.
2. Check that $F_1 \neq \infty$, otherwise, abort.
3. Compute $F_2 = [\ell]F_1$.
4. If $F_2 = \infty$, return ‘valid’, otherwise, abort.

For each of the steps 1 and 3, if $\ell = 2$, perform a DOUBLE operation; if $\ell = 3$, perform a TRIPLE operation.

This approach implicitly checks and ensures that K is exactly of order ℓ^e.

The above approach can be applied to PQDH. We note that each party uses both bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ to compute isogenies. Hence, while computing isogenies with the corresponding kernels, a user can verify at the same time the correct order of these points.

The cost of this countermeasure for PQDH is less than one percent. In practice, each user needs to compute at most four extra DOUBLE operations and two TRIPLE operations, which is negligible compared to the entire computation.

As a result, the proposed countermeasure for this fault attack by an active adversary at a nearly free cost of computation.

6 PQDH-Based PAKE

In this section, we present a password-authenticated key exchange protocol based on isogenies, namely a PQDH-based PAKE. The only other known isogeny-based PAKE protocol is one by Taraskin et al. [16], which is based on SIDH. Our PAKE protocol uses the same construction, but the underlying key agreement is PQDH. It is an alternative, rather than an improvement, which provides a protocol that is symmetric in terms of actions of the initiator and the responder, though this comes at a computational cost.

6.1 Protocol

We adapt the main isogeny-based PAKE protocol construction presented in [16] to PQDH.

Suppose parties U_1 and U_2 share a common password pwd. The global parameters are defined as before. Let H_A and H_B be random oracles mapping inputs to masking functions, and let KDF be a key derivation function.

Remark 5. The resulting masking functions Ψ are not explicitly described here. The reader may refer to [16] for details on masking function construction. For this paper, we note that to mask, we apply the corresponding matrix to the auxiliary points and to unmask, it is enough to apply the unmasking scalar multiplication operation to the second auxiliary point.

Each party performs the following steps:

1. Choose $n_A \in \mathbb{Z}_{\ell_A^{e_A}}$ uniformly and set $K_A = P_A + n_A Q_A$.
2. Choose $n_B \in \mathbb{Z}_{\ell_B^{e_B}}$ uniformly and set $K_B = P_B + n_B Q_B$.
3. Obtain E_A using the kernel $\langle K_A \rangle$ for the isogeny $\phi_A\colon E \to E_A = E/\langle K_A \rangle$.
4. Obtain E_B using the kernel $\langle K_B \rangle$ for the isogeny $\phi_B\colon E \to E_B = E/\langle K_B \rangle$.
5. Compute the images of the values P_B and Q_B under ϕ_A, namely $\phi_A(P_B)$ and $\phi_A(Q_B)$.
6. Compute the images of the values P_A and Q_A under ϕ_B, namely $\phi_B(P_A)$ and $\phi_B(Q_A)$.
7. Compute $\Psi_A = H_A(j(E_A) \| \text{pwd})$ and set
$$\begin{bmatrix} X_A \\ Y_A \end{bmatrix}\Bigg(= \Psi_A \cdot \begin{bmatrix} \phi_A(P_B) \\ \phi_A(Q_B) \end{bmatrix}\Bigg($$
8. Compute $\Psi_B = H_B(j(E_B) \| \text{pwd}_A)$ and set
$$\begin{bmatrix} X_B \\ Y_B \end{bmatrix}\Bigg(= \Psi_B \cdot \begin{bmatrix} \phi_B(P_A) \\ \phi_B(Q_A) \end{bmatrix}\Bigg($$
9. Send $\{E_A, X_A, Y_A, E_B, X_B, Y_B\}$ to the other party.

Upon receiving $\{E_1, X_1, Y_1, E_2, X_2, Y_2\}$ from the party, does the following steps:

1. Check that $e_{E_1}(X_1, Y_1) = e_E(P_B, Q_B)^{\ell_A^{e_A}}$—if not, abort.
2. Check that $e_{E_2}(X_2, Y_2) = e_E(P_A, Q_A)^{\ell_B^{e_B}}$—if not, abort.
3. Compute $\Psi_1 = H_A(j(E_1) \| \text{pwd})$.
4. Compute $\Psi_2 = H_B(j(E_2) \| \text{pwd})$.
5. Compute $j_1 = j(E_1/\langle X_1 + n_B^{\Psi_1^{-1}} Y_1 \rangle)$.
6. Compute $j_2 = j(E_2/\langle X_2 + n_A^{\Psi_2^{-1}} Y_2 \rangle)$.
7. Construct the key K, given by

$$K = \text{KDF}((E_A, X_A, Y_A, E_B, X_B, Y_B) \oplus (E_1, X_1, Y_1, E_2, X_2, Y_2) \| j_1 \oplus j_2 \| (\Psi_A, \Psi_B) \oplus (\Psi_1, \Psi_2)).$$

Complexity We compare the complexity of the given approach to that of the original isogeny-based PAKE protocol, which is based on SIDH. Given that each party in phase one and phase two performs about twice the number of operations in PQDH-based PAKE, the complexity is about double.

Security For security purposes, one can break the protocol into two separate instances, except the last step, first running the odd steps and then running the even steps and combining the two results to obtain the key. Thus, we get the same security arguments with slight modifications that on one side it might be harder to attack the given protocol because there is a double amount of weight, but on the other hand due to the fact that we are symmetrically combing the secret results, it will make the advantage negligible and hence provide us with about the same security as that of the original PAKE protocol.

7 Implementation

We implemented PQDH including the fault attack countermeasures. The implementations are based on the optimized implementation of the SIKE submission to the NIST post-quantum standardization process[3] [10]. Our additions apply to both the version in portable C, as well as to the ASM optimized versions for ARM and Intel x86-64. The supported primes are:

- `p503` $= 2^{250} \cdot 3^{159} - 1$
- `p751` $= 2^{372} \cdot 3^{239} - 1$

.

The implementation uses Montgomery curves $E_{A,B}$ over $\mathbb{F}_q$ that satisfy the curve equation $By^2 = x^3 + Ax^2 + x$. Arithmetic on the elliptic curves is done efficiently using projective coordinates. Isogeny computations are done using an optimal tree traversal strategy as described in [10]. Multi-precision arithmetic is optimized using ARMv8 and Intel x86-64 assembly.

Our additions and modifications are the following: PQDH is added as defined in Algorithm 1. For a detailed specification of the underlying algorithms, we refer to the SIKE specification ([10], Alg. 3-22). The detailed explicit algorithms for implementing the countermeasures are defined in Appendix A.

Algorithm 1: PQDH = (PQDHGen, PQDHDer)

```
function PQDHGen
    Input: ()
    Output: $(sk, pk)$
    $(sk_2, sk_3) \leftarrow_R (\mathcal{K}_2, \mathcal{K}_3)$
    $(pk_2, pk_3) \leftarrow (\texttt{isogen}_2(sk_2), \texttt{isogen}_3(sk_3))$
    return $((sk_2, sk_3), (pk_2, pk_3))$

function PQDHDer
    Input: $(sk, pk) = ((sk_2, sk_3), (pk_2, pk_3))$
    Output: $(K)$
    $j_2 \leftarrow \texttt{isoex}_2(pk_3, sk_2)$
    $j_3 \leftarrow \texttt{isoex}_3(pk_2, sk_3)$
    $K \leftarrow j_2 \oplus j_3$
    return $K$
```

7.1 Evaluation Results

The performance of PQDH and the countermeasures were evaluated on an Intel Core i7-8559U 2.7 GHz (Coffee Lake) CPU, on CentOS 7. Hyperthreading and Turbo Boost were disabled as a standard practice. The software was compiled using GCC version 4.8.5 with ”-O3” optimization level.

We denote the instantiations for the schemes as `SIDHp503`, `SIDHp751`, `PQDHp503`, `PQDHp751`.

Table 1 shows the performance of SIDH and PQDH. All schemes are evaluated using (a) no countermeasures, (b) the countermeasure from Sec. 5.1 is applied to the `KeyDer` phase of all schemes, and (c) the countermeasure from Sec. 5.2 is applied to PQDH.

[3] `https://github.com/Microsoft/PQCrypto-SIDH`

Performance In absolute terms on our test platform, `PQDH KeyGen` is performed in 8 msec. and 22.3 msec. for `P503` and `P751`, respectively. `PQDH KeyGen` in 6.7 msec. and 18.6 msec., respectively.

The performance of `PQDH KeyGen` is equivalent to `SIDH KeyGen A` and `SIDH KeyGen B` added up. The same holds for The `PQDH KeyDer` phase, with a performance of `SIDH KeyDer A` and `SIDH KeyDer B` added up.

All countermeasures are applied to the `KeyDer` phases. The first countermeasure version shows a 22% – 24% increase of cycles in the schemes. Compared to that, the second countermeasure version applied to PQDH shows a negligible overhead that lies within the measurement tolerance. It can be seen as a rare case of a countermeasure without a performance impact and applies therefore even for very performance-constrained environments.

Scheme	KeyGen A	KeyGen B	KeyDer A	KeyDer B
No countermeasures				
SIDHp503	10’165	11’302	8’284	9’513
SIDHp751	28’342	31’769	23’043	27’679
PQDHp503	21’466		17’779	
PQDHp751	60’254		50’155	
Efficient countermeasure for SIDH (Sec. 5.1)				
SIDHp503	-		10’134	11’815
SIDHp751	-		28’566	34’444
PQDHp503	-		21’945	
PQDHp751	-		62’187	
Efficient countermeasure for PQDH (Sec. 5.2)				
PQDHp503	-		18’060	
PQDHp751	-		50’234	

Table 1: Performance of SIDH and PQDH, in thousands of cycles. Evaluation of three versions: No countermeasures, optimal countermeasure for SIDH, and an efficient countermeasure for PQDH. On Core i7-8559U 2.7 GHz.

Key sizes The sizes of the PQDH private and public keys are double the size of the SIDH keys. Note that the net size of the 2-torsion private key of `SIDHp751` is 372 bit and would fit in 47 bytes. To match the size of the 3-torsion keys, we zero-pad it to 48 bytes. All sizes are depicted in Table 2.

8 Conclusion

Isogeny-based cryptography continues to be researched and developed. The schemes are being optimized, more cryptanalysis is being performed, new variants of schemes are being created for various applications, and new protocols are being designed. As cryptography is an integral part of today’s IT infrastructure, we also need to find methods allowing for an efficient migration to new schemes.

In this paper, we have presented PQDH, a variant of SIDH, which removes the requirement to distinguish between the initiator and the responder. This scheme is not only one of the first isogeny-based, but also one of the first

Scheme	**private key**	**public key**	**shared secret**
	`sk`	`pk`	`ss`
`SIDHp503`	32	378	126
`SIDHp751`	48	564	188
`PQDHp503`	64	756	126
`PQDHp751`	96	1128	188

Table 2: Private/public key, and shared secret sizes of SIDH and PQDH in bytes.

quantum-resistant schemes with such property. This will be very helpful in transitioning from conventional to post-quantum cryptography. Also, we have presented an efficient countermeasure against the side-channel fault attack. The attack applies not only to the static version of SIDH and PQDH, but also to the ephemeral one, which makes this countermeasure highly desirable. Finally, we have shown that same isogeny-based PAKE construction, as shown in [16], applies to the PQDH scheme.

References

1. Steven M. Bellovin and Michael Merritt. Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks. In *IEEE SYMPOSIUM ON RESEARCH IN SECURITY AND PRIVACY*, pages 72–84, 1992.
2. Xavier Bonnetain. Improved Low-qubit Hidden Shift Algorithms. *CoRR*, abs/1901.11428, 2019.
3. Xavier Bonnetain and André Schrottenloher. Quantum Security Analysis of CSIDH and Ordinary Isogeny-based Schemes. *IACR Cryptology ePrint Archive*, 2018:537, 2018.
4. Reinier Bröker, Denis Charles, and Kristin Lauter. Evaluating Large Degree Isogenies and Applications to Pairing Based Cryptography. In *Pairing '08: Proceedings of the 2nd International Conference on Pairing-Based Cryptography*, pages 100–112, 2008.
5. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An Efficient Post-Quantum Commutative Group Action. In *ASIACRYPT*, 2018.
6. Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *J. Math. Cryptol.*, 8(1):1–29, 2014.
7. Craig Costello, Patrick Longa, and Michael Naehrig. Efficient Algorithms for Supersingular Isogeny Diffie-Hellman. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, pages 572–601, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
8. Jintai Ding, Saed Alsayigh, Jean Lancrenon, Saraswathy RV, and Michael Snook. Provably Secure Password Authenticated Key Exchange Based on RLWE for the Post-Quantum World. In Helena Handschuh, editor, *Topics in Cryptology – CT-RSA 2017*, pages 183–204, Cham, 2017. Springer International Publishing.
9. Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *J. Mathematical Cryptology*, 8:209–247, 2014.
10. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Amir Jalali, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Encapsulation. 2017. Submission to Post-Quantum Cryptography Standardization Process.
11. David Jao and Vladimir Soukharev. A subexponential algorithm for evaluating large degree isogenies. In *Algorithmic number theory*, volume 6197 of *Lecture Notes in Comput. Sci.*, pages 219–233. Springer, Berlin, 2010.
12. National Institute of Standards and Technology. Post-Quantum Cryptography Standardization Process, 2017. `https://csrc.nist.gov/Projects/Post-Quantum-Cryptography`.
13. Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM J. Comput.*, 26(5):1484–1509, 1997. Preliminary version in FOCS '94. `arXiv:quant-ph/9508027v2`.
14. Joseph H. Silverman. *The arithmetic of elliptic curves*, volume 106 of *Graduate Texts in Mathematics*. Springer-Verlag, New York, 1992. Corrected reprint of the 1986 original.
15. Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. Math. Commun.*, 4(2):215–235, 2010.
16. Oleg Taraskin, Vladimir Soukharev, David Jao, and Jason LeGrow. An Isogeny-Based Password-Authenticated Key Establishment Protocol. *IACR Cryptology ePrint Archive*, 2018:886, 2018.
17. Yan Bo Ti. Fault Attack on Supersingular Isogeny Cryptosystems. In Tanja Lange and Tsuyoshi Takagi, editors, *Post-Quantum Cryptography*, pages 107–122, Cham, 2017. Springer International Publishing.
18. Jacques Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sci. Paris Sér. A-B*, 273:A238–A241, 1971.
19. Jiang Zhang and Yu Yu. Two-Round PAKE from Approximate SPH and Instantiations from Lattices. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology – ASIACRYPT 2017*, pages 37–67, Cham, 2017. Springer International Publishing.

A Explicit algorithms

We provide a collection of explicit algorithms used for the fault attack countermeasures: Differential addition `xADD` is used for the SIDH fault attack countermeasure from Sec. 5.1 is added as in Algorithm 2. The optimal countermeasures for SIDH from Sec. 5.1 are integrated to `isoex`$_2$ and `isoex`$_3$. They are defined in Algorithms 5-6. For the PQDH countermeasure from Sec. 5.2, the optimal tree traversal strategies for computing isogenies are modified as defined in Algorithms 3-4.

Algorithm 2: Differential addition

```
function xADD
    Input: $P = (X_P : 1)$, $Q = (X_Q : 1)$, $PQ = (X_{Q-P} : 1)$
    Output: $(X_{P+Q} : Z_{P+Q})$
$t_0 \leftarrow X_P + 1$
$t_1 \leftarrow X_P - 1$
$X_{P+Q} \leftarrow X_Q - 1$
$Z_{P+Q} \leftarrow X_Q + 1$
$t_0 \leftarrow X_{P+Q} \cdot t_0$
$t_1 \leftarrow Z_{P+Q} \cdot t_1$
$Z_{P+Q} \leftarrow t_0 - t_1$
$X_{P+Q} \leftarrow t_0 + t_1$
$Z_{P+Q} \leftarrow Z_{P+Q}^2$
$X_{P+Q} \leftarrow X_{P+Q}^2$
$Z_{P+Q} \leftarrow X_{Q-P} \cdot Z_{P+Q}$
return $(X_{P+Q} : Z_{P+Q})$
```

Algorithm 3: Computing and evaluating a 2^e-isogeny, with the countermeasure for PQDH

```
function 2_e_iso_pqdh
    Static parameters: Integer $\mathtt{e}_2$ from the public parameters, a strategy $(s_1, \ldots, s_{\mathtt{e}_2/2-1}) \in (\mathbb{N}^+)^{\mathtt{e}_2/2-1}$
    Input: Constants $(A_{24}^+ : C_{24})$ corresponding to a curve $E_{A/C}$, $(X_S : Z_S)$ where $S$ has exact order $2^{\mathtt{e}_2}$ on $E_{A/C}$
    Optional input: $(X_1 : Z_1)$, $(X_2 : Z_2)$ and $(X_3 : Z_3)$ on $E_{A/C}$
    Output: $(A_{24}^{+}{}' : C_{24}')$ corresponding to the curve $E_{A'/C'} = E/\langle S \rangle$
    Optional output: $(X_1' : Z_1')$, $(X_2' : Z_2')$ and $(X_3' : Z_3')$ on $E_{A'/C'}$
Initialize empty deque $\mathbf{S}$
push($\mathbf{S}$, ($\mathtt{e}_2/2$, $(X_S : Z_S)$)))(
$i \leftarrow 1$, $j \leftarrow 0$
while $\mathbf{S}$ not empty do
    $(h, (X : Z)) \leftarrow$ pop($\mathbf{S}$)
    if $h = 1$ then
        if $j = 0$ then
            $(X_d : Z_d) \leftarrow$ xDBL$\big((X : Z), (A_{24}^+ : C_{24})\big)$(                // Alg.3 [10]
            if $Z_d = 0$ then
                Error: Potential fault attack
            $(X_d : Z_d) \leftarrow$ xDBL$\big((X_d : Z_d), (A_{24}^+ : C_{24})\big)$(            // Alg.3 [10]
            if $Z_d \neq 0$ then
                Error: Potential fault attack
        $\big((A_{24}^+ : C_{24}), (K_1, K_2, K_3)\big)$($\leftarrow$ 4_iso_curve $((X : Z))$   // Alg.11 [10]
        Initialize empty deque $\mathbf{S}'$
        while $\mathbf{S}$ not empty do
            $(h, (X : Z)) \leftarrow$ pull($\mathbf{S}$)
            $(X : Z) \leftarrow$ 4_iso_eval $((K_1, K_2, K_3), (X : Z))$                 // Alg.12 [10]
            push($\mathbf{S}'$, $(h - 1, (X : Z))$)(
        $\mathbf{S} \leftarrow \mathbf{S}'$
        for $(X_j : Z_j)$ in optional input do
            $(X_j : Z_j) \leftarrow$ 4_iso_eval $\big((K_1, K_2, K_3), (X_j : Z_j)\big)$(          // Alg.12 [10]
        $j \leftarrow 1$
    else if $0 < s_i < h$ then
        push($\mathbf{S}$, $(h, (X : Z))$)(
        $(X : Z) \leftarrow$ xDBLe$\big((X : Z), (A_{24}^+ : C_{24}), 2 \cdot s_i\big)$(          // Alg.4 [10]
        push($\mathbf{S}$, $(h - s_i, (X : Z))$)(
        $i \leftarrow i + 1$
    else
        Error: Invalid strategy
return $(A_{24}^+ : C_{24})$, $[(X_1 : Z_1), (X_2 : Z_2), (X_3 : Z_3)]$(
```

Algorithm 4: Computing and evaluating a 3^e-isogeny, with the countermeasure for PQDH

```
function 3_e_iso_pqdh
    Static parameters: Integer $\mathtt{e}_3$ from the public parameters, a strategy $(s_1, \ldots, s_{\mathtt{e}_3-1}) \in (\mathbb{N}^+)^{\mathtt{e}_3-1}$
    Input: Constants $(A_{24}^+ : A_{24}^-)$ corresponding to a curve $E_{A/C}$, $(X_S : Z_S)$ where $S$ has exact order $3^{\mathtt{e}_3}$ on $E_{A/C}$
    Optional input: $(X_1 : Z_1)$, $(X_2 : Z_2)$ and $(X_3 : Z_3)$ on $E_{A/C}$
    Output: $({A_{24}^+}' : {A_{24}^-}')$ corresponding to the curve $E_{A'/C'} = E/\langle S \rangle$
    Optional output: $(X_1' : Z_1')$, $(X_2' : Z_2')$ and $(X_3' : Z_3')$ on $E_{A'/C'}$
Initialize empty deque S
push(S, $(\mathtt{e}_3, (X_S : Z_S))$)
$i \leftarrow 1, j \leftarrow 0$
while S not empty do
    $(h, (X : Z)) \leftarrow$ pop(S)
    if $h = 1$ then
        if $j = 0$ then
            if $Z = 0$ then
                Error: Potential fault attack
            $(X_d : Z_d) \leftarrow$ xTPL $\big((X : Z), (A_{24}^+ : A_{24}^-)\big)$                // Alg. 6 [10]
            if $Z_d \neq 0$ then
                Error: Potential fault attack
        $\big((A_{24}^+ : A_{24}^-), (K_1, K_2)\big) \leftarrow$ 3_iso_curve $((X : Z))$                // Alg. 13 [10]
        Initialize empty deque S′
        while S not empty do
            $(h, (X : Z)) \leftarrow$ pull(S)
            $(X : Z) \leftarrow$ 3_iso_eval $((K_1, K_2), (X : Z))$                // Alg. 14 [10]
            push(S′, $(h - 1, (X : Z))$)
        S ← S′
        for $(X_j : Z_j)$ in optional input do
            $(X_j : Z_j) \leftarrow$ 3_iso_eval $\big((K_1, K_2), (X_j : Z_j)\big)$                // Alg. 14 [10]
        $j \leftarrow 1$
    else if $0 < s_i < h$ then
        push(S, $(h, (X : Z))$)
        $(X : Z) \leftarrow$ xTPLe $\big((X : Z), (A_{24}^+ : A_{24}^-), s_i\big)$                // Alg. 7 [10]
        push(S, $(h - s_i, (X : Z))$)
        $i \leftarrow i + 1$
    else
        Error: invalid strategy
return $(A_{24}^+ : A_{24}^-)$, $[(X_1 : Z_1), (X_2 : Z_2), (X_3 : Z_3)]$
```

Algorithm 5: Establishing shared keys in the 2-torsion, with the fault attack countermeasure for SIDH.

```
function isoex_2
    Input: Secret key $\mathrm{sk}_2 \in \mathbb{Z}$, public key $pk_3 = (x_1, x_2, x_3) \in (\mathbb{F}_{p^2})^3$, and parameter e2
    Output: A $j$-invariant $j_2$
$(A : C) \leftarrow$ (get_A$(x_1, x_2, x_3) : 1)$                                   // Alg. 10 [10]
$(A_{24}^+ : C_{24}) \leftarrow (A + 2 : 4)$
$Q \leftarrow$ xADD$(x1, x2, x3)$                                                       // Alg. 2
$Q \leftarrow$ xDBLe$(Q, (A_{24}^+ : C_{24}), e2 - 1)$                                   // Alg. 4 [10]
if $Z_Q = 0$ then
    Error: Potential fault attack
$Q \leftarrow$ xDBL$(Q, (A_{24}^+ : C_{24}))$                                            // Alg. 3 [10]
if $Z_Q \neq 0$ then
    Error: Potential fault attack
$Q \leftarrow$ xDBLe$((x1 : 1), (A_{24}^+ : C_{24}), e2 - 1)$                            // Alg. 4 [10]
if $Z_Q = 0$ then
    Error: Potential fault attack
$Q \leftarrow$ xDBL$(Q, (A_{24}^+ : C_{24}))$                                            // Alg. 3 [10]
if $Z_Q \neq 0$ then
    Error: Potential fault attack
$(X_S : Z_S) \leftarrow$ Ladder3pt$(\mathrm{sk}_2, (x_1, x_2, x_3), (A : C))$            // Alg. 8 [10]
$(A_{24}^+ : C_{24}) \leftarrow$ 2_e_iso$\Big((A_{24}^+ : C_{24}), (X_S : Z_S)\Big)\Big($   // Alg. 15 [10]
$(A : C) \leftarrow (4A_{24}^+ - 2C_{24} : C_{24})$
$j =$ j_inv$((A : C))$                                                                   // Alg. 9 [10]
return $j$
```

Algorithm 6: Establishing shared keys in the 3-torsion, with the fault attack countermeasure for SIDH

```
function isoex$_3$
    Input: Secret key $sk_3 \in \mathbb{Z}$, public key $pk_2 = (x_1, x_2, x_3) \in (\mathbb{F}_{p^2})^3$, and parameter e3
    Output: A $j$-invariant $j_3$
$(A : C) \leftarrow$ (get_A$(x_1, x_2, x_3)$: 1)                                  // Alg. 10 [10]
$(A^+_{24} : A^-_{24}) \leftarrow (A + 2 : A - 2)$
$Q \leftarrow$ xADD$(x1, x2, x3)$                                                    // Alg. 2
$Q \leftarrow$ xTPLe$(Q, (A^+_{24} : A^-_{24}), e3 - 1)$                           // Alg. 7 [10]
if $Z_Q = 0$ then
    Error: Potential fault attack
$Q \leftarrow$ xTPL$(Q, (A^+_{24} : A^-_{24}))$                                    // Alg. 6 [10]
if $Z_Q \neq 0$ then
    Error: Potential fault attack
$Q \leftarrow$ xTPLe$((x1 : 1), (A^+_{24} : A^-_{24}), e3 - 1)$                    // Alg. 7 [10]
if $Z_Q = 0$ then
    Error: Potential fault attack
$Q \leftarrow$ xTPL$(Q, (A^+_{24} : A^-_{24}))$                                    // Alg. 6 [10]
if $Z_Q \neq 0$ then
    Error: Potential fault attack
$(X_S : Z_S) \leftarrow$ Ladder3pt$(\mathrm{sk}_3, (x_1, x_2, x_3), (A : C))$      // Alg. 8 [10]
$(A^+_{24} : A^-_{24}) \leftarrow$ 3_e_iso$\big((A^+_{24} : A_{24}), (X_S : Z_S)\big)\Big($   // Alg. 16 [10]
$(A : C) \leftarrow (2 \cdot (A^-_{24} + A^+_{24}) : A^+_{24} - A^-_{24})$
$j$ = j_inv$((A : C))$                                                            // Alg. 9 [10]
return $j$
```