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-know SIDH protocol. Then, we provide an isogeny-based password-authenticated key exchange protocol based on our scheme. A summary of security and computational complexities are also presented. Finally, we present an efficient countermeasure against a side-channel attack that applies to both static and ephemeral versions of SIDH and our scheme.

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

1 Introduction

Recently, NIST has started the first post-quantum cryptography standardization process [12]. Cryptographers around the world have submitted in total 82 proposals and 69 were accepted for the first round, 26 of which were selected for the second round. The research and cryptanalysis in this area has 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 and transmits it to the other party. In such a case the 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 clearly 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]. Although SIKE is a KEM, it 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 obtain an efficient post-quantum PAKE scheme.

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 about isogeny-based cryptography. In Section 4 we present PQDH followed by post-quantum PAKE in Section 5. In Section 6 we discuss the side-channel attack countermeasure. The evaluation results are shown in Section 7. 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 present a PAKE (password-authenticated key exchange) scheme based on PQDH. Finally, we contribute an efficient countermeasure against a side-channel fault-attack on isogeny-based schemes, which applies to both static and ephemeral versions.

2.1 Post-Quantum Diffie-Hellman

SIDH is one of the most promising candidates for a quantum-resistant key agreement schemes. We focus on the ephemeral version as the most frequently used version in pracice, 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 compatibility is due to the fact that 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 API and users are still performing different actions. We found a way to overcome the obstacle of differentiating between initiator and responder and will present a scheme, named PQDH, which is fully based on SIDH, but works with the Diffie-Hellman API.

2.2 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 that it allows 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]. We propose a PAKE protocol based on isogenies between supersingular elliptic curves. It builds on top of PQDH and its speed characteristics is between SIDH and PQDH. The first isogeny-based PAKE protocol was proposed by Taraskin et al. [16], which is based on SIDH. The protocol presented in this paper is based on PQDH. Given that PQDH truly resembles Diffie-Hellman-type schemes, building other protocols from it results them to be less complex. The PQDH-based PAKE demonstrates that property well. Our protocol is more efficient and less complex, however it does not provide forward secrecy, while the SIDH-based one does. Hence, we propose a choice of isogeny-based PAKE with trade-offs.

2.3 Fault Attack Countermeasure

A cryptographic scheme may be secure from the theoretical point of view, however, there is more to look at 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 the 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 if 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 to compute an isogeny on a random point. The uniqueness of this attack is that 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 totally 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 user's basis points. Otherwise, if we know the isogeny value on 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 user A to compute his isogeny ϕ_A on 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. Note that users A and B are generic here, thus, we can symmetrically switch all A's and B's in our description.

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-channels 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 with a keyboard for PIN entry. One more important application is that it is now used in the IEEE 802.11 WLAN standard. There is a number of other applications of PAKE, including its usage in client-server networks.

3 Background

3.1 Isogenies

We provide a brief review of the necessary background information 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 over some finite field $\mathbb{F}_q$ of cardinality q, an *isogeny* ϕ is an algebraic morphism from E_1 to E_2 of the form

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

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 ϕ, denoted $\deg(\phi)$, is its degree as an algebraic morphism. Two elliptic curves are *isogenous* if there exists an isogeny between them.

Given an isogeny $\phi\colon E_1 \to E_2$ of degree n, there exists another isogeny $\hat{\phi}\colon E_2 \to E_1$ of the same degree n satisfying $\phi \circ \hat{\phi} = \hat{\phi} \circ \phi = [n]$ (where $[n]$ is the multiplication by n map). The isogeny $\hat{\phi}$ is called the *dual isogeny* of ϕ.

For any natural number n, 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 the *endomorphism ring* $\mathrm{End}(E)$ to be the set of all isogenies from elliptic cuvre E to itself, defined over the algebraic closure $\bar{\mathbb{F}}_q$ of $\mathbb{F}_q$. The endomorphism ring is a ring under the operations of pointwise addition and functional composition. If $\dim_{\mathbb{Z}}(\mathrm{End}(E)) = 2$, then we say that E is *ordinary*; otherwise $\dim_{\mathbb{Z}}(\mathrm{End}(E)) = 4$ and we say that E is *supersingular*. Two curves, between which there exists an isogeny, are either both ordinary or both supersingular. All elliptic curves used in this work are supersingular. The size of the kernel of that isogeny is equal to the degree of that isogeny (as an algebraic map) [14, III.4.10(c)]. The kernel uniquely defines the isogeny up to isomorphism. Methods 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 to perform 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

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

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

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 $\{\phi_A(P_B), \phi_A(Q_B)\} \subset E_A$ 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 $\phi_B(P_A), \phi_B(Q_A) \in E_B$ 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 symetrically. Alice and Bob can then use the common j-invariant of

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

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 initiator and different one for 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 in order 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 smooth order) to recover K_A. The same applies for Bob. The side-channel attack presented in [17] tries to force the user to compute their isogeny on own basis points.

Remark 2. The textbook version of SIDH protocol assumes that the secret scalars are m, n, 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 that we present 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 class group is commutative. It is an interesting solution. One thing to note is that due to its commutativity, 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 "more 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 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*) keypair 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: The user 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 $\phi_A \colon E \to E_A = E/\langle K_A \rangle$.
5. Obtain E_B using the kernel $\langle K_B \rangle$ for the isogeny $\phi_B \colon E \to E_B = E/\langle K_B \rangle$.
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 user either sends or publishes the 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 user'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. 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.

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

Fig. 2: PQDH Diagram

In the diagram, the densely dotted edges represent the secret isogenies of one user and the loosely dashed edges represent the secret isogenies of the other user. 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 exactly 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 exactly on the same assumptions as SIDH. We can see that breaking the scheme is equivalent to breaking two instances of SIDH. This means that the security of the scheme is about the same as of SIDH. In fact, the complexity of breaking the scheme is actually two times the complexity of SIDH, meaning that the same parameters yield us a scheme which has one extra bit of security compared to SIDH. This extra bit applies to both conventional and quantum security.

5 Isogeny-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 is based on PQDH and is easier to use in practice, but it has less security features. Hence, it is an alternative, rather than an improvement. We will also provide complexity analysis and brief security analysis.

5.1 Protocol

Let the parties be U_1 and U_2. Let E be the starting curve. Let pwd be the shared password. We also define hash function H to be a secure (pre-image and collision resistant) hash function mapping to $\mathbb{Z}_{\ell_B^{e_B}}$.

Key Generation: Each party U_i performs the following:

1. Selects random $n_i \in \mathbb{Z}_{\ell_A^{e_A}}$.
2. Computes $K_{A_i} = P_A + [n_i]Q_A$.
3. Computes $\phi_{A_i} \colon E \to E_{A_i} = E/\langle K_{A_i} \rangle$.
4. Evaluates $\phi_{A_i}(P_B)$ and $\phi_{A_i}(Q_B)$.

As usual, E_{A_i}, $P_i = \phi_{A_i}(P_B)$ and $Q_i = \phi_{A_i}(Q_B)$ is the public key. At this point, the parties exchange their public keys.

Key Derivation: On obtaining the other party's (U_{1-i}) public key, $\{E_{A_{1-i}}, P_{1-i}, Q_{1-i}\}$, the party U_i performs the following steps to derive the common key:

1. Computes $n_p = H(\text{pwd}\|j(E_{A_i}) \oplus j(E_{A_{1-i}}))$.
2. Computes $K_{A_iB} = \phi_{A_i}(P_B) + [n_p]\phi_{A_i}(Q_B)$.
3. Computes $E_{A_iB} = E_{A_i}/\langle K_{A_iB} \rangle$.

4. Computes $K_{A_{1-i}B} = \phi_{A_{1-i}}(P_B) + [n_p]\phi_{A_{1-i}}(Q_B)$.
5. Computes $E_{A_{1-i}B} = E_{A_{1-i}}/\langle K_{A_{1-i}B}\rangle$.
6. Evaluates $j_1 = j(E_{A_iB})$ and $j_2 = j(E_{A_{1-i}B})$.
7. Computes the session key $k = KDF(j_1 + j_2)$.

Diagram We present the diagram of the protocol.

$$\begin{tikzcd}
E_{A_2B} & E_B \arrow[l, dashed] \arrow[r, dotted] & E_{A_1B} \\
E_{A_2} \arrow[u, dotted] & E \arrow[l, dashed] \arrow[r, dotted] \arrow[u] & E_{A_1} \arrow[u, dashed]
\end{tikzcd}$$

Fig. 3: Isogeny-Based PAKE Protocol

In this diagram, the densely dotted edges represent the isogenies computed by one user and the loosely dashed edges represent the isogenies computed by the other user, while the solid line is the isogeny based on the shared secret password. The red curves are the common derived secret values used for the common secret key.

Complexity In this protocol, the parties exchange standard public keys. Thus, the overhead is equal to the overhead of the original SIDH protocol. Looking at the computational complexity, SIDH requires performing two isogeny computations, while in the given protocol it is done three times, meaning that its complexity is 1.5 times higher. Comparing to the computational complexity of PQDH, our PAKE protocol's complexity is 3/4 of the former.

Security For the given scheme, we can consider obtaining of E_{A_2B} by the first user and of E_{A_1B} by the second user as two separate events. Thus, each party uses a common known (secret to everyone else) isogeny based on the shared secret password and other party's public key. This means that they gain no extra information about each other's secret keys, identical to what happens in SIDH. Looking from an outside party's perspective, and given that the shared secret information (password, and isogeny computations based on password) is never transmitted over a public channel, we can see that external parties observe even less information than in the SIDH protocol, as only $\ell_A^{e_A}$-isogenies evaluated on B-basis are transmitted, while in SIDH, one also sees $\ell_B^{e_B}$-isogenies evaluated on A-basis.

Next, we need to observe the possibility of offline dictionary attacks for the PAKE protocol. An offline dictionary attack occurs when an adversary observes and records the communication between entities, and then tries every possible password to attempt to guess which password was used. In order for the scheme to be secure, adversary should not be able to distinguish whether the attempted password was the one used or not. In the provided scheme, no public or private information about the shared secret is transmitted. Moreover, the public information about users' own keys that is being exchanged, has no relationship to the shared secret and the results of the computation are not displayed publicly. The adversary has no way to check if the guess was correct as he does not see information that would provides that possibility; no password related/involved information is transmitted and no information against which password guessing can be checked is available to the adversary. Hence, we can see that the given protocol is also secure against the offline dictionary attack.

We do note that this scheme was not intended to provide forward secrecy, compared to isogeny-based PAKE protocol in [16]. This is an alternative, which is more efficient, at the cost of forward secrecy.

Textbook Version There is a textbook version of the protocol for easier understanding and seeing how it is related to the PQDH protocol. This scheme is not as efficient as the one presented above. The key generation process is identical and the key derivation process is as follows:

1. Compute $n_p = H(\text{pwd}\|j(E_{A_i}) \oplus j(E_{A_{1-i}}))$.
2. Compute $K_B = P_B + [n_p]Q_B$.
3. Compute $\phi_B \colon E \to E_B = E/\langle K_B\rangle$.
4. Evaluate $\phi_B(P_A)$ and $\phi_B(Q_A)$.
5. Compute $K_{A_iB} = \phi_{A_i}(P_B) + [n_p]\phi_{A_i}(Q_B)$.
6. Compute $E_{A_iB} = E_{A_i}/\langle K_{A_iB}\rangle$.
7. Compute $K_{A_{1-i}B} = \phi_{A_{1-i}}(P_B) + [n_p]\phi_{A_{1-i}}(Q_B)$.
8. Compute $E_{A_{1-i}B} = E_{A_{1-i}}/\langle K_{A_{1-i}B}\rangle$.
9. Evaluate $j_1 = j(E_{A_iB})$ and $j_2 = j(E_{A_{1-i}B})$.
10. Compute the session key $k = KDF(j_1 + j_2)$.

5.2 Adapting the PAKE Scheme for Devices with Computational Limitations

The presented PAKE scheme can be modified, if required by an application, to shift more load on precomputations, thus deceasing the load of the actual computation during the key establishment phase.

The scheme would be as follows.

Key Generation: Each party U_i performs the following:

1. Selects random $n_i \in \mathbb{Z}_{\ell_A^{e_A}}$.
2. Computes $K_{A_i} = P_A + [n_i]Q_A$.
3. Computes $\phi_{A_i} \colon E \to E_{A_i} = E/\langle K_{A_i}\rangle$.
4. Evaluates $\phi_{A_i}(P_B)$ and $\phi_{A_i}(Q_B)$.

As usual, E_{A_i}, $P_i = \phi_{A_i}(P_B)$ and $Q_i = \phi_{A_i}(Q_B)$ is the public key.

Precomputation Phase: Each party U_i performs the following:

1. Computes $n_p = H(\text{pwd})$.
2. Computes $K_{A_iB} = \phi_{A_i}(P_B) + [n_p]\phi_{A_i}(Q_B)$.
3. Computes $E_{A_iB} = E_{A_i}/\langle K_{A_iB}\rangle$.
4. Evaluates $j_1 = j(E_{A_iB})$.

At this point, the parties exchange their public keys.

Key Derivation: On obtaining the other party's (U_{1-i}) public key, $\{E_{1-i}, P_{1-i}, Q_{1-i}\}$, the party U_i performs the following steps to derive the common key:

1. Computes $K_{A_{1-i}B} = \phi_{A_{1-i}}(P_B) + [n_p]\phi_{A_{1-i}}(Q_B)$.
2. Computes $E_{A_{1-i}B} = E/\langle K_{A_{1-i}B}\rangle$.
3. Evaluates $j_2 = j(E_{A_{1-i}B})$.
4. Computes the session key $k = KDF(j_1 + j_2)$.

In this variant, we have removed the values of the j-invariants of the curves for obtaining n_p. This might make some formal proof techniques more involved. also there should be an increased caution about the aspect of reusing some data, which becomes possible in this adaptation of the scheme. This could be a preferred or an unwanted feature. Thus, this variant should be chosen only if it meets the criteria for the application it is used for.

The diagram for this variant stays the same, as on the overall level, we are doing almost the same operations, except that we are moving some of them to the precomputational stage.

We will discuss complexity and security of this variant with respect to the first variant.

Complexity The overhead complexity with respect to the first variant does not change, as the parties are still exchanging the public keys of the same format. For the computational complexity, because two of three isogeny computations are now being precomputed, we get a complexity of $1/3$ of the first variant, or $1/2$ of the original SIDH protocol. Comparing to the computational complexity of PQDH, our PAKE protocol's complexity is $1/4$ of the former.

Security We are still conceptually doing the same steps as the in the first variant, except moving some of them, where possible, to the precomputation stage, and not including j-invariants in the computation of n_p. Hence, we get the same security as for the first variant, modulo a possibility for replay attacks, where applicable. As already mentioned, since we have removed the values of the curves from the hash function used for obtaining n_p, it might make some formal proof techniques more involved. We recommend an easy way to overcome these issues is to use counters, i.e. $n_p = H(\text{pwd}||\mathit{counter})$, if such security property is required for the given computationally limited device.

6 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 2.3. A direct countermeasure to this attack is to check the order of the other user'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 points in the basis, we could have a cost of 200 percent. In this section, we propose a more efficient approach to providing such countermeasure.

6.1 Efficient Approach for SIDH

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

Remark 4. We recommend in this case to choose $\phi(P), \phi(Q)$ as values of S, T, respectively. This would also prevent a fault-injection attack that could occur during the course of computation itself.

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 $S_2 = T_2 = \infty$, in which case this means that $S, T \in E[\ell^e]$ or $S_2 = -T_2 \neq \infty$, in which case this means that $S, T \in E \setminus E[\ell^e]$. ■

The above claim 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. Another application of this approach is that when we are using Montgomery curves, we could already be provided with $S, T, S - T$, in which case we can choose to verify $S - T$ and one of S or T.

Perform the following:

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

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

Compared to the general countermeasure (computing the orders of points) approach, this one is expected to be about 10 times faster, depending on the curve format.. We avoid expensive computation of 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 Sec. 7.1 show that the cost of this countermeasure is approximately 22-24 percent.

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 $i \in \{0, 1, \ldots, e\}$.

6.2 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, but 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 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 if we are using a multiplication-based, isogeny-based, or optimal strategy. These three strategies can be found 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 user uses both bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$ to compute isogenies. Hence, while computing isogenies with the corresponding kernels, the 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 with respect to the entire computation.

As a result, the proposed countermeasure for this attack at nearly free cost of computation. Though the likelihood of this side-channel attack being applied to PQDH is lower than for SIDH, given that this attack still could happen, we should take advantage of this almost free countermeasure, presented in this paper.

7 Implementation

We implemented PQDH and PAKE, 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, `PAKE` is added as defined in 2 and Algorithm 3, and is implemented in the following variants:

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

· PAKE: The efficient PAKE version as described in Section 5.
· PAKE-P: The PAKE version with precomputations for computationally limited devices and for faster key derivation. Described in Section 5.2.
· PAKE-TB: The PAKE textbook version entirely based on PQDH, as described in Section 5.
· PAKE-TB-P: The PAKE textbook version with precomputations for computationally limited devices and for faster key derivation.

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)

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

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

7.1 Evaluation Results

The performance of PQDH, PAKE and the countermeasures was 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, PAKEp503, PAKEp751.

Table 1 shows the performance of SIDH, PQDH and PAKE. All schemes are evaluated using (a) no countermeasures, (b) the countermeasure from Sec. 6.1 is applied to the KeyDer phase of all schemes, and (c) the countermeasure from Sec. 6.2 is applied to PQDH.

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. PAKE KeyGen in 3.7 msec. and 10.4 msec., respectively, and PAKE KeyDer in 7.4 msec and 20.7 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.

Algorithm 2: Algorithms used for PAKE:
PAKE = (PAKEGen, PAKEDer)
PAKE-P = (PAKEGen, PAKEPrecomp, PakeDer-P)
PAKE-TB = (PAKEGen, PAKEDer-TB)
PAKE-TB-P = (PAKEGen, PAKEPrecomp-TB, PAKEDer-P)

```
1  function PAKEGen
       Input: ()
       Output: $(sk, pk)$
2      $sk \leftarrow_R \mathcal{K}_2$
3      $pk \leftarrow$ isogen$_2(sk)$
4      return $(sk, pk)$

5  function PAKEDer
       Input: $(pk, pk', pwd)$
       Output: $(K)$
6      $n_p$ = pake_common_param$(pk, pk', pwd)$
7      $j_2 \leftarrow$ isoex$_3(pk, n_p)$
8      $j'_2 \leftarrow$ isoex$_3(pk', n_p)$
9      $K \leftarrow j_2 \oplus j'_2$
10     return $K$

11 function PAKEPrecomp
       Input: $(pk, pwd)$
       Output: $(j_2, n_p)$
12     $n_p \leftarrow H(pwd)$
13     $j_2 \leftarrow$ isoex$_3(pk, n_p)$
14     return $(j_2, n_p)$

15 function PAKEDer-P
       Input: $(n_p, pk', j)$
       Output: $(K)$
16     $j'_2 \leftarrow$ isoex$_3(pk', n_p)$
17     $K \leftarrow j \oplus j'_2$
18     return $K$

19 function PAKEPrecomp-TB
       Input: $(sk, pwd)$
       Output: $(j_3, n_p)$
20     $n_p \leftarrow H(pwd)$
21     $pk_3 \leftarrow$ isogen$_3(n_p)$
22     $j_3 \leftarrow$ isoex$_2(pk_3, sk)$
23     return $(j_3, n_p)$

24 function PAKEDer-TB
       Input: $(sk, pk, pk', pwd)$
       Output: $(K)$
25     $n_p$ = pake_common_param$(pk, pk', pwd)$
26     $pk'' \leftarrow$ isogen$_3(n_p)$
27     $j \leftarrow$ PQDHDer$((sk, n_p), (pk, pk''))$
28     $K \leftarrow j$
29     return $K$
```

Algorithm 3: Establishing common parameters for PAKE

```
function pake_common_param
    Input: Password $pwd$, public keys $pk = (x_1, x_2, x_3) \in (\mathbb{F}_{p^2})^3$, and
           $pk' = (x'_1, x'_2, x'_3) \in (\mathbb{F}_{p^2})^3$, $e_A$
    Output: A $j$-invariant $j_2$
1 $(A : C) \leftarrow ($get_A$(x_1, x_2, x_3) : 1)$            // Alg. 10 in [10]
2 $j \leftarrow$ j_inv$(A, C)$                                  // Alg. 9 in [10]
3 $(A' : C') \leftarrow ($get_A$(x'_1, x'_2, x'_3) : 1)$       // Alg. 10 in [10]
4 $j' \leftarrow$ j_inv$(A', C')$                               // Alg. 9 in [10]
5 $n_p \leftarrow H(pwd \| j \oplus j')$    // H is, e.g. SHAKE256 with output length
    eA bits
6 return $n_p$
```

The `PAKE KeyGen` performance is equivalent to `SIDH KeyGen A`. This holds among all `PAKE` variants. The `PAKE KeyDer` phase takes approx. 10% more cycles than the respective phase in `PQDH`. It basically consists of the `pake_common_param` plus twice the SIDH `KeyDerB` step.

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

The evaluations of the four PAKE variants are depicted in Table 2. The `KeyGen` steps of all variants are equivalent. The textbook version `PAKE-TB` KeyDer includes one SIDH `KeyDer B`, one PQDH `KeyDer`, and one `pake_common_param` step which leads to a performance-tradeoff of 45%-50% compared to `PAKE`. An advantage of the `PAKE-P` version is that it halves the `KeyDer` complexity because it moves parts of its computation to a `Precomp` phase. In practice, the ephemeral `KeyGen` and `Precomp` step can be combined and performed offline, while the device is otherwise idling, thus minimizing the more time-critical `KeyDer` phase. The same can be said about the textbook version with precomputation `PAKE-TB-P`, where two-thirds of the `KeyDer` phase can be moved to the `Precomp` phase.

Key sizes The sizes of PQDH private and public key are double the size of the SIDH keys. The PAKE keys are the same size and structure of the SIDH keys. Note that net size of the 2-torsion private key of `SIDHp751` and `PAKEp751` 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 3.

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 which allow 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 quantum-resistant scheme with such property. This will be very helpful in transitioning from conventional to post-quantum cryptography. Also, we have shown a new isogeny-based PAKE protocol, which provides efficiency at the cost of forward secrecy with respect to the PAKE protocol presented in [16], and thus provides more applicable options with possible tradeoffs. Finally, we have presented an efficient countermeasure against the side-channel fault attack. The attack is applicable not only to the static version of SIDH and PQDH, but also to the ephemeral one, which makes this countermeasure highly desirable.

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	
PAKEp503	10’049		19’496	
PAKEp751	28’164		55’524	
Efficient countermeasure for SIDH (Sec. 6.1)				
SIDHp503	-		10’134	11’815
SIDHp751	-		28’566	34’444
PQDHp503	-		21’945	
PQDHp751	-		62’187	
PAKEp503	-		24’260	
PAKEp751	-		68’735	
Efficient countermeasure for PQDH (Sec. 6.2)				
PQDHp503	-		18’060	
PQDHp751	-		50’234	

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

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.

Scheme	KeyGen	Precomp	KeyDer
P503			
PAKE	10’049	-	20’016
PAKE-TB		-	29’285
PAKE-P		9’411	9’348
PAKE-TB-P		19’529	
P751			
PAKE	28’164	-	55’909
PAKE-TB		-	83’504
PAKE-P		27’302	27’101
PAKE-TB-P		54’983	

Table 2: Performance of PAKE variants: PAKE, PAKE-TB (textbook), PAKE-P (with precomputation), PAKE-TB-P (textbook with precomputation). In thousands of cycles.

Scheme	**private key** sk	**public key** pk	**shared secret** ss
SIDHp503	32	378	126
SIDHp751	48	564	188
PQDHp503	64	756	126
PQDHp751	96	1128	188
PAKEp503	32	378	126
PAKEp751	48	564	188

Table 3: Private/public key, and shared secret sizes of SIDH, PQDH, and PAKE in bytes.

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. Math. Cryptol.*, (to appear). `http://eprint.iacr.org/2011/506`.
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. 6.1 is added as in Algorithm 4. The optimal countermeasures for SIDH from Sec. 6.1 are integrated to $\texttt{isoex}_2$ and $\texttt{isoex}_3$. They are defined in Algorithms 7-8. For the PQDH countermeasure from Sec. 6.2, the optimal tree traversal strategies for computing isogenies are modified as defined in Algorithms 5-6.

Algorithm 4: Differential addition

```
function xADD
    Input: $P = (X_P : 1)$, $Q = (X_Q : 1)$, $PQ = (X_{Q-P} : 1)$
    Output: $(X_{P+Q} : Z_{P+Q})$
1  $t_0 \leftarrow X_P + 1$
2  $t_1 \leftarrow X_P - 1$
3  $X_{P+Q} \leftarrow X_Q - 1$
4  $Z_{P+Q} \leftarrow X_Q + 1$
5  $t_0 \leftarrow X_{P+Q} \cdot t_0$
6  $t_1 \leftarrow Z_{P+Q} \cdot t_1$
7  $Z_{P+Q} \leftarrow t_0 - t_1$
8  $X_{P+Q} \leftarrow t_0 + t_1$
9  $Z_{P+Q} \leftarrow Z_{P+Q}^2$
10 $X_{P+Q} \leftarrow X_{P+Q}^2$
11 $Z_{P+Q} \leftarrow X_{Q-P} \cdot Z_{P+Q}$
12 return $(X_{P+Q} : Z_{P+Q})$
```

Algorithm 5: Computing and evaluating a 2^e-isogeny, with the countermeasure for PQDH

```
function 2_e_iso_pqdh
    Static parameters: Integer $\mathsf{e}_2$ from the public parameters, a strategy $(s_1, \ldots, s_{\mathsf{e}_2/2-1}) \in (\mathbb{N}^+)^{\mathsf{e}_2/2-1}$
    Input: Constants $(A_{24}^+ : C_{24})$ corresponding to a curve $E_{A/C}$, $(X_S : Z_S)$ where $S$ has exact order $2^{\mathsf{e}_2}$ 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 $E_{A'/C'}$
1  Initialize empty deque S
2  push(S, $(\mathsf{e}_2/2, (X_S : Z_S))$)
3  $i \leftarrow 1$, $j \leftarrow 0$
4  while S not empty do
5      $(h, (X : Z)) \leftarrow$ pop(S)
6      if $h = 1$ then
7          if $j = 0$ then
8              $(X_d : Z_d) \leftarrow$ xDBL$\big((X : Z), (A_{24}^+ : C_{24})\big)$            // Alg.3 [10]
9              if $Z_d = 0$ then
10                 Error: Potential fault attack
11             $(X_d : Z_d) \leftarrow$ xDBL$\big((X_d : Z_d), (A_{24}^+ : C_{24})\big)$            // Alg.3 [10]
12             if $Z_d \neq 0$ then
13                 Error: Potential fault attack
14         $\big((A_{24}^+ : C_{24}), (K_1, K_2, K_3)\big) \leftarrow$ 4_iso_curve$((X : Z))$  // Alg.11 [10]
15         Initialize empty deque S′
16         while S not empty do
17             $(h, (X : Z)) \leftarrow$ pull(S)
18             $(X : Z) \leftarrow$ 4_iso_eval$((K_1, K_2, K_3), (X : Z))$       // Alg.12 [10]
19             push(S′, $(h - 1, (X : Z))$)
20         S ← S′
21         for $(X_j : Z_j)$ in optional input do
22             $(X_j : Z_j) \leftarrow$ 4_iso_eval$\big((K_1, K_2, K_3), (X_j : Z_j)\big)$  // Alg.12 [10]
23         $j \leftarrow 1$
24     else if $0 < s_i < h$ then
25         push(S, $(h, (X : Z))$)
26         $(X : Z) \leftarrow$ xDBLe$\big((X : Z), (A_{24}^+ : C_{24}), 2 \cdot s_i\big)$            // Alg.4 [10]
27         push(S, $(h - s_i, (X : Z))$)
28         $i \leftarrow i + 1$
29     else
30         Error: Invalid strategy
31 return $(A_{24}^+ : C_{24})$, $[(X_1 : Z_1), (X_2 : Z_2), (X_3 : Z_3)]$
```

Algorithm 6: 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 $\mathbf{S}$
push($\mathbf{S}$, ($\mathtt{e}_3$, $(X_S : Z_S)$))
$i \leftarrow 1$, $j \leftarrow 0$
while $\mathbf{S}$ not empty do
    $(h, (X : Z)) \leftarrow$ pop($\mathbf{S}$)
    if $h = 1$ then
        if $j = 0$ then
            if $Z = 0$ then
                Error: Potential fault attack
            $(X_d : Z_d) \leftarrow$ xTPL$\left((X : Z), (A_{24}^+ : A_{24}^-)\right)$        // Alg. 6 [10]
            if $Z_d \neq 0$ then
                Error: Potential fault attack
        $\left((A_{24}^+ : A_{24}^-), (K_1, K_2)\right) \leftarrow$ 3_iso_curve$((X : Z))$    // Alg. 13 [10]
        Initialize empty deque $\mathbf{S}'$
        while $\mathbf{S}$ not empty do
            $(h, (X : Z)) \leftarrow$ pull($\mathbf{S}$)
            $(X : Z) \leftarrow$ 3_iso_eval$((K_1, K_2), (X : Z))$        // Alg. 14 [10]
            push($\mathbf{S}'$, $(h - 1, (X : Z))$)
        $\mathbf{S} \leftarrow \mathbf{S}'$
        for $(X_j : Z_j)$ in optional input do
            $(X_j : Z_j) \leftarrow$ 3_iso_eval$\left((K_1, K_2), (X_j : Z_j)\right)$    // Alg. 14 [10]
        $j \leftarrow 1$
    else if $0 < s_i < h$ then
        push($\mathbf{S}$, $(h, (X : Z))$)
        $(X : Z) \leftarrow$ xTPLe$\left((X : Z), (A_{24}^+ : A_{24}^-), s_i\right)$        // Alg. 7 [10]
        push($\mathbf{S}$, $(h - s_i, (X : Z))$)
        $i \leftarrow i + 1$
    else
        Error: invalid strategy
return $(A_{24}^+ : A_{24}^-)$, $[(X_1 : Z_1), (X_2 : Z_2), (X_3 : Z_3)]$
```

Algorithm 7: 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 $j_2$
1  $(A : C) \leftarrow$ (get_A$(x_1, x_2, x_3)$ : 1)                          // Alg. 10 [10]
2  $(A^+_{24} : C_{24}) \leftarrow (A + 2 : 4)$
3  $Q \leftarrow$ xADD$(x1, x2, x3)$                                         // Alg. 4
4  $Q \leftarrow$ xDBLe$(Q, (A^+_{24} : C_{24}), e2 - 1)$                     // Alg. 4 [10]
5  if $Z_Q = 0$ then
6      Error: Potential fault attack
7  $Q \leftarrow$ xDBL$(Q, (A^+_{24} : C_{24}))$                              // Alg. 3 [10]
8  if $Z_Q \neq 0$ then
9      Error: Potential fault attack
10 $Q \leftarrow$ xDBLe$((x1 : 1), (A^+_{24} : C_{24}), e2 - 1)$              // Alg. 4 [10]
11 if $Z_Q = 0$ then
12     Error: Potential fault attack
13 $Q \leftarrow$ xDBL$(Q, (A^+_{24} : C_{24}))$                              // Alg. 3 [10]
14 if $Z_Q \neq 0$ then
15     Error: Potential fault attack
16 $(X_S : Z_S) \leftarrow$ Ladder3pt$(\mathrm{sk}_2, (x_1, x_2, x_3), (A : C))$  // Alg. 8 [10]
17 $(A^+_{24} : C_{24}) \leftarrow$ 2_e_iso$\Big((A^+_{24} : C_{24}), (X_S : Z_S)\Big)$  // Alg. 15 [10]
18 $(A : C) \leftarrow (4A^+_{24} - 2C_{24} : C_{24})$
19 $j$ = j_inv$((A : C))$                                                    // Alg. 9 [10]
20 return $j$
```

Algorithm 8: 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 $j_3$
1  $(A : C) \leftarrow$ (get_A($x_1, x_2, x_3$): 1)                          // Alg. 10 [10]
2  $(A^+_{24} : A^-_{24}) \leftarrow (A + 2 : A - 2)$
3  $Q \leftarrow$ xADD($x1, x2, x3$)                                        // Alg. 4
4  $Q \leftarrow$ xTPLe($Q, (A^+_{24} : A^-_{24}), e3 - 1$)                  // Alg. 7 [10]
5  if $Z_Q = 0$ then
6      Error: Potential fault attack
7  $Q \leftarrow$ xTPL($Q, (A^+_{24} : A^-_{24})$)                           // Alg. 6 [10]
8  if $Z_Q \neq 0$ then
9      Error: Potential fault attack
10 $Q \leftarrow$ xTPLe($(x1 : 1), (A^+_{24} : A^-_{24}), e3 - 1$)           // Alg. 7 [10]
11 if $Z_Q = 0$ then
12     Error: Potential fault attack
13 $Q \leftarrow$ xTPL($Q, (A^+_{24} : A^-_{24})$)                           // Alg. 6 [10]
14 if $Z_Q \neq 0$ then
15     Error: Potential fault attack
16 $(X_S : Z_S) \leftarrow$ Ladder3pt($\mathrm{sk}_3, (x_1, x_2, x_3), (A : C)$)   // Alg. 8 [10]
17 $(A^+_{24} : A^-_{24}) \leftarrow$ 3_e_iso$\Big((A^+_{24} : A_{24}), (X_S : Z_S)\Big)$   // Alg. 16 [10]
18 $(A : C) \leftarrow (2 \cdot (A^-_{24} + A^+_{24}) : A^+_{24} - A^-_{24})$
19 $j$ = j_inv($(A : C)$)                                                    // Alg. 9 [10]
20 return $j$
```