Post-quantum Cryptography: Analysis of Supersingular Isogeny Key Encapsulation (`SIKE`)

Yannick Bormuth[*], Dario Kermanschah[†], Michael Burkhalter[‡], Lauro Böni[§]

September 2, 2020

Abstract

This report aims to give thorough insights and analyses of Supersingular Isogeny Key Exchange (`SIKE`), one of the remaining candidates for public key encryption in NIST's selection for quantum-safe cryptographic standards. We will motivate its importance in post-quantum cryptography, introduce its mathematical foundations and discuss its strengths and weaknesses. We present known attacks including potential quantum speed-ups, computational resource requirements and put it in perspective with current cryptographic standards. A Python script implementing the key features of isogeny based cryptography for a simple set of parameters is provided along with this report.

Key Words: Post Quantum Cryptography, Supersingular Isogeny Key Exchange, Elliptic Curves, Public-key Exchange

[*] *Email address:* yannick.bormuth@gmail.com
[†] *Email address:* dario.kermanschah@gmail.com
[‡] *Email address:* michael.burckhalter@protonmail.com
[§] *Email address:* laboeni@gmail.com

Contents

1 Introduction

While awaiting the advent of large scale quantum computers the cryptographers' community started in the first decade of the third millennium the search for cryptographic algorithms that can whithstand attacks not only from classical computers but from quantum computers as well. Such algorithms are the research object of post-quantum cryptography and are also known as quantum-proof, quantum-safe, or quantum-resistant algorithms. The need for novel public key primitives is undisputed as it is known that the present most popular public-key algorithms are not quantum-safe.

The National Institute of Standards and Technology (NIST) acknowledged this need and *initiated a process to solicit, evaluate, and standardize one or more quantum-resistant public-key cryptographic algorithms* ([13]) in 2016. There were 17 public-key encryption and key-establishment (PKE/KEM) algorithms and 9 digital signature algorithms (DSA) in NIST's second round. In July 2020, NIST announced the 7 finalists - of which 4 on PKE/KEM and 3 on DSA - for round 3. The finalists will continue to be reviewed for consideration for standardization at the conclusion of the third round. In addition, eight alternate candidates selected for a potential fourth round of evaluation. Many of these alternate candidates have worse performance than the finalists but might be selected for standardization based on a high confidence in their security.

In this project, we focus on one of the alternate candidates for PKE/KEM, namely `SIKE` (Supersingular Isogeny Key Encapsulation). The choice of this protocol was made before the announcement of the round 3 finalists, see [14].

2 Supersingular Isogeny Diffie-Hellman Key Exchange

Consider the following cryptographic problem: Two parties want to communicate securely but are only given a public channel. How can they exchange a secret between them? Asymmetric cryptography provides us with a solution. Each of the two parties chooses a private key for themselves and computes a public key based on some public parameters and their own private key. While one party's public key is accessible for everyone, it is only useful for the other party. In fact, using the other party's public key together with their own private key, they can compute a secret, shared between the two parties. The computations both parties have to perform are based on a mathematical problem that generates one-way functions. These one-way functions are the reason a secure key exchange is possible. More precisely, a one-way function guarantees that it is computationally unfeasible for an outsider to compute the two party's shared secret, not even just part of it, given public parameters only. On the other hand, it is comparably easy for the two parties to compute their shared secret, as they not only know the public parameters but also have *direct* access to their own secret key.

The Supersingular Isogeny Diffie-Hellman (`SIDH`) key exchange is one particular construction of such an asymmetric cryptographic algorithm. Before we dive into the rather overwhelming mathematics of supersingular elliptic curves and isogenies between them, we first want to motivate some of the terminology of `SIDH`. As the name suggests, `SIDH` works in an analogous fashion as the conventional Diffie-Hellman (`DH`) key exchange. To introduce the new concepts of `SIDH`, it can therefore help to directly link existing notions in the DH protocol with their corresponding counterparts in SIDH.

Overview of the DH key exchange. Let us begin by repeating the mathematical foundations of the DH key exchange that relies on finite field arithmetics. The protocol requires two parties, Alice and Bob, to agree on a prime number p defining a work space, the multiplicative group of integers modulo p, denoted as $(\mathbb{Z}/p\mathbb{Z})^{\times}$. Alice and Bob also agree on a special public element in this group, one of its *generators* g. The protocol then requires Alice to choose a secret key, an integer $1 < a < p$, and to compute a public key g^a. Analogously, Bob chooses his secret key $1 < b < p$ and computes his public key g^b. After exchanging the public keys, the two parties can then compute their shared secret g^{ab} by exponentiating the other party's public keys with their own secret key.

General cryptographic requirements. The DH protocol, as well as any asymmetric cryptographic protocol as e.g. SIDH, has to satisfy three key properties, such that it provides a practical and secure key exchange: 1. the shared secret cannot be computed in reasonable time given the public parameters, 2. the shared secret is reasonably easy to compute for one party given their secret key and 3. the size of the parameters of the protocol are such that it takes a reasonable amount of resources to exchange and to store them. What it means to be reasonable in this context is subject to the state of the art regarding computational resources and can therefore change with time.

The development of quantum computers has given rise to new algorithms that have faster run times than their classical counterparts. More specifically, the problem of integer factorisation can currently be solved classically in sub-exponential-time by means of a highly optimised General Number Field Sieve algorithm. Shor's algorithm however, provides a polynomial-time quantum solution. In fact, the quantum speed up lies in a period-finding subroutine. Since the security of many popular cryptographic protocols, as the DH key exchange, rely on the fact that period-finding is a hard problem (for DH see discrete logarithm problem), quantum computers pose a serious security risk for current cryptographic standards. The development of quantum-safe cryptographic protocols is therefore a necessity to maintain uncompromised privacy standards.

While the conventional DH protocol is clearly vulnerable by quantum algorithmic attacks, the SIDH key exchange promises to provide a quantum-safe security protocol.

SIDH in a nutshell. In simple words, the core idea of the SIDH protocol is to define a graph on the set of supersingular j-invariants (which uniquely characterize isomorphic elliptic curves). The vertices are given by the different j-invariants and the edges are given by the respective isogenies (functions with some desirable properties mapping points from one elliptic curve to another). Alice and Bob start on separate (public) generators $P_A, Q_A \in E_0$ and $P_B, Q_B \in E_0$, respectively on a pre-defined (and public) initial curve E_0. They linearly combine these two points by a (secret) constant to obtain (secret) points S_A and S_B, respectively. These two points are then be mapped (secretly) to new elliptic curves (characterized by their j-invariants) by the use of the isogenies. After some iterations, they exchange their curves and the mapping of their starting points and redo the whole iteration again. Finally, they land on the same j-invariant, which then is the shared secret of the protocol.

Similarities between SIDH and DH. As in the DH key exchange, SIDH requires two parties, Alice and Bob, to agree on a prime p, defining a work space, the set of *supersingular* elliptic curves $\mathcal{S}$ over the Galois field $\mathbb{F}_{p^2}$. Therefore, we still rely on finite field arithmetics, however SIDH adds a higher level of operations: maps between elliptic curves, so called isogenies. More precisely, an isogeny is a group homomorphism mapping one point from one elliptic curve to another, $\Phi : E \to E'$, therefore connecting the two elements E and E' within $\mathcal{S}$. Alice and Bob then agree on a public element in this set $\mathcal{S}$, one particular supersingular elliptic curve E_0. The protocol requires Alice to choose a private generator S_A, a point on E_0, which generates the corresponding private isogeny $\Phi_{S_A} : E_0 \to E_0/\langle S_A \rangle$ that computes the elliptic curve $E_0/\langle S_A \rangle$. Analogously, Bob chooses another point on E_0 , his private generator S_B, which generates his private isogeny $\Phi_{S_B} : E_0 \to E_0/\langle S_B \rangle$ that computes $E_0/\langle S_B \rangle$. Alice and Bob then exchange $E_0/\langle S_A \rangle$ and $E_0/\langle S_B \rangle$ and compute their shared secret, corresponding to the elliptic curve $E_0/\langle S_A, S_B \rangle$ using their private keys and the respective isogenies $\Phi_{\Phi_{S_B}(S_A)}$ and $\Phi_{\Phi_{S_A}(S_B)}$.

There is an obvious analogy between the DH and the SIDH protocol. It relies on the correspondence between the work spaces $(\mathbb{Z}/p\mathbb{Z})^\times$ and $\mathcal{S}$, the operations multiplication $\times$ and isogeny Φ and their starting elements g and E_0. However, after only reading this brief outline of the SIDH protocol one would remain with several unanswered questions. The more obvious ones are: How does one find an isogeny $\Phi_S : E_0 \to E_0/\langle S \rangle$ for a given point S on the elliptic curve E_0? What elliptic curve does the quotient $E_0/\langle S \rangle$ refer to? In Section 2.1, we will see that a point S on E_0 contains enough information to find both the isogeny Φ_S and the curve $E_0/\langle S \rangle$. However, this brief summary of the protocol has some more subtleties that we will address now.

Differences between SIDH and DH. While the isogenies Φ_{S_A} and Φ_{S_B} can be computed directly using the public parameters and each party's own private generator S_A or S_B, there appears to be a problem when computing $\Phi_{\Phi_{S_B}(S_A)}$ or $\Phi_{\Phi_{S_A}(S_B)}$ as these isogenies depend on both party's private generators. More explicitly, they depend on the image under the other party's private isogeny $\Phi_{S_B}(S_A)$ or $\Phi_{S_A}(S_B)$. How can Alice and Bob now compute their isogenies without having to know each others private generators, effectively breaking the security of the protocol?

The SIDH protocol provides an answer to this question. The solution is to extend the public parameters with two basis points for each party, P_A, Q_A, P_B, Q_B each on the public curve E_0. Furthermore, each party has to propagate the other party's basis points through its own isogeny and publish them, $\Phi_{S_A}(P_B)$, $\Phi_{S_A}(Q_B)$, $\Phi_{S_B}(P_A)$, $\Phi_{S_B}(Q_A)$, as part of its public key. The public basis points are then linked to the private generators through $S_A = P_A + [k_A]Q_A$ and $S_B = P_B + [k_B]Q_B$, where k_A and k_B are Alice's and Bob's respective private keys and $[k]$ is the multiplication-by-k operation. Having access to each others public keys then allows to compute their own private generator on the other party's curve, using $\Phi_{S_B}(S_A) = \Phi_{S_B}(P_A) + [k_A]\Phi_{S_B}(Q_A)$ and $\Phi_{S_A}(S_B) = \Phi_{S_A}(P_B) + [k_B]\Phi_{S_A}(Q_B)$, which is always holds true since isogenies are group homomorphisms. Extending the public parameters and each party's public key with those basis points therefore guarantees the privacy of their own private generator.

Now, one might wonder why the conventional DH key exchange works as it is, without publishing any more parameters. The answer lies in the fact that the generator g as well as the public keys g^a and g^b lie in the same group, in which both parties know how to operate, i.e. apply multiplication, without having any information about a nor b. Although in the SIDH protocol, the supersingular elliptic curves E_0, $E_0/\langle S_A\rangle$ and $E_0/\langle S_B\rangle$ also lie in the same set $\mathcal{S}$, the parties can only operate, i.e. build their respective isogeny, if they know the image of their private generator S_A or S_B on their current elliptic curve. In other words, the operation within the set of supersingular elliptic curves $\mathcal{S}$ is more involved than the one of the multiplicative group $(\mathbb{Z}/p\mathbb{Z})^\times$.

Outline. The previous overview in its brevity lacks several mathematical and implementational details and concepts that are however necessary requirements for the protocol to be mathematically well-defined and cryptographically applicable. In Section 2.1, we will give a more precise collection of the mathematical concepts of SIDH and list other necessary ingredients, like the isogenies used in SIDH. This will lead us to introduce the notions of the j-invariant and the l-torsion of an elliptic curve, the degree of an isogeny as well as isogeny graphs, visualising the operations in $\mathcal{S}$.

Just like the DH protocol, the practicality and security of the SIDH key exchange rely on the same three properties as mentioned before already: difficult computation of shared secret given the public key, easy computation of a shared secret given one private key, and the small size of stored parameters. We demonstrate the security relevant components to the SIDH protocol in Section 6 while the practical resource requirements, for optimised and more secure versions of SIDH as Supersingular Isogeny Key Encapsulation (SIKE), are discussed in Section 5. Known attacks, both on SIDH and SIKE, are presented in Section4.

2.1 Terminology, Notation and Prerequisites

The following section provides the notation and useful results from classical group theory and/or cryptography required to rigorously understand the SIDH and SIKE protocol.

We will work on a Galois Field $\mathbb{F}_{p^2}$ for some prime p. For the sake of brevity, we can think of this field as the field $\mathbb{F}_p(\mathrm{i})$, where i fulfils $\mathrm{i}^2 + 1 = 0$. Hence, elements of $\mathbb{F}_{p^2}$ can be written as $u + \mathrm{i}v$ for $u, v \in \mathbb{F}_p$.

Definition 1 (Elliptic Curve). An Elliptic Curve E over a finite field $\mathbb{K}$ is the set of solutions $(x, y) \in \mathbb{K} \times \mathbb{K}$ of the equation

$$y^2 = x^3 + ax^2 + x, \tag{1}$$

for $a \in \mathbb{K}$. This representation is referred to as the **Montgomery**[1] form and denoted by E_a or $E_a(\mathbb{K})$ to emphasize the underlying field.

We note that we will work subsequently on the projective field $\mathbb{P}^2_{\mathbb{K}}$ over $\mathbb{K}$. A proper derivation and definition of projective spaces is not in the scope of this report - however,

[1]There exist also other representations such as Weierstrass or Legendre for an elliptic curve.

for the sake of simplicity, the reader can think of the projective space as the field $\mathbb{K}^2$ equipped with a so-called **point at infinity** $\mathcal{O}$, the neutral element with respect to the group law[2], i.e. the addition of points on an elliptic curve E.

Example 1 (point addition, [18], [3])**.** Let us consider the addition of two points $P_1 = (x_1, y_1)$ and $P_2 = (x_2, y_2)$, with $P_1 \neq P_2$ on a given elliptic curve E_a. Then $P_3 = (x_3, y_3) := P_1 + P_2$ has coordinates

$$x_3 = \frac{(y_2 - y_1)^2}{(x_2 - x_1)^2} - a - x_1 - x_2 \tag{2}$$

$$y_3 = \frac{(2x_1 + x_2 + a)(y_2 - y_1)}{(x_2 - x_1)} - \frac{(y_2 - y_1)^3}{(x_2 - x_1)^3} - y_1. \tag{3}$$

The addition has the geometric interpretation that the points P_1, P_2 and $-P_3$, satisfying $P_1 + P_2 - P_3 = \mathcal{O}$, are the intersections with a line and the elliptic curve E_a.

Definition 2 (j-invariant, [3])**.** The j-invariant of an Elliptic Curve E_a in Montgomery form is given by the term

$$j(E_a) = 256\frac{(a^2 - 3)^3}{(a^2 - 4)} \in \mathbb{K}. \tag{4}$$

Note that the computation of $j(E_a)$ is performed in $\mathbb{K}$ and relies on the computation of a multiplicative inverse. For $\mathbb{K} = \mathbb{F}_{p^2}$, the problem of finding a multiplicative inverse in $\mathbb{F}_{p^2}$ can be reduced to finding one in $\mathbb{F}_p$ since

$$\frac{1}{(u + \mathrm{i}v)} = \frac{(u - \mathrm{i}v)}{u^2 + v^2}, \tag{5}$$

where $(u^2 + v^2)^{-1} \in \mathbb{F}_p$ can be computed using the extended Euclidean algorithm.

Theorem 1 (uniqueness of the j-invariant, [17])**.** Every elliptic curve has a unique j-invariant, and two elliptic curves are isomorphic if and only if they have the same j-invariant.

Hence, j-invariants give us a nice way of uniquely characterizing elliptic curves. Furthermore, as we will see in Section 2.2, points on an elliptic curve E with finite order play a crucial role in the study of the `SIDH` and `SIKE` protocol. It is therefore useful to consider the set of points on E with order n, the n-torsion.

Definition 3 (n-torsion, [17], Section 3.1)**.** The n-torsion is given by

$$E[n] := \{P \in E(\overline{\mathbb{K}}) \mid nP = \mathcal{O}\},$$

where $\overline{\mathbb{K}}$ is the algebraic closure of the Galois field $\mathbb{K}$ and $\mathcal{O}$ is the point at infinity in the projective space.

[2]Further details can for example be found in [17], Section 2.3.

Example 2 ([17], Section 3.1)**.** With some assumptions to $\mathbb{K}$ (characteristic[3] of $\mathbb{K}$ is not 2), it follows that $E[2] = \{\mathcal{O}, (e_1, 0), (e_2, 0), (e_3, 0)\}$, since the points $(e_i, 0)$ have an infinite slope on E. As an abstract group, this is isomorphic to $\mathbb{Z}_2 \times \mathbb{Z}_2$.

The next theorem generalizes the results from above example.

Theorem 2 (n-torsion isomorphism, [17], Theorem 3.2)**.** Let E be an elliptic curve over $\mathbb{K}$ and let n be a positive integer. If the characteristic of $\mathbb{K}$ does not divide n, or is 0, then

$$E[n] \cong \mathbb{Z}_n \times \mathbb{Z}_n. \tag{6}$$

Definition 4 (Supersingular Elliptic Curve, [17], Section 3.1)**.** We say that an Elliptic Curve E over a field $\mathbb{K}$ with characteristic p is supersingular, if $E[p] \cong 0$. Otherwise, E is called ordinary and $E[p] \cong \mathbb{Z}_p$.

Supersingular elliptic curves in characteristics p do not have points of order p, even in the algebraic closure $\overline{\mathbb{K}}$. We will work with supersingular curves, as they have many interesting properties. For example, the order (i.e. number of points) of supersingular curves of characteristics p is $(p+1)$ ([17], Corollary 4.32). Also the number of different j-invariants can be calculated explicitly, which is stated in the next theorem.

Theorem 3 ([17], Corollary 4.40)**.** The subset $\mathcal{S} \subseteq \mathbb{F}_{p^2}$ containing all supersingular j-invariants in $\mathbb{F}_{p^2}$ is of size $\kappa := \lfloor \frac{p}{12} \rfloor + z$ for some $z \in \{0, 1, 2\}$.

From Theorem 3 we get a set of κ j-invariants. Together with Theorem 1 we also get a set of different (supersingular) elliptic curves which are unique up to isomorphism. That is, for E_1 such that $j(E_1) \in \mathcal{S}$, we have an isomorphism Ψ to another elliptic curve, say E_2

$$\begin{aligned} \Psi \colon E_1 &\longrightarrow E_2 \\ \Psi^{-1} \colon E_2 &\longrightarrow E_1, \end{aligned}$$

that fulfills $j(E_1) = j(E_2)$. The isomorphism Ψ is a special case of a so-called isogeny. Let us now turn our attention to general isogenies between elliptic curves.

Definition 5 (isogeny (abstract))**.** An **isogeny** between two elliptic curves E_1 and E_2 is a nonconstant homomorphism $\phi\colon E_1 \to E_2$ that is given by rational functions.

In particular, this means that $\phi([\lambda]P + [\mu]Q) = [\lambda]\phi(P) + [\mu]\phi(Q)$ for $P, Q \in E_1$ and the multiplication maps $[\lambda]$ and $[\mu]$ i.e. a linear combination of points on E_1 mapped to the curve E_2 is equivalent to the linear combination of the mapped points in E_2.

Moreover, for $(u, v) = \phi(x, y)$, we get that $u = r_1(x)$ and $v = yr_2(x)$ for two rational functions $r_1(x) = \frac{p(x)}{q(x)}$ and $r_2(x) = \frac{r(x)}{s(x)}$.[4] Based on this derivation, we are now able to define the degree of an isogeny.

[3]The characteristic of a finite field is the smallest integer p such that $\underbrace{(1 + 1 + \cdots + 1)}_{\text{p times}} = 0$.

[4]This argument is elaborated in full detail in [17], Section 12.2.

Definition 6 (degree of an isogeny, [17], Section 12.1)**.** The degree of an isogeny is given by

$$\deg(\phi) = \max(\deg\, p(x), \deg\, q(x)).$$

If the derivative $r_1'(x)$ is not the zero-map, we say that the isogeny is **separable**. In this case, the degree is given by

$$\deg(\phi) = |\ker(\phi)|.$$

Isogenies of degree l will subsequently be denoted by l-isogenies.

We will subsequentially work with separable isogenies as they have nice properties as shown in the following theorem:

Theorem 4 (one-to-one correspondence of isogenies, [3])**.** A separable **isogeny** is in one-to-one correspondence with a finite subgroup, i.e. every subgroup G of points on an elliptic curve E gives rise to a unique isogeny $\phi\colon E \to E'$ whose kernel is G (and vice versa). We can therefore write the codomain as $E' = E/G$.

Theorem 5 (Vélu's formula, [17], Section 12.3)**.** This theorem connects a given group of points on an elliptic curve with separable isogenies such that this group of points is the kernel of the separable isogeny and provides an explicit expression for this isogeny. We do not state the whole theorem and the explicit formula here. Interested readers can consult [17], Section 12.3.

Before brining our attention to the `SIDH/SIKE` key exchange, we present two useful isogenies to carry out the protocol. In the following maps, we will leave away the transformations of the y-coordinate since arithmetics on Montgomery elliptic curves can be efficiently carried out with only the x-coordinate. In fact, any isogeny satisfies

$$(x, y) \mapsto (f(x), cyf'(x)), \tag{7}$$

where f' is the der(ivative of f and c is a fixed constant.

Example 3 (point-doubling, [17], [3])**.** Let us consider the point-doubling formula of a point $P = (x, y)$ on a given elliptic curve E_a

$$\begin{aligned} [2]\colon E_a &\to E_a \\ x &\mapsto \frac{(x^2-1)^2}{4x(x^2+ax+1)}. \end{aligned} \tag{8}$$

Points of order two on E_a are those points, where $y = 0$. By simple calculations, we get that there exist four points $\{\mathcal{O}, (0,0), (\alpha, 0), (1/\alpha, 0)\} =: G$, where $\alpha^2 + \alpha a + 1 = 0$ that are of order 2 on E_a, in accordance with to the 2-torsion ($E[2] = G$). If we now input G together with E_a into Vélu's formula, we get as output the unique isogeny with kernel G (i.e. the [2]-map, a 4-isogeny) together with E_a itself. However, if we input a subset of G, e.g. $G' = \{\mathcal{O}, (\alpha, 0)\}$ with $\alpha \neq 0$ together with the curve E_a, we get as output the following 2-isogeny:

$$\begin{aligned} [\phi]\colon E_a &\to E_{a'} \\ x &\mapsto \frac{x(\alpha x - 1)}{x - \alpha}, \end{aligned} \tag{9}$$

for $a' = 2(1 - 2\alpha^2)$. The y-coordinate can be recovered eq. (7) with $c^2 = \alpha$.

Example 4 (point-tripling, [17], [3])**.** Let us consider the point-tripling formula on a given elliptic curve E_a

$$\begin{aligned} [3]\colon E_a &\to E_a \\ x &\mapsto \frac{x(x^4 - 6x^2 - 4ax - 3)^2}{(3x^4 + 4ax^3 + 6x^2 - 1)^2}. \end{aligned} \tag{10}$$

If we now input all points of order 3 together with E_a into the Vélu formula, we get as output the unique isogeny with kernel G (i.e. the [3]-map) together with E_a itself. However, if we input a subgroup of G, e.g. $G' = \{\mathcal{O}, (\beta, \gamma), (\beta, -\gamma)\}$ together with E_a, we get as output the following 3-isogeny:

$$\begin{aligned} [\phi]\colon E_a &\to E_{a'} \\ x &\mapsto \frac{x(\beta x - 1)}{(x - \beta)^2}, \end{aligned} \tag{11}$$

for $a' = (a\beta - 6\beta^2 + 6)\beta$. The y-coordinate can be recovered eq. (7) with $c = \alpha$.

Note. Note that the point tripling method in Example 4 could also be achieved by a sequential application of the point doubling method in Example 3 and the point addition in Example 1. More general, every multiplication-by-k operation can be expressed as a composition of point doubling and point addition.

2.2 SIDH Protocol

We still work on a Galois Field $\mathbb{F}_{p^2}$ for some prime p. We will denote a point on the elliptic curve only by its x-coordinate. The respective y-coordinate can be obtained by the application of the graph equation.

We now elaborate the protocol in full detail: We set $p = 2^{e_A}3^{e_B} - 1$ and work on an initial (predefined) supersingular elliptic curve E_0 in Montgomery form $y^2 = x^3 + ax^2 + x$.

Table 1 presents the protocol in a chronological way:

Alice	Bob
Given: (public) generators P_A, Q_A on E_0 of her subgroup $\langle P_A, Q_A \rangle = E_0[2^{e_A}] \cong \mathbb{Z}_{2^{e_A}} \times \mathbb{Z}_{2^{e_A}}$	Given: (public) generators P_B, Q_B on E_0 of his subgroup $\langle P_B, Q_B \rangle = E_0[3^{e_B}] \cong \mathbb{Z}_{3^{e_B}} \times \mathbb{Z}_{3^{e_B}}$
Computes secret generator points $S_{A,0} = P_A + [k_A]Q_A$ for some private key $k_A \in \{0, \dots, 2^{e_A} - 1\}$	Computes secret generator points $S_{B,0} = P_B + [k_B]Q_B$ for some private key $k_B \in \{0, \dots, 3^{e_B} - 1\}$
Performing isogenies Φ_A	**Performing isogenies Φ_B**
Sends to Bob: $(\Phi_A(E_0), \Phi_A(P_B), \Phi_A(Q_B))$	Sends to Alice: $(\Phi_B(E_0), \Phi_B(P_A), \Phi_B(Q_A))$
Sets $\Phi_B(E_0)$ as the new starting point and computes $S'_{A,0} = \Phi_B(S_{A,0}) = \Phi_B(P_A) + [k_A]\Phi_B(Q_A)$ (k_A as above)	Sets $\Phi_A(E_0)$ as the new starting point and computes $S'_{B,0} = \Phi_A(S_{B,0}) = \Phi_A(P_B) + [k_B]\Phi_A(Q_B)$ (k_B as above)
Performing isogenies Φ_A	**Performing isogenies Φ_B**
Computes shared secret: j-invariant corresponding to $\Phi_A(\Phi_B(E_0))$	Computes shared secret: j-invariant corresponding to $\Phi_B(\Phi_A(E_0))$

Table 1: SIDH protocol

The steps "**Performing isogenies Φ**" consist of the follwing procedure, which will be explained in detail from the point of view of Alice.[5]

Performing isogenies Φ

1. Given $S_{A,0}$ (which is a point in E_0 of order 2^{e_A}) or $S'_{A,0}$ (which is a point in $\Phi_B(E_0)$ of order 2^{e_A}) as defined above with the secret integer k_A, Alice applies

[5]The point of view for Bob can be elaborated analogically.

the point doubling operation[6] $(e_A - 1)$-times to $S_{A,0}$ leading to a point of order 2, which we will denote by $R_{A,0}$.

2. Using Vélu's formula, she then obtains an isogeny $\phi_0 : E_0 \longrightarrow E_1 := E_0/\langle R_{A,0}\rangle$ of degree 2 (as its kernel consists exactly of $\mathcal{O}$ and $S_{A,0}^{e_A-1}$).[7]

3. She then updates her starting points P_A, Q_A to $P'_A = \phi_0(P_A), Q'_A = \phi_0(Q_A)$. We note that the order of these points does not change under ϕ - however the order of $S_{A,1} = \phi_0(S_{A,0})$ decreased by a factor of 2.

4. She then uses the new point $S_{A,i}$ and the curve $E_i = E_{i-1}/\langle R_{A,i-1}\rangle$ and recursively iterates through this procedure by doubling the point $S_{A,i}$ until it has order 2 (always one doubling iteration less for each iteration) which will be denoted by $R_{A,i}$ and then applies Vélu's formula obtaining a 2-isogeny $\phi_i : E_i \longrightarrow E_{i+1} = E_i/\langle R_{A,i}\rangle$ bringing her to the next elliptic curve.
This iteration is done e_A times obtaining $\phi_0, \phi_1, \dots, \phi_{e_A-1}$. She then sets $\Phi_A = \phi_{e_A-1} \circ \cdots \circ \phi_0$, which is a 2^{e_A}-isogeny by construction and maps from E_0 to $E_{e_A} = \Phi_A(E_0)$.

5. Her public key is then $(\Phi_A(E_0), \Phi_A(P_B), \Phi_A(Q_B))$, hence she maps the initial curve E_0 and Bob's (public) initial basis points under her (secret) isogeny Φ_A.
It is important that Alice and Bob also publish the image under Φ of the other's starting points P_i, Q_i $(i \in \{A, B\})$. Unlike the traditional Diffie-Hellman, where the exponent commute $((g^a)^b = (g^b)^a)$, this is no longer the case in `SIDH` (even the codomains of Φ_A and Φ_B do not match hence a composition does not make any sense). Moving the initial points through the isogenies allows to tackle this problem: Alice (and Bob) is now able to redo the above procedure on a new elliptic curve (after the exchange) E_B (or E_A for Bob), as she knows the images of her starting point under Bob's isogeny and can therefore determine the new S_A.

6. In the case where Alice and Bob perform the step "performing isogenies Φ" *after* the key exchange, the image of the initial basis points under Φ is no longer needed.

On the first sight, it may seem unintuitive, why Alice and Bob will land on the very same j-invariant after following the protocol as described in Table 1. However, the following result from classical group theory should underline why this is the case:

$$E/\langle P, Q\rangle \cong (E/\langle P\rangle)/\langle \Phi(Q)\rangle,$$

for $\Phi : E \longrightarrow E/\langle P\rangle$. In our case, we have that

$$(E_0/\langle S_{A,0}\rangle)/\langle \Phi_A(S_{B,0})\rangle \cong E_0/\langle S_{A,0}, S_{B,0}\rangle$$

and

$$(E_0/\langle S_{B,0}\rangle)/\langle \Phi_B(S_{A,0})\rangle \cong E_0/\langle S_{B,0}, S_{A,0}\rangle.$$

[6]Bob of course applies then the point-tripling operation on $S_{B,0}$ respectively on $S'_{B,0}$. More general, Alice and Bob apply the multiplication-by-b_i operation for $i \in \{A, B\}$.

[7]In general: of degree b_A

Therefore, Alice and Bob will land on the same j-invariant corresponding to

$$E_0/\langle S_{A,0}, S_{B,0}\rangle.$$

We point out that the special form of p (namely that it only factors into primes b_A and b_B plays a crucial role in above argument. If there would be a third prime factor, say b_C, above argument would no longer be valid.

Secret parameters The following parameters are kept secret:

- k_A and k_B respective S_A and S_B.
- The isogenies Φ_A and Φ_B.
- The common secret j-invariant corresponding to $E_0/\langle S_{A,0}, S_{B,0}\rangle$

Public parameters Before the two parties compute their shared secret, they agree on multiple parameters over a public channel. These public parameters are:

- **A field $\mathbb{F}_{p^2}$**
  By defining the field, we (implicitly) also define a set of different supersingular j-invariants corresponding to different elliptic curves.

- **A supersingular elliptic curve E_0 over $\mathbb{F}_{p^2}$**
  On the set of different j-invariants, we define one to be our starting point/starting curve. The supersingular j-invariants define the nodes of a supersingular l-isogeny graph (l will be either 2 or 3), whose edges are l-isogenies (modulo isomorphisms) between elliptic curves associated a j-invariant. The graph is both *connected* and $(l+1)$-*regular*, meaning that there is a path between any two nodes and each node has $(l+1)$ (possibly degenerate) neighbouring nodes. The latter property follows from the fact that the l-torsion $E[l]$ of an elliptic curve E has $(l+1)$ subgroups.

- **A prime p that is chosen as $p = b_A^{e_A} b_B^{e_B} - 1$**
  The prime is chosen as $p = b_A^{e_A} b_B^{e_B} - 1$ such that we are guaranteed to find exactly two subgroups with orders $b_A^{e_A}$ and $b_B^{e_B}$ (recall that the order of a subgroup divides the order of the group itself). Commonly, we choose $b_A = 2$, $b_B = 3$ (in analogy to l as above). The parameters e_A and e_B are chosen such that p is large (see sect. 6). The choices of b_A, b_B, e_A and e_B restrict the potential walks in the supersingular isogeny graph. The walk of both Alice and Bob starts at the node j_0. Alice's secret key selects one particular walk through a b_A-isogeny graph with e_A steps. Analogously, Bob's secret key selects one particular walk through a b_B-isogeny graph with e_B steps. Guessing one party's walk is unlikely since in general, each vertex defines $b_A + 1$ or respectively $b_B + 1$ (possibly degenerate) edges.

- **Alice's and Bob's starting points (P_A, Q_A) and (P_B, Q_B)**
  Although the private keys k_A and k_B (and therefore the generator points S_A and S_B) are secret, the basis elements (P_A, Q_A) and (P_B, Q_B) of the two dimensional

e_A- and e_B-torsion are public. The propagated basis points $(\Phi(P), \Phi(Q))$ for Alice and Bob respectively are also public, as they are shared with their public key. This concept is missing in the classical Diffie-Hellman key exchange since there, all computations are performed in the same group. Here however, we go from one group, the respective elliptic curve, to another through the isogeny maps. For the two parties to continue their respective calculations in different groups, it is necessary to know how their secret generating point transforms under isogenies. There is the dilemma that only Alice knows how Bob's secret generator transforms under her $b_A^{e_A}$-isogeny. Instead of B revealing his secret generator S_B he can reveal his basis points and keep the particular linear combination secret. Since isogenies are group homormorphisms, he can reconstruct his secret generator on Alice's destination curve without knowing her isogeny and without Alice knowing his secret generator.

- **The public keys $(\Phi_A(E_0), \Phi_A(P_B), \Phi_A(Q_B))$ and $(\Phi_B(E_0), \Phi_B(P_A), \Phi_B(Q_A))$ of Alice and Bob**

3 From SIDH to SIKE

The transition from the SIDH to the SIKE protocol is motivated by a potential security flaw in SIDH [6]: If one of the two parties, say Alice, involved in the key exchange keeps reusing their secret key k_A, Bob can then iteratively generate tampered public keys to reconstruct Alice's secret key k_A:
Suppose Bob wants to find Alice's secret k_A - the plan is to do this bit by bit, write $k_A = \sum_{j=0}^{e_A-1} \alpha_j 2^j$. As a first step, Bob wants to find α_0. For this, he computes his public key $(\Phi_B(E_0), \Phi_B(P_A), \Phi_B(Q_A))$ but instead of sending this (honest) public key to Alice, he will provide her with a tampered version by adding an order-two point to $\Phi_B(Q_A)$, as for example $2^{e_A-1}\Phi_B(P_A)$, such that: $PK'_B = \big(\Phi_B(E_0), \Phi_B(P_A), \Phi_B(Q_A)+2^{e_A-1}\Phi_B(P_A)\big)$. Alice will then, in her next step, compute $S_A = \Phi_B(P_A)+[k_A](\Phi_B(Q_A)+2^{e_A-1}\Phi_B(P_A))$. Now we see that since $[k_A]2^{e_A-1}\Phi_B(P_A) = 0 \iff \alpha_0 = 0$, the protocol succeeds (i.e. Alice and Bob end up on the same curve) if and only if k_A is even. Working with similar tricks, Bob can go on and discover $\alpha_1, \ldots, \alpha_{e_A-3}$, the remaining two bits can be found by brute force. To protect herself against such attacks, Alice can either change her secret k_A every time she participates in the SIDH protocol or she can find a way to check that the public key she receives from Bob has been generated in an honest way. This leads us to SIKE (supersingular isogeny key encapsulation), where Bob has to use Alice's public key and his secret k_B to do his part of the protocol before he sends anything to Alice. More precisely, he chooses a random value m and constructs his private key in dependence of Alice's public key: $k_B = H(PK_A, m)$. The public key he then provides to Alice is $\Big(PK_B, H'(j) \oplus m\Big)$, where $\oplus$ denotes the XOR operation. Note that Bob calculates the shared secret $j = j(E_{AB})$ before sending anything to Alice and that his random value is encapsulated in the public key by XORing it with $H'(j)$. Alice can use that that information to compute the shared secret j and recover Bob's random value m by simple XORing. Knowing that value, she can re-generate Bob's public key

and verify that he did not tamper with the protocol. Note that key encapsulation makes the protocol less symmetric in the sense that only one of the participants gets to re-use their key while the other should change it after each iteration as it is actually shared with the counterparty during execution. In practice, such an asymmetric scenario is beneficial when one party (server) communicates with many clients as it reduces the computational effort.

4 Known Attacks

In this section we describe the two most prominent classical attacks against the `SIDH` and `SIKE` protocol. In the quantum world, these attacks are slightly faster but do not yield exponential speed-up. In general, the security of a cryptographic protocol is compromised when parts of any private parameters, such as private keys or the share secret, can be intercepted. In the context of isogeny based cryptography, the underlying mathematical problem to solve is the following:

> Given two supersingular elliptic curves E_0, E_A which are $b_A^{e_A}$-isogenous,
> find a connecting isogeny.

Note that given the isogeny, an attacker can use Velu's formula/theorem to find the corresponding subgroup of E_0 whose generator is the secret key. The fact that we consider *supersingular* curves is an essential part of the security. There is a subexponential quantum algorithm for the ordinary case, but only an exponential quantum algorithm for the supersingular case. The latter is described in Section 4.3. More mathematical details can be found in [7].

4.1 Meet-in-the-middle attack

In the situations we consider for the `SIKE` and `SIDH` protocol, we are work on supersingular elliptic curves over a finite field $\mathbb{F}_{p^2}$ with $p = b_A^{e_A} b_B^{e_B} - 1$. While there are $\mathcal{O}(p)$ (isomorphism classes of) supersingular elliptic curves over $\mathbb{F}_{p^2}$, only $\mathcal{O}(\sqrt{p})$ are connected to the initial curve E_0 by $b_A^{e_A}$-isogenies (because such isogenies are in 1:1 correspondence to subgroups of E_0 with order $b_A^{e_A}$ of which there are $(b_A - 1)b_A^{e_A+1}$). Hence, thinking of a graph whose vertices are supersingular elliptic curves and edges correspond to degree - b_A isogenies, possible endpoints of paths starting from the initial curve are increasingly sparse as p gets large. To find the path Alice has traveled to go from E_0 to E_A (which has length e_A), we save all paths of length $\lfloor e_A/2 \rfloor$ going out from E_0 and walk one-by-one from E_A on paths of length $\lceil e_A/2 \rceil$, always checking if the endpoint is in the saved list. If it is, we have found Alice's $b_A^{e_A}$-isogeny $E_0 \to E_A$. Overall, this gives an attack which requires $\mathcal{O}(p^{1/4})$ memory and runs in $\mathcal{O}(p^{1/4})$. The $\mathcal{O}(p^{1/4})$ memory requirement shows that this attack is not realistic for sizes considered here (in the smallest case of `SIKEp434` we would need to store $> 2^{108}$ bits which is not feasible.) Therefore it has been suggested to fix an upper bound on memory available (like a still unrealistic 2^{80} units) and check the runtimes of the best algorithms under these conditions.

4.2 Van Oorschot-Wiener Collision Finding

Assuming a memory capacity of 2^{80}, there is a better way to fill it than to do meet-in-the-middle until it's full and replace certain elements when capacity is reached. We introduce a set S containing all (j-invariants of) curves which are either $b_A^{e_A/2}$-isogenous to E_0 or E_A. Of this elements, we choose in some way 2^{80} distinguished elements which are the only ones we will allow ourselves to write to memory. The second ingredient will be a deterministic pseudo-random function $f : S \to S$ which takes $x_i \in S$, feeds it into a hash function and then uses the resulting first bit to decide if the new curve $f(x_i)$ will be isogenous to E_0 or E_A and the remaining bits will be used to choose an isogeny (i.e. a subgroup which will be the kernel of the isogeny). For the process, we start at a random $x_0 \in S$ and apply f until we reach a distinguished element $x_n \in S$ which is then written to memory together with the starting element x_0. If we find $x \neq y \in S$ such that $f(x) = f(y) = z$ and x, y are isogenous to E_0, E_A respectively, then z is a mid-point between E_0, E_A in the isogeny graph. Most probably, z will not be a distinguished element. However, since f is deterministic, the walk from z to a distinguished element z_{dist} will be equal both times. So the first time we go through z we will write to memory (x_1, z_{dist}), the second time we will write (x_2, z_{dist}) so that we can recover x and y by walking (using f) from x_1, x_2.

The general problem of finding (for given functions $f : A \to B$ and $g : C \to B$) points $a \in A, c \in C$ such that $f(a) = g(c)$ is called the claw problem. The collision finding algorithm above is a solver which classically runs in $\mathcal{O}(p^{1/4})$. In the quantum world, the runtime can be reduced to $\mathcal{O}(p^{1/6})$ (Tani's algorithm).

4.3 Quantum attack : Grover search

We can formulate the problem of finding a certain isogeny between two supersingular elliptic curves E_0, E_A so that it can be solved by Grover's search algorithm. Recall that Grover's quantum search algorithm finds the unique value x which is mapped to 1 by a function $f : X \to \{0, 1\}$ in quantum time $\mathcal{O}(\sqrt{|X|})$. In our setting, we can define X to be all pairs (ϕ_1, ϕ_2) of $b_A^{e_A/2}$-isogenies with domain E_0, respectively E_A and the function $f : X \to \{0, 1\}$ has $f((\phi_1, \phi_2)) = 1$ if and only if $j(\varphi_1(E_0)) = j(\varphi_2(E_A))$. Since there are $(b_A + 1)b_A^{(e_A - 1)} \sim p^{1/4}$ subgroups in E (and these are in 1:1 correspondence to isogenies), Grover's algorithm finds our isogeny in $\mathcal{O}(\sqrt{|X|}) \sim p^{1/4}$ (note that X consists of *pairs* of isogenies, so $|X| \sim p^{1/2}$).

Despite the quantum version of Tani's collision finding algorithm is faster than Grover search ($\mathcal{O}(p^{1/6})$ compared to $\mathcal{O}(p^{1/4})$), it turns out that Grover search is in fact more cost-effective considering other cost functions (including space/gate count).

5 Resource Requirements

5.1 Memory

The `SIKE` algorithm starts from an initial curve and the public parameters P_i and Q_i for Alice and Bob. For efficiency reasons, it is represented as a triplet of field elements

(P_i, Q_i, R_i). This section presents the four sets of parameters `SIKEp434`, `SIKEp503`, `SIKEp610`, and `SIKEp751`, named so because of the bit length of the underlying prime field. In each case, the parameters are: the prime p, the values e_2 and e_3 and the x- and y-coordinates of the points (P_i, Q_i, R_i) of which each coordinate consists of two values describing it as a complex number. All these parameters can be found in the source code of Microsoft's implementation of `SIKE` (e.g. P434.c in [11]).

First, in Table 2 we reproduced the sizes, in terms of bytes, of the different inputs and outputs required by the key encapsulation mechanism (KEM) as presented in the `SIKE` documentation. The public key consists of P,Q and R encoded by their x-coordinates only. Hence,the size of the public key (in bytes) is calculated as $3 * 2 * N_p$ where $N_p = \lceil \log_2 p \rceil$. The secret key is stored as a concatenation of the random message m of the KEM (see Section 3), the private key k and the public key p_k of Bob. The concatenation is necessary in order to comply with NIST's API guidelines. NIST's decapsulation API does not include an input for the public key, so it needs to be included as part of the secret key. The byte length of randomly chosen value m is defined in the code for each parameter set of SIKE. For Alice, the private key corresponds to integers in the range $(0,1,\ldots,2^{e_2} - 1)$. It is therefore encoded as an octet string of length $N = \lceil s/8 \rceil$ with $s = \lfloor \log_2 2^{e_2} = e_2 \rfloor$. For Bob, the private key corresponds to integers in the range $(0,1,\ldots,3^{e_3} - 1)$. It is therefore encoded as an octet string of length $N = \lceil s/8 \rceil$ with $s = \lfloor \log_2 3^{e_3} \rfloor$. In this setting, Bob uses the static secret and Alice performs the key encapsulation (names are swapped compared to Section 3). It is advantageous to execute Bob's isogeny first and only once for the static key as it is computationally more intense.

It turns out that we can actually compress the public keys much further by focusing on the second and third element in the public key. The high level idea is that the $\mathbb{F}_{p^2}$ elements used to transmit the points $\phi(P)$ and $\phi(Q)$ are rather large compared to the size of the integer coefficients that are needed to represent them with respect to a given basis. In such a basis, the points can be represented as: $\phi(P_3) = [\alpha]R + [\beta]S$ where coefficients α and β are integers in the range $(0,1,\ldots,3^{e_3} - 1)$. With some more technical details, the compressed public key can be stored as three times the byte size of the coefficient, the byte size $N_{p^2} = 2 \times N_p$ of a field element and three additional bytes. The secret key is again a concatenation of m, the private key k of Bob, the compressed public key cp_k and an additional field element. Using this, we reproduced the key sized for the compressed implementation as shown in Table 3.

Scheme			public key	random value	private key	secret key	shared secret
	e_2	e_3	p_k	m	k	s_k (m, k, p_k)	ss
`SIKEp434`	216	137	330	16	28	374	16
`SIKEp503`	250	159	378	24	32	434	24
`SIKEp610`	305	192	462	24	38	524	24
`SIKEp751`	372	239	564	32	48	644	32

Table 2: Size (in bytes) of inputs and outputs in `SIKE`.

Scheme			compressed public key	random value	private key	compressed secret key	shared secret
	e_2	e_3	cp_k	m	k	cs_k (m, k, cp_k)	ss
`SIKEp434`	216	137	197	16	27	350	16
`SIKEp503`	250	159	225	24	32	407	24
`SIKEp610`	305	192	274	24	39	491	24
`SIKEp751`	372	239	335	32	47	602	32

Table 3: Size (in bytes) of inputs and outputs in the compressed implementation of `SIKE`.

5.2 Performance

We evaluated the performance of the optimized and x64-assembly implementations by running the provided benchmarking tests on a machine at our disposal. The machines was powered by a 2.7 GHz Intel Core i5-5350U (Broadwell) processor. We found that our results are in good agreement with the performance test in Table 2.1 of [11], which was run on a 3.4GHz Intel Core i7-6700 (Skylake).

Implementation using x64 assembly

	Key generation	Encapsulation	Decapsulation	
`SIKEp434`	5 570	8 996	9 668	
`SIKEp503`	7 592	13 148	14 054	
`SIKEp610`	14 143	26 288	26 184	
`SIKEp751`	23 909	38 615	41 134	

Table 4: Performance (in thousands of cycles) of `SIKE` on a 2.7 GHz Intel Core i5-5350U (Broadwell) processor. Cycle counts are rounded to the nearest 10^3 cycles.

In order to assess the performance requirement of `SIKE`, we compared it with classical Ellptic Curve (`EC`) key exchange for 256-bit prime provided on [5]. Both, `SIKEp434` and the reference correspond to a `AES-128` symmetric key exchange and therefore the NIST security level 1. For `SIKEp434`, the number of cycles are taken as the sum from Table 4. For `EC`, the cycles for key pair generation and shared secret computation are given in [5] for a similar 2015 Intel Core i5-5350U (Broadwell) processor. The total number of cycles is calculated from two key pair generations and two shared secret computation. The comparison is shown in Table 5.

Compared to other post-quantum cryptographic protocols, `SIKE` is computationally more expensive than competing alternatives. The implementation of [11] needs $O(10^3)$ milliseconds (ms) for encapsulation and decapsulation. An implementation on 32-bit ARM Cortex-M4 microcontroller has benchmark results of 184 million clock cycles (i.e. 1.09 seconds @168MHz) [16]. However, new architectures ([12]) may reduce that gap significantly, demonstrating that `SIKE`, which features the smallest public keys in the NIST PQC process, can be efficiently implemented for embedded applications.

	prime bits	secret key bytes	public key bytes	shared secret bytes	cycles
EC	256	32	64	64	$\sim 4\,000\,000$
SIKE	434	330	374	16	$\sim 25\,000\,000$

Table 5: Comparison of key lengths and cycle count for classical Elliptic Curve with 256-bit prime and `SIKEp434`, both corresponding to security level 1 (`AES-128`).

5.3 Power

The submission [4] includes an additional implementation for the ARM Cortex-M4 processor in which the field arithmetic is written with hand-optimized ARMv7 assembly targeting the Cortex-M4 processor. These cores are optimized for low-cost and energy-efficient microcontrollers which have been implemented in many devices. The implementation of SI;E needs $O(10^8)$ cycles for key generation, encapsulation and decapsulation. All parameter sets lead to an energy consumption between 70 and 80 mW during the run.

6 Security analysis

As a reminder for the reader: The current state-of-the-art in cryptanalysis against `SIDH` takes on the problem of computing the corresponding isogeny for two l^e-isogenous curves E an E' in the supersingular isogeny graph.

The following table shows time estimates of the classical attacks described in section 4 for a prime $p \sim 2^{448}$ of length 448 bits. This means that with a 448-bit prime p, we get classical 128-bit security.

	Processors	Space	Total Time
Meet-in-the-middle	48	64	154
	48	80	138
	64	80	138
VoW Collision Finding	48	64	136
	48	80	128
	64	80	128

Table 6: Time Complexity of Attacks against 2-Isogeny Problem for $p \sim 2^{448}$. All numbers in $\log_2$-scale.

NIST anticipates that there will be significant uncertainties in estimating the security strengths of post-quantum cryptosystems. Therefore, NIST will base its classification on the range of security strengths offered by the existing NIST standards in symmetric cryptography (see 4.A.5 Security Strength Categories in the Call for Proposals ([13])). Here, computational resources may be measured using a variety of different metrics (e.g., number of classical elementary operations, quantum circuit size, etc.). Also, NIST has suggested that one quantum gate can be assigned a cost equivalent to $O(1)$ classical gates. The number of gates (size) of a quantum circuit is calculated as width (space) times depth (time). [8] provides an estimation for number of qubits for

NIST level	classical gates	reference algorithms	factoring	discrete logarithm key	discrete logarithm group	Elliptic curve	`SIKE`
1	2^{143}	`AES-128`	3 072	256	3 072	256	`SIKEp434`
2	2^{146}	`SHA3-256`					`SIKEp503`
3	2^{207}	`AES-192`	7 680	384	7 680	384	`SIKEp610`
5	2^{272}	`AES-256`	15 360	512	15 360	512	`SIKEp751`

Table 7: Comparison of NIST security levels (in terms of classical gates for PQC) of `SIKE` parameter sets with classical key exchange protocols (sizes in bits).

reversible circuits that implement the full Advanced Encryption Standard `AES`-k. To classify PQC, NIST suggests an approach where quantum attacks are restricted to a fixed running time, or circuit depth. Call this parameter `MAXDEPTH`. This restriction is motivated by the difficulty of running extremely long serial computations. Plausible values for `MAXDEPTH` range from 2^{40} logical gates (the approximate number of gates that presently envisioned quantum computing architectures are expected to serially perform in a year) through 2^{64} logical gates (the approximate number of gates that current classical computing architectures can perform serially in a decade), to no more than 2^{96} logical gates (the approximate number of gates that atomic scale qubits with speed of light propagation times could perform in a millennium). In addition to the classical case described above, we estimate the cost of an attack against `SIDH` using Grover search because it turns out (see [1]) that although it runs in $\mathcal{O}(p^{1/4}$ compared to Tani's algorithm's $\mathcal{O}(p^{1/6})$, it's still more cost-effective because it needs much less memory. With `MAXDEPTH` of 2^D, using Grover search to attack the protocol for a k-bit prime would require $(2^{k/4}/2^D)^2$ quantum circuits. This means that an 448-bit prime would require $(2^{112}/2^{40})^2 = 2^{144}$ quantum circuits and corresponds to classical 128-bit security.

Only [10] gives concrete cost estimates for solving the computational supersingular isogeny problem in different scenarios imposing a depth restriction on quantum circuits. They convert a quantum circuit model in a classical random access machine (RAM) which acts as a controller for memory peripheral such as an array of bit or quibits. Allowing depth 2^{96} for example, they conclude that no known quantum algorithm can break `SIKE` in their model of computation with less than 2^{143} classical gates for `SIKEp434`. These results lead to the classification shown in Table 7 used throughout the report.

In Table 7, we categorize the `SIKE` parameter sets in NIST's security levels and compare it with classical protocols such as `DH` based on discrete logarithm, ECC or RSA. The values of the reference protocols are provided in [2]. From the KEM (see Section 3), we see that the size of the random value m defines an upper bound for the strength of the protocol by a brute-force exploitation. However, as described above, algorithms that attack on the isogeny computation can do significantly better.

7 Conclusion

In this report we have given an introduction to isogeny based cryptography, namely the `SIDH` key exchange as well as the improved `SIKE` protocol, resolving potential security flaws with an additional focus on enhanced performance. We have summarised state-of-the-art attacks on `SIKE` and have shown that even with quantum speed-ups the protocol satisfies current standards. Other cryptographic parameters, as key size and computational resource requirements were discussed in detail, providing comparisons within `SIKE`'s own security levels and the classical Diffie-Hellman key exchange.

In conclusion, regarding the NIST's mission of establishing a security standard in a post-quantum world, one can say that in general, `SIKE` has two main advantages: The key sizes to achieve security levels targeted by NIST are comparatively small (e.g. 564B public keys/48B private keys for security level 5, where other candidates in round 2/3 of the NIST competition require key lengths in the kB/MB range). Also, since elliptic curves have already been used for years in cryptographic applications, we can assume that there will be less implementation-specific vulnerabilities as an engineer with experience in ECC should be able to implement `SIKE` safely without much trouble. On the other hand, there is a tradeoff: we pay for very small keysizes with increased (compared to other protocols) runtimes by a factor of around 100. Hence `SIKE` is especially suitable in situations where space/bandwidth is more of a rare commodity than computing power.

References

[1] G. Adj, D. Cervantes-Vázquez, J.-J. Chi-Domínguez, A. Menezes, and F. Rodríguez-Henríquez. On the cost of computing isogenies between supersingular elliptic curves. Cryptology ePrint Archive, Report 2018/313, 2018. https://eprint.iacr.org/2018/313.

[2] BlueKrypt. Cryptographic Key Length Recommendation. https://www.keylength.com/en/4/, 2020. [Online; accessed 28-August-2020].

[3] C. Costello. Supersingular isogeny key exchange for beginners. In K. G. Paterson and D. Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 21–50, Cham, 2020. Springer International Publishing.

[4] David Jao. Sike specifiation document. https://sike.org/files/SIDH-spec.pdf, 2020. [Online; accessed 30-August-2020].

[5] ECRYPT-EU. eBACS: ECRYPT Benchmarking of Cryptographic Systems. https://bench.cr.yp.to/results-dh.html, 2020. [Online; accessed 28-August-2020].

[6] S. D. Galbraith, C. Petit, B. Shani, and Y. B. Ti. On the security of supersingular isogeny cryptosystems. In J. H. Cheon and T. Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 63–91, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[7] S. D. Galbraith and F. Vercauteren. Computational problems in supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2017/774, 2017. https://eprint.iacr.org/2017/774.

[8] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt. Applying grover8217;s algorithm to aes: quantum resource estimates. In *Proceedings of the 7th International Conference on Post-Quantum Cryptography (PQCrypto'16), Fukuoka, Japan*, volume 9606 of *Lecture Notes in Computer Science*, pages 29–43. Springer, September 2016. See also arXiv preprint arXiv:1512.04965.

[9] D. Jao and L. De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In B.-Y. Yang, editor, *Post-Quantum Cryptography*, pages 19–34, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[10] S. Jaques and J. M. Schanck. Quantum cryptanalysis in the ram model: Claw-finding attacks on sike. Cryptology ePrint Archive, Report 2019/103, 2019. https://eprint.iacr.org/2019/103.

[11] P. Longa. PQCrypto-SIDH v3.3. https://github.com/Microsoft/PQCrypto-SIDH, 2020. [Online; accessed 31-August-2020].

[12] P. M. C. Massolino, P. Longa, J. Renes, and L. Batina. A compact and scalable hardware/software co-design of sike. Cryptology ePrint Archive, Report 2020/040, 2020. https://eprint.iacr.org/2020/040.

[13] NIST. Post-Quantum Cryptography. https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization, 2020. [Online; accessed 31-August-2020].

[14] NIST. Post-quantum cryptography standardization process: Third round candidate announcement. https://csrc.nist.gov/News/2020/pqc-third-round-candidate-announcement, 2020. [Online; accessed 31-August-2020].

[15] J. Renes. Computing isogenies between montgomery curves using the action of (0,0). Cryptology ePrint Archive, Report 2017/1198, 2017. https://eprint.iacr.org/2017/1198.

[16] H. Seo, M. Anastasova, A. Jalali, and R. Azarderakhsh. Supersingular isogeny key encapsulation (sike) round 2 on arm cortex-m4. Cryptology ePrint Archive, Report 2020/410, 2020. https://eprint.iacr.org/2020/410.

[17] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography, Second Edition.* Chapman amp; Hall/CRC, 2 edition, 2008.

[18] Wikipedia. Montgomery curve — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Montgomery%20curve&oldid=950440406, 2020. [Online; accessed 31-August-2020].

A Python Implementation

Attached to this report is a Python implementation of the `SIDH` protocol, for a simple set of public parameters, following the definitions in [3]. The code comes with five different classes:

`GFp`*(x,p)* implements the element $x \mod p$ of the field $\mathbb{F}_p$ over a prime p and its operations, including multiplicative inversion.

`GFp2`*((u,v),p)* implements the element $u + \mathrm{i}v \mod p$ of the field $\mathbb{F}_{p^2}$ over a prime p and its operations, including multiplicative inversion (see eq. (5)) and a naive (and slow) computation of the modular square root (required for the computation of the y-coordinate under the 2-isogeny).

`MontgomeryEllipticCurve`*(a)* implements the elliptic curve $E(a)$ over $\mathbb{F}_{p^2}$ in Montgomery form given the parameter $a \in \mathbb{F}_{p^2}$ including the computation of its j-invariant (see eq. (4)) and the transformations of the curve under a 2- or 3-isogeny (given the x-coordinate of the non-neutral element in corresponding kernel, see eqs. (9) and (11))).

`EllipticCurvePoint`*((x,y),E,is_infinity=False)* implements a point (x, y), with $x, y \in \mathbb{F}_{p^2}$ or the neutral element $\mathcal{O}$ (*is_infinity=True*) of the group of points on the elliptic curve E, the group law of addition including multiplication by a positive integer (see eqs. (2) and (8)) and the transformation of the point under a 2- or 3-isogeny (given the x-coordinate of the non-neutral element in corresponding kernel, see eqs. (9) and (11))).

`SupersingularIsogenyGraphWalker`*(a,e_2,e_3,b_2_torsion,b_3_torsion)* and its member function `perform_l_isogeny_walk`*(l,key,public_key=None)* implement a walk through a $l = 2$- or $l = 3$-isogeny graph with *e_2* and *e_3* steps, defined by the prime $p = 2^{e_2}3^{e_3} - 1$, the basis of the 2^{e_A}- and 3^{e_3}-torsion *b_2_torsion* $= (P_2, Q_2)$ and *b_3_torsion* $= (P_3, Q_3)$ on a initial Montgomery curve with parameter a, a *key*. If a *public_key = (a_prime,b_l_torsion_prime)* is specified, the l-isogeny walk is performed according to the image of the original basis *b_l_torsion_prime* on the Montgomery curve with parameter *a_prime*. The index 2 traditionally refers to Alice and 3 to Bob.

The code can carry out the `SIDH` key exchange protocol, as described in sect. 2.2, for arbitrary keys[8]. First, each of the two party's isogeny walk is performed, given the initial public parameters, which generates their respective public keys. Then follows another isogeny walk for each of the two parties, given the other parties public key, which generates their shared secret. One can check that the key exchange was successful by comparing two party's result for the shared secret.

[8]In the current version of the script Alice's private key, *key_2*, can only be chosen odd. This is due to the fact that even keys will generate a 2-isogeny corresponding to the kernel $\{\mathcal{O}, (\alpha, 0)\}$ with $\alpha = 0$, for which eq. (9) cannot be applied. Odd values for *key_2* however, always satisfy $\alpha \neq 0$. The correct 2-isogeny for $\alpha = 0$ can be computed with Vélu's formulas and is given on page 10 in [15] following [9] for a general Montgomery curve $by^2 = x^3 + ax^2 + x$. Since currently, we only support $b = 1$, one has to compose the isogeny with another isomorphism to make it directly compatible with our code.

The initial parameters for the prime p, the Montgomery parameter a and the bases of the 2^{e_2}- and 3^{e_3}-torsion can in principle be replaced with any other allowed choice satisfying the mathematical constraints. Remember though that a valid protocol has to satisfy that the prime has a decomposition $p = 2^{e_2}3^{e_3} - 1$, the curve is supersingular and the basis points are in their respective 2^{e_2}- and 3^{e_3}-torsion. In practice, therefore, these parameters should only be changed if it is understood that these mathematical constraints are indeed fulfilled.