Towards Post-Quantum Key-Updatable Public-Key Encryption via Supersingular Isogenies

Edward Eaton, David Jao, Chelsea Komlo, and Youcef Mokrani

University of Waterloo

Abstract. We present the first post-quantum secure Key-Updatable Public-Key Encryption (UPKE) construction. UPKE has been proposed as a mechanism to improve the forward-secrecy and post-compromise security of secure messaging protocols, but the hardness of all existing constructions rely on discrete logarithm assumptions. We focus our assessment on isogeny-based cryptosystems due to their suitability for performing a potentially unbounded number of update operations, a practical requirement for secure messaging where user conversations can occur over months, if not years.
We begin by formalizing two UPKE variants in the literature as Symmetric and Asymmetric UPKE, which differ in how encryption and decryption keys are updated. We argue that Asymmetric UPKE constructions in the literature [25] cannot be straightforwardly instantiated using SIDH nor CSIDH. We then describe a SIDH construction that partially achieves the required security notions for Symmetric UPKE, but due to existing mathematical limitations, cannot provide fine-grained forward secrecy. Finally, we present a CSIDH Symmetric UPKE construction that requires a parameter set in which the class group structure is fully known. We discuss open problems which are applicable to any cryptosystem with similar requirements for continuous operations over the secret domain.

Keywords: Secure messaging, post-quantum cryptography, isogenies, key-updateable encryption

1 Introduction

Secure communication protocols are quickly evolving [3, 28, 29], driven by the need to meet simultaneous usability and security requirements, such as asynchronous communication while ensuring forward secrecy and post-compromise security for conversations that can occur over months, if not years. *Key-Updatable Public-Key Encryption* (UPKE) schemes have been proposed as a solution to improve weak forward secrecy properties of existing secure messaging protocols such as the Signal and Message Layer Security (MLS) protocols [1, 2, 10, 21, 25, 34, 35]. In addition to standard public-key encryption functionality, UPKE schemes allow encryption and decryption keys to be asynchronously *updated* with fresh entropy, thereby *healing* the protocol by restoring security even after exposure of secret values. Unfortunately, the security of all UPKE schemes proposed to date relies on the hardness of the discrete logarithm problem.

In this work, we perform the first assessment of the viability of quantum-secure UPKE schemes. We begin by formalizing two UPKE variants presented in the literature which we call *Symmetric UPKE* and *Asymmetric UPKE*[1], and assess the extent to which existing isogeny-based cryptosystems can instantiate both variants. We model Asymmetric UPKE after a construction proposed by

[1] Note that symmetric and asymmetric here refers to the requirements of how the update operation is performed, not the style of encryption.

Jost, Maurer, and Mularczyk [25], in which encryption keys are updated using elements in the public domain, while decryption keys are updated using private values. We model Symmetric UPKE after a construction proposed by Alwen et al. [1] to improve the forward-secrecy and post-compromise security of TreeKEM [5], the group key-exchange primitive used by MLS, where both encryption and decryption keys are updated using the *same* secret update value. Further, we introduce the notion of IND-CPA-Usecurity, a generalization of a security model by Alwen et al. [1] for UPKE constructions.

We argue that Asymmetric UPKE constructions as currently defined in the literature *cannot* be instantiated by either Supersingular Isogeny Diffie Hellman (SIDH) nor Commutative Supersingular Isogeny Diffie-Hellman (CSIDH). We then present a series of steps demonstrating that while SIDH can in theory be used for Symmetric UPKE constructions, a viable construction in practice is hindered by existing mathematical limitations. We then present a CSIDH-based Symmetric UPKE construction which can be used today with the existing CSIDH-512 parameter set, or any CSIDH parameter set where the class group structure is fully known. Knowing the class group structure ensures unique group element representation and uniform sampling of secret key material. Taken together, these properties ensure that knowledge of a secret key prior to an update will not leak information about the key *after* an update operation, thereby fulfilling forward secrecy and post-compromise security. We prove that our CSIDH construction fulfills IND-CPA-Usecurity.

We focus our analysis on isogeny-based cryptosystems, as alternative quantum-secure cryptographic primitives have undesirable usability or efficiency trade-offs for secure messaging protocols, or simply cannot support the algebraic structure required for UPKE. In the setting of secure messaging, user conversations can potentially endure months, if not years, and so supporting ongoing protocol actions without bounds on the number of consecutive update operations is desirable. By contrast, lattice-based cryptosystems accumulate errors for each additional operation, and so require either bounding the number of operations or performing some expensive compression function to limit growth of errors [17, 18]. Code-based primitives similarly accumulate errors over repeated operations, and so have similar restrictions on the number of possible operations that can be performed [30]. Finally, multivariate and hash-based primitives are not a good fit for key-exchange protocols in general, much less protocols with more advanced requirements such as updatability[2]. Further, since updates occur only periodically, performance of update operations is less of a concern.

Contributions. In this work, we assess the viability of post-quantum secure key-updatable public-key encryption (UPKE) schemes, and define constructions using isogeny-based cryptosystems for a subset of these schemes. Towards this end, we present the following contributions:

- We give formal definitions of Symmetric UPKE and Asymmetric UPKE as two UPKE variants presented in the literature. We also present IND-CPA-U, a generalized security model for proving IND-CPA security specifically for UPKE schemes, a setting in which the adversary is assumed to be able to adaptively choose updates and corrupt secret key material.
- We argue that the most prominent Asymmetric UPKE construction currently in the literature [25] *cannot* be straightforwardly instantiated by either SIDH or CSIDH.
- We then describe a SIDH-based Symmetric UPKE construction that is possible in theory, but requires further mathematical advancements and careful cryptanalytic scrutiny to be instanti-

[2] Hash functions only rely on one-way functions and thus cannot provide the structure needed for key exchange. Multivariate schemes are generally built from a surjective trapdoor function that is difficult to build a key exchange protocol from, and with no obvious algebraic structure to allow for updating.

ated in practice. We present this scheme in order to make clear these gaps and possible future research directions.
– We present a Symmetric UPKE construction that can be used today with CSIDH-512, or any CSIDH parameter set where the class group has been fully computed. We prove this construction to be IND-CPA-Usecure, and provide an implementation.

Related Work The most closely related work to our own is the already mentioned work of Alwen et al. [1] as a mechanism to improve forward secrecy and post-compromise security of TreeKEM [5]. Our Symmetric UPKE primitive is modeled after their construction, and our work is an effort to define a post-quantum UPKE variant suitable for similar use. Further, we prove security in a more robust model that models the adversary's capability to both adaptively choose update values for the victim as well as corrupt their local state. The work by Alwen et al. was in turn based upon work by Jost et al. [25], which is most akin to our Asymmetric UPKE notion.

Both secure messaging protocols and post-quantum protocols are still in active development. Efforts to combine the two into post-quantum secure messaging are so far rare in the literature, although we refer to [7] as a recent example of exactly this. Their work constructs a version of Signal's X3DH protocol out of the (ring)-LWE problem.

Alternative (Unrelated) Notions of UPKE. There exists a separate notion of "updatable encryption" in the literature [6, 24]. In these schemes, a *ciphertext* is updated using an update token such that the encrypted message becomes an encryption under a new public key without decrypting the message. These schemes should not be confused with *key-updatable* UPKE schemes.

Organization. We present preliminaries and an overview of isogenies and isogeny-based cryptosystems in Section 2, and of updatable public-key encryption schemes in Section 3. In Section 4 we assess the extent to which SIDH and CSIDH can be used to instantiate Asymmetric and Symmetric UPKE constructions. We present a series of steps toward a SIDH Symmetric UPKE construction in Section 5 and a CSIDH Symmetric UPKE construction in Section 6. We conclude in Section 8.

2 Preliminaries

Let λ denote the security parameter in unary representation. We denote sampling a value a from a non-empty set S uniformly at random as $a \xleftarrow{\$} S$.

As we are constructing a hybrid encryption scheme, we will rely on the standard notions of a key encapulation mechanism, or *KEM*, and a data encapsulation mechanism, or *DEM*. However we do not define our scheme as a *KEM* in order to match the interface presented in previous work in this area [1]. In order to use our protocol as a hybrid encryption scheme we will use a data encapsulation mechanism, which we keep as an abstract interface for flexibility.

Definition 1. *A Data Encapsulation Mechanism DEM [19] is a tuple of of three algorithms: a non-deterministic key generation algorithm KeyGen(λ) that accepts a security parameter λ and outputs a randomized key K, a non-deterministic encryption algorithm Encrypt(K, m) that accepts K and a message m and outputs a ciphertext ctxt, and a deterministic decryption algorithm Decrypt(K, ct) that outputs the message m.*

In Appendix D we will show that our CSIDH-based construction satisfies our IND-CPA-Udefinition so long as CSIDH combined with the *DEM* scheme is IND-CPA. Thus we do not specify the security requirements of the *DEM* specifically, only that it is enough to ensure that the hybrid system is IND-CPA.

We now give further background on isogenies and their use in public-key cryptosystems.

2.1 Isogenies and Isogeny-Based Cryptography

Let p be a prime number and $\mathbb{F}_p$ be a finite field of characteristic p, and let E_0 and E_1 be elliptic curves defined over $\mathbb{F}_p$. An isogeny $\phi\colon E_0 \to E_1$ is a rational map from $E_0(\overline{\mathbb{F}_p})$ to $E_1(\overline{\mathbb{F}_p})$ which is also a group homomorphism [37], where $\overline{\mathbb{F}_p}$ is the algebraic closure of $\mathbb{F}_p$. When two curves are isomorphic, they share the same *j-invariant*. which remains a constant value for all isomorphic curves.

An endomorphism of an elliptic curve $\phi\colon E \to E$ is a rational map from E to itself, defined over an extension field $\mathbb{F}_{p^n}$. The set of all endomorphisms for an elliptic curve (over an algebraic closure) forms a ring under the operations of point-wise addition and composition; we denote this ring of endomorphisms as $\mathrm{End}(E)$. When $\mathrm{End}(E)$ is isomorphic to an order in a quaternion algebra, the curve is classified as *supersingular*, otherwise, $\mathrm{End}(E)$ is isomorphic to to an imaginary quadratic field and the curve is classified as *ordinary*[37].

We next describe two existing cryptosystems—SIDH and CSIDH—whose security has been demonstrated to reduce to the hardness of the Supersingular Isogeny Problem (or variants thereof), described in Definition 2. We include additional background information on both in Appendix A.

Definition 2. ***Supersingular Isogeny Problem*** *[23] Given a finite field K and two supersingular elliptic curves E_1, E_2 defined over K such that $|E_1| = |E_2|$, compute an isogeny $\phi\colon E_1 \to E_2$*

Supersingular Isogeny Diffie-Hellman (SIDH). Introduced by Jao and De Feo in 2011 [23], SIDH is a Diffie-Hellman like scheme defined using secret isogenies between supersingular elliptic curves to perform a key exchange protocol. SIDH can also be constructed as a PKE scheme [23]. SIDH has been adapted as a KEM with additional Fujisaki-Okamoto techniques [20] as an IND-CCA2 secure candidate for the ongoing NIST competition to standardize quantum-resistant key exchange protocols [22].

Performing key exchange via SIDH begins with each party agreeing to a starting public curve $E_0(\mathbb{F}_{p^2})$, where p is a prime of the form $2^{e_1}3^{e_2}-1$, and two sets of basis points $\{P_A, Q_A\}, \{P_B, Q_B\} \subset E_0$, which are generators of the 2^{e_1} and 3^{e_2}-torsion subgroups respectively. For this work, we assume secret values—which define the kernel of an isogeny—are of the form $\langle [1]P + [n]Q\rangle$, such that participants only need to randomly generate the scalar n to define their secret key. Alice begins by selecting $n_A \xleftarrow{\$} \mathbb{Z}_{2^{e_1}}$ which defines her secret isogeny $\phi_A\colon E_0 \to E_A$, such that $E_A = E_0/\langle [1]P_A + [n_A]Q_A\rangle$. Similarly, Bob selects $n_B \xleftarrow{\$} \mathbb{Z}_{3^{e_2}}$, which defines his secret isogeny $\phi_B\colon E_0 \to E_B$, such that $E_B = E_0/\langle [1]P_B + [n_B]Q_B\rangle$. Alice publishes her public key $(E_A, \phi_A(P_B), \phi_A(Q_B))$, while Bob publishes his public key $(E_B, \phi_B(P_A), \phi_B(Q_A))$.

After obtaining each other's public curves, their shared secret is a common curve E_{AB}, which is the same for Alice and Bob up to isomorphism. Alice arrives at E_{AB} by calculating $E_B/\langle \phi_B(P_A) + [n_A]\phi_B(Q_A)\rangle$ using her secret term n_A, whereas Bob calculates $E_A/\langle \phi_A(P_B) + [n_B]\phi_A(Q_B)\rangle$ using his secret term n_B. Alice and Bob obtain the same shared secret by finding the j-invariant of E_{AB}, as their resulting values $E_B/\langle \phi_B(P_A) + [n_A]\phi_B(Q_A)\rangle \cong E_A/\langle \phi_A(P_B) + [n_B]\phi_A(Q_B)\rangle$ are equal up to isomorphism.

Commutative Supersingular Isogeny Diffie-Hellman (CSIDH). As described in Section 2.1, SIDH performs "Diffie-Hellman-like" operations over a set of supersingular elliptic curves over the quadratic extension field $\mathbb{F}_{p^2}$ and isogenies between these curves. SIDH ensures commutativity of key-exchange operations between two participants by additional sending auxiliary points on each participant's public curves. Participants arrive at the same curve up to isomorphism by "dividing out" the starting elliptic curve by two non-intersecting subgroups. CSIDH [8] builds upon the Couveignes-Rostovtsev-Stolbunov [11, 36] scheme, but instead uses the graph of supersingular curves. The security of CSIDH is based on the Supersingular Isogeny problem defined in Definition 2, but CSIDH restricts the supersingular isogeny graph (where nodes are supersingular curves, and edges are isogenies) to curves defined over $\mathbb{F}_p$.

While the full ring of endomorphisms of supersingular curves over $\mathbb{F}_{p^2}$ is non-commutative, restricting consideration to the subring of endomorphisms defined over $\mathbb{F}_p$ yields a (commutative) imaginary quadratic order $\mathcal{O}$. A consequence of this restriction is that the isogeny graph must also be restricted to curves and isogenies defined over $\mathbb{F}_p$. To ensure that isogeny operations can be computed efficiently using Vélu's formulas [38], the prime p in CSIDH is defined to be of the form $p = 4 \cdot \ell_1 \cdot \ell_2, \ldots, \cdot \ell_d - 1$ for some set of small primes ℓ_d generating the class group $cl(\mathcal{O})$.

Similarly to SIDH, participants performing a key-exchange must agree to some starting curve curve $E_0(\mathbb{F}_p)$. A secret key in CSIDH is a vector $\vec{e} \in \mathbb{Z}^d$; each element in $\vec{e}$ is within some bound to ensure the values are "small." The vector $\vec{e}$ represents a secret ideal $\prod_{i=1}^{d} {\ell_i}^{e_i}$. By the Deuring correspondence, this ideal corresponds to exactly one isogeny from the starting curve to another curve in the graph.

While CSIDH is normally presented as if it were a group action, there are limitations to interpreting it as such. Sampling uniformly random elements from the group and efficiently computing the group action on those elements requires computing the structure of the class group $cl(\mathcal{O})$, which takes subexponential time in general. The authors of CSI-FiSh [4] solved this problem by explicitly computing the structure of $cl(\mathcal{O})$ for the CSIDH-512 parameter set, which is a specific parameter set using a 511-bit prime p. Such a computation requires subexponential time, and cannot reasonably be extended to much larger parameter sets using present technology. In this case, the group is cyclic, and the group order N is now known. Furthermore, the authors computed a basis of short vectors generating the relation lattice of the set of small prime generators, allowing one to convert elements of $\mathbb{Z}_N$ to vectors $\vec{e} \in \mathbb{Z}^d$ given a choice of group generator, so that the representation $\prod_{i=1}^{d} \ell_i^{e_i}$ can be used for isogeny computation. Fully computing $cl(\mathcal{O})$ allows for efficient and uniform sampling of elements in $\mathbb{Z}_N$ and canonical representation in $\mathbb{Z}_N$. While the authors of CSI-FiSh used these properties to define a signature scheme, we employ the same structure for the purposes of constructing UPKEs.

Efficient Algorithms for the ℓ-Isogeny Path Problem. The *Deuring correspondence* establishes a mapping between supersingular curves over the quadratic extension field $\mathbb{F}_{p^2}$ and maximal orders in a quaternion algebra.[3] The endomorphism ring $\mathrm{End}(E)$ of a supersingular curve E is isomorphic to a maximal order $\mathcal{O}$ in $\mathbf{Q}_{p,\infty}$ (the quaternion algebra over $\mathbf{Q}$ ramified at p and ∞). For each maximal order $\mathcal{O} \in \mathbf{Q}_{p,\infty}$, there are at most two supersingular curves (up to isomorphism) with endomorphism rings isomorphic to $\mathcal{O}$. The correspondence also provides information about isogenies. If there is an isogeny $\phi\colon E_0 \to E_1$, then looking at the corresponding maximal orders $\mathcal{O}_0$ and

[3] The mapping is not quite bijective. Curves with conjugate j-invariants are mapped to the same maximal order, so the mapping is at most two-to-one for isomorphic curves.

$\mathcal{O}_1$, there is a left ideal in $\mathcal{O}_0$ which is also a right ideal in $\mathcal{O}_1$. If the isogeny has degree ℓ^k for a prime ℓ, the ideals will have the same norm.

Kohel et al. demonstrated that finding a path in the ℓ-isogeny graph is easier when working over the maximal orders and ideals [26]. Their algorithm (commonly referred to as the KLPT algorithm), takes as input the endomorphism ring of an elliptic curve $\mathrm{End}(E_1)$ and a prime ℓ and finds an ideal with norm ℓ^k for some k that is a left ideal of $\mathrm{End}(E_0)$ and a right ideal of $\mathrm{End}(E_1)$. This ideal can then be converted back to an isogeny $\phi\colon E_0 \to E_1$.

The reason this does not break SIDH is because of the requirement that the endomorphism ring of E_1 is known. Calculating the endomorphism ring of E_1 is believed to be difficult—unless an isogeny $\psi\colon E_0 \to E_1$ is known. This means that for most cases KLPT can only find an isogeny from E_0 to another curve E_1 if an isogeny between the two is already known.

These algorithms have proven to be quite useful in constructing cryptographic protocols. Finding a second isogeny between two curves is useful for proving knowledge of a secret isogeny $\phi\colon E_0 \to E_1$. By concatenating a commitment isogeny $\psi\colon E_1 \to E_2$, KLPT can be used to generate an isogeny $\eta\colon E_0 \to E_2$ that does not go through E_1. This technique has been used to construct signature schemes, beginning with [16], and most recently to construct the signature scheme SQI-Sign [12], based upon an improved version of the KLPT algorithm, which we call KLPT*. This improved version reduces the degree of the output isogeny from $\frac{9}{2}\log_\ell(p)$ to $\frac{15}{4}\log_\ell(p)$. Note however that KLPT* relies on two assumptions (Assumptions 1 and 2) to prove that the output of KLPT* leaks no information about its input. In summary, KLPT* assumes that an isogeny of fixed degree between two curves can be found with high probability, and that the output is statistically close to a uniform sampling (a random walk).

3 Key-Updatable Public-Key Encryption (UPKE)

We begin by formalizing the notion of a generalized UPKE scheme. We then present two variants of UPKE schemes in the literature [1, 25], which differ in how update operations are performed.

Definition 3. Key-Updatable Public-Key Encryption (UPKE) *A UPKE scheme $\mathcal{U}$ is a tuple of six algorithms: a key generation algorithm KeyGen, an encryption algorithm Encrypt, a decryption algorithm Decrypt, an algorithm to generate update values GenUpdate, and algorithms to update private and public keys UpdatePrivate, UpdatePublic, respectively.*

UPKE schemes must fulfill the following correctness, security, and usability notions.

- *Correctness*: The scheme should correctly perform public-key encryption and decryption both before and after a series of updates.
- *Forward secrecy*: If an attacker learns the secret key for epoch n, the updated secret keys in epochs $1 \ldots, n-1$ should not be recoverable.
- *Post-compromise security*: If an attacker learns the secret key for epoch n, all updated secret keys in epochs $n+1, n+2, \ldots$ should not be recoverable.
- *Asynchronicity*: Anyone with knowledge of a public key should be able to initiate an update, so that the update operation to the public key is immediately available, and only the update operation to the secret key should be performed *eventually*.

– *Key Indistinguishability*: An adversary that has access to both a freshly generated keypair and an updated keypair has a negligible advantage to distinguish between the two. Note that while such a property may be desirable in practice for privacy reasons, our reason for requiring this property is for the reduction to IND-CPA in our proof. However, alternative proof strategies may not require this property.

Symmetric UPKE. We model Symmetric UPKE after a construction described by Alwen et al. [1], which is presented as a mechanism to improve the forward-secrecy and post-compromise security properties of TreeKEM.

In this construction, the sender of a message also generates the update value, which is transmitted privately to the holder of the decryption key along with the ciphertext and message. The sender of a message applies the update value to the other party's encryption key, and then the receiver applies the *same* update value to their decryption key.

Definition 4. Symmetric UPKE *A Symmetric UPKE scheme instantiates $\mathcal{U}$ as follows:*

- $KeyGen(\lambda) \rightarrow (sk, pk)$: Accepts a security parameter λ and outputs a public encryption key pk and secret decryption key sk.
- $Encrypt(pk, m) \rightarrow c$: Encrypts a message m using pk, resulting in a ciphertext c.
- $Decrypt(sk, c) \rightarrow m$: Decrypts c using sk, producing the plaintext message m.
- $GenUpdate(\lambda) \rightarrow \mu$: Accepts λ and outputs a randomly-generated update value μ.
- $UpdatePrivate(sk, \mu) \rightarrow sk'$: Takes as input sk and the update value μ and produces a deterministic output that is the updated sk'.
- $UpdatePublic(pk, \mu) \rightarrow pk'$: Takes as input pk and update value μ, and produces a deterministic output that is the updated pk'.

Correctness. Symmetric UPKE constructions are correct if they correctly perform *Encrypt* and *Decrypt* operations both before and after a series of *UpdatePublic* and *UpdatePrivate* operations.

Asymmetric UPKE. We first define a generic Asymmetric UPKE construction, and then present Asymmetric UPKE† that is modeled after an existing construction in the literature. In the Asymmetric UPKE setting, encryption and decryption keys are updated using *distinct* update values.

Definition 5. Asymmetric UPKE *An Asymmetric UPKE scheme instantiates $\mathcal{U}$ as follows, where KeyGen, Encrypt, Decrypt remain identical to the Symmetric setting:*

- $GenUpdate(\lambda) \rightarrow (\mu_{sk}, \mu_{pk})$: Produces the update for the encryption key μ_{pk} and an update for the decryption key μ_{sk}.
- $UpdatePrivate(sk, \mu_{sk}) \rightarrow sk'$: Accepts as input sk and a secret update value μ_{sk}, and produces an updated secret key sk'.
- $UpdatePublic(pk, \mu_{pk}) \rightarrow pk'$: Accepts as input pk and a update value μ_{pk}, and produces as output a updated public key pk'.

Correctness. As in the Symmetric setting, Asymmetric UPKE constructions are correct if *Encrypt* and *Decrypt* operations can be correctly performed. before and after a series of *UpdatePublic* and *UpdatePrivate* operations.

Asymmetric UPKE†. Jost, Maurer, and Mularczyk [25] presented an ElGamal-based Asymmetric UPKE construction that we refer to as Asymmetric UPKE†. In Asymmetric UPKE†, the update value for the encryption key is in the *public* domain, while the update value for the decryption key is in the *secret* domain. Updates require a homomorphic one-way function f, such that $pk' = pk \circ \mu_{pk} = f(sk') = f(sk \circ \mu_{sk})$.

3.1 Security

We now present a generalization of IND-CPA (Indistinguishability under Chosen Message Attack) security for UPKE schemes described by Alwen et al. [1], which we define as "Indistinguishability under Chosen Plaintext Attacks with Updatability", or IND-CPA-U. We present this notion of IND-CPA-Usecurity in Figure 1. Our notion assumes a Symmetric UPKE construction, but extends to the Asymmetric UPKE setting by simply allowing the adversary to learn public update values.

In Alwen et al.'s definition, the adversary is given the public key pk_0 and provides a sequence of updates $\mu_1, \ldots, \mu_\tau$. The public and private keys are updated accordingly and the adversary is issued an IND-CPA challenge under pk_τ. The public and secret key are updated again, this time with a secret update, and the adversary is given the resulting public *and* secret key. They then must respond to the IND-CPA challenge.

Their model illustrates the fundamental idea behind how security works for updatable encryption: the adversary may learn (or even control) either the update value or the secret key, but as long as they do not have both, the updated secret key remains secure. However they have the restriction that the adversary controls the updates prior to the IND-CPA challenge, and receives the secret key afterwards. We generalize this by allowing the adversary to adaptively choose whether they want to control the update or learn a secret key, with the restriction that the IND-CPA challenge can only be issued on a public key that has not been compromised in a straightforward way.

We begin by generating a keypair (pk_0, sk_0) and sending pk_0 to $\mathcal{A}$. We initialize $i \leftarrow 0$ (the most recent version of the keypair will be (pk_i, sk_i)). After this we let the adversary decide how the key will be updated. To this end, we provide our adversary with the following oracles:

- The $\mathsf{GiveUpdate}(\mu)$ oracle takes in an update value μ. It increments $i \leftarrow i+1$, and then generates

$$(pk_i, sk_i) \leftarrow \mathit{UpdatePublic}(pk_{i-1}, \mu), \mathit{UpdatePrivate}(sk_{i-1}, \mu)$$

  before providing the new pk_i to the adversary.
- The $\mathsf{FreshUpdate}()$ oracle corresponds to updates happening that the adversary does not control or know the update value for. It generates a random $\mu \xleftarrow{\$} \mathit{GenUpdate}()$ and then calls $\mathsf{GiveUpdate}(\mu)$, providing the new pk_i to the adversary.
- The $\mathsf{Corrupt}(j)$ oracle provides sk_j for a index $j \leq i$.

Eventually, the adversary requests a challenge on an index $j \leq i$ and provides messages (m_0, m_1). A bit $b \xleftarrow{\$} \{0,1\}$ is sampled and $c_b \leftarrow \mathit{Encrypt}(m_b, pk_j)$ is provided back to the adversary. After making further queries to the update and corruption oracles, the adversary must issue a bit b'. They are said to win if $b = b'$ and the index j is *fresh*. The freshness requirement ensures that the adversary cannot trivially win.

An index j is considered fresh if:

- The adversary has not called $\mathsf{Corrupt}(j)$, and

IND-CPA-U(Indistinguishability Under Chosen Plaintext Attack with Updatability) Game:

$(pk_0, sk_0) \leftarrow KeyGen(\lambda);\ i = 0$ // *Derive the starting keypair*

$j, m_0, m_1, \mathsf{st} \leftarrow \mathcal{A}_0(pk_0)$ // *Adversary queries oracles, returns index and challenge messages*

$b \xleftarrow{\$} \{0,1\},\ c_b \leftarrow Encrypt(pk_j, m_b)$ // *Generate challenge ciphertext*

$b' \xleftarrow{\$} \mathcal{A}_1(c_b, \mathsf{st})$ // *The adversary outputs a guess for b*

Adversary wins if $\mathsf{IsFresh}(j)$ and $b' = b$.

where in addition to the normal IND-CPA oracles, the adversary has access to the oracles:

- $\mathsf{GiveUpdate}(\mu) \rightarrow pk_i$, after performing // *The adversary can update keys with chosen value*
  $i = i + 1; (pk_i, sk_i) \leftarrow UpdatePublic(pk_{i-1}, \mu), UpdatePrivate(sk_{i-1}, \mu);$
  $U = U \cup i$ // *Keep track of updates the adversary has provided*
- $\mathsf{FreshUpdate}() \rightarrow pk_i$, after performing // *Update with value not chosen by adversary*
  $i = i + 1;\ \mu \xleftarrow{\$} GenUpdate();\ (pk_i, sk_i) \leftarrow UpdatePublic(pk_{i-1}, \mu), UpdatePrivate(sk_{i-1}, \mu)$
- $\mathsf{Corrupt}(j)$, returning sk_j after performing the following steps: // *Allow the adversary to learn the j^{th} keypair*
  $C = C \cup j$
  $i = j;$ while $i \in U$ do : $C = C \cup i,\ i = i - 1;$ // *Left-adjacent updates chosen by $\mathcal{A}$ are now corrupt*
  $k = j;$ while $(k+1) \in U$ do : $C = C \cup k,\ k = k + 1;$ // *Right-adjacent updates from $\mathcal{A}$ are also corrupt*

We define $\mathsf{IsFresh}(j)$ to return true if and only if $j \notin C$

Fig. 1. IND-CPA-Uexperiment, where the adversary issues a guess in the regular IND-CPA game after performing a series of arbitrary updates and corruptions. To ensure the adversary cannot trivially win, we require the secret key under which the challenge is issued to be fresh, meaning the adversary cannot derive its value from simply having corrupted that key or a prior key from which its value can be derived.

– There is not a sequence of updates (in either direction) all of which from $\mathsf{GiveUpdate}$ that connects the index j to an index k for which $\mathsf{Corrupt}(k)$ has been called.

In Figure 2 we visualize how the queries that the adversary has performed cause a given index to be considered fresh or not.

Let W_b denote the event that Experiment b defined in Figure 1 outputs b. We define the advantage of an adversary $\mathcal{A}$ against a UPKE scheme $\mathcal{U}$ as $\mathsf{Adv}(\mathcal{A},\mathcal{U}) = \Pr[W_b] - 1$.

Definition 6. *A UPKE scheme is IND-CPA-Usecure if for any polynomial-time adversary $\mathcal{A}$, the value of $\mathsf{Adv}(\mathcal{A},\mathcal{U})$ is negligible.*

Unlike IND-CPA games for plain PKE schemes, the IND-CPA-Udefinition presented in Definition 6 captures the notion of forward secrecy and post-compromise security by allowing $\mathcal{A}$ to learn any secret key material and provide whatever update values that it wishes, with conditions preventing the adversary from trivially winning the IND-CPA game.

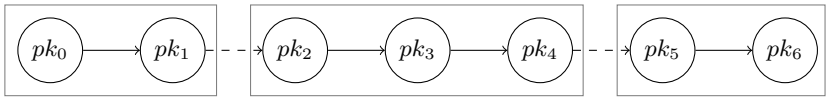


Fig. 2. A series of updates applied to public keys. In this diagram, updates that occur as a result of GiveUpdate queries are solid, while updates that are a result of FreshUpdate queries are dashed. Keys that are connected by updates that the adversary has provided or has knowledge of can be viewed as a block. If one index in a given block is compromised, leaking the secret key, then the secret key for any index in the block can be calculated. In our security model, such indices are not considered fresh, and if the adversary requests the challenge on such an index, they are penalized.

4 Assessing Isogeny-Based UPKE

As seen in Section 3, UPKE constructions require that *UpdatePrivate* output an sk_u whose distributed is *independent* from sk, to ensure forward secrecy and post-compromise security. Further, this operation should be *asynchronous*, so that external parties can update the public key without requiring the keyholder to be online.

With these requirements in mind, isogeny-based cryptography presents an attractive option for UPKE constructions. While lattice-based cryptography can support key-exchange operations, in the setting where an unbounded number of update operations can be performed, the security of existing lattice-based constructions unfortunately degrades with each update operation. [4]

We now discuss the extent to which existing isogeny-based schemes—SIDH and CSIDH—can support Symmetric and Asymmetric UPKE constructions.

Isogeny-Based Symmetric UPKE. As described in Section 4, Symmetric UPKE constructions apply the same update value μ to both sk and pk. Unfortunately, several practical limitations impact both SIDH and CSIDH-based UPKE constructions, which we discuss in Sections 5 and 6.

We compare our constructions against a naive "online" UPKE construction that we call *Double Encrypt*, that does *not* achieve the desired goals of asynchronicity and fine-grained forward secrecy, which we aim to improve upon. As a naive "starting point" construction, *Double Encrypt* simply allows any party to select a keypair at random and perform nested encryption to the recipient using the recipient's long-lived keypair and this ephemeral public key (after sending the recipient the corresponding secret key, encrypted to their long-lived keypair). When the recipient comes online, they simply generate a fresh long-lived keypair. *Double Encrypt*achieves all properties required of a UPKE scheme as defined in Section 3 except for asynchronicity and fine-grained forward secrecy. Specifically, *Double Encrypt* maintains a static key for each "window" before and after the keyholder performs an update, and so the forward secrecy of *Double Encrypt* is maintained only for each window. We provide specific details of *Double Encrypt* in Appendix B,

Isogeny-Based Asymmetric UPKE. Definition 5 describes a generalized notion of Asymmetric UPKE, followed by Asymmetric UPKE†, a concrete construction in the literature [25].

[4] For example, the obvious thing to do with a lattice-based system is to add together two (ring)-LWE samples to update a public key. However the corresponding secret key will then be the sum of two (ring)-LWE secrets. The distribution of the resulting secret will be dependent on the previous secret, making it difficult to argue for the security of such a system.

We now describe why there is no straightforward way to use SIDH or CSIDH to instantiate a Asymmetric UPKE† style construction. We narrow our assessment to Asymmetric UPKE† in order to effectively determine the assumed mathematical structure.

In Definition 5, *UpdatePublic* applies a public update value μ_{pk} to pk, whereas *UpdatePrivate* applies a *private* update value to sk. As such, Asymmetric UPKE† requires a homomorphic structure between the public and private domains.

More formally, the Asymmetric UPKE† construction assumes the existence of a function f that maps the private domain to a public one. For example, $pk = f(sk)$, and $f(\mu_{sk}) = \mu_{pk}$. For a discrete logarithm-based system like Asymmetric UPKE†, f is the simple mapping $f : x \mapsto g^x$. The public key update operation then works because the domain and codomain of f have group operations $\times$ and $\star$, and $f(\mu_{sk} \times sk) = f(\mu_{sk}) \star f(sk) = \mu_{pk} \star pk$.

The Failure of both SIDH and CSIDH in trying to instantiate Asymmetric UPKE† is that the protocol requires f to be a group homomorphism, and current isogeny schemes offer, at best, a group action. SIDH defines only one operation—applying isogenies via Vélu's formula—over elements in the set of supersingular curves over $\mathbb{F}_{p^2}$. While SIDH can "combine" public and private values by some (non-group) operation, SIDH does not define a group operation between group elements. More clearly, it is difficult to imagine how to apply one public value in SIDH—a supersingular curve—to another. As such, SIDH *cannot* be used in a straightforward way as the underlying public-key encryption primitive to construct the Asymmetric UPKE† scheme.

For similar reasons, CSIDH cannot be used for an Asymmetric UPKE† construction. The private values in CSIDH form a commutative group, so one might hope for more algebraic structure to be useful. But for Asymmetric UPKE, we need some sort of structure in the public domain. Public values in CSIDH are still elliptic curves, and without an operation that can be applied between elliptic curves or a new way to separate private and public values, CSIDH as currently defined also cannot support Asymmetric UPKE constructions.

5 Symmetric UPKE via SIDH

A naive SIDH-based Symmetric UPKE construction would simply generate an isogeny μ as a secret update, and perform a plain SIDH key exchange using μ to update the keyholder's public key $(E_A, \phi_A(P_B), \phi_A(Q_B))$ under $\mu(E_A)$, to obtain the updated public key $(E_{A\mu}, \phi_{A\mu}(P_B), \phi_{A\mu}(Q_B))$. After sending μ via an encrypted channel to the keyholder, the keyholder would similarly update their secret key $\phi_A : E_0 \to E_A$ by performing a plain SIDH key exchange obtaining $sk' = \phi_{A\mu} = \langle \phi_\mu(P_A) + [n_A]\phi_\mu(Q_A) \rangle$. Unfortunately, this naive construction is *not* forward secure, as the torsion points remain linearly dependent after each update. We discuss this limitation further in Appendix C.

We now describe an "online" SIDH UPKE construction that achieves roughly the same windowed forward secrecy as *Double Encrypt*. Our construction requires the keyholder to publish their updated public key after updating their corresponding private key using KLPT*. However, other participants can perform a "partial update" of other participant's public keys non-interactively to achieve some measure of forward secrecy. Note that we present this construction purely as a proof of concept and to demonstrate where gaps exist in the effort to instantiate SIDH-based Symmetric UPKE.

At a high level, the output of KLPT* will be broken into SIDH-sized chunks. The elliptic curve after each chunk will form part of the public key. This means that we will be performing an SIDH-style key exchange with the secret isogeny of one party being significantly longer than

the other. Instances where one party's isogeny is longer than the others are called 'unbalanced' or 'overstretched', and have been considered in a cryptanalytic context previously [27]. We touch on this further in Section 7. Due to these potential security problems and because we only introduce this scheme to motivate a potential approach, we do not provide a security proof, nor do we claim the existence of one.

An "Online" Construction. Since KLPT* outputs isogenies of degree greater than 2^{e_1} and the point generating the isogeny's kernel lies outside of $E_0(\mathbb{F}_{p^2})$, our construction represents secrets as a composition of k SIDH-sized isogenies, and public keys as a set of k curves. Let the curves E_{A_1} to E_{A_k} represent part of Alice's secret key, and the curves E_{A_1B} to E_{A_kB} be a shared secret between Alice and Bob obtained by performing one SIDH operation per curve in each party's keys. The issue to avoid is that after performing an update, Bob cannot send both of his auxiliary points to Alice on the shared curve E_{A_iB}, as an attacker could compute the curve using these two points. This is because elliptic curves used in SIDH have two parameters and the knowledge of two point on a curve generates a solvable linear equation system. Hence, an attacker having access the intermediate shared secret curves would then only need to break the final SIDH exchange, making the scheme less secure. We describe a fix that sends only both auxiliary points on the first curve but only a single point for each subsequent curve.

Let (P_{A_i}, Q_{A_i}) be a basis of the 2^{f_1}-torsion of E_{A_i} and let $(P_{A_{i+1}}, Q_{A_{i+1}})$ be a basis of the 2^{f_1}-torsion of $E_{A_{i+1}}$. Alice can write $\phi_{A_{i+1}}(P_{A_i})$ and $\phi_{A_{i+1}}(Q_{A_i})$ as linear combinations of $P_{A_{i+1}}$ and $Q_{A_{i+1}}$. This can be done by using a pairing defined on points of $E_{A_{i+1}}$, for example, the Weil pairing, to reduce the problem of finding a linear combination to finding a discrete logarithm. Since $\phi_{A_{i+1}}$ has a cyclic kernel, we have that $(P_{A_{i+1}}, \phi_{A_{i+1}}(P_{A_i}), \phi_{A_{i+1}}(Q_{A_i}))$ or $(Q_{A_{i+1}}, \phi_{A_{i+1}}(P_{A_i}), \phi_{A_{i+1}}(Q_{A_i}))$ generates the entire 2^{f_1}-torsion of $E_{A_{i+1}}$. Using linear algebra, Alice can then check which of $P_{A_{i+1}}$ or $Q_{A_{i+1}}$ is required to generate the entire torsion, using the corresponding auxiliary point $\phi_{B_{i+1}}(P_{A_{i+1}})$ sent by Bob. From this, Alice can use the commutativity of SIDH key exchange to compute the other auxiliary point. Once all k SIDH schemes have been completed, the shared secret between Alice and Bob is the j-invariant of the final curve E_{A_kB}. We call this variation on the scheme "extended SIDH" as the secret key is a chain of isogenies whose composition is of much higher degree than the usual SIDH isogeny.

This induces the following "online" UPKE scheme. Note that this construction deviates from the definition of Symmetric UPKE presented in Definition 4, in that *UpdatePrivate* outputs the completely updated public key (achieving better forward secrecy), whereas *UpdatePublic* outputs a partially updated public key (achieving partial forward secrecy). Note that *UpdatePrivate* is necessary to obtain forward secrecy since, otherwise, the initial secret chain of isogenies would simply be a subchain of the updated secret key.

- *KeyGen*(λ): Sample $(\alpha_1, \beta_1), \dots (\alpha_k, \beta_k) \xleftarrow{\$} \mathbb{Z}_{2^{e_1}}^2$. Let $sk = (\phi_{A_1}, \dots, \phi_{A_k})$ as a chain of 2^{f_i}-isogenies $E = E_{A_0} \xrightarrow{\phi_{A_1}} E_{A_1} \xrightarrow{\phi_{A_2}} E_{A_2} \xrightarrow{\phi_{A_3}} \cdots \xrightarrow{\phi_{A_k}} E_{A_k}$ with $f_i \leq e_1$. Set pk to be the tuple $(E_A, \phi_{A_1}(P_B), \phi_{A_1}(Q_B))$ and a list of tuples $(E_{A_{i+1}}, \phi_{A_{i+1}}(P_{B_i}), \phi_{A_{i+1}}(Q_{B_i}), G_i)$. Both sk and pk are defined recursively by doing the following:
  - P_{A_i} and Q_{A_i} form the canonical basis of the 2^{e_1}-torsion of E_{A_i}. P_{B_i} and Q_{B_i} form the canonical basis of the 3^{e_2}-torsion of E_{A_i}.
  - $\phi_{A_{i+1}}$ is the $2^{f_{i+1}}$-isogeny from E_{A_i} whose kernel is the cyclic group generated by $\alpha_{i+1}P_{A_i} + \beta_{i+1}Q_{A_i}$.

- $E_{A_{i+1}}$ is the codomain of $\phi_{A_{i+1}}$.
- G_i is the additional auxiliary point on E_{A_iB} used to complete the SIDH scheme for the $(i+1)$th isogeny, either $\phi_{B_{i+1}}(P_{A_{i+1}})$ or $\phi_{B_{i+1}}(Q_{A_{i+1}})$. To determine which, we define a *ChooseAuxiliary* algorithm, described in Section 5.1.
- Output (sk, pk).

– *Encrypt*(pk, m): Sample $b \xleftarrow{\$} \mathbb{Z}_{3^{e_2}}$ which defines $\phi_B \colon E_0 \to E_{\mathsf{enc}}$. From E_{enc}, compute E_{A_kB} using Algorithm 3 of Section 5.2. Compute $K \xleftarrow{\$} KDF(j(E_{A_kB}))$ and $ctxt \leftarrow DEM.Encrypt(K, m)$. Let $ct = (E_{\mathsf{enc}}, ctxt)$. Output ct.

– *Decrypt*(sk, c): Parse ct as $(E_{\mathsf{enc}}, ctxt)$; set $K \xleftarrow{\$} KDF(j(E_{\mathsf{enc}}))$. Output $DEM.Decrypt(K, ctxt)$.

– *GenUpdate*$(\lambda) \to \mu$: Sample $(\alpha', \beta') \xleftarrow{\$} \mathbb{Z}_{2^{e_1}}^2$. Produce the isogeny $\mu = \phi_{A'} \colon E_{A_k} \to E_{A'}$, of degree $2^{f'}$ with $f' \leq e_1$ and kernel generated by $\alpha' P_{A_k} + \beta' Q_{A_k}$. Output μ.

– *UpdatePublic*$(pk, \mu) \to pk'$: Add the tuple $(E_{A'}, \phi_{A'}(P_{B_k}), \phi_{A'}(Q_{B_k}), G_k)$ to pk, resulting in pk'.

– *UpdatePrivate*$(sk, \mu) \to (sk', pk'')$: Append $\mu = \phi_{A'}$ to the isogeny chain sk. Apply KLPT* on the composition of the isogenies in sk to obtain a new chain of isogenies $E \xrightarrow{\phi_{A'_1}} E_{A'_1} \xrightarrow{\phi_{A'_2}} E_{A'_2} \xrightarrow{\phi_{A'_3}} \cdots \xrightarrow{\phi_{A'_\ell}} E_{A'}$. Output $sk' = (\phi_{A'_1}, \ldots, \phi_{A'_\ell})$ and completely updated public key pk''.

So long as Alice and Bob choose the same canonical torsion basis of each elliptic curve, G_i can be encoded using a single bit and does not depend on Bob's ephemeral key. This can be done by having Alice and Bob use the same algorithm to find the basis.

We now elaborate on the extent to which our construction can achieve the required properties of a UPKE scheme.

Correctness: Since both the initial and updated keys are a chain of SIDH protocols, correctness is respected.

Forward secrecy: Because the output of KLPT* will not leak information about its inputs (if the underlying assumptions of KLPT* regarding fixed-length outputs and the randomness of the output hold), an attacker obtaining the updated key gains no information on the keys before the last application of KLPT*. As such, our scheme achieves forward secrecy roughly comparable to *Double Encrypt*, since all participants get some measure of "windowed" forward secrecy between when an external party performs *UpdatePublic*, and the keyholder performs *UpdatePrivate*.

Post-compromise security: Since the output of KLPT* will not leak information about its inputs (again, if its underlying assumptions hold), an attacker looses all information once KLPT* is reapplied during the next update.

Asynchronicity: This scheme cannot support asynchronous updates.

Key Indistinguishability: This scheme does not provide key indistinguishability.

As this construction does not fully instantiate the requirements for Symmetric UPKE (due to the lack of asynchronicity), we purely present this construction as a proof of concept not intended for use in practice. For this reason as well as the potential security problems introduced by overstretching SIDH parameters, we do not make formal claims of security. Hence, while we discuss how to avoid current torsion point attacks in Section 7, we omit a formal proof of security.

5.1 Choosing and Getting Auxiliary Points

Recall that in *KeyGen*, the public key includes one additional auxiliary point on E_{A_iB}, which is used to complete the SIDH scheme for the $(i+1)$th isogeny. We now describe the implementation of *ChooseAuxiliary*, an algorithm to determine whether this auxiliary point is on $\phi_{B_{i+1}}(P_{A_{i+1}})$ or $\phi_{B_{i+1}}(Q_{A_{i+1}})$.

Let G_i be the received auxiliary point (either $\phi_{B_{i+1}}(P_{A_{i+1}})$ or $\phi_{B_{i+1}}(Q_{A_{i+1}})$) and let $\psi_{A_{i+1}} : E_{A_iB} \to E_{A_{i+1}B}$ be isogeny induced by the ith SIDH exchange. Let $\mathsf{WeilPairing}(E, P, Q, n)$ be the Weil pairing of two points P and Q of order dividing n in the elliptic curve E, and $\mathsf{DiscreteLog}(a, b, n)$ be the discrete logarithm of a by b, two elements in a group of order dividing n. Let $\mathsf{IsLinearCombination}([U_1, \ldots, U_k], V)$ be a function returning True if V is a linear combination of $U_1, \ldots, U_k$ and False otherwise. Finally, let $\mathsf{LinearCombination}([U_1, \ldots, U_k], V)$ be a function returning $(a_1, \ldots, a_k)$ such that $V = \sum_{i=1}^{k} a_i U_i$.

The *ChooseAuxiliary* algorithm selects the information required to evaluate the auxiliary points for the next exchange, by the following steps.

- Use Weil pairings and discrete logarithms to define $\phi_{A_i}(P_{A_i})$ and $\phi_{A_i}(Q_{A_i})$ as linear combinations of $P_{A_{i+1}}$ and $Q_{A_{i+1}}$.
- Use linear algebra to check if $Q_{A_{i+1}}$ is a linear combination of $P_{A_{i+1}}$, $\phi_{A_i}(P_{A_i})$ and $\phi_{A_i}(Q_{A_i})$. If that is the case, choose $\phi_{B_{i+1}}(P_{A_{i+1}})$. Otherwise, choose $\phi_{B_{i+1}}(Q_{A_{i+1}})$.

We define *ChooseAuxiliary* more precisely in Algorithm 1.

Algorithm 1 Auxiliary points decision algorithm

procedure ChooseAuxiliary$(E_{A_i}, E_{A_{i+1}}, P_{A_i}, Q_{A_i}, P_{A_{i+1}}, Q_{A_{i+1}}, \phi_{A_{i+1}})$
  $w_1 \leftarrow \mathsf{WeilPairing}(E_{A_{i+1}}, P_{A_{i+1}}, Q_{A_{i+1}}, A)$
  $w_1 \leftarrow \mathsf{WeilPairing}(E_{A_{i+1}}, Q_{A_{i+1}}, P_{A_{i+1}}, A)$
  $w_3 \leftarrow \mathsf{WeilPairing}(E_{A_{i+1}}, P_{A_{i+1}}, \phi_{A_{i+1}}(P_{A_i}), A)$
  $w_4 \leftarrow \mathsf{WeilPairing}(E_{A_{i+1}}, P_{A_{i+1}}, \phi_{A_{i+1}}(Q_{A_i}), A)$
  $w_5 \leftarrow \mathsf{WeilPairing}(E_{A_{i+1}}, Q_{A_{i+1}}, \phi_{A_{i+1}}(P_{A_i}), A)$
  $w_6 \leftarrow \mathsf{WeilPairing}(E_{A_{i+1}}, Q_{A_{i+1}}, \phi_{A_{i+1}}(Q_{A_i}), A)$
  $v_{11} \leftarrow \mathsf{DiscreteLog}(w_5, w_2, A)$
  $v_{12} \leftarrow \mathsf{DiscreteLog}(w_3, w_1, A)$
  $v_{21} \leftarrow \mathsf{DiscreteLog}(w_6, w_2, A)$
  $v_{22} \leftarrow \mathsf{DiscreteLog}(w_4, w_1, A)$
  $V_1 \leftarrow (v_{11}, v_{12})$ ▷ $\phi_{A_{i+1}}(P_{A_i}) = v_{11}P_{A_{i+1}} + v_{12}Q_{A_{i+1}}$
  $V_2 \leftarrow (v_{21}, v_{22})$ ▷ $\phi_{A_{i+1}}(Q_{A_i}) = v_{21}P_{A_{i+1}} + v_{22}Q_{A_{i+1}}$
  if $\mathsf{IsLinearCombination}([V_1, V_2], (1, 0))$ **then**
    $b \leftarrow 1$ ▷ $b = 0$ is Alice requires the auxiliary point of $P_{A_{i+1}}$ and $b = 1$ if $Q_{A_{i+1}}$ is required instead.
  else if $\mathsf{IsLinearCombination}([V_1, V_2], (0, 1))$ **then**
    $b \leftarrow 0$
  else
    $b \xleftarrow{\$} \{0, 1\}$
  return (b, V_1, V_2)

Once the additional point G_i has been received, the *GetAuxiliary* algorithm evaluates the auxiliary points by solving the equation system obtained with *ChooseAuxiliary*. We define *GetAuxiliary* more precisely in Algorithm 2.

Algorithm 2 Auxiliary points evaluation algorithm

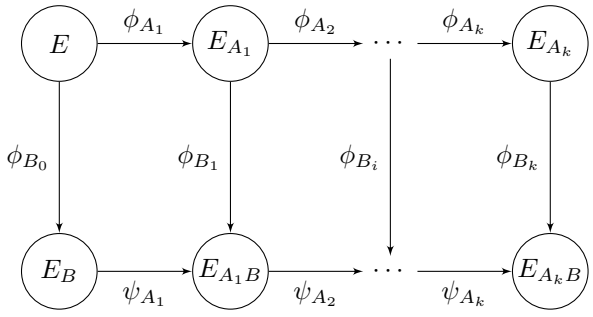
procedure GetAuxiliary($E_{A_i}, E_{A_{i+1}}, P_{A_i}, Q_{A_i}, P_{A_{i+1}}, Q_{A_{i+1}}, \phi_{A_{i+1}}, \psi_{A_{i+1}}, \phi_{B_i}(P_{A_i}), \phi_{B_i}(Q_{A_i}), G_i$)
  $(b, V_1, V_2) \leftarrow$ ChooseAuxiliary($E_{A_i}, E_{A_{i+1}}, P_{A_i}, Q_{A_i}, P_{A_{i+1}}, Q_{A_{i+1}}, \phi_{A_{i+1}}$)
  if $b = 0$ **then**
    $P \leftarrow G_i$
    $(a_1, a_2, a_3) \leftarrow$ LinearCombination($[V_1, V_2, (1, 0)], (0, 1)$)
    $Q \leftarrow a_1\psi_{A_{i+1}}(\phi_{B_i}(P_{A_i})) + a_2\psi_{A_{i+1}}(\phi_{B_i}(Q_{A_i})) + a_3G_i$
  else
    $Q \leftarrow G_i$
    $(a_1, a_2, a_3) \leftarrow$ LinearCombination($[V_1, V_2, (0, 1)], (1, 0)$)
    $P \leftarrow a_1\psi_{A_{i+1}}(\phi_{B_i}(P_{A_i})) + a_2\psi_{A_{i+1}}(\phi_{B_i}(Q_{A_i})) + a_3G_i$
  return (P, Q) ▷ $P = \phi_{B_{i+1}}(P_{A_{i+1}})$ and $Q = \phi_{B_{i+1}}(Q_{A_{i+1}})$

The idea of using Weil pairings to verify if a set of points generates the torsion group of an elliptic curve was first discussed in [31]. The algorithm in that paper, although different from Algorithm 2, uses the same fundamental idea of using Weil pairings to reduce the problem to a discrete logarithm computation. However, the construction in Section 5 is the first application of that algorithm to allow SIDH exchanges with isogenies with degree larger than p. We refer to these exchanges as "extended SIDH", which we further explain in Section 5.2.

5.2 Extended SIDH

Recall that "extended SIDH" refers to the operation of performing one plain SIDH operation for each element in sk, as sk is a chain of isogenies whose composition is of higher degree than p. We employ extended SIDH in *Encrypt* and *Decrypt* to derive the shared key K.

We now describe how to perform this extended SIDH operation in detail. Let $sk = (\phi_{A_1}, \ldots, \phi_{A_k})$ be Alice's private isogeny chain with $\ker \phi_{A_{i+1}} = \alpha_{i+1}P_{A_i} + \beta_{i+1}Q_{A_i}$. Let (P_{B_0}, Q_{B_0}) be a basis of the 3^{e_2}-torsion on $E = E_{A_0}$. Let $(P_{B_{i+1}}, Q_{B_{i+1}}) = (\phi_{A_{i+1}}(P_{B_i}), \phi_{A_{i+1}}(Q_{B_i}))$. Let ϕ_B be Bob's secret isogeny on E with kernel $\langle P_{B_0} + sk_BQ_{B_0}\rangle$ for a secret integer sk_B. Let ϕ_{B_i} be the associated 3^{e_2}-isogeny on E_{A_i} with kernel $\langle P_{B_i} + sk_BQ_{B_i}\rangle$. The idea behind Extended SIDH is the following commutative diagram:



Bob computes E_{A_kB} when generating ϕ_{B_k}. Alice uses the following algorithm.

Let $\phi_B(P_A)$ and $\phi_B(Q_A)$ be the auxiliary points given by Bob on the initial curve E. Let E_B be the public codomain of ϕ_B Let $G = (G_1, \ldots, G_{k-1})$ be the list of auxiliary points given by Bob on each of curve $E_{A_1}, \ldots, E_{A_{k-1}}$. Let $\mathsf{IsogenyFromKernel}(E, K)$ be a function returning the isogeny with kernel generated by the point $K \in E$, and $\mathsf{IsogenyCodomain}(\phi)$ be a function returning the codomain of the isogeny ϕ.

The *ExtSIDH* computes E_{A_kB} by repeating the following steps for i from 0 to $k-1$:

- Using Algorithm 2, compute the auxiliary points $(\phi_{B_i}(P_{A_i}), \phi_{B_i}(Q_{A_i})$ of the 3^{e_2}-torsion basis of E_{A_i}. This step is not necessary for the first isogeny ϕ_{A_1} as both its auxiliary points are given.
- Compute $\psi_{A_{i+1}}$ the isogeny from E_{A_iB} with kernel generated by $\alpha_{i+1}\phi_{B_i}(P_{A_i}) + \beta_{i+1}\phi_{B_i}(P_{A_i})$.
- Compute $E_{A_{i+1}B}$ the codomain of $\psi_{A_{i+1}}$.

We define *ExtSIDH* more precisely in Algorithm 3.

Algorithm 3 Extended SIDH

procedure ExtSIDH($sk, E_B, \phi_B(P_A), \phi_B(Q_A), G$)
  $(P, Q) \leftarrow (\phi_B(P_A), \phi_B(Q_A))$
  $\psi_{A_1} \leftarrow \mathsf{IsogenyFromKernel}(E_B, \alpha_1\phi_B(P_A) + \beta_1\phi_B(Q_A))$
  $E_{A_1B} \leftarrow \mathsf{IsogenyCodomain}(\psi_{A_1})$
  for $0 \le i \le k-2$ **do**
    $(P, Q) \leftarrow \mathsf{GetAuxiliary}(E_{A_i}, E_{A_{i+1}}, P_{A_i}, Q_{A_i}, P_{A_{i+1}}, Q_{A_{i+1}}, \phi_{A_{i+1}}, \psi_{A_{i+1}}, P, Q, G_i)$
    $\psi_{A_{i+2}} \leftarrow \mathsf{IsogenyFromKernel}(E_B, \alpha_{i+1}P + \beta_{i+1}Q)$
    $E_{A_{i+2}B} \leftarrow \mathsf{IsogenyCodomain}(\psi_{A_{i+2}})$
  return E_{A_kB}

6 Symmetric UPKE Construction via CSIDH

We described in the prior section the difficulties in constructing a SIDH-based UPKE scheme due to the simultaneous requirements for asynchronicity, forward secrecy, and post-compromise security. We now show how CSIDH, combined with knowledge of the class group structure, can overcome these challenges and construct a scheme that satisfies all notions of a secure and useful UPKE.

CSIDH UPKE, first attempt. CSIDH admits operations that are much closer to those used in the classical construction from Alwen et al. [1]. Recall that secret keys in CSIDH are represented by a vector of ℓ integers. For efficiency reasons, the integers are usually chosen to be within a bound B, for example, $B = 5$ so that all entries are between -5 and 5. Then the group element $[e_1, e_2, \ldots, e_\ell]$ represents the group element

$$\mathfrak{g}_1^{e_1}\mathfrak{g}_2^{e_2}\cdots\mathfrak{g}_\ell^{e_\ell},$$

for a set of canonical generators $\{\mathfrak{g}_i\}$. Since the group is commutative, we have that if g is represented by $[e_1, \ldots, e_\ell]$ and h is represented by $[f_1, \ldots, f_\ell]$ then $g \cdot h$ can be represented by $[e_1+f_1, \ldots, e_\ell+f_\ell]$. A basic design for a symmetric UPKE scheme would then be for the update value to be a random

group element, to update the public key by applying the group action, and to update the secret key by adding the group elements together.

Unfortunately, this simple design is not secure. If each entry for the update value is drawn uniformly from $-B$ to B, then the distribution of each entry of the new public key is centered at the old public key. This leaks a certain amount of information about the old secret key. For example, if only one update has occurred, if an entry is $2B$ then the adversary immediately knows that the corresponding entry before the update must have been B.

One fix may be to increase the bound B in an attempt to show that leaking the secret key between certain updates still doesn't reveal enough of the secret key to allow a break. Such an analysis must be done carefully, but reveals another fundamental problem. As more updates occur, the size of each entry in the vector is likely to grow. The efficiency of CSIDH is directly dependent on the ℓ_1-norm of this vector, and so allowing it to grow with updates will result in a slower and slower decryption process, eventually becoming unacceptable.

Note that this is almost exactly the same problem that a first attempt at a lattice-based scheme would run into. If one were to define a scheme based on the LWE problem, then updates could be generated by sampling an LWE secret. The secret key would then be updated by adding the update value to the old secret. But as described for CSIDH, this will cause the error term to grow over time, eventually causing the system to fail. Furthermore, because errors are not chosen uniformly, the distribution of a secret will always be dependent on the previous secret, meaning some information about previous keys is leaked in the event of a compromise.

One technique to circumvent this problem that has been to employ rejection sampling, as in the signature scheme SeaSign [14]. However, rejection sampling only works when we can reject the group elements that would leak information on the secret key. Since the party selecting the update value is not the owner of the public key, rejection sampling is not an option in our scenario. Instead, we will need the group elements to be represented in a way that has better properties.

As mentioned in Section 2.1, the signature scheme CSI-FiSh uses a different representation for group elements. Let N denote the order of the group. To compute the group action (i.e., apply the isogeny to an elliptic curve) one converts an element of $\mathbb{Z}_N$ (represented simply by an integer) to an ideal in $\mathbb{Z}^\ell$ and then applies the action as in CSIDH. Representing group elements as an integer in $\mathbb{Z}_N$ gives a unique representation. It is also still very easy to apply the group operation in this representation — it is just addition modulo N.

Our Construction. To prevent leakage from secret key updates described above, our construction requires a class group structure that is fully known, so that the secret key and update value can both be represented in $\mathbb{Z}_N$. The calculation of this value N as well as the methodology to convert the representation was a major contribution of the CSI-FiSh paper [4]. To update a public key, we apply the group action, and to update the secret key we add modulo N. Because we can sample uniformly over $\mathbb{Z}_N$, we have that the updated secret key leaks no information about the previous secret key, as desired.

We now describe the scheme in full, relying heavily on group action notation. Let N be the order of the class group $cl(\mathcal{O}) \cong \mathbb{Z}_N$. To apply the group action onto a supersingular elliptic curve E (denoted $g \star E$), we first need to convert the element to a representation in $\mathbb{Z}^\ell$ with a low L_1 norm, and then apply the action as in the original CSIDH paper.

- $KeyGen(\lambda)$: Sample $g_{sk} \xleftarrow{\$} \mathbb{Z}_N$ and set $E_{pk} := g_{sk} \star E_0$. Output $(sk, pk) = (g_{sk}, E_{pk})$.

- *Encrypt*(pk, m): Sample $g_{\mathsf{enc}} \xleftarrow{\$} \mathbb{Z}_N$ and compute $K \leftarrow KDF(g_{\mathsf{enc}} \star pk)$, $E_{\mathsf{enc}} \leftarrow g_{\mathsf{enc}} \star E_0$, and $ctxt \leftarrow DEM.Encrypt(K, m)$. Output $ct = (E_{enc}, ctxt)$.
- *Decrypt*(sk, ct): Parse ct as $(E_{\mathsf{enc}}, ctxt)$, set $K \leftarrow KDF(sk \star E_{enc})$, output $DEM.Decrypt(K, ctxt)$.
- *GenUpdate*(): Sample $\mu \xleftarrow{\$} \mathbb{Z}_N$.
- *UpdatePrivate*(sk, μ): Output $sk' \leftarrow sk + \mu \pmod N$.
- *UpdatePublic*(pk, μ): Output $pk' \leftarrow \mu \star pk$.

Theorem 1. *Let $\mathcal{A}$ be an adversary capable of winning the IND-CPA-Ugame with advantage ϵ that makes q_{gen} queries to the* FreshUpdate *oracle. We will construct an adversary capable of winning an IND-CPA game in time approximately equal to the running time of $\mathcal{A}$ with advantage $\epsilon/(q_{gen}+1)$.*

We demonstrate that our construction attains IND-CPA-Usecurity, by showing a reduction from an adversary capable of winning the IND-CPA-Ugame to one that can win a plain IND-CPA game. By a plain IND-CPA game, we mean a game in which no calls to the *GenUpdate*, GiveUpdate, or Corrupt oracles are made.. We present our complete proof in Appendix D.

Implementation. Because our scheme requires the structure of the class group to be known, our CSIDH-based scheme can only be instantiated if such a computation has been performed. At the present time, this requirement limits us to the CSIDH-512 parameter set, which claims 64 bits of post-quantum security. Peikert [32] has questioned this security claim, and more recent analysis [9] indicates that CSIDH-4096 is necessary for NIST level 1 security. Computing the structure of the class group is a sub-exponential computation, and so becomes feasible with the availability of a quantum computer to perform the computation. As such, the scheme may not be able to be instantiated until it is most needed.

Other than computing the class group, the main challenge in an implementation is in computing the group action. To compute the group action, the element of $\mathbb{Z}_N$ is converted to a vector in $\mathbb{Z}^\ell$, which represents the group element $\prod_{i=1}^{\ell} \mathfrak{g}_i^{e_i}$ for a vector $\vec{e}$ and set of generators $\{\mathfrak{g}_i\}_i$. This vector is then applied to the elliptic curve as is done in CSIDH.

Thus the additional complication over any other CSIDH implementation is in converting the element of $\mathbb{Z}_N$ to a vector of integers. This process is described in the CSI-FiSh paper, and the authors have provided code to do this (for the CSIDH-512 parameter set). The authors of CSI-FiSh found that the process of converting to a vector only makes a key negotiation 15% slower. Using their implementation of CSI-FiSh, we have a proof of concept script that illustrates the process of updating the secret and public keys. Our script is available at `https://github.com/tedeaton/CSIDH-UPKE`.

7 Future Research Directions

In UPKE constructions, to ensure post-compromise and forward secrecy, the update value be properly 'mixed in' with the secret key when performing an update.

Recall our SIDH construction presented in Section 5. The KLPT* algorithm performs a functionality very close to what is needed; however, the outputs of KLPT* are too long to be computed within SIDH. As we have seen, while 'breaking up' the KLPT* output into SIDH-sized segments allows for computation, it does not result in an asynchronous protocol. Furthermore, since each

update increases the length of Alice's total isogeny, a large number of updates would make Alice's isogeny much longer than both Bob's isogeny and the usual isogenies used in SIDH. The security analysis of such a version of SIDH is challenging since prior work has shown that 'unbalanced' [33] and 'overstretched' [13, 27] versions of SIDH where the degree of one isogeny is much larger than the other may be less secure. We remark that these attacks require knowledge of the initial curve's endomorphism ring, and thus could potentially be countered by choosing the initial curve E to be a random elliptic curve with unknown $\mathrm{End}(E)$, and including this curve in the original public key. It may also be of interest to note that while several published schemes in the literature employ unbalanced SIDH parameters, as far as we know our scheme is the first cryptosystem proposal that actually uses overstretched SIDH parameters in an essential way. Note that while simply re-initializing the protocol can prevent arbitrarily unbalanced parameters, but such an approach creates complexity tradeoffs for implementations, in turn opening to door to alternative security issues.

Overcoming these issues likely means improving both KLPT* or widening the range of isogenies SIDH can compute with. However, there are limitations to how much KLPT* variants can be improved. A simple counting argument shows that paths of length e_1 on the ℓ-isogeny graph will only reach a small fraction of available elliptic curves. The cryptographer's dream is to have the output of KLPT* match the input of SIDH, but this cannot happen only with improvements to KLPT*; new versions of SIDH working with a wider class of isogenies must be designed and shown to be secure.

8 Conclusion

In this work, we have performed the first assessment of the post-quantum readiness for *key-updatable* public-key encryption schemes by determining the extent to which two isogeny-based cryptosystems can be used to instantiate Symmetric and Asymmetric UPKE constructions. We provided formalizations for both Asymmetric and Symmetric UPKE and a generalized security notion, denoted IND-CPA-U. Because neither SIDH nor CSIDH define a group action among elements in the public domain, neither supports Asymmetric UPKE designs that require update operations between public update values and encryption keys. However, both SIDH and CSIDH can be used for Symmetric UPKE constructions. The SIDH-based Symmetric UPKE construction, while possible in theory, requires mathematical improvements for a construction in practice. Our CSIDH-based construction can be instantiated today using CSIDH-512 as the parameter set. We highlighted several open problems that would improve our constructions, including the need for security analysis for unbalanced or overstretched SIDH parameters, as well as stronger CSIDH parameter sets. Such improvements will benefit any protocol that requires ongoing and asynchronous randomization of secret terms.

Acknowledgments. We thank Martin Albrecht, Alex Davidson, and Fernando Virdia for discussion of lattice-based UPKE operations. We thank Douglas Stebila for his review of our proof and suggestions on modeling an adaptive adversary that can both select update values and compromise a victim's local state. We thank Chris Leonardi for understanding limitations on the KLPT algorithm, and Richard Barnes for his help in understanding the details of the MLS protocol.

References

1. Joël Alwen, Sandro Coretti, Yevgeniy Dodis, and Yiannis Tselekounis. Security analysis and improvements for the IETF MLS standard for group messaging. *IACR Cryptology ePrint Archive*, 2019:1189,

2019.
2. Fatih Balli, Paul Rösler, and Serge Vaudenay. Determining the Core Primitive for Optimally Secure Ratcheting. *IACR Cryptol. ePrint Arch.*, 2020:148, 2020.
3. R. Barnes, B. Beurdouche, J. Millican, E. Omara, K. Cohn-Gordon, and R. Robert. The Message Layer Security (MLS) Protocol. `https://tools.ietf.org/pdf/draft-ietf-mls-protocol-09.pdf`, March 2020.
4. Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019*, pages 227–247, Cham, 2019. Springer International Publishing.
5. Karthikeyan Bhargavan, Richard Barnes, and Eric Rescorla. TreeKEM: Asynchronous Decentralized Key Management for Large Dynamic Groups A protocol proposal for Messaging Layer Security (MLS). Research report, Inria Paris, May 2018.
6. Dan Boneh, Saba Eskandarian, Sam Kim, and Maurice Shih. Improving speed and security in updatable encryption schemes. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 559–589, Cham, 2020. Springer International Publishing.
7. Jacqueline Brendel, Marc Fischlin, Felix Günther, Christian Janson, and Douglas Stebila. Towards post-quantum security for Signal's X3DH handshake. In Michael J. Jacobson Jr., Orr Dunkelman, and Colin O'Flynn, editors, *Proc. 27th Conference on Selected Areas in Cryptography (SAC) 2020*, LNCS. Springer, October 2020. To appear. Cryptology ePrint Archive, Report 2019/1356. `http://eprint.iacr.org/2019/1356`.
8. Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In *Advances in Cryptology – ASIACRYPT 2018 – 24th International Conference, Proceedings, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, 2018.
9. Jorge Chávez-Saab, Jesús-Javier Chi-Domínguez, Samuel Jaques, and Francisco Rodríguez-Henríquez. The SQALE of CSIDH: Square-root vélu Quantum-resistant isogeny Action with Low Exponents. Cryptology ePrint Archive, Report 2020/1520, 2020. `https://eprint.iacr.org/2020/1520`.
10. K. Cohn-Gordon, C. Cremers, B. Dowling, L. Garratt, and D. Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS P)*, pages 451–466, 2017.
11. Jean-Marc Couveignes. Hard homogeneous spaces. Cryptology ePrint Archive, Report 2006/291, 2006. `https://eprint.iacr.org/2006/291`.
12. Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. SQISign: Compact post-quantum signatures from quaternions and isogenies. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020*, pages 64–93, Cham, 2020. Springer International Publishing.
13. Victoria de Quehen, Pťer Kutas, Chris Leonardi, Chloe Martindale, Lorenz Panny, Christophe Petit, and Katherine E. Stange. Improved torsion point attacks on SIDH variants, 2021.
14. Luca De Feo and Steven D. Galbraith. SeaSign: Compact isogeny signatures from class group actions. In *Advances in Cryptology – EUROCRYPT 2019 – 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Proceedings, Part III*, volume 11478 of *LNCS*, pages 759–789. Springer, 2019.
15. Steven D. Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 63–91, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
16. Steven D. Galbraith, Christophe Petit, and Javier Silva. Identification protocols and signature schemes based on supersingular isogeny problems. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017, Proceedings, Part I*, volume 10624 of *Lecture Notes in Computer Science*, pages 3–33. Springer, 2017.
17. Craig Gentry. Fully Homomorphic Encryption Using Ideal Lattices. In *Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing*, STOC '09, page 169–178, New York, NY, USA, 2009. Association for Computing Machinery.

18. Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*. Springer Berlin Heidelberg, 2013.
19. Javier Herranz, D. Hofheinz, and Eike Kiltz. Kem/dem: Necessary and sufficient conditions for secure hybrid encryption. *IACR Cryptology ePrint Archive*, 2006.
20. Dennis Hofheinz, Kathrin Hövelmanns, and Eike Kiltz. A modular analysis of the fujisaki-okamoto transformation. In Yael Kalai and Leonid Reyzin, editors, *Theory of Cryptography*, pages 341–371, Cham, 2017. Springer International Publishing.
21. Joseph Jaeger and Igors Stepanovs. Optimal channel security against fine-grained state compromise: The safety of messaging. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 33–62, Cham, 2018. Springer International Publishing.
22. David Jao, Reza Azarderakhsh, Matthew Campagna, Craig Costello, Luca De Feo, Basil Hess, Aaron Hutchinson, Amir Jalali, Koray Karabina, Brian Koziel, Brian LaMacchia, Patrick Longa, Michael Naehrig, Geovandro Pereira, Joost Renes, Vladimir Soukharev, and David Urbanik. Supersingular Isogeny Key Exchange. `https://sike.org/files/SIDH-spec.pdf`, 2019. last accessed 2020-04-20.
23. David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 19–34, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
24. Yao Jiang. The Direction of Updatable Encryption does not Matter Much. *IACR Cryptol. ePrint Arch.*, 2020:622, 2020. To appear in Asiacrypt 2020.
25. Daniel Jost, Ueli Maurer, and Marta Mularczyk. Efficient ratcheting: Almost-optimal guarantees for secure messaging. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 159–188, Cham, 2019. Springer International Publishing.
26. David Kohel, Kristin Lauter, Christophe Petit, and Jean-Pierre Tignol. On the quaternion ℓ-isogeny path problem. *LMS Journal of Computation and Mathematics*, 17(A):418–432, 2014.
27. Péter Kutas, Simon-Philipp Merz, Christophe Petit, and Charlotte Weitkämper. One-way functions and malleability oracles: Hidden shift attacks on isogeny-based protocols. Cryptology ePrint Archive, Report 2021/282, 2021. `https://eprint.iacr.org/2021/282`.
28. Moxie Marlinspike and Trevor Perrin. The Double Ratchet Algorithm, 2016. `https://signal.org/docs/specifications/doubleratchet/`.
29. Moxie Marlinspike and Trevor Perrin. The X3DH Key Agreement Protocol, 2016. `https://signal.org/docs/specifications/x3dh/`.
30. R. J. McEliece. A Public-Key Cryptosystem Based On Algebraic Coding Theory. *Deep Space Network Progress Report*, 44:114–116, January 1978.
31. Victor Miller. The weil pairing, and its efficient calculation. *J. Cryptology*, 17:235–261, 09 2004.
32. Chris Peikert. He gives C-sieves on the CSIDH. Cryptology ePrint Archive, Report 2019/725, 2019. `https://eprint.iacr.org/2019/725`.
33. Christophe Petit. Faster algorithms for isogeny problems using torsion point images. In *Advances in Cryptology - ASIACRYPT 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 330–353. Springer, 2017.
34. Bertram Poettering and Paul Rösler. Towards bidirectional ratcheted key exchange. In *Annual International Cryptology Conference*, pages 3–32. Springer, 2018.
35. Bertram Poettering and Paul Rösler. Asynchronous ratcheted key exchange. Cryptology ePrint Archive, Report 2018/296, 2018. `https://eprint.iacr.org/2018/296`.
36. Alexander Rostovtsev and Anton Stolbunov. Public-key cryptosystem based on isogenies. Cryptology ePrint Archive, Report 2006/145, 2006. `https://eprint.iacr.org/2006/145`.
37. Joseph Silverman. *The Arithmetic of Elliptic Curves*, volume 106. Springer New York, 01 2009.
38. Jacques Vélu. Isogénies entre courbes elliptiques. *C. R. Acad. Sci. Paris Sér. A-B*, 273:A238–A241, 1971.

A Algebraic Structures for SIDH and CSIDH

In Section 2.1, we presented background on both SIDH and CSIDH. We now provide additional analysis of the algebraic structure for both.

SIDH. Unlike groups such as $\mathbb{Z}_p$ or elliptic curves over some finite field, the endomorphism ring for supersingular curves over $\mathbb{F}_{p^2}$ is neither commutative nor defines a group action at all. To perform SIDH, participants require agreeing upon additional "auxiliary points" which ensure both parties arrive at the same shared secret after completing the protocol. This structure defined by SIDH can be thought of as a *semigroup*; although note that since any function can be considered a semigroup, this classification is not particularly informative.

CSIDH. Couveignes [11] first defined the concept of a *hard homogeneous space* as a finite commutative group G acting over some set X, a generalization of the structure for a group suitable to performing Diffie-Hellman-like operations. Similarly to SIDH, CSIDH defines a finite commutative group consisting of the ideal-class group $cl(\mathcal{O})$ acting over the set of supersingular curves through the application of isogenies. However, importantly, the graph of supersingular curves in CSIDH is *restricted* to the field $\mathbb{F}_p$, thereby ensuring that $cl(\mathcal{O})$ can act freely and transitively over the group. As $cl(\mathcal{O})$ is commutative, CSIDH can support commutativity of operations over this set of supersingular curves via the group action.

As discussed in Section 2.1, group elements are represented by a vector $\vec{e} \in \mathbb{Z}^d$, representing the ideal $\prod_{i=1}^{d} \ell_i^{e_i}$. This representation is used because in general applying the group action is computationally difficult. To get around this, CSIDH uses a canonical set of generators $\mathfrak{g}_1, \ldots, \mathfrak{g}_d$ and works be only applying small powers of these generators to the starting curve E. Two major limitations of this representation are that it does not allow for uniform sampling over the class group, and the representation of group elements is not necessarily unique.

Beullens et al. [4] solved this problem, at least for the CSIDH-512 parameter set by explicitly computing the structure of the class group. For this parameter set, they found an N such that $cl(\mathcal{O}) \cong \mathbb{Z}_N$ and described a method to swap between the representation of group elements as a member of $\mathbb{Z}_N$ and of $\mathbb{Z}^d$. This allows for both uniform sampling and unique representation.

B A Naive Online-Only UPKE Scheme.

We briefly describe a scheme that achieves the notions required for UPKE described in Section 3, except for that of asynchronicity. We use this scheme to measure the effectiveness of our constructions.

We refer to our naive scheme as *Double Encrypt*. In summary, the scheme uses a long-lived keypair as well as an ephemeral keypair, to ensure forward secrecy. *Double Encrypt* is defined as follows:

- $KeyGen(\lambda) \rightarrow (sk_I, pk_I)$: Accepts a security parameter λ and output a public encryption key pk_I and secret decryption key sk_I.
- $Encrypt(pk_I, pk_U m) \rightarrow c$: Encrypts a message m using first the public key pk_I and then a second time using an ephemeral update public key pk_U, resulting in a ciphertext c.
- $Decrypt(sk_I, sk_U, c) \rightarrow m$: Decrypts the ciphertext c using sk_U to decrypt the outer layer and sk_I to decrypt the inner layer, producing as output the plaintext message m.

– $GenUpdate(\lambda) \rightarrow (pk_U, sk_U)$: Accepts a security parameter λ and outputs a set of encryption and decryption keys (pk_U, sk_U). This is performed by the keyholder themselves.

– $Update(\lambda) \rightarrow sk'$: Performed by the keyholder; takes as input a security parameter λ, and performs $KeyGen(\lambda) \rightarrow (sk'_I, pk'_I)$. The keyholder deletes (sk_I, pk_I) and sets (sk'_I, pk'_I) as their new encryption and decryption keypair.

In *Double Encrypt*, the keyholder directly updates their encryption and decryption keys, which requires the keyholder to be online in order to update pk_I to pk'_I.

C Naive SIDH Construction

We now present a first naive SIDH UPKE construction and discuss its limitations with respect to forward secrecy. These limitations led to our use of KLPT* and the "online" SIDH-based UPKE construction that we present in Section 5.

SIDH-based UPKE, first attempt. A first step towards a Symmetric UPKE scheme using SIDH has a similar form to plain public-key encryption via SIDH defined by Jao and De Feo [23]. However, here, the update step is performed using a SIDH key exchange operation, where the resulting curve from the exchange becomes the updated public key as opposed to the negotiated shared secret.

Alice performs key generation in exactly the same way as in plain SIDH, by first sampling a scalar $n_A \xleftarrow{\$} \mathbb{Z}_{2^{e_1}}$ which then defines her secret encryption key sk_A is the isogeny $\phi_A \colon E_0 \rightarrow E_A$, such that $E_A = E_0/\langle [1]P_A + [n_A]Q_A \rangle$. Alice then generates her public encryption key pk_A is the curve with auxiliary points $(E_A, \phi_A(P_B), \phi_A(Q_B))$. exactly the same way as in plain SIDH

Let's say now that Bob wishes to generate an update for Alice's keypair. To do so, he simply performs an SIDH *KeyGen* in exactly the same manner as Alice. He samples a secret update value $n_\mu \xleftarrow{\$} \mathbb{Z}_{3^{e_2}}$ which he uses to define a secret isogeny $\mu \colon E_0 \rightarrow E_\mu$. Bob then applies μ to derive Alice's new public key $E_{A\mu}$ by performing a plain SIDH *KeyGen*, and then transmits μ to Alice (using an encrypted channel). Alice applies μ to her secret term similarly by performing a plain SIDH key exchange operation.

Differently to plain SIDH key exchange, Alice's updated public key now becomes the resulting curve $E_{A\mu}$, along with updated auxiliary points.

$$(E_{A\mu}, \phi_{A\mu}(P_B), \phi_{A\mu}(Q_B))$$

Alice's updated secret key sk_A correspondingly becomes the updated isogeny $\phi_{A\mu} \colon E_0 \rightarrow E_{A\mu}$, such that

$$\phi_{A\mu} = \langle \phi_\mu(P_A) + [n_A]\phi_\mu(Q_A) \rangle.$$

Note that transmitting μ instead of just the auxiliary points prevents active attacks as described previously in the literature [15].

Problem: The naive scheme does *not* preserve forward secrecy or post-compromise security. While the naive scheme does enable performing updates to SIDH public and private keys via a secret update value, this approach does *not* fulfill the notions of forward secrecy or post-compromise security as required for UPKE schemes. We now describe how the scheme fails to be IND-CPA-Usecure.

Recall that in the IND-CPA-Ugame described in Section 3.1, the adversary is allowed to perform the GiveUpdate query with update values of the adversary's own choosing, and can learn any keypair by performing Corrupt on some index j denoting the secret key of interest. The adversary wins the game if they have a non-negligible chance at guessing the contents of some ciphertext that has been encrypted after some series of τ update, for which the adversary cannot derive the corresponding sk_τ secret key.

In this naive construction, after applying the series of τ update values, the challenger's secret key sk_A will have the following form:

$$\phi_{(A1,\ldots,\tau)} = \langle \phi_1(\ldots(\phi_\tau(P_A))) + [n_A]\phi_1(\ldots(\phi_\tau(Q_A)))\rangle$$

Even after applying τ updates, the n_A term remains static, and consequently the torsion points across updates remain linearly dependent, meaning that the naive scheme *cannot* provide forward secrecy or post-compromise security. An adversary can derive sk_A simply by compromising a prior "window" and then applying the τ updates to derive $\phi_{(A1,\ldots,\tau)}$, by solving a system of linear equations. Even if $\phi_{(A1,\ldots,\tau+1)}$ were instead represented as a group element $P_{\tau+1}$ that generates the kernel of $\phi_{(A1,\ldots,\tau+1)}$ (to avoid persisting the static n_A value), the adversary could still learn $\phi_{(A1,\ldots,\tau+1)}$ simply by solving the discrete logarithm problem for the generator point $Q_{\tau+1} = [n_A]\phi_1(\ldots(\phi_\tau + 1(Q_A)))$. Solving for the discrete logarithm in this setting is easy even for a classical computer, as the order of the group is smooth and hence can be performed in polynomial time.

Proposed Fix: Use KLPT* to output a fresh secret. As described in Section 2.1, KLPT* finds an isogeny path between two supersingular curves with known endomorphism rings. As such, KLPT* can be used to efficiently find the "composition" of an update value and the original secret, and ensures that knowledge of the output composition will not leak information about inputs.

D Proof of CSIDH-Based UPKE

In Section 6, we present a CSIDH-based UPKE construction. We present the proof of its IND-CPA-Usecurity here.

Proof. As we are showing a reduction to a plain IND-CPA game, we will start by being given a public key pk^*. To begin, select a uniformly random index $i \xleftarrow{\$} \{0,\ldots,q_{gen}\}$. The idea of the proof is to set the public key after the ith FreshUpdate query to be pk^*, and hope that the adversary requests the IND-CPA-Uchallenge to be issued on a public key that occurs before the next FreshUpdate. If we are correct, then the adversary's ability to distinguish which message was encrypted under pk^* (or a related key) will allow us to win the IND-CPA game.

At the start of the game, if $i = 0$ then we set $pk_0 \to pk^*$. Otherwise, we sample a new uniform pk_0 from *KeyGen*. From here we proceed as normal. If the adversary makes a corruption query, then we provide them with the corresponding private key. When a GiveUpdate(μ) query is made, we update the secret and public key and make note of the μ value.

When the ith query to FreshUpdate is made, we set the resulting public key to pk^*. We carry on, and when the next FreshUpdate query is made we sample a fresh public key from $KeyGen$. If the adversary ever makes a Corrupt query on any of the keys between these FreshUpdate queries, then we abort. We will consider the probability of having to abort occurring momentarily.

Eventually, the adversary requests the IND-CPA-Uchallenge on a public key with index j. We hope that this index means a key that falls between the ith FreshUpdate and the $i+1$th call to FreshUpdate. When this happens, the adversary submits m_0, m_1 as part of the challenge.

We then forward m_0, m_1 to receive back an encryption of m_b, consisting of $C = g \star E_0$ for a random g, as well as $DEM.Encrypt(K, m_b)$. Let $\mu_1, \mu_2, ..., \mu_k$ be k queries to GiveUpdate after the ith FreshUpdate query. We provide the adversary with $(-\mu_1 - \mu_2 - \cdots - \mu_k) \star C$ and $DEM.Encrypt(K, m_b)$.

Note that $K = KDF(g \star pk^*) = KDF((-\mu_1 - \cdots - \mu_k) \star g \star (\mu_1 + \cdots + \mu_k) \star pk^*)$, which means that the message is encrypted under the correct key. So, when the adversary submits a guess for b, we can guess the same value, and if the adversary is correct, so are we.

When we set the public key to pk^* after the ith call to FreshUpdate, the adversary cannot notice that we have not genuinely updated the public key, unless they issue a Corrupt query. If such a Corrupt query is issued, we must abort. However, note that if our guess is correct, and the IND-CPA query is requested in this segment of public keys, then no Corrupt query will be issued, or else the adversary's advantage is 0.

Because updates are sampled uniformly over $\mathbb{Z}_p$, the resulting public key is uniformly random over the public key space (this follows from the fact that the group action is regular). So after a FreshUpdate has occurred, the adversary has no information on the distribution of the secret key, and we can thus replace the public key with the challenge public key pk^*. The adversary has no advantage in distinguishing that we have done this. As a result, we have a $1/(1 + q_{gen})$ chance of correctly guessing where the challenge will be requested. If we are correct, the adversary does not change their behavior at all, as they have no advantage in distinguishing that we are not managing the game honestly. This means the chance that we abort is exactly $q_{gen}/(1 + q_{gen})$.

Our advantage in winning the IND-CPA game is thus the adversary's advantage in winning the IND-CPA-Ugame times the probability we do not abort, which is $\epsilon/(1 + q_{gen})$, as desired.

We note that the techniques in this proof can also be applied to the classical construction of Alwen et al. [1]. While they couple together the public key update and encryption functions, the same general strategy can be used to show that the stronger IND-CPA-Unotion can be satisfied by their construction.