SJÄLVSTÄNDIGA ARBETEN I MATEMATIK

MATEMATISKA INSTITUTIONEN, STOCKHOLMS UNIVERSITET

Post-Quantum Cryptography: Supersingular Isogeny Diffie-Hellman Key Exchange

av

Erik Thormarker

2017 - No 42

Post-Quantum Cryptography: Supersingular Isogeny Diffie-Hellman Key Exchange

Erik Thormarker

Självständigt arbete i matematik 30 högskolepoäng, avancerad nivå

Handledare: Jonas Bergström, John Mattsson

2017

Abstract

If large-scale quantum computers can be built, then we need to replace our most commonly used public key cryptosystems with post-quantum public key cryptosystems. These are the public key cryptosystems that are believed to remain secure even if large-scale quantum computers can be built. One relatively recent proposal for such a cryptosystem is the Supersingular isogeny Diffie-Hellman (SIDH) key exchange by Jao and De Feo. The purpose of this thesis is to explain the theoretical background of SIDH and to evaluate the current status of the cryptosystem. Along the way we will discuss how the isogenies in SIDH give rise to non-backtracking walks in supersingular isogeny graphs and we perform simulations in order to study the behaviour of these walks. To our knowledge such simulations have not been documented elsewhere in the literature about SIDH. We also study a simple reduction between two computational problems related to SIDH on supersingular isogeny graphs. A very similar reduction was recently mentioned in independent work by Galbraith and Vercauteren. While studying the implementation status of SIDH we describe a KEM built on SIDH that is IND-CCA2 secure in the random oracle model. To our knowledge, in doing so we answer an open question posed by Kirkwood et al. in the context of SIDH. Finally we give theoretical estimates of computational work, memory usage and key sizes in different variants of SIDH as functions of the quantum security level. This is straightforward thanks to analyses by earlier authors in work on SIDH.

Acknowledgements

I want to thank my supervisors Jonas Bergström, Håkan Englund, Christine Jost and John Mattsson for all of their patience, help and ideas.

Contents

1 Introduction

Many applications on today's Internet rely on the existence of fast public key cryptography that can be done with low bandwidth. In RSA and Elliptic curve Diffie-Hellman (ECDH) we have two well understood (and at least in the case of ECDH, very efficient) public key cryptosystems. Unfortunately there is a threat on the horizon. Both cryptosystems are built on problems that can be efficiently solved if sufficiently large quantum computers are built. Whether large-scale quantum computers will ever be built is an open research question. However, selecting and evaluating new cryptosystems is a necessarily slow process, therefore preparing for the worst and evaluating our alternatives for doing public key cryptography in a post-quantum world is necessary. There is also the threat of encrypted data being collected and archived with the intent to break its encryption if/when large-scale quantum computers are successfully built.

In this thesis we look at the Supersingular isogeny Diffie-Hellman (SIDH) key exchange. Other suggested post-quantum public key cryptosystem include code-based and lattice-based cryptosystems. The main advantage of SIDH is that is has relatively small key sizes. Unfortunately, the performance of the cryptosystem has been relatively poor in the past. The main purpose of this thesis is to explain the theoretical foundation of SIDH and to evaluate the status of the cryptosystem.

We start in Section 2 with defining elliptic curves and a group operation on them. In Section 3 we explain how we can implement a Diffie-Hellman key exchange with this group operation and we talk about the computational problem, the Elliptic curve discrete logarithm problem (ECDLP), on which it is based. Section 4 gives a very brief introduction to quantum computers and some of the most important quantum algorithms. It turns out that the ECDLP is no longer infeasible to solve if we have a quantum computer. This motivates our interest in SIDH and other post-quantum public key cryptosystems. In Section 5 we define isogenies between elliptic curves. It turns out that, although both take place on elliptic curves, ECDH and SIDH have very little in common. In ECDH we used secret scalars, in SIDH we use secret isogenies instead. In Section 6 we finally present SIDH. We discuss its underlying computational problems in Section 7 and explain how the isogenies in SIDH give rise to non-backtracking walks in supersingular isogeny graphs. These walks are too short to reach every vertex of such a graph, but through simulations for small examples we note that the distributions of the variables $j(E_A)$, $j(E_B)$ and $j(E_{AB})$ (which are vertices of the graphs) in SIDH appear to be essentially the best possible. Inspired by our simulation results, we give a simple reduction between two computational problems related to SIDH. In Section 8 we describe the best known attacks against SIDH. The best attack turns out to be a "generic" algorithm for the claw problem. It also turns out that there is a powerful attack that Bob can do when Alice is using a static key in SIDH to recover Alice's secret key. In Section 9 we review the progress made in improving the performance of SIDH. We also describe a proposed counter-measure that Alice can perform when using a static key to thwart Bob's attack mentioned above. In addition to this counter-measure, we propose a slightly different one for which we can prove IND-CCA2 security. We also use analyses from earlier papers about SIDH to give total estimates of computational work, memory usage and key size as functions of the quantum security in several variants of SIDH. At last, in Section 10, we present our conclusions about SIDH as a viable post-quantum alternative.

The reader is assumed to be familiar with:

- Basic abstract algebra: the order of a group element, finite fields, Lagrange's theo-

rem, group homomorphisms, etc.

- Basic computational complexity theory: Ω-notation, O-notation, Θ-notation, polynomial time algorithms, etc.

- Basic cryptography: asymmetric encryption, symmetric encryption, cryptographic hash functions, negligible functions, security strength in bits, etc.

All logarithms in this thesis have base 2 unless stated otherwise.

1.1 The contributions of this thesis

- We give a construction of a KEM based on SIDH that is similar to the construction suggested in [KLM+15, p. 14]. As noted in [GPST16, Section 2.5], [KLM+15] do not give a formal security proof for their construction. We prove that our KEM is secure in the sense of IND-CCA2 in the random oracle model under the SSDDH assumption. The message overhead for our KEM is the same as the overhead for the suggestion in [KLM+15]. To this author's knowledge this answers an open question in [KLM+15, p. 18] in the context of SIDH. Our KEM is based on the Fujisaki-Okamoto transform in [FO00]. See Section 9.8.4 for details.

- Through simulations for small examples we observe that the number of distinct j-invariants $j(E_A)$ that Alice's isogeny $\phi_A : E_0 \to E_A$ can take her to is on average very close to the number of distinct j-invariants we get when we choose uniformly at random (with repetition) from the set of all supersingular j-invariants defined over $\mathbb{F}_{p^2}$. To this author's knowledge, simulations similar to these for SIDH have not been documented elsewhere in the literature. See Section 7.3 for details.

- Through simulations for small examples we observe that the distribution of $j(E_{AB})$ in SIDH appears to be essentially the best possible. To this author's knowledge, simulations similar to these for SIDH have not been documented elsewhere in the literature. See Section 7.4 for details.

- As other authors have noted before us, we make the simple observation that the isogenies in SIDH give non-backtracking walks in supersingular isogeny graphs. This observation and our simulation results for $j(E_A)$ inspire us to give a simple heuristic reduction from the CSSI problem to the DSSI problem. A very similar reduction was recently mentioned in independent work in [GV17, Section 6.2]. We work with a slightly different formulation of the DSSI problem than that in [GV17] and also give simulation results that prove convincingly that the reduction works. See Section 7.6 for details.

- We give estimations of computational work, memory usage and public key size in state-of-the-art implementations of SIDH as functions of the quantum security in bits λ. These estimations are straightforward to do thanks to earlier analyses in [CLN16], [CJL+16b] and [DFJP14]. See Section 9.10 for details.

1.2 List of notation

Notation	Meaning
a	Finite field addition (see Section 9.10)
codomain(f)	The codomain of the function $f : A \to B$, i.e., B
char(K)	The characteristic of the field K
CSSI	Computational supersingular isogeny problem (see Problem 7.5.1)
deg(ϕ)	The degree of the isogeny ϕ (see Definition 5.1.2)
domain(f)	The domain of the function $f : A \to B$, i.e., A
DSSI	Decisional supersingular isogeny problem (see Problem 7.5.2)
$E, E(\overline{K})$	Will be an elliptic curve defined over $\overline{K}$ in general (see Section 2.1)
$[E]$	The equivalence class of E in $V(\mathcal{G}_\ell)$ (see Definition 7.1.1)
$E(L)$	The L-rational points on the elliptic curve E (see Section 2.1)
$E[m], E(\overline{K})[m]$	The m-torsion points on the elliptic curve E (see Section 2.2)
$E(L)[m]$	The L-rational m-torsion points on the elliptic curve E (see Section 2.2)
End(E)	The endomorphism ring of the elliptic curve E (see Definition 5.2.4)
(EC)DH	(Elliptic curve) Diffie-Hellman key exchange (see Section 3)
(EC)DHP	(Elliptic curve) Diffie-Hellman problem (see Section 3)
(EC)DLP	(Elliptic curve) discrete logarithm problem (see Section 3)
$E(\mathcal{G}_\ell)$	The edge set of $\mathcal{G}_\ell$ (see Definition 7.1.1)
$e_m(\cdot, \cdot)$	The Weil-pairing on $E[m]$ (see Section 2.2)
e_A, e_B	exponents in the public parameter p of SIDH (see Section 6.2)
$\mathcal{G}_\ell$	The supersingular isogeny graph with respect to ℓ (see Definition 7.1.1)
I	Finite field inversion (see Section 9.10)
id	The identity isogeny (see Section 5.1)
$j(E)$	The j-invariant of the elliptic curve E (see Definition 5.1.11)
K, L	Will be fields in general
K^*	The multiplicative group of the field K, i.e., the non-zero elements
K^n	Affine n-dimensional space with respect to a field K (see Section 2.1)
$K[x]$	The ring of polynomials with coefficients in K
KEM	Key encapsulation mechanism (see Appendix A)
ℓ_A, ℓ_B	Prime factors in the public parameter p of SIDH (see Section 6.2)
$[m]$	The multiplication by m-map (see the proof of Theorem 2.2.2)
M	Finite field multiplication (see Section 9.10)
non-backtracking	See Section 7.2
O	The point at infinity (see Section 2.1)
$\langle P \rangle$	The subgroup generated by a group element P
$\langle P, Q \rangle$	All linear combinations of the group elements P and Q (see Section 2.2)
$o(P)$	The order of a group element P (see Section 2.2)
$\mathbb{P}^2(K)$	Projective 2-space with respect to a field K (see Section 2.1)
S	Finite field squaring (see Section 9.10)
separable	See Definition 5.1.3

Notation	Meaning
SIDH	Supersingular isogeny Diffie-Hellman key exchange (see Section 6)
SSDDH	Supersingular isogeny decisional Diffie-Hellman problem (see Problem 9.8.1)
Trace(A)	The sum of the elements on the main diagonal of the $n \times n$-matrix A
$V(\mathcal{G}_\ell)$	The vertex set of $\mathcal{G}_\ell$ (see Definition 7.1.1)
$x(P)$	The x-coordinate of the elliptic curve point P
$\|x\|$	The length of a bitstring x
$(x)_i$	The ith leftmost bit of a bitstring x
$x\|\|y$	The concatenation of the bitstrings x and y
$y(P)$	The y-coordinate of the elliptic curve point P
θ	Will be an isomorphism in general (see Section 5)
λ	Will be the quantum security in bits of SIDH in general
$\Pi = (\mathcal{E}, \mathcal{D})$	A public key encryption scheme (see Appendix A)
π_q	The Frobenius endomorphism (see Section 5)
ϕ	Will be an isogeny in general (see Section 5)
$\widehat{\phi}$	The dual of the isogeny ϕ (see Theorem 5.1.15)
$\perp$	Decryption failure for a public key encryption scheme
$\#A$	The cardinality of the set A

2 Elliptic curves

We mainly follow [Was08] in this section. We define elliptic curves and a group operation on them in Section 2.1. In Section 2.2 we talk about torsion subgroups with respect to the group operation, a concept that will be central to our applications in Section 6. In what follows K will be a field and $\overline{K}$ its algebraic closure. Unless otherwise stated, L will be a field such that $\overline{K} \supseteq L \supseteq K$. For our applications later in this text K will always be a finite field $\mathbb{F}_q$ with q a prime power. Therefore our focus in this overview of elliptic curves will be on curves defined over fields with positive characteristic p. We will also assume that $p \notin \{2, 3\}$. It turns out that we can limit ourselves to particularly simple equations defining our elliptic curves when we make this assumption [Sil09, Remark 1.3] and it will not be a restriction in our applications.

We will refer the reader to other texts for proofs of most of the theorems and propositions that we state. But we will often say something about a proof or for instance prove one direction of an if and only if-statement.

2.1 A group operation on elliptic curves

We start with a **Weierstrass equation**. This is an equation in variables x and y of the form

$$y^2 = x^3 + Ax + B \tag{1}$$

with $A, B \in \overline{K}$.

Definition 2.1.1. An **elliptic curve** $E(\overline{K})$ is a set

$$\{(x, y) \in \overline{K} \times \overline{K} : y^2 = x^3 + Ax + B\} \cup \{O\}$$

for fixed $A, B \in \overline{K}$ such that $4A^3 + 27B^2 \neq 0$. The element O is called **the point at infinity**. When $A, B \in L$ we say that $E(\overline{K})$ is **defined over** L. As is standard, we will call the elements of $E(\overline{K})$ **points**. There is another way to define elliptic curves that requires more algebraic geometry than we have space to develop here, the standard textbook is [Sil09]. Definition 2.1.1 is sufficient for our purposes.

We let E be short-hand for $E(\overline{K})$. In general we need to work over the algebraic closure $\overline{K}$ when developing the theory of elliptic curves. For example, the functions (isogenies) between elliptic curves that we will study will in general be defined over $\overline{K}$ and this is where they will be surjective. However in our applications we will want to work with the subset of points on $E(\overline{K})$ that lie in L. For a given elliptic curve E defined over K we therefore let the set $E(L)$ be the points

$$\{(x, y) \in L \times L : y^2 = x^3 + Ax + B\} \cup \{O\}$$

The set $E(L)$ is often referred to as the **L-rational points** on E. The space

$$L^n = \underbrace{L \times \cdots \times L}_{n \text{ times}}$$

is called **affine** n-dimensional space (with respect to L).

To explain the point at infinity O we need the following definition.

Definition 2.1.2. Projective 2-space over L (denoted by $\mathbb{P}^2(L)$) are the equivalence classes of the set

$$\{(x_0, x_1, x_2) \in L^3 : x_i \neq 0 \text{ for some i}\}$$

under the equivalence relation

$$(x_0, x_1, x_2) \sim (y_0, y_1, y_2)$$

if there exists a λ in L^* such that $x_i = \lambda y_i$ for $0 \leq i \leq 2$. As usual, L^* denotes the multiplicative group of L, i.e., the non-zero elements of L. We write the equivalence class of (x_0, x_1, x_2) as $(x_0 : x_1 : x_2)$.

A **homogeneous Weierstrass equation** is an equation of the form

$$y^2z = x^3 + Axz^2 + Bz^3 \tag{2}$$

with $A, B \in \overline{K}$. The equation is homogeneous since $(x, y, z) \in \overline{K}^3$ is a solution if and only if $(\lambda x, \lambda y, \lambda z)$ is a solution for every $\lambda \in \overline{K}^*$. This means it makes sense to ask which equivalence classes $(x : y : z) \in \mathbb{P}^2(\overline{K})$ are solutions to the equation. By inspection we see that $(0 : 1 : 0)$ is always a solution of a homogeneous Weierstrass equation. Keep in mind that some component must be non-zero so $(0 : 0 : 0)$ is not a solution. One can verify that $(0 : 1 : 0)$ is the only solution with $z = 0$. Also, since

$$(x : y : z) = (x/z : y/z : z/z) = (x/z : y/z : 1) \tag{3}$$

when $z \neq 0$, we may assume that $z = 1$ when $z \neq 0$. But when $z = 1$, equation (2) becomes

$$y^2 = x^3 + Ax + B$$

so the solutions $(x : y : 1)$ correspond exactly to the solutions (x, y) of the non-homogeneous Weierstrass equation (1) with the same constants A and B. The extra solution $(0 : 1 : 0)$ that we get in projective space is the point at infinity O. Washington [Was08, p.18] suggests that we think of the point at infinity as dividing by $z = 0$ in (3) and obtaining ∞ in the non-zero y-coordinate.

Remarkably, we are able to define a binary operation $+$ on the points in $E(L)$, under which $E(L)$ becomes an abelian group. The point at infinity O acts as the identity element under $+$. For any $O \neq P \in E$ we refer to the x and y-coordinate of P as $x(P)$ and $y(P)$, respectively. Let $P, Q \in E(L)$. The rules for computing $P + Q$ are as follows, where the first rule that applies is used:

i) If $P = O$, then $P + Q = Q$.

ii) If $Q = O$, then $P + Q = P$.

iii) If $x(P) \neq x(Q)$, then

$$x(P + Q) = m^2 - x(P) - x(Q), \quad y(P + Q) = m(x(P) - x(P + Q)) - y(P)$$

where $m = \frac{y(Q)-y(P)}{x(Q)-x(P)}$.

iv) If $x(P) = x(Q)$ and $y(P) \neq y(Q)$, then $P + Q = O$.

v) If $P = Q$ and $y(P) \neq 0$, then

$$x(P+Q) = m^2 - 2x(P), \quad y(P+Q) = m(x(P) - x(P+Q)) - y(P)$$

where $m = \frac{3x(P)^2+A}{2y(P)}$.

vi) If $P = Q$ and $y(P) = 0$, then $P + Q = O$.

The rules are derived from the idea that three points on a line in L^2 should sum, with regards to $+$, to O. We refer to [Was08, p.12] for details. Some extra care is needed for special cases such as $P + P$. To motivate the rule for this case one looks at the tangent of the curve defining $E(L)$ at the point P. Note that Rule (iii) for adding P and $Q \neq P$ differ from Rule (v) for adding P to itself (**doubling**).

Theorem 2.1.3. *$E(L)$ is an abelian group under $+$. The element O is the identity element.*

Proof. See [Was08, Theorem 2.1]. The part of the proof that requires the most work is proving that $+$ is associative. It is intuitively clear that $+$ is abelian since the line through P and Q is the same as the line through Q and P. Note that if $P = (x, y)$ is in $E(L)$, then $(x, -y)$ is also in $E(L)$. By Rule (iv), the point $(x, -y)$ is the inverse of P. The inverse is indeed unique since the equation $y^2 = a$, for $a = x^3 + Ax + B$, has at most two solutions in $\overline{K}$. □

2.2 Torsion points and the Weil pairing

For a positive integer m we write

$$\underbrace{P + \cdots + P}_{m \text{ times}}$$

as mP. We also let $0P = O$ and if m is a negative integer we let $mP = (-m)(-P)$

Definition 2.2.1. Let m be a positive integer. The set

$$\{P \in E(L) : mP = O\} \subseteq E(L)$$

is denoted by $E(L)[m]$. We let $E[m]$ be short-hand for $E(\overline{K})[m]$.

It is easy to verify that $E(L)[m]$ is a subgroup since $E(L)$ is abelian. The points in $E(L)[m]$ are referred to as the **m-torsion** points of $E(L)$. As usual, when we talk about the **order of a point** P (denoted by $o(P)$) we mean the smallest positive integer m such that $mP = O$.

The following theorem will be very important in our applications in Section 6.

Theorem 2.2.2. *Let E be an elliptic curve defined over $\overline{K}$ and let m be a positive integer. If $char(K) = 0$ or $char(K)$ does not divide m, then*

$$E[m] \cong \mathbb{Z}_m \oplus \mathbb{Z}_m$$

Proof. See [Was08, Theorem 3.2]. By studying the rules for adding points [Was08, Section 3.2], it is possible to show that the multiplication by m-map $[m] : E \to E$ defined by

$$[m](P) = mP, \quad \text{for all } P \in E$$

is group homomorphism with respect to $+$. It is also possible to show that $\#\ker[m] = m^2$. By the fundamental structure theorem for finite abelian groups [BB06, Theorem 7.5.7]

$$\ker[m] \cong \mathbb{Z}_{n_1} \oplus ... \oplus \mathbb{Z}_{n_k} \tag{4}$$

with $n_i \mid n_{i+1}$ for all $1 \leq i < k$. Just like for m, one can show that $\#\ker[\ell] = \ell^2$ for any prime $\ell \mid n_1$. So we must have $k \leq 2$ in (4) since $\ell \mid n_1$ and $n_i \mid n_{i+1}$ for all $1 \leq i < k$. Since also any point P in $\ker[m]$ has $o(P) \mid m$, we must have $n_k \leq m$ and hence

$$\ker[m] \cong \mathbb{Z}_m \oplus \mathbb{Z}_m$$

Finally, note that $E[m] = \ker[m]$. □

Let $E[m]$ be as in Theorem 2.2.2, it then follows from the theorem that there are points $P, Q \in E[m]$ such that linear combinations of P and Q generate all of $E[m]$, i.e.,

$$\{aP + bQ : a, b \in \mathbb{Z}_m\} = E[m]$$

In general for $P, Q \in E$, we denote the set of all linear combinations of P and Q by $\langle P, Q \rangle$.

Theorem 2.2.3. *Let E be an elliptic curve defined over $\overline{K}$. Let m be a positive integer such that char($\overline{K}$) does not divide m (if char($\overline{K}$) is non-zero) and let μ_m be the mth roots of unity in $\overline{K}$. Then there is a pairing*

$$e_m : E[m] \times E[m] \to \mu_m$$

called the ***Weil pairing*** *such that for all $P, Q, S \in E[m]$*

i) e_m is bilinear, that is

$$e_m(P + Q, S) = e_m(P, S)e_m(Q, S)$$

$$e_m(P, Q + S) = e_m(P, Q)e_m(P, S)$$

ii) $e_m(P, P) = 1$

iii) $e_m(P, Q) = e_m(Q, P)^{-1}$

iv) e_m is non-degenerate. That is, if

$$(P, T) = 1, \quad \textit{for all } T \in E[m],$$

then $P = O$. Similarly, if

$$(P, T) = 1 \textit{ for all } P \in E(\overline{K})[m],$$

then $T = O$.

Proof. See [Was08, Section 11.2]. □

3 Elliptic curve Diffie-Hellman key-exchange

We mainly follow [Was08] and [BSS99] in this section. We let E be an elliptic curve defined over a finite field $\mathbb{F}_q$ and P be a fixed point in $E(\mathbb{F}_q)$ whose order N is close in bit size to that of $\#E(\mathbb{F}_q)$. In Section 3.1 we define the Elliptic curve Diffie-Hellman (ECDH) key exchange and the related Discrete logarithm problem (DLP). In Section 3.2.1 we give an algorithm that will be used to solve any DLP instances that arise in our applications. In Section 3.2.2 we describe an exponential time random walk algorithm for solving the DLP on elliptic curves. For well chosen elliptic curves, exponential time attacks on ECDH such as this are the best known [BSS99, I.3]. Another exponential time algorithm (with large space requirements) that we do not discuss here is Shank's baby-step giant-step algorithm (see for example [Was08, Section 5.2.1]). In Section 3.2.3 we describe the MOV algorithm that is especially well suited for solving the DLP on the particular class of curves called Supersingular elliptic curves that we will use in our applications in Section 6. This has given the supersingular elliptic curves somewhat of bad reputation in cryptographic applications with elliptic curves. However, as we will see, this is no longer relevant in our applications.

3.1 Key exchange

The Elliptic curve Diffie-Hellman (ECDH) key exchange is as follows:

1. Alice picks a secret key $sk_A = a \in_R \mathbb{Z}_N$ and sends her public key $pk_A = aP$ to Bob. Bob similarly picks a secret key $sk_B = b \in_R \mathbb{Z}_N$ and sends his public key $pk_B = bP$ to Alice.

2. Alice computes
$$a(bP) = (ab)P$$
Similarly, Bob computes
$$b(aP) = (ba)P = (ab)P$$
They can now use $(ab)P$ as a shared key SK.

The problem facing an eavesdropping adversary is the following.

Problem 3.1.1 (Elliptic curve Diffie-Hellman problem (**ECDHP**))**.** Given P, aP and bP, find $(ab)P$.

Our interest in ECDH comes from our belief that ECDHP is hard (on a classical computer) combined with the fact that there are efficient ways to compute the multiplication mP for an integer m. One of the simplest effective ways to compute such a multiplication is the Double-and-add algorithm [Sil09, XI.1.1] that is similar to the Square-and-multiply algorithm used for exponentiation modulo an integer. It is an open problem whether ECDHP is as hard as the following problem.

Problem 3.1.2 (Elliptic curve discrete logarithm problem (**ECDLP**))**.** Given P and a point $Q \in \langle P \rangle$, find an integer m such that $mP = Q$.

If we can solve ECDLP, then we can solve ECDHP. So by our discussion above we believe ECDLP is hard in the classical setting as well. There are some special classes of elliptic curves for which ECDLP is easier than in the general case, one such class is

that of supersingular elliptic curves (see Section 3.2.3). For these curves, by reduction to a discrete logarithm problem in $\mathbb{F}_{q^2}^*$, subexponential algorithms such as index calculus [Was08, Section 5.1] are available. For an elliptic curve that is chosen with known attacks in mind, only exponential (in $\log N$) time algorithms are known. Since no sub-exponential algorithms are known for ECDLP, we have significantly smaller key sizes than for say RSA cryptosystems of the same security level, where the sub-exponential General number field sieve applies [BSS99, Section I.3]. In the next section we survey some algorithms for solving ECDLP.

3.2 Attacks on ECDLP

Now we assume that we have been given P and $Q \in \langle P \rangle$ and are trying to find an integer m such that

$$mP = Q$$

We will assume that $N = \#\langle P \rangle$ is known to us.

3.2.1 Pohlig-Hellman algorithm

Suppose for now that N is smooth, i.e., that it factors as

$$p_1^{\alpha_1} \dots p_n^{\alpha_n}$$

for small primes p_i. Factoring smooth numbers is easy and we may assume that we know the factorisation.

By assumption

$$Q = mP$$

for some $m \in \mathbb{Z}_N$. For each p_i, we will do the following. We know m can be written in base p_i

$$m = e_0 + e_1 p_i + e_2 p_i^2 + \dots$$

for some coefficients $e_0, e_1, ... < p_i$ that are unknown to us. We want to determine m mod $p_i^{\alpha_i}$ and therefore we seek the coefficients $e_0, e_1, ..., e_{\alpha_i - 1}$. We start by computing a searchable list A of $e\frac{N}{p_i}P$ for $0 \leq e < p_i$.

We set $Q_0 = Q$ and compute

$$\frac{N}{p_i} Q_0 \tag{5}$$

The expression (5) will be in A since

$$\frac{N}{p_i} Q_0 = \frac{N}{p_i} mP = e_0 \frac{N}{p_i} P + e_1 NP + e_2 N p_i P + ... = e_0 \frac{N}{p_i} P$$

So we can identify e_0. We then set

$$Q_1 = Q - e_0 P = (e_1 p_i + e_2 p_i^2 + ...)P$$

and compute

$$\frac{N}{p_i^2} Q_1 \tag{6}$$

The expression (6) will be in A since

$$\frac{N}{p_i^2}Q_1 = \frac{N}{p_i^2}(e_1 p_i + e_2 p_i^2 + ...)P = e_1\frac{N}{p_i}P + e_2 NP + ... = e_1\frac{N}{p_i}P$$

So we can identify e_1. We then set

$$Q_2 = Q - (e_0 + e_1 p_i)P = (e_2 p_i^2 + ...)P$$

Continuing in this manner we will be able to identify all the coefficients $e_0, \ldots, e_{\alpha_i - 1}$.

We can now determine $m_i = m \mod p_i^{\alpha_i}$ for all i. Using the Extended euclidean algorithm we can find integers a_i and b_i such that

$$a_i\frac{N}{p_i^{\alpha_i}} + b_i p_i^{\alpha_i} = 1$$

Note that

$$a_i\frac{N}{p_i^{\alpha_i}} = 1 - b_i p_i^{\alpha_i} = 1 \mod p_i^{\alpha_i}$$

Hence

$$m = \sum_{i=1}^{n} a_i\frac{N}{p_i^{\alpha_i}}m_i \mod N$$

by the Chinese remainder theorem.

If some p_i in the factorisation of N is very large, then computing the list $e\frac{N}{p_i}P$ for $0 \leq e < p_i$ becomes infeasible. This is the reason we usually assume that N is a large prime (or that there is some large prime p in the factorisation of N).

3.2.2 Pollard's rho algorithm

The following probabilistic algorithm due to Pollard runs in expected time $\mathrm{O}\big(N^{1/2}\big)$.

1. Partition $E(\mathbb{F}_q)$ into K disjoint sets S_i for $0 \leq i \leq K-1$.
2. Define $f : E(\mathbb{F}_q) \to \mathbb{Z}_K$ by $f(T) = i$, where i is the index of the S_i that contains T.
3. For each $0 \leq i \leq K-1$, pick $a_i, b_i \in_R \mathbb{Z}_N$ and let
$$M_i = a_i P + b_i Q$$
4. Pick $a, b \in_R \mathbb{Z}_N$ and let
$$P_0 = aP + bQ$$
5. Iteratively compute P_i for $i \geq 1$ using the formula
$$P_i = P_{i-1} + M_{f(P_{i-1})}$$

The idea is that the sequence $\{P_i\}$ gives us walk in $\langle P \rangle$ that behaves sufficiently random. So random that by the Birthday paradox we will have $P_j = P_i$ for some $j < i \leq \sqrt{p}$ with good probability. Then, since $P_j = cP + dQ$ and $P_i = eP + fQ$ for some $c, d, e, f \in \mathbb{Z}_N$,

$$P_j = P_i \quad \Rightarrow \quad cP + dQ = eP + fQ \quad \Rightarrow \quad (c-e)P = (f-d)mP$$

So we can take

$$m = (f-d)^{-1}(c-e) \mod N$$

as long as $(f-d)$ is invertible modulo N.

Remark 1. The number $K = 20$ appears to do well in practice [BSS99, p.95].

As the algorithm is stated above we need to save the whole sequence $\{P_i\}$ to check for repetitions. We can do better space-wise by running two sequences P_i and P_{2i} simultaneously in step 5. Note that once $P_j = P_i$ for some $j < i$, then we will have $P_{j+\ell} = P_{i+\ell}$ for all positive integers ℓ since P_{i+1} only depends on P_i. If we save only the current values of P_i and P_{2i} then it is easy to see that we will have $P_i = P_{2i}$ for some i after P_i has become periodic.

Another alternative is to have many sequences $\{P_i\}$ with different start values a, b running in parallel on client computers. If one sequence P_i hits a point T that another sequence P_i' was at ℓ iterations earlier, then we will have

$$P_{i+\ell} = P_i' \tag{7}$$

for all i from then on. By saving *distinguished* points (chosen by e.g. patterns in the least significant bits of the x-coordinate) in sequences to a central server, the server can detect when two sequences have a relation such as (7). The more distinguished points that are saved, the sooner the server will detect a relation. So it is a time-space trade-off.

3.2.3 The MOV algorithm

We will assume that N is prime in this section to simplify matters. See [Was08, Section 5.3.1] for the more general case when N is composite. We also assume N is relatively prime to q.

The idea of the MOV algorithm is to reduce the ECDLP in $E(\mathbb{F}_q)$ to a discrete logarithm problem in $\mathbb{F}_{q^k}^*$. There we can use other tools, such as index calculus [Was08, Section 5.1]. Whether the reduction is helpful or not depends on how large k is.

Definition 3.2.1. The **embedding degree of** $E(\mathbb{F}_q)$ with respect to N is the smallest positive integer k such that

$$E(\mathbb{F}_{q^k})[N] = E[N]$$

or by Theorem 2.2.2 equivalently (since N is relatively prime to q) the smallest positive integer k such that

$$E(\mathbb{F}_{q^k})[N] \cong \mathbb{Z}_N \oplus \mathbb{Z}_N$$

Note that k exists since

i) $E[N]$ is a finite group by Theorem 2.2.2,

ii) $\overline{\mathbb{F}}_q = \cup_{i \geq 1} \mathbb{F}_{q^i}$ by [NX09, Theorem 1.2.1], and

iii) $\mathbb{F}_{q^i} \subseteq \mathbb{F}_{q^{ij}}$ for any integer $j \geq 1$ by [NX09, Lemma 1.1.4].

If $E(\mathbb{F}_{q^k})[N] = E[N]$, then $\mu_N \subset \mathbb{F}_{q^k}^*$ [Was08, Corollary 3.11]. Also, when N and q are relatively prime then we have $\#\mu_N = N$ [Was08, p.86]. We want to find to find the integer m such $mP = Q$. Note that we can only retrieve m modulo N from Q, where N is the order of P. An outline of the MOV algorithm is as follows:

1. Pick $T \in_R E(\mathbb{F}_{q^k})$ where k is the embedding degree of $E(\mathbb{F}_q)$ with respect to N.

2. Compute the point $T' = \frac{\#E(\mathbb{F}_{q^k})}{N} T$.

3. Note that $T' \in E[N]$. Compute the Weil pairings

$$a = e_N(P, T')$$

$$b = e_N(Q, T')$$

If $a = 1$, go back to step 1.

4. Solve the discrete logarithm problem in $\mathbb{F}_{q^k}^*$ of finding an integer j such that

$$b = a^j$$

The integer j is a solution (mod N) to our ECDLP instance since

$$e_N(Q, T') = e_N(mP, T') = e_N(P, T')^m$$

and at the same time

$$b = a^j \quad \Rightarrow \quad e_N(Q, T') = e_N(P, T')^j$$

Thus, $e_N(P, T')^{m-j} = 1$. But this means that

$$m - j \mod N = 0$$

since $1 \neq a = e_N(P, T') \in \mu_N$ and $\#\mu_N = N$ is prime.

For a discussion on computing the Weil pairing in step 3 and more details of the MOV algorithm, see [HPSS08, Section 6.9.1].

As hinted at in the introduction this section, (under certain conditions) supersingular elliptic curves have the particularly low embedding degree 2 with respect to N (see [Was08, Proposition 5.3]). This will be irrelevant to our applications with supersingular elliptic curves in Section 6. Besides, any discrete logarithm problems on those curves will be easy anyway with the Pohlig-Hellman algorithm. In fact, the key compression for SIDH that we will discuss in 9.4 relies on this.

4 Quantum computing and post-quantum cryptography

This section provides a very brief and informal introduction to quantum computing and some of the most important quantum algorithms. We will mainly follow [DPV06, Chapter 10] and [Vaz05]. In Section 4.1 we introduce the qubit and quantum superpositions. In Sections 4.2 and 4.3 we outline two quantum algorithms that are usually singled out as the two most important quantum algorithms, Grover's algorithm and Shor's algorithm. Both algorithms, but perhaps Shor's in particular, have important practical applications in cryptanalysis. Another important problem with a related algorithm that we do not discuss here is the Hidden subgroup problem [BBD09, Section 4]. In Section 4.4 we survey some alternative post-quantum cryptosystems that have been studied extensively.

4.1 Quantum computing

The quantum analogue of the classical bit is the *qubit*. A qubit can be represented by a vector of unit length in a complex two-dimensional Hilbert space $\mathcal{H}$. Recall that a Hilbert space is a complete vector space with an inner product. Let $\overline{0}$ and $\overline{1}$ be two orthonormal basis vectors in $\mathcal{H}$. While the classical bit is either in state 0 or 1, the qubit is in the *superposition* state

$$\alpha\overline{0} + \beta\overline{1}$$

for $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$. Following standard notation we will write column vectors $\overline{\psi}$ in *ket*-notation $|\psi\rangle$. So our qubit is in the state $\alpha\,|0\rangle + \beta\,|1\rangle$. This is the mathematical model. Physically there is unfortunately no way to "read off" the coefficients α and β by looking at our system. What we can do is to choose a basis such as $\{|0\rangle, |1\rangle\}$ and "measure" our qubit in this basis. Then we will see $|0\rangle$ with probability $|\alpha|^2$ and $|1\rangle$ with probability $|\beta|^2$. At the same time our qubit will change state to the basis vector we observed in our measurement. For our discussion here we assume that such a measurement is possible. The coefficients α and β are commonly called *amplitudes*.

We manipulate qubits using *quantum gates*. A quantum gate is a unitary complex matrix of suitable dimension. Recall that a matrix U is unitary if

$$U^\dagger U = I$$

where $U^\dagger$ is the matrix obtained from U by taking the transpose, and then the complex conjugate of each entry. A quantum gate U acts linearly on a qubit's state through matrix multiplication. A quantum computation (applying a series of quantum gates to our qubit) is therefore a reversible process. We can work with systems of n qubits. Our system is then in the superposition

$$\sum_{i\in\{0,1\}^n} \alpha_i\,|i\rangle$$

where $\sum_{i\in\{0,1\}^n} |\alpha_i|^2 = 1$ just like in the single qubit system. Now suppose that we have a system $|\psi\rangle$ of two qubits. That is,

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle$$

Suppose now that we measure the first qubit of $|\psi\rangle$ and the outcome is $|0\rangle$. How does that effect the whole state of $|\psi\rangle$? Well what happens is that $|\psi\rangle$ changes into a superposition

of the basis states that are consistent with the measurement, i.e., the basis states $|00\rangle$ and $|01\rangle$,

$$|\psi\rangle = \frac{\alpha_{00}}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} |00\rangle + \frac{\alpha_{01}}{\sqrt{|\alpha_{00}|^2 + |\alpha_{01}|^2}} |01\rangle \tag{8}$$

This phenomena extends to general n-qubit systems and is very useful. It is crucial to all algorithms we will describe.

4.2 Grover's algorithm

Suppose we are given black box access to a function $f : \{0,1\}^n \to \{0,1\}$. Suppose also that we know that there is a unique $a \in \{0,1\}^n$ such that $f(a) = 1$. Classically we need 2^{n-1} queries in expectation to find a. Remarkably, on a quantum computer Grover's algorithm finds a in $\mathrm{O}\big(\sqrt{2^n}\big) = \mathrm{O}\big(2^{n/2}\big)$ queries in expectation.

We will now outline Grover's algorithm [Vaz05, Lecture 11]. We will assume that the amplitudes α_i are real through all of the algorithm. As before a is the unique bitstring such that $f(a) = 1$ that we are trying to find.

1. Start with a uniform superposition

$$\sum_{i \in \{0,1\}^n} \alpha_i |i\rangle, \quad \alpha_i = \frac{1}{2^{n/2}} \text{ for all } i$$

2. Repeat $\mathrm{O}\big(2^{n/2}\big)$ many times:

   (a) Do

$$\sum_{i \in \{0,1\}^n} \alpha_i |i\rangle \mapsto \sum_{i \in \{0,1\}^n} \alpha_i (-1)^{f(i)} |i\rangle$$

   (b) Apply inversion about the mean. By this we mean the following. Let $\mu = \frac{\sum_{i \in \{0,1\}^n} \alpha_i}{2^n}$ and do

$$\sum_{i \in \{0,1\}^n} \alpha_i |i\rangle \mapsto \sum_{i \in \{0,1\}^n} (2\mu - \alpha_i) |i\rangle$$

3. We claim that $|a\rangle$ now has amplitude α_a such that $|\alpha_a|^2 \geq \frac{1}{2}$. Thus, we measure and see $|a\rangle$ with at least probability $\frac{1}{2}$.

After the first time we apply step 2a, we have

$$\mu - \alpha_a \approx 2\mu$$

when we approximate μ with $\frac{1}{2^{n/2}}$ (α_a is $-\frac{1}{2^{n/2}}$ and not $\frac{1}{2^{n/2}}$, which slightly effects μ). After applying inversion about the mean in step 2b (which makes α_a positive again) we have

$$\alpha_a - \mu \approx 2\mu$$

For each iteration in step 2, the distance between α_a and μ will continue to increase with approximately 2μ. After at most $\mathrm{O}\big(2^{n/2}\big)$ iterations we therefore have $|\alpha_a|^2 \geq \frac{1}{2}$ as claimed in step 3.

It is not at all clear that the algorithm we described can be implemented with a sequence of quantum gates. We refer the interested reader to [Gro96].

Researchers estimate that Grover's algorithm can retrieve a 128 bit AES-key in about 2^{86} (quantum) operations [GLRS16, Table 5]. The attack requires about 3000 qubits.

Grover's algorithm cannot be efficiently parallelised. The obvious parallelisation is to split the domain of f in K equally large parts over K quantum computers. Then Grover's algorithm runs in time

$$\mathrm{O}\big(\sqrt{2^n/K}\big)) = \mathrm{O}\big(2^{n/2}/\sqrt{K}\big)),$$

which is only a factor $\sqrt{K}$ speed-up, and this is the best we can do [Zal99, Section 1].

4.3 Shor's algorithm

The hardness of the RSA cryptosystem depends on the hardness of integer factoring. For classical computers, the state-of-the-art factoring algorithm is the general number field sieve which runs in sub-exponential, but super-polynomial time (in the bit size of the number to be factored). On a quantum computer we can do better, Shor's algorithm factors a number in polynomial time. Along with Grover's algorithm, Shor's algorithm is one of the great discoveries of quantum computing.

The quantum Fourier transform (QFT) is an essential part of Shor's algorithm. The discrete Fourier transform (DFT) acts on a complex vector $\alpha = (\alpha_0, ..., \alpha_{N-1})$ producing the complex output $\beta = (\beta_0, ..., \beta_{N-1})$, where

$$\beta_j = \sum_{k=0}^{N-1} \alpha_k e^{\frac{2kj\pi i}{N}}, \quad 0 \leq j \leq N-1 \tag{9}$$

A naive implementation of DFT runs in $\mathrm{O}\big(N^2\big)$ operations. The fast Fourier transform (FFT) computes the DFT classically in $\mathrm{O}\big(N \log N\big)$ operations. With a quantum computer we can do even better. For $N = 2^n$, the QFT takes a superposition

$$\sum_{k \in \{0,1\}^n} \alpha_k \left|k\right\rangle$$

to

$$\sum_{j \in \{0,1\}^n} \beta_j \left|j\right\rangle, \tag{10}$$

where β_j are as in (9), in $\mathrm{O}\big(\log^2 N\big) = \mathrm{O}\big(n^2\big)$ quantum operations. Keep in mind though that we have no way to physically "read off" the coefficients β_j. All we can do is measure (10) and see $\left|j\right\rangle$ with probability $|\beta_j|^2$.

Suppose ℓ is a non-trivial factor of a positive integer N. Suppose also that we have a superposition α whose amplitudes have uniform periodic support with period ℓ and offset $l < \ell$, that is,

$$\alpha = \sum_{k=0}^{N/\ell-1} \sqrt{\frac{\ell}{N}} \left|k\ell + l\right\rangle,$$

Here we write the basis vectors in decimal base instead of binary base. It can be shown [DPV06, p. 320] that the QFT applied to α produces

$$\beta = \sum_{j=0}^{\ell-1} \frac{1}{\sqrt{\ell}} e^{\frac{lj2\pi i}{\ell}} \left|j\frac{M}{\ell}\right\rangle \tag{11}$$

So if we measure β we see any of the first ℓ multiples of $\frac{M}{\ell}$ with equal probability.

An outline of Shor's algorithm is as follows [DPV06, Section 10.7]:

1. Choose an integer $2 \leq x \leq N-1$ uniformly at random. Let M be a power of two such that $M \approx N^2$. Start with two quantum registers (a register is a sequence of qubits), both set to 0. The first register should be large enough to fit M values and the second one N values.
$$\alpha = |0, 0\rangle$$
2. For $1 \leq i \leq 2 \log N$, do:
   (a) Apply QFT to the first register of α to get a uniform superposition
   $$\beta = \sum_{k=0}^{M-1} \frac{1}{\sqrt{M}} |a, 0\rangle$$
   This works since the first register can be considered to be a periodic superposition with period M, that is, $\ell = M$ in the discussion above (and no offset). Thus, the QFT produces a uniform superposition β over all possible values of the first register, according to (11).
   (b) Apply the function $f(a) = x^a \mod N$ to the value a in the first register and save the result in the second register.
   $$\beta = \sum_{k=0}^{M-1} \frac{1}{\sqrt{M}} |a, x^a \mod N\rangle$$
   (c) Measure the second register. Suppose we get the result $|b\rangle$. Let the multiplicative order of x modulo N be r. By (8), this means that β is now in the superposition
   $$\beta = \sum_{k=0}^{M/r-1} \sqrt{\frac{r}{M}} |kr + l, b\rangle$$
   where $0 \leq l \leq r-1$ is the unique integer such that $x^l \mod N = b$. Applying the quantum Fourier transform to β and measuring gives a multiple of $\frac{M}{r}$ as discussed above. Let g_i be the measured value.
3. Let g be the gcd of all g_i. Then with high probability $g = \frac{M}{r}$. It can be shown that, with high probability, g is even and $\gcd(x^{g/2} + 1, N)$ is a non-trivial factor of N [DPV06, p. 320].

We have made the same simplification as in [DPV06, Section 10.7] and ignored the fact that r does not necessarily divide M. We refer to [DPV06, Section 10.7] for details.

A version of Shor's algorithm described in [Bea02] runs on $2n + 3$ qubits and uses $\mathrm{O}\big(n^3 \log(n)\big)$ elementary quantum gates, where $n = \log N$. Another version of Shor's algorithm solves the discrete logarithm problem [PZ03, Section 2.3.3].

4.4 Post-quantum cryptography

Some researchers estimate that a quantum computer capable of attacking current cryptosystems could possibly be built by 2030 at a cost of one billion dollars [CJL+16a, p. 6]. While there may be some as-yet undiscovered critical obstacle in implementing quantum computers, the research community knows of no such problem at this time. In fact, many researchers now believe it to be merely a significant engineering challenge to construct a large-scale quantum computer, according to National Institute of Standards and Technology (NIST)[1]. NIST is currently accepting submissions in its post-quantum cryptography standardisation process with submission deadline 30 November 2017[2]. Besides SIDH, which provides a key exchange, the most studied alternatives for post-quantum cryptosystems are [BL17]:

- **Code-based encryption**
  Code-based encryption uses error-correcting codes to achieve asymmetric encryption. The security of the original proposal by McEliece [McE78] is well studied. The parameters suggested by McEliece were intended to provide 64 bits of security [BL17, p. 6] and a state-of-the-art attack runs in 2^{60} operations [BLP08]. The drawback of the system is the large key size, almost 1 MB at 128 bits of quantum security [ABB+, Section 4][BL17, p. 7].

- **Lattice-based encryption**
  There are several public key encryption schemes built on lattice problems. One of the oldest and most studied is the NTRU encryption system [HPS98]. There is also an encryption system based on the *Learning with errors (LWE)* problem that has a security reduction to a lattice problem which is believed to be hard [BBD09, Section 5.4] [Reg09]. Although lattice systems show a lot of promise as post-quantum alternatives, some authors argue that much study remains before declaring these systems secure [BL17, p. 8].

- **Lattice-based signatures**
  BLISS and its variants are promising candidates for an efficient post-quantum signature scheme that also offer relatively small signatures [DDLL13]. Older proposals include NTRUSign [HHGP+] and GGH [GGH97]. Cryptanalysis for these two systems were done in [NR06]. A version of NTRUSign that was updated as a result of this cryptanalysis remains relatively efficient [BBD09, p. 182].

- **Multivariate equations-based signatures**
  Several multivariate equations-based signature schemes have been considered. These are based on the NP-hard problem of solving multivariate quadratic polynomial equations over finite fields. However, special subclasses of equation systems with trapdoors must be considered in the signature schemes. One promising system is Rainbow [DS05]. The system that the authors of [BL17] recommend (among the multivariate systems) is HFE [Pat96] in its $v-$-variant. In general these systems have large keys, but small signatures.

[1] http://csrc.nist.gov/groups/ST/post-quantum-crypto/

[2] http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf

- **Hash-based signatures** An early scheme is Lamport's one-time signatures [p. 650][DH76]. The signer holds a private key consisting of $2n$ bitstrings

  $$((x_{1,0}, x_{1,1}), (x_{2,0}, x_{2,1}), ..., (x_{n,0}, x_{n_1}))$$

  The public key is

  $$((H(x_{1,0}), H(x_{1,1})), (H(x_{2,0}), H(x_{2,1})), ..., (H(x_{n,0}), H(x_{n_1})))$$

  where H is a fixed hash function that outputs n bits. The signature of a message m with

  $$H(m) = 11010...1$$

  is then

  $$(x_{1,1}, x_{2,1}, x_{3,0}, x_{4,1}, x_{5,0}, ..., x_{n,1})$$

  The problem is that the signer can only sign one message per key.

  A more advanced scheme uses a Merkle binary tree structure where the leafs are Lamport one-time signature keys. This idea has been refined in the XMSS system [BDH11]. One problem with the XMSS system is that it is stateful; the signer must remember which keys have been used.

We will compare several PQ-cryptosystems when used as KEM schemes in Section 9.9.2.

5 Isogenies and supersingular elliptic curves

We mainly follow [Was08] and [Sut15] in this section. The most important concepts in our applications in Section 6 will be isogenies and supersingular elliptic curves. Like in Section 2, we assume that we have fixed fields K, L and $\overline{K}$ such that $\overline{K} \supseteq L \supseteq K$. Unless otherwise stated, E, E' and E_i will be elliptic curves defined over $\overline{K}$. In Section 5.1 we define elliptic curve isogenies, one of the most central concepts in this thesis. Another central concept, that of supersingular elliptic curves, is introduced in Section 5.2. In Section 5.3 we discuss how we can construct isogenies and in Section 5.4 we talk about supersingular elliptic curves defined over finite fields, which will be the setting for our applications in Section 6.

5.1 Isogenies

Definition 5.1.1. An **isogeny** is a non-constant group homomorphism

$$\phi : E_1 \to E_2$$

given by rational functions with coefficients in $\overline{K}$. That is, there are polynomials $r_1, r_2, r_3, r_4 \in \overline{K}[x, y]$ such that

$$\phi(P) = (R_1(x, y), R_2(x, y)) = \left(\frac{r_1(x, y)}{r_2(x, y)}, \frac{r_3(x, y)}{r_4(x, y)}\right)$$

for all $O \neq P = (x, y) \in E_1$ such that $r_2(P) \neq 0$ and $r_4(P) \neq 0$. We will discuss the points (x, y) where r_2 or r_4 vanish below. Note that $\phi(O) = O$ since ϕ is a homomorphism.

We can put an isogeny ϕ in a standard form by noting that

$$\frac{r_1(x, y)}{r_2(x, y)} = \frac{f_1(x) + y f_2(x)}{f_3(x) + y f_4(x)}$$

for some $f_1, f_2, f_3, f_4 \in \overline{K}[x]$ since $y^2 = x^3 + Ax + B$. Then also

$$\left(\frac{f_1(x) + y f_2(x)}{f_3(x) + y f_4(x)}\right)\left(\frac{f_3(x) - y f_4(x)}{f_3(x) - y f_4(x)}\right) = \frac{f_5(x) + y f_6(x)}{f_7(x)} = \frac{f_5(x)}{f_7(x)} + y\frac{f_6(x)}{f_7(x)}$$

for some $f_5, f_6, f_7 \in \overline{K}[x]$. For any $O \neq P = (x, y)$,

$$\phi(-P) = -\phi(P) \quad \Rightarrow \quad (R_1(x, -y), R_2(x, -y)) = (R_1(x, y), -R_2(x, y)),$$

so we must have $\frac{f_6(x)}{f_7(x)} = 0$. Hence

$$R_1(x, y) = \frac{p(x)}{q(x)}$$

for some $p, q \in \overline{K}[x]$. Similarly one can show that

$$R_2(x, y) = \frac{s(x)y}{t(x)}$$

for some $s, t \in \overline{K}[x]$. We therefore always assume that our isogenies are on standard form

$$\phi(x, y) = \left(\frac{p(x)}{q(x)}, \frac{s(x)y}{t(x)} \right)$$

where p and q have no common factor $f \in \overline{K}[x] \setminus \overline{K}$, and s and t have no such common factor either. We say that ϕ is **defined over** L if $\lambda p, \lambda q, \lambda' s, \lambda' t \in L[x]$ with $\lambda, \lambda' \in L^*$. Using that ϕ maps points satisfying a Weierstrass equation to points satisfying another Weierstrass equation, one can show that $q(x)$ and $t(x)$ have the same set of roots (not counting multiplicity) in $\overline{K}$ [Sut15, Lemma 5.22]. If $(x, y) \in E_1$ and $q(x) = 0$, then we should have that $\phi(x, y) = O$.

We will assume that $\phi : E_1 \to E_2$ and

$$\phi(x, y) = \left(\frac{p(x)}{q(x)}, \frac{s(x)y}{t(x)} \right) \tag{12}$$

when we speak of an isogeny ϕ for the rest of this section.

Definition 5.1.2. The **degree of an isogeny** ϕ (denoted by $\deg(\phi)$) is the maximum of the degrees of p and q. That is,

$$\deg(\phi) = \max\{\text{degree}(p(x)), \text{degree}(q(x))\}$$

Definition 5.1.3. An isogeny ϕ is **inseparable** if $\frac{d}{dx}\left(\frac{p(x)}{q(x)}\right) = 0$. If ϕ is not inseparable, then it is **separable**.

To assume that our isogenies are on standard form (12) is very helpful when proving some of the theorems in this section.

Proposition 5.1.4. *A separable isogeny $\phi : E_1 \to E_2$ has*

$$\deg \phi = \# \ker \phi$$

An inseparable isogeny ϕ has

$$\deg \phi > \# \ker \phi$$

Proof. See [Was08, Proposition 12.8]. We will say something about the proof when ϕ is separable. Suppose ϕ is on standard form (12), then

$$\frac{d}{dx}\left(\frac{p(x)}{q(x)}\right) \neq 0 \quad \Rightarrow \quad p'(x)q(x) - p(x)q'(x) \neq 0 \tag{13}$$

One then looks at a point $(a, b) \in \phi(E_1)$ that satisfies certain properties. One required property is that

$$\max\{\text{degree}(p(x)), \text{degree}(q(x))\} = \deg(\phi) = \text{degree}(p(x) - aq(x))$$

Note that

$$p(x) - aq(x) = 0$$

for any $(x, y) \in E_1$ such that $\phi((x, y)) = (a, b)$. We then use (13) to show that $p(x) - aq(x)$ has $\deg \phi$ distinct roots of multiplicity one. After that we argue that there are $\deg \phi$ number of distinct $(x, y) \in E_1$ such that $\phi(x, y) = (a, b)$. Since ϕ is a homomorphism this means

$$\# \ker \phi = \deg \phi$$

□

It follows directly from Proposition 5.1.4 that the kernel of an isogeny is finite. It is also a subgroup of E_1 since an isogeny is a group homomorphism.

The identity map

$$\mathrm{id}(P) = P, \quad \text{for all } P \in (\overline{K})$$

is an isogeny. Usually in group theory two groups are isomorphic if there are bijective group homomorphisms between them. For two elliptic curves to be isomorphic we will also require that the bijective homomorphisms are isogenies, i.e., that they are given by rational functions.

Definition 5.1.5. Two elliptic curves E_1 and E_2 are **isomorphic** if there are isogenies

$$\phi : E_1 \to E_2, \quad \psi : E_2 \to E_1$$

defined over $\overline{K}$ such that

$$\phi \circ \psi = \mathrm{id}_{E_2}, \quad \psi \circ \phi = \mathrm{id}_{E_1}$$

Theorem 5.1.6. *An isogeny $\phi : E_1 \to E_2$ is surjective.*

Proof. See [Was08, Theorem 12.9]. We will say something about the proof here. We know that $\phi(O) = O$. If $O \neq (a, b) \in E_2$, we want to find (x, y) such that $\phi((x, y)) = (a, b)$. Since ϕ is on standard form (12) this means

$$p(x) - aq(x) = 0$$

We will only discuss the case when $p(x) - aq(x)$ is not constant here. We can use that we are working in $\overline{K}$ to find $(x_0, y_0) \in E_1$ such that $p(x_0) - aq(x_0) = 0$. Then

$$\phi((x_0, y_0)) = (a, b')$$

for some $y_0, b' \in \overline{K}$. This means $(a, b) = (a, \pm b')$, so $\phi((x_0, y_0)) = (a, b)$ or

$$\phi((x_0, -y_0)) = \phi(-(x_0, y_0)) = -\phi((x_0, y_0)) = -(a, b') = (a, -b') = (a, b)$$

□

One can verify that a composition of isogenies is an isogeny.

Proposition 5.1.7. *A composition $\psi = \psi_n \circ \cdots \circ \psi_1$ of separable isogenies $\psi_1, \ldots, \psi_n$ is separable.*

Proof. See [Sut15, Lecture 14 p. 1]. □

Proposition 5.1.8. *Let the isogeny $\psi = \psi_n \circ \cdots \circ \psi_1$ be a composition of isogenies $\psi_1, \ldots, \psi_n$. Then*

$$\deg \psi = (\deg \psi_n) \ldots (\deg \psi_1)$$

Proof. See [Sut15, Lemma 7.8]. We will say something about the proof when we compose separable isogenies. Suppose we have

$$\phi = \phi_2 \circ \phi_1$$

for separable isogenies ϕ_1 and ϕ_2. Let $P \in \ker \phi_2$, then

$$\#\phi_1^{-1}(P) = \#\ker \phi_1$$

since ϕ_1 is a surjective homomorphism by Theorem 5.1.6. Using also that ϕ is separable by Proposition 5.1.7 and that separable isogenies have degree equal to the order of their kernel by Proposition 5.1.4 we get

$$\deg \phi = \#\ker \phi = (\#\ker \phi_1)(\#\ker \phi_2) = (\deg \phi_1)(\deg \phi_2)$$

By induction the proposition holds for any number of composed separable isogenies. We will only compose separable isogenies in our applications. □

The following theorem will be important for our applications in Section 6.

Theorem 5.1.9. *If there are separable isogenies*

$$\phi : E_1 \to E_2, \quad \psi : E_1 \to E_3$$

such that

$$\ker \phi = \ker \psi,$$

then E_2 and E_3 are isomorphic. In fact, there is an isomorphism θ such that

$$\phi = \theta \circ \psi$$

Proof. See [Was08, Proposition 12.12]. □

An isogeny from E to itself is called an **endomorphism**. One can show [Sut15, Definition 5.19] that for any E, defined over $\mathbb{F}_q$ for a prime power $q = p^k$, there is an endomorphism $\pi_q : E \to E$ defined by

$$\pi_q(x, y) = (x^q, y^q)$$

The endomorphism π_q is called **the Frobenius endomorphism**. One can also study the p-power Frobenius isogeny π_p defined by

$$(x, y) \to (x^p, y^p)$$

When E is defined over $\mathbb{F}_{p^k}$, but not over $\mathbb{F}_p$, π_p is an isogeny but not an endomorphism [Sut15, Lecture 5, p. 7]. The codomain of π_p is commonly denoted by $E^{(p)}$ and is defined by the Weierstrass equation

$$y^2 = x^3 + A^p x + B^p$$

Theorem 5.1.10. *Let ϕ be an isogeny defined over a finite field $\mathbb{F}_q$ of characteristic p. Then ϕ can be written as*

$$\phi = \phi_{sep} \circ \pi_p^n$$

for some integer $n \geq 0$ and a separable isogeny ϕ_{sep}.

Proof. See [Sut15, Corollary 6.4]. □

Note that π_p^n is inseparable. Since π_p^n has degree p^n, Theorem 5.1.10 and Proposition 5.1.8 imply that any isogeny, over a finite field of characteristic p, whose degree is not a multiple of p, is separable. All the isogenies we will work with in our applications will be separable.

Definition 5.1.11. The **j-invariant** of an elliptic curve E given by the Weierstrass equation

$$y^2 = x^3 + Ax + B$$

is

$$j(E) = 1728\frac{4A^3}{4A^3 + 27B^2}$$

Note that $4A^3 + 27B^2 \neq 0$ by assumption in Definition 2.1.1. If E is defined over K, then the $j(E) \in K$.

Proposition 5.1.12. *For any $j \in L$, there is an elliptic curve E defined over L such that*

$$j(E) = j$$

Proof. See [Was08, p. 47]. When $j \neq 0, 1728$, one can verify that j is the j-invariant of the elliptic curve E defined over L that is given by

$$y^2 = x^3 + \frac{3j}{1728 - j}x + \frac{2j}{1728 - j}$$

There are known Weierstrass equations (with coefficients $A, B \in L$ such that $4A^3+27B^2 \neq 0$) that give the j-invariants 0 and 1728 as well [Was08, p. 46]. □

The j-invariant gets its name from being invariant under isomorphism.

Proposition 5.1.13. *The elliptic curves E_1 and E_2 are isomorphic if and only if $j(E_1) = j(E_2)$.*

Proof. See [Sut15, Theorem 15.11]. Here we only discuss one of the directions in the statement. Suppose that $j(E_1) = j(E_2)$. As one part of the proof that E_1 and E_2 are isomorphic we want an isogeny $\phi : E_1 \to E_2$ with $\ker \phi = \{O\}$. Let E_1 and E_2 be given by the Weierstrass equations

$$y^2 = x^3 + Ax + B, \quad y^2 = x^3 + A'x + B,$$

respectively. It turns out [Was08, Theorem 2.19] that when $j(E_1) = j(E_2)$ it is always possible to find a $\lambda \in \overline{K}^*$ such that

$$A' = \lambda^4 A, \quad B' = \lambda^6 B$$

One can then show that ϕ can be defined by $\phi(x, y) = (\lambda^2 x, \lambda^3 y)$. □

In our applications we will work with isogenies that are defined over $\mathbb{F}_{p^2}$.

Proposition 5.1.14. *Let E_1 and E_2 be elliptic curves defined over $\mathbb{F}_q$ for some prime power q. If there is an isogeny $\phi : E_1 \to E_2$ that is defined over $\mathbb{F}_q$, then*

$$\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$$

Proof. Let N be the degree of ϕ. Following [Was08, Excersise 12.12] we fix a prime $\ell \nmid qN$. Let $n \geq 1$ be an integer. For any $P \in E_1[\ell^n]$ we have $o(\phi(P)) \mid \ell^n$, so $\phi(P) \in E_2[\ell^n]$. Using Lagrange's theorem it is also easy to see that ϕ restricted to $E_1[\ell^n]$ is injective since $\#\ker\phi \mid N$ by [Sut15, Corollary 6.9]. Thus, ϕ restricted to $E_1[\ell_n]$ is a group isomorphism

from $E_1[\ell^n]$ to $E_2[\ell^n]$. Pick $\{P_1, P_2\} \subseteq E_1[\ell^n]$ such that $\langle P_1, P_2\rangle = E_1[\ell^n]$. Then the Frobenius endomorphism π_q acts like

$$\pi_q(P_1) = aP_1 + bP_2$$

$$\pi_q(P_2) = cP_1 + dP_2$$

for some $a, b, c, d \in \mathbb{Z}_{\ell^n}$. Let $(\pi_q)_{\ell^n}$ be the matrix

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

By [Was08, Proposition 4.11],

$$\text{Trace}((\pi_q)_{\ell^n}) = q + 1 - \#E_1(\mathbb{F}_q) \mod \ell^n$$

As discussed in [Was08, proof of Lemma 4.5], π_q commutes with ϕ when ϕ is defined over $\mathbb{F}_q$. Since $\langle \phi(P_1), \phi(P_2)\rangle = E_2[\ell^n]$, it follows that

$$q + 1 - \#E_1(\mathbb{F}_q) = q + 1 - \#E_2(\mathbb{F}_q) \mod \ell^n$$

But this holds for arbitrarily large n so we must have

$$\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$$

□

Tate in fact proved that the converse of Proposition 5.1.14 holds as well [Tat66, Theorem 1.(c)].

Theorem 5.1.15. *Let E_1 and E_2 be defined over L. For any isogeny $\phi : E_1 \to E_2$ defined over L there exists an isogeny $\widehat{\phi} : E_2 \to E_1$ defined over L, called the **dual isogeny** of ϕ, such that*

$$\widehat{\phi} \circ \phi = [\deg \phi]$$

Proof. See [Gal12, Theorem 9.6.21]. □

When we study supersingular isogeny graphs in Section 7.2, Theorem 5.1.15 will tell us that if there is a directed edge from vertex A to vertex B, then there is a directed edge from B to A as well.

5.2 Supersingular elliptic curves

Proposition 5.2.1. *For an elliptic curve E defined over K with $\text{char}(K) = p > 0$ we have*

i) $E[p] = \{O\}$, or

ii) $E[p] \cong \mathbb{Z}_p$

*If i) holds, then we say that E is **supersingular**. If ii) holds, then we say that E is **ordinary**.*

Proof. See [Was08, Theorem 3.2]. It turns out that the multiplication by p-map $[p]$, i.e.,

$$[p](P) = pP, \quad \text{for all } P \in E,$$

is an inseparable endomorphism of degree p^2 (when $\text{char}(K) = p > 0$). Then by Proposition 5.1.4, $\deg[p] > \#\ker[p]$. So

$$p^2 = \deg[p] > \#\ker[p] = \#E[p]$$

It follows from the fundamental structure theorem for finite abelian groups that $E[p]$ has an element of order ℓ for any prime ℓ dividing $\#E[p]$. At the same time all elements in $E[p]$ have order dividing p. Since $\#E[p] < p^2$, the only possibilities are $\#E[p] = 1$ or $\#E[p] = p$. □

Theorem 5.2.2. *Let E be an elliptic curve defined over $\mathbb{F}_q$, where $q = p^k$ for a prime p and a positive integer k. Then E is supersingular if and only if $p \mid a$ where*

$$a = q + 1 - \#E(\mathbb{F}_q)$$

Proof. See [Was08, Theorem 4.31]. We will say something about one of the directions. Suppose that

$$\#E(\mathbb{F}_q) = q + 1 - a$$

with $p \mid a$. It turns out [Was08, Theorem 4.12] that there is a recursively defined sequence a_n such that

$$\#E(\mathbb{F}_{q^n}) = q^n + 1 - a_n \tag{14}$$

for all positive integers n. The sequence is defined by

$$a_0 = 2, \quad a_1 = a, \quad a_{n+2} = a_1 a_{n+1} - q a_n, \quad \text{for all } n \geq 0$$

where a is as in the statement of the theorem. If $p \mid a$, then

$$a_n \mod p = 0$$

for all n. Thus,

$$\#E(\mathbb{F}_{q^n}) \mod p = 1$$

for all n by (14). So there can be no elements of order p in any group $E(\mathbb{F}_{q^n})$. Since $\overline{\mathbb{F}}_q = \cup_{i \geq 1} \mathbb{F}_{q^i}$ by [NX09, Theorem 1.2.1], this means

$$E[p] = \{O\}$$

□

As we saw in the proof of Proposition 5.1.14, we can represent the restriction of ϕ_q to $E[m]$ (with m relatively prime to q) by a 2×2 matrix A_m. For any such m, it turns out that the integer a in Theorem 5.2.2 is the trace of A_m modulo m [Was08, Theorem 4.10]. For this reason a is called the **Trace of Frobenius**.

Theorem 5.2.3. *(Hasse's theorem) The trace of Frobenius a in Theorem 5.2.2 satisfies*

$$|a| \leq 2\sqrt{q} = 2\sqrt{p^k}$$

Proof. See [Was08, Theorem 4.2]. Let $(x, y) = P \in E(\overline{\mathbb{F}}_q)$. It can be shown that

$$(x, y) = \phi_q(x, y) = (x^q, y^q)$$

if and only if $(x, y) \in E(\mathbb{F}_q)$. One can also show that

$$\phi_q - \mathrm{id}$$

is a separable endomorphism. But then

$$a = q + 1 - \#E(\mathbb{F}_q) = q + 1 - \ker(\phi_q - \mathrm{id}) = q + 1 - \deg(\phi_q - \mathrm{id})$$

and it is possible to prove the theorem by studying $\deg(\phi_q - \mathrm{id})$. □

Remark 2. Let E be a supersingular elliptic curve defined over $\mathbb{F}_p$ for a prime $p \geq 5$. By Theorem 5.2.2 we have

$$\#E(\mathbb{F}_p) = p + 1 - a$$

with i) $a = 0$, or ii) $p \leq |a|$. But by Theorem 5.2.3

$$|a| \leq 2\sqrt{p},$$

so case ii) can not occur since $p \geq 5$.

Definition 5.2.4. The **endomorphism ring** of E (denoted by $\mathrm{End}(E)$) is the set of all endomorphisms $\phi : E \to E$ (together with the constant zero-map) defined over $\overline{K}$ with the following ring structure.

i) The neutral addition element is the constant zero-map $\phi(P) = O$, for all $P \in E(\overline{K})$.

ii) Addition is given by
$$(\phi_1 + \phi_2)(P) = \phi_1(P) + \phi_2(P)$$
for all $\phi_1, \phi_2 \in \mathrm{End}(E)$.

iii) The neutral multiplication element is the identity isogeny.

iv) Multiplication is given by
$$(\phi_1 \cdot \phi_2)(P) = \phi_1 \circ \phi_2(P)$$
for all $\phi_1, \phi_2 \in \mathrm{End}(E)$

One can verify that multiplication is distributive with respect to addition.

Note that $\mathrm{End}(E)$ is not necessarily a commutative ring. The following theorem is important background for our applications in Section 6.

Theorem 5.2.5. *If E is a supersingular elliptic curve, then $\mathrm{End}(E)$ is a non-commutative ring. If E is ordinary, then $\mathrm{End}(E)$ is commutative.*

Proof. See [Sil09, Theorem III.9.4 and V.3.1]. □

5.3 Vélu's formulae

Given a finite subgroup $G \subseteq E_1$, the following formulae give us a separable isogeny ϕ and an elliptic curve E_2 such that

$$\phi : E_1 \to E_2$$
$$\ker \phi = G$$

By Theorem 5.1.9, E_2 is unique up to isomorphism. Here we follow the presentation of [Was08, Theorem 12.16]. The original formulae are due to Vélu [Vél71]. In general ϕ and E_2 will be defined over $\overline{K}$. If $G \subseteq E_1(L)$ (so E_1 is also defined over L by the definition of $E_1(L)$), then ϕ and E_2 will also be defined over L. For our applications we are particularly interested in $L = \mathbb{F}_q$ for a prime power q, see Remark 3.

Let E_1 be given by the Weierstrass equation

$$y^2 = x^3 + Ax + B$$

with $A, B \in \overline{K}$. Let G be a finite subgroup of E_1. For any $O \neq Q = (x_Q, y_Q) \in G$ let

$$g_Q^x = 3x_Q^2 + A, \quad g_Q^y = -2y_Q$$

$$v_Q = \begin{cases} g_Q^x, & \text{if } 2Q = O \\ 2g_Q^x, & \text{otherwise} \end{cases}, \quad u_Q = (g_Q^y)^2$$

Let the following be a disjoint union

$$G = \{O\} \sqcup G_2 \sqcup R \sqcup (-R)$$

where G_2 are all points of order 2 in G. So R contains exactly one of Q and $-Q$ for all $Q \in G \setminus (\{O\} \cup G_2)$. Let $S = R \cup G_2$ and define

$$v = \sum_{Q \in S} v_Q$$

$$w = \sum_{Q \in S} (u_Q + x_Q v_Q)$$

The codomain of ϕ, the elliptic curve E_2, will be given by the Weierstrass equation

$$Y^2 = X^3 + (A - 5v)X + (B - 7w) \tag{15}$$

The isogeny $\phi = (X(x, y), Y(x, y))$ is given by

$$X(x, y) = x + \sum_{Q \in S} \left(\frac{v_Q}{x - x_Q} + \frac{u_Q}{(x - x_Q)^2} \right) \tag{16}$$

$$Y(x, y) = y - \sum_{Q \in S} \left(u_Q \frac{2y}{(x - x_Q)^3} + v_Q \frac{y - y_Q}{(x - x_Q)^2} - \frac{g_Q^x g_Q^y}{(x - x_Q)^2} \right) \tag{17}$$

See [Was08, Theorem 12.16] for a proof of correctness of the formulae. Note that (16) and (17) require us to iterate over all elements in S.

Remark 3. By inspecting (15), (16) and (17) we see that if E is defined over a finite field $\mathbb{F}_q$ and $G \subseteq E(\mathbb{F}_q)$, then E_2 and ϕ are defined over $\mathbb{F}_q$. Hence

$$\#E_1(\mathbb{F}_q) = \#E_2(\mathbb{F}_q)$$

by Proposition 5.1.14.

5.4 Supersingular elliptic curves defined over $\overline{\mathbb{F}}_p$

Let $p \geq 5$ be a fixed prime in this section.

Theorem 5.4.1. *Let E be a supersingular elliptic curve defined over $\overline{\mathbb{F}}_p$. Then $j(E) \in \mathbb{F}_{p^2}$.*

Proof. See [Sil09, Theorem V.3.1.(a)(iii)]. □

Proposition 5.4.2. *Let j_0 be a supersingular j-invariant in $\mathbb{F}_{p^2}$. Then there exists a supersingular elliptic curve E defined over $\mathbb{F}_{p^2}$ with $j(E) = j_0$ and*

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p+1} \oplus \mathbb{Z}_{p+1}$$

Proof. We will follow the proof of Lemma 3.21 in [BGJGP05]. By the proof of Proposition 5.1.13 there is a supersingular elliptic curve E_1 with $j(E_1) = j_0$. By [Was08, Theorem 4.3] there exists a supersingular elliptic curve E_2 defined over $\mathbb{F}_p$ with Trace of Frobenius $a = 0$. By the sequence a_n from the proof of Theorem 5.2.2 we have that

$$\#E_2(\mathbb{F}_{p^2}) = p^2 + 1 - (-2p) = (p+1)^2$$

By [Gal12, Theorem 25.3.17], all supersingular elliptic curves defined over $\overline{\mathbb{F}}_p$ are isogenous[3]. Let $\phi : E_2 \to E_1$ be an isogeny. By Theorem 5.1.10 we can write ϕ as

$$\phi = \phi_s \circ \pi_p^n$$

for some non-negative integer n and some separable isogeny ϕ_s. We know that π_p is an endomorphism since E_2 is defined over $\mathbb{F}_p$. Hence ϕ_s is a separable isogeny from E_2 to E_1. By [Was08, Theorem 4.10], π_p satisfies the endomorphism equation

$$\pi_p^2 - \pi_p a + [p] = 0$$

on E_2. Hence $\pi_p^2 = [-p]$, since $a = 0$. Then

$$\phi_s \circ \pi_p^2 = \phi_s \circ [-p] = [-p] \circ \phi_s$$

So $\pi_p^2(\ker \phi_s) \subseteq \ker \phi_s$. By (the discussion after) [Sut15, Theorem 6.8] and [BSS99, p. 45] there is then a separable isogeny $\phi_s' : E_2 \to E_2/\ker \phi_s$ that is defined over $\mathbb{F}_{p^2}$, where the elliptic curve $E_2/\ker \phi_s$ is defined over $\mathbb{F}_{p^2}$ and isomorphic to E_1. Let $E = E_2/\ker \phi_s$. Since ϕ_s' is defined over $\mathbb{F}_{p^2}$ we have that

$$\#E(\mathbb{F}_{p^2}) = \#E_2(\mathbb{F}_{p^2})$$

by Proposition 5.1.14. It now follows from [Was08, Theorem 4.4] that

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p+1} \oplus \mathbb{Z}_{p+1}$$

□

[3] In fact, [Gal12, Theorem 25.3.17] says something stronger. It says that given two supersingular elliptic curves defined over $\overline{\mathbb{F}}_p$ and a prime $\ell \neq p$ we can find an isogeny, whose degree is a power of ℓ, between them. [Gal12] refers to [Mes86] for a proof, who in turn attributes the proof to Serre according to [Gal12].

6 Supersingular isogeny Diffie-Hellman key exchange

In this section we present SIDH. We describe the key exchange in Section 6.2, and discuss picking secret keys and computing isogenies of power-smooth degree in Sections 6.3 and 6.4, respectively.

6.1 Background

In a way the history of SIDH starts before Jao and De Feo introduced it in [JDF11] in 2011. Stolbunov suggested in [Sto10] to use isogenies between ordinary elliptic curves to obtain a post-quantum key exchange. But the commutative structure of the endomorphism rings of those curves was exploited in [CJS14] to give a sub-exponential attack. See [CJS14] for details of the time complexity of the attack. Also, as pointed out in [JDF11, p. 1], the performance of an early implementation of the cryptosystem was poor: about 230 seconds on a desktop PC for a single exchange at 128-bits of classical security. While the cryptosystem of [Sto10] could possibly be adapted with the attack of [CJS14] in mind, the already poor performance of the cryptosystem led [CJS14] to suggest that *"...attacks such as ours would seem to disqualify such systems from consideration in a post-quantum world."* Supersingular elliptic curves do not have a commutative endomorphism ring and it appears that this led [JDF11] to present SIDH in 2011. Note however that there are other differences between the two cryptosystems than just using different sets of elliptic curves.

6.2 SIDH key exchange

We now describe the supersingular isogeny Diffie-Hellman (SIDH) key-exchange that was introduced in [JDF11]. The public parameters of the cryptosystem are p, E_0, (P_A, Q_A) and (P_B, Q_B) such that:

i) p is a prime of the form
$$p = \ell_A^{e_A} \ell_B^{e_B} f - 1 \tag{18}$$
where ℓ_A and ℓ_B are small distinct primes, and f is a small positive cofactor such that p is prime. The number f is meant to give flexibility in selecting p and can be set to 1. The cryptographic strength of the cryptosystems with regards to known attacks depend on the size of $\ell_A^{e_A}$ or $\ell_B^{e_B}$. Hence we want
$$\ell_A^{e_A} \approx \ell_B^{e_B}$$
Most implementations use a prime p of the form $p = 2^{e_a} 3^{e_B} - 1$. As an example, [CLN16] uses
$$p = 2^{372} 3^{239} - 1$$
when aiming for 128 bits of quantum security and 192 bits of classical security. Actually, [JDF11] define p to be of the form $p = \ell_A^{e_A} \ell_B^{e_B} f \pm 1$. We have restricted ourselves to the form in (18) since most implementations use this form [AFJ14][CLN16][KAJMK16a] and since it fits well with our results in Section 5.4. However, it turns out that one can prove results corresponding to those in Section 5.4 in the case of $p = \ell_A^{e_A} \ell_B^{e_B} f + 1$ as well. See Remark 9 in Section 7.1 for details. In this author's experience, finding a prime p of suitable form is not a problem in practice, but we refer to [JDF11, Section 4.1] for a discussion.

ii) E_0 is a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ such that

$$E_0(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}f} \oplus \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}f} \tag{19}$$

By our results in Section 5.4 we know that any supersingular elliptic curve defined over $\overline{\mathbb{F}}_p$ is isomorphic to one defined over $\mathbb{F}_{p^2}$ with the same group structure as that in (19). In this sense we are not missing anything by working over $\mathbb{F}_{p^2}$. All the vertices of the supersingular isogeny graph that we will study in Section 7.1 can be represented in $\mathbb{F}_{p^2}$. The group structure in (19) is important for us to be able to find suitable rational points (P_A, Q_A) and (P_B, Q_B) in $E(\mathbb{F}_{p^2})$. On the problem of actually finding a suitable curve E_0 we note that [Brö09] gives an efficient algorithm to find a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ with Trace of Frobenius $-2p$. Also, there are several known Weierstrass equations for elliptic curves defined over $\mathbb{F}_{p^2}$ with Trace of Frobenius $-2p$. For example, $y^2 = x^3 + x$ is one such equation when $p = 3 \mod 4$ [Was08, Proposition 4.37], which happens in particular if $\ell_A = 2$ (assuming $e_A \geq 2$). This is the Weierstrass equation used in [CLN16]. Once we have a curve defined over $\mathbb{F}_{p^2}$ with Trace of Frobenius $-2p$, [Was08, Theorem 4.4] guarantees that it has the same $\mathbb{F}_{p^2}$-rational group structure as in (19). In fact, we shall see in Section 7.1 and Theorem 7.1.4 that once we have one curve with the correct group structure, then we can generate a "random" curve with the same group structure. The curve will be random in the sense that its j-invariant will be chosen (roughly) uniformly at random from the set of all supersingular j-invariants in $\mathbb{F}_{p^2}$.

iii) (P_A, Q_A) is Alice's public parameter. Recall that

$$E_0[\ell_A^{e_A}] \cong \mathbb{Z}_{\ell_A^{e_A}} \oplus \mathbb{Z}_{\ell_A^{e_A}}$$

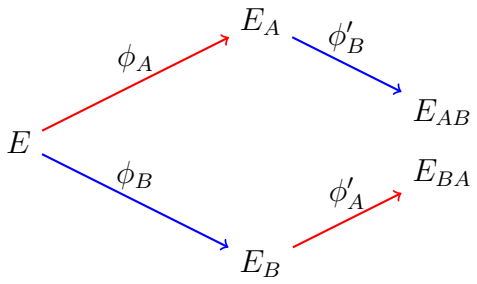
by Proposition 2.2.2. By the group structure of $E_0(\mathbb{F}_{p^2})$ in (19) we have

$$E_0(\mathbb{F}_{p^2})[\ell_A^{e_A}] = E_0[\ell_A^{e_A}]$$

The points $P_A, Q_A \in E[\ell_A^{e_A}]$ are chosen to be independent in the sense that linear combinations of them generate all of $E[\ell_A^{e_A}]$. As in Section 2.2, we denote this by $\langle P_A, Q_A \rangle = E_0[\ell_A^{e_A}]$. One can verify that if the Weil pairing $e_{\ell_A^{e_A}}(P_A, Q_A)$ has order $\ell_A^{e_A}$, then $\langle P_A, Q_A \rangle = E_0[\ell_A^{e_A}]$. The points (P_A, Q_A) can be found by simply testing random points and using the Weil pairing to check independence [JDF11, Section 4.1]. There are also more efficient methods for certain starting curves E_0 [CLN16, Choosing generator points for torsion subgroups]. Bob's corresponding public parameter is (P_B, Q_B) such that $\langle P_B, Q_B \rangle = E_0[\ell_B^{e_B}]$.

Figure 1 explains the idea of SIDH.

Figure 1: Alice will compute the red isogenies (with sub-index A) and Bob will compute the blue ones (with sub-index B). Alice will know the codomain E_{BA} of ϕ'_A and Bob will know the codomain E_{AB} of ϕ'_B. We will show that $j(E_{BA}) = j(E_{AB})$, so they can use the j-invariants of the respective codomains as a shared key.



Given the fixed public parameters p, E_0, (P_A, Q_A) and (P_B, Q_B) the SIDH key exchange is as follows:

1. Alice picks $m_A, n_A \in_R \mathbb{Z}_{\ell_A^{e_A}}$, under the constraint that not both of m_A and n_A are divisible by ℓ_A. These two numbers make up Alice's secret key $sk_A = (m_A, n_A)$. She lets
$$G_A = \langle m_A P_A + n_A Q_A \rangle$$
By Remark 4, the order of G_A is $\ell_A^{e_A}$. Alice computes (using for instance Vélu's formulae) a separable isogeny $\phi_A : E_0 \to E_A$ that has kernel G_A. Alice sends her public key
$$pk_A = (E_A, \phi_A(P_B), \phi_A(Q_B))$$
to Bob. Bob similarly picks $m_B, n_B \in_R \mathbb{Z}_{\ell_B^{e_B}}$, computes a separable isogeny $\phi_B : E_0 \to E_B$ with kernel $G_B = \langle m_B P_B + n_B Q_B \rangle$ and sends
$$pk_B = (E_B, \phi_B(P_A), \phi_A(Q_A))$$
to Alice.

2. Upon receiving
$$pk_B = (E_B, \phi_B(P_A), \phi_A(Q_A))$$
from Bob, Alice computes a separable isogeny $\phi'_A : E_B \to E_{BA}$ that has kernel
$$\langle m_A \phi_B(P_A) + n_A \phi_B(Q_A) \rangle = \phi_B(\langle m_A P_A + n_A Q_A \rangle) = \phi_B(G_A)$$
Similarly, Bob computes a separable isogeny $\phi'_B : E_A \to E_{AB}$ with kernel $\phi_A(G_B)$. By Remark 5, E_{BA} and E_{AB} are isomorphic as elliptic curves, so, by Proposition 5.1.13, Alice and Bob can take the j-invariant
$$j(E_{BA}) = j(E_{AB})$$
as their shared key.

We stress that Alice can only evaluate the composition
$$\phi = \phi'_A \circ \phi_B$$

on $\langle P_A, Q_A \rangle = E_0[\ell_A^{e_A}]$ and Bob can only evaluate the composition

$$\phi' = \phi'_B \circ \phi_A$$

on $\langle P_B, Q_B \rangle = E_0[\ell_B^{e_B}]$. If Alice could evaluate $\phi(P_B)$ and $\phi(Q_B)$, then she retrieves Bob's secret key by solving a DLP instance in $E_{BA}(\mathbb{F}_{p^2})$ since

$$\phi(P_B) = -m_B^{-1} n_B \phi(Q_B)$$

(assuming $m_A \in \mathbb{Z}^*_{\ell_B^{e_B}}$). We will see in next section that Alice in this case can assume that Bob's secret key is

$$(m'_B, n'_B) = (1, -m_B^{-1} n_B \mod \mathbb{Z}_{\ell_B^{e_B}})$$

The following remarks are meant to help the reader who wants justification for some of the claims made in our description of the key exchange.

Remark 4. By construction $m_A P_A + n_A Q_A$ lies in $E[\ell_A^{e_A}]$ so its order is at most $\ell_A^{e_A}$. Without loss of generality we can assume that $m_A \in \mathbb{Z}^*_{\ell_A^{e_A}}$ so

$$k m_A \neq 0 \mod \ell_A^{e_A}$$

for any integer $1 \leq k < \ell_A^{e_A}$. This means that

$$k(m_A P_A + n_A Q_A) = k m_A P_A + k n_A Q_A \neq O$$

since

$$\langle P_A, Q_A \rangle = E_0[\ell_A^{e_A}] \cong \mathbb{Z}_{\ell_A^{e_A}} \oplus \mathbb{Z}_{\ell_A^{e_A}}$$

Hence the order of $m_A P_A + n_A Q_A$ is $\ell_A^{e_A}$.

Remark 5. Let $\phi : E_0 \to E_{BA}$ be defined by

$$\phi = \phi'_A \circ \phi_B$$

We want to show that

$$\langle G_A, G_B \rangle = \langle m_A P_A + n_A Q_A, m_B P_B + n_B Q_B \rangle = \ker \phi$$

It is clear that by construction

$$\langle m_A P_A + n_A Q_A, m_B P_B + n_B Q_B \rangle \subseteq \ker \phi$$

Now suppose that $P \in \ker \phi$. If $P \notin \ker \phi_B = \langle m_B P_B + n_B Q_B \rangle$, then we must have

$$\phi_B(P) = \phi_B(k(m_A P_A + n_A Q_A))$$

for some integer k. Thus, for some integer k'

$$P - k(m_A P_A + n_A Q_A) = k'(m_B P_B + n_B Q_B) \in \ker \phi_B$$

so $P \in \langle m_A P_A + n_A Q_A, m_B P_B + n_B Q_B \rangle$. Similarly one can show that

$$\langle G_A, G_B \rangle = \langle m_A P_A + n_A Q_A, m_B P_B + n_B Q_B \rangle = \ker \phi'$$

for $\phi' : E_0 \to E_{AB}$ defined by

$$\phi' = \phi'_B \circ \phi_A$$

Hence E_{AB} and E_{BA} are isomorphic as elliptic curves by Theorem 5.1.9.

6.3 Normalised secret keys

As noted in [CLN16] and [GPST16] there are more ways to pick m_A and n_A in step 1 of the key exchange than there are cyclic subgroups of order $\ell_A^{e_A}$ in $E[\ell_A^{e_A}]$. As explained in [GPST16, Lemma 2.1], an attacker can assume that $m_A = 1$ or $n_A = 1$ in Alice's secret key. To see this, assume without loss of generality that $m_A \in \mathbb{Z}^*_{\ell_A^{e_A}}$. Then

$$\langle m_A P_A + n_A Q_A \rangle = \mathbb{Z}_{\ell_A^{e_A}} m_A^{-1}(m_A P_A + n_A Q_A) = \mathbb{Z}_{\ell_A^{e_A}}(P_A + m_A^{-1} n_A Q_A) = \langle P_A + m_A^{-1} n_A Q_A \rangle$$

If we write Alice's secret key as a tuple $(1, \alpha)$ (or $(\alpha, 1)$), then it is easy to see that there are $\ell_A^{e_A - 1}(\ell_A + 1)$ distinct (with respect to the subgroups that they generate) keys:

i) If $\alpha \in \mathbb{Z}^*_{\ell_A^{e_A}}$, then we can assume the private key has the form $(1, \alpha)$, since $(\alpha \cdot \alpha^{-1}, \alpha \cdot 1)$ has this form. There are $\ell_A^{e_A - 1}(\ell_A - 1)$ such keys $(1, \alpha)$.

ii) If $\alpha \notin \mathbb{Z}^*_{\ell_A^{e_A}}$ and the key has the form $(1, \alpha)$, then we can write the key as $(1, \ell_A \alpha')$ with $\alpha' \in \mathbb{Z}_{\ell_A^{e_A - 1}}$. So there are $\ell_A^{e_A - 1}$ such keys.

iii) If $\alpha \notin \mathbb{Z}^*_{\ell_A^{e_A}}$ and the key has the form $(\alpha, 1)$, then we can write the key as $(\ell_A \alpha', 1)$ with $\alpha' \in \mathbb{Z}_{\ell_A^{e_A - 1}}$. So there are $\ell_A^{e_A - 1}$ such keys.

We call secret keys of the forms in i)-iii) **normalised**. Since any cyclic subgroup $G \subseteq E_0[\ell_A^{e_A}]$ has a generator of the form $m_A P_A + n_A Q_A$ the normalised secret keys are in one-to-one correspondence with the the cyclic subgroups of $E[\ell_A^{e_A}]$ of order $\ell_A^{e_A}$. In [CLN16, Sampling full order 2-torsion points] only secret keys from ii) are used.

6.4 Computing power-smooth isogenies efficiently

A critical question in implementing SIDH is how to efficiently compute the isogenies of the key exchange. The isogeny ϕ_A computed by Alice in step 1 of the key exchange has a kernel of order $\approx \sqrt{p}$. For realistic choices of p it is infeasible to directly apply Vélu's formulas as they are written in Section 5.3, since that would involve iterating over roughly half of the points in the kernel. The answer is to build ϕ_A as a composition of e_A separable isogenies of degree ℓ_A. Recall that a separable isogeny of degree ℓ_A has a kernel of order ℓ_A.

Start with

$$R_0 = m_A P_A + n_A Q_A$$

Iteratively for $0 \leq i < e_A$ we do the following (we drop the subindex A from e_A and ℓ_A to simplify notation). Let E_0 be as in the key exchange description.

1. Let ϕ_i be the separable isogeny from E_i with kernel

$$\langle \ell^{e-i-1} R_i \rangle \tag{20}$$

that Vélu's formulae give. By Remark 6, $\#\langle \ell^{e-i-1} R_i \rangle = \ell$.

2. Let E_{i+1} be the codomain of ϕ_i. This is the elliptic curve

$$E_{i+1} = E_i / \langle \ell^{e-i-1} R_i \rangle$$

that Vélu's formulae give.

3. Let $R_{i+1} = \phi_i(R_i)$.

Then Alice can set

$$E_A = E_e$$
$$\phi_A = \phi_{e-1} \circ ... \circ \phi_0 \tag{21}$$

by Remark 7. Note that ϕ is separable by Proposition 5.1.7 since it is the composition of separable isogenies.

Remarks 6 and 7 provide justification for the claims made above. Remark 8 is an observation that will be useful later.

Remark 6. We claim that

$$o(R_i) = \ell^{e-i}$$

for all $0 \leq i \leq e$. We have already seen that $o(R_0) = \ell^e$. We proceed by induction. Suppose that

$$o(R_{i-1}) = \ell^{e-(i-1)} = \ell^{e-i+1}$$

for some $0 < i \leq e-1$. By definition of R_i, $o(R_i) = o(\phi_{i-1}(R_{i-1}))$. Since

$$\ker \phi_{i-1} = \langle \ell^{e-(i-1)-1} R_{i-1} \rangle = \langle \ell^{e-i} R_{i-1} \rangle$$

we have

$$\ell^{e-i} R_i = \phi_{i-1}(\ell^{e-i} R_{i-1}) = O$$

and

$$kR_i = \phi_{i-1}(kR_{i-1}) \neq O,$$

for all $1 \leq k < \ell^{e-i}$. So

$$o(R_i) = \ell^{e-i}$$

Remark 7. We need to argue why

$$\ker \phi_A = \langle R_0 \rangle = \langle m_A P_A + n_A Q_A \rangle$$

By Remark 6, $o(\phi(R_0)) = o(R_e) = 1$. So

$$\langle R_0 \rangle \subseteq \ker \phi$$

with $\#\langle R_0 \rangle = \ell^e$. The degree of the composition is the product of the degrees of the composed isogenies by Proposition 5.1.8. Hence

$$\langle R_0 \rangle = \ker \phi$$

since $\deg \phi = \ell^e$ and $\deg \phi \geq \# \ker \phi$ by Proposition 5.1.4.

Remark 8. Note that we can never have $\ker \phi_i = \ker \widehat{\phi_{i-1}}$ for $1 \leq i \leq e_A - 1$ in (21). If we did, then we would have (keep in mind that $\ker \widehat{\phi_{i-1}} = \phi_{i-1}(E_{i-1}[\ell])$)

$$\ell^{e-i-1} R_i = \ell^{e-i-1} \phi_{i-1}(R_{i-1}) = \phi_{i-1}(S)$$

for some $S \in E_{i-1}[\ell]$. Here we have used that $\ell^{e-i-1} R_i$ generates $\ker \phi_i$. Hence

$$\ell^{e-i-1} R_{i-1} - S \in \ker \phi_{i-1} \subseteq E_{i-1}[\ell],$$

but $o(\ell^{e-i-1} R_{i-1}) = \ell^2$ by Remark 6, which is a contradiction.

The approach to computing ϕ_A given in steps 1-3 above is the multiplication-based approach of [JDF11]. Another approach to computing ϕ_A in [JDF11] is the isogeny-based:

1. Compute $Q_i = \ell^i R_0$ for all $0 \leq i < e$.

2. Note that $\langle Q_{e-1} \rangle = \langle \ell^{e-1} R_0 \rangle$, so (just like in step 1 of the multiplication-based approach) we can use it to obtain ϕ_0. now apply ϕ_0 to all Q_i for $0 \leq i < e-1$. Then
$$\langle Q_{e-2} \rangle = \langle \ell^{e-2} \phi_0(R_0) \rangle = \langle \ell^{e-2} R_1 \rangle$$
so we can use it to obtain ϕ_1. now apply ϕ_1 to all Q_i for $0 \leq i < e-2$. Then
$$\langle Q_{e-3} \rangle = \langle \ell^{e-3} \phi_1(R_1) \rangle = \langle \ell^{e-3} R_2 \rangle$$
so we can use it to obtain ϕ_2. Continuing in this manner gives ϕ.

The methods we discussed in this section work in general for all isogenies that Alice and Bob need to compute in the key exchange. However, any competitive implementation of SIDH must use a third more complex but much more efficient method introduced in [DFJP14, Section 4.2.2]. The method is sort of a middle way between the two approaches described here and it takes into account the cost ratio between scalar ℓ-multiplications and ℓ-degree isogeny evaluations.

7 Computational problems on supersingular isogeny graphs

In Section 7.1 we define supersingular isogeny graphs whose vertices are the supersingular j-invariants in $\mathbb{F}_{p^2}$. Recall that any supersingular elliptic curve defined over $\overline{\mathbb{F}}_p$ had its j-invariant in $\mathbb{F}_{p^2}$ by our results in Section 5.4. In Section 7.2 we explain how the isogenies in SIDH give rise to walks of a specific form in these graphs. In Section 7.3 and 7.4 we do simulations for small examples of p to see how these walks behave. In Section 7.5 we discuss some computational problems on supersingular isogeny graphs that are related to SIDH. We give a heuristic reduction between two of them in Section 7.6. In Section 7.7 and 7.8 we discuss a cryptographic application and computational problems on supersingular isogeny graphs that predate SIDH.

7.1 Supersingular isogeny graphs

Definition 7.1.1. The **supersingular isogeny graph** $\mathcal{G}_\ell$ for a prime $p \geq 5$ and a prime $\ell \neq p$ is the directed multi-graph $G_\ell = G(V, E)$, where:

i) The vertex set $V(\mathcal{G}_\ell)$ is the set of equivalence classes of the set of all supersingular elliptic curves defined over $\overline{\mathbb{F}}_p$ under the equivalence relation of being isomorphic. The equivalence classes can be represented by j-invariants in $\mathbb{F}_{p^2}$ by Theorems 5.1.13 and 5.4.1.

ii) To get the edge set $E(\mathcal{G}_\ell)$ we pick an elliptic curve representative E for each equivalence class in $V(\mathcal{G}_\ell)$. We know by Proposition 5.4.2 that, for each equivalence class, we can pick a representative E defined over $\mathbb{F}_{p^2}$ such that

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p+1} \oplus \mathbb{Z}_{p+1}$$

Each of the $\ell+1$ many cyclic ℓ-order subgroups of E gives a separable ℓ-isogeny with domain E. For each such isogeny we add a directed edge from the equivalence class $[E]$ to the equivalence class of the codomain of the isogeny. Note that loops may occur, i.e., the equivalence class of the codomain of the isogeny can be $[E]$. One can verify that the edge set $E(\mathcal{G}_\ell)$ does not depend on the choice of representatives.

Because of the dual isogeny, if there is an edge $([E_1], [E_2]) \in E(\mathcal{G}_\ell)$, then there is always an edge $([E_2], [E_1]) \in E(\mathcal{G}_\ell)$. However, some special cases can occur where there for instance are two edges $([E_1], [E_2]) \in E(\mathcal{G}_\ell)$, but only one edge $([E_2], [E_1]) \in E(\mathcal{G}_\ell)$ [Gal12, Remark 25.3.2].

Theorem 7.1.2. *$\mathcal{G}_\ell$ is connected.*

Proof. See [Koh96, Corollary 78] and [Gal12, 25.3.17]. □

Remark 9. In Section 6.2 we assumed that p was of the form $p = \ell_A^{e_A}\ell_B^{e_B}f - 1$, but we mentioned that [JDF11] also considered primes p of the form $p = \ell_A^{e_A}\ell_B^{e_B}f + 1$. In that case we want to work with curves E with $\mathbb{F}_{p^2}$-rational group structure

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p-1} \oplus \mathbb{Z}_{p-1} = \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}f} \oplus \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}f} \tag{22}$$

in SIDH. It turns out that it is still possible to choose a representative for each vertex in $V(\mathcal{G}_{\ell_A})$ (and $V(\mathcal{G}_{\ell_B})$) with group structure as in (22). By [Was08, Theorems 4.3 and 4.4],

a curve E defined over $\mathbb{F}_{p^2}$ with such group structure exists. We may assume that E is the representative of its equivalence class (vertex) j in $\mathcal{G}_{\ell_A}$. Since all ℓ_A-order subgroups of E are $\mathbb{F}_{p^2}$-rational it follows from Remark 3 that we can choose representatives with group structure as in (22) for all neighbours of j in $\mathcal{G}_{\ell_A}$. Continuing in this manner and using that $\mathcal{G}_{\ell_A}$ is connected we can show that we can choose a representative for each vertex that has the sought group structure. We also note that one can use the sequence a_n in the proof of Theorem 5.2.2 to show that any such representative must be minimally defined over $\mathbb{F}_{p^2}$, i.e., the coefficients of the representative's Weierstrass equation can not all lie in $\mathbb{F}_p$.

Theorem 7.1.3.

$$\#V(\mathcal{G}_\ell) = \left\lfloor \frac{p}{12} \right\rfloor + \epsilon$$

with $\epsilon \in \{0, 1, 2\}$.

Proof. See [Gal12, Theorem 9.11.11]. □

It turns out that $\mathcal{G}_\ell$ is an expander graph. In fact, it has very strong expansion properties even among expander graphs. We refer to [JDF11, Section 2.1] for more details. The following theorem is one consequence.

Theorem 7.1.4. *If we walk at least*

$$\frac{\log(2\#V(\mathcal{G}_\ell))}{\log((\ell+1)/(2\sqrt{\ell}))}$$

random steps in $\mathcal{G}_\ell$*, starting from any fixed vertex* $[E] \in V(\mathcal{G}_\ell)$*, then we end up at any vertex* $[E'] \in V(\mathcal{G}_\ell)$ *with at least probability* $\frac{1}{2\#V(\mathcal{G}_\ell)}$*.*

Proof. See [DFJP14, Proposition 2.1] for references. □

As mentioned in Section 6.2 ii), Theorem 7.1.4 gives us a way to generate a random starting curve E_0 for the SIDH key exchange. However, the starting curve is held fixed in any SIDH implementation this author is aware of.

7.2 Non-backtracking walks in $\mathcal{G}_{\ell_A}$

When Alice computed her isogeny ϕ_A of the SIDH protocol as in Section 6.4 she got a composition of ℓ_A-isogenies

$$\phi_A = \phi_{e_A-1} \circ ... \circ \phi_0 \tag{23}$$

These isogenies give a walk of length e_A in $\mathcal{G}_{\ell_A}$ in the following natural way. We may assume that E_0 is the representative of the vertex $[E_0]$ in $\mathcal{G}_{\ell_A}$. Write ϕ_A as

$$\phi_A = (\phi_{e_A-1} \circ \theta_{e_A-2}^{-1}) \circ (\theta_{e_A-2} \circ \phi_{e_A-2} \circ \theta_{e_A-3}^{-1}) \circ ... \circ (\theta_1 \circ \phi_1 \circ \theta_0^{-1}) \circ (\theta_0 \circ \phi_0) \tag{24}$$

where each θ_i is the isomorphism that takes $\text{codomain}(\phi_i)$ to the representative of $[\text{codomain}(\phi_i)]$ in $\mathcal{G}_{\ell_A}$. Thus, (24) is a walk in $\mathcal{G}_{\ell_A}$. One can verify that each possible normalised secret key $sk_A = (m_A, n_A)$ gives a unique walk with respect to edges traversed. These walks are **non-backtracking** in the sense that we have

$$\ker(\theta_i \circ \phi_i \circ \theta_{i-1}^{-1}) \neq \ker((\theta_{i-1} \circ \phi_{i-1} \circ \theta_{i-2}^{-1})^\wedge)$$

for any i. We give the details of this argument in Remark 10. Here, $\psi^\wedge$ for an isogeny ψ means $\widehat{\psi}$.

Remark 10. Note that

$$(\theta_{i-2} \circ \widehat{\phi_{i-1}} \circ \theta_{i-1}^{-1}) \circ (\theta_{i-1} \circ \phi_{i-1} \circ \theta_{i-2}^{-1}) = [\ell]$$

Hence

$$\ker((\theta_{i-1} \circ \phi_{i-1} \circ \theta_{i-2}^{-1})^{\wedge}) = \theta_{i-1}(\ker \widehat{\phi_{i-1}})$$

Note also that

$$\ker(\theta_i \circ \phi_i \circ \theta_{i-1}^{-1}) = \theta_{i-1}(\ker \phi_i)$$

But by Remark 8 in Section 6.4

$$\ker \phi_i \neq \ker \widehat{\phi_{i-1}}$$

Thus,

$$\ker(\theta_i \circ \phi_i \circ \theta_{i-1}^{-1}) \neq \ker((\theta_{i-1} \circ \phi_{i-1} \circ \theta_{i-2}^{-1})^{\wedge})$$

7.3 Estimating the distribution of $j(E_A)$

There are $(\ell_A+1)\ell_A^{e_A-1}$ many non-backtracking walks of length e_A from $[E_0]$ in $\mathcal{G}_{\ell_A}$. Each is given by a normalised secret key $sk_A = (m_A, n_A)$. Hence there are as many possible values of $j(E_A)$ as there are distinct j-invariants we can end up at when we take non-backtracking walks of length e_A in $\mathcal{G}_{\ell_A}$ starting at $[E_0]$. Note that we have similar situations for the other isogenies $\phi_B : E_0 \to E_B$, $\phi_{BA} : E_B \to E_{BA}$ and $\phi_{AB} : E_A \to E_{AB}$ that are computed in the key exchange. As mentioned in [DFJP14, Section 5.1] and [GPST16, Section 6] we are not guaranteed mixing under Theorem 7.1.4 when only walking e_A steps in $\mathcal{G}_{\ell_A}$. It is therefore interesting to estimate the number of distinct j-invariants that Alice can get to by taking non-backtracking e_A-walks in $\mathcal{G}_{\ell_A}$. We will make the following heuristic estimation. For this estimation to make sense one should think of the starting curve E_0 as randomly generated among the representatives of the vertices of $V(\mathcal{G}_{\ell_A})$.

Estimation 7.3.1. We estimate that $j(E_A)$ in the SIDH key exchange is distributed as when we pick a random j from the multiset S that we construct in the following way.

1. Pick $\ell_A^{e_A-1}(\ell_A + 1)$ vertices of $\mathcal{G}_{\ell_A}$ uniformly at random with repetition. Each time a vertex j is picked, put a copy of it in S.

We will assume that the corresponding estimation holds for $j(E_B)$. To provide an indication that this is a good heuristic estimation we run simulations for three small values of p. The Sage code is available for download[4]. We start from a fixed vertex $[E]$ of $\mathcal{G}_{\ell_B}$. We then walk enough random steps to ensure mixing under Theorem 7.1.4 to obtain a starting vertex $[E_0]$. We then record the number of distinct j-invariants $j(E_A)$ that Alice can get to by taking non-backtracking e_A-walks in $\mathcal{G}_{\ell_A}$ starting at $[E_0]$. There are $(\ell_A + 1)\ell_A^{e_A-1}$ such non-backtracking walks starting at $[E_0]$, so this is the maximum possible number of distinct j-invariants. We repeat this whole procedure 500 times and use the results to get an estimate on the distribution of the number of distinct j-invariants $j(E_A)$ that Alice can get. We do this for a small example

$$p = \ell_A^{e_A}\ell_B^{e_B} f - 1 = 2^8 3^5 - 1$$

[4]https://github.com/eriktho/thesis-sage-code

in Figure 2. There we show the fraction of times (out of 500) that Alice got x distinct j-invariants, where x is the value on the horizontal axis. Figure 3 shows the fraction of times (out of 500) that we got x distinct elements j in S, where x is the value on the horizontal axis, when S is constructed as in Estimation 7.3.1. The mean of both simulations (i.e., the mean of x) is given in Alice's row in Table 1. The third column gives the theoretical expected number of distinct elements j in S, which is

$$\#V(\mathcal{G}_{\ell_A})\left(1-\left(\frac{\#V(\mathcal{G}_{\ell_A})-1}{\#V(\mathcal{G}_{\ell_A})}\right)^{(\ell_A+1)\ell_A^{e_A-1}}\right) \tag{25}$$

by linearity of expectation. We do the corresponding simulations for Bob and his isogeny $\phi_B : E_0 \to E_B$ as well, the results are given in Figures 4 and 5, and in Bob's row in Table 1. The corresponding simulation results for $p = 2^9 3^6 5 - 1$ and $p = 2^{10} 3^6 7 - 1$ are given on the pages following the next.

Remark 11. The simulations for S incorrectly use $\#V(\mathcal{G}_{\ell_A}) = \lceil p/12 \rceil + \epsilon$ when the correct number of vertices is $\#V(\mathcal{G}_{\ell_A}) = \lfloor p/12 \rfloor + \epsilon$ by Theorem 7.1.3. As the difference would not be noticeable in the simulation results we have not corrected this.

As mentioned above our walks are too short to guarantee mixing under Theorem 7.1.4. This is obvious since the number of walks ($\approx \sqrt{p}$) is strictly less than the number of vertices ($\#V(\mathcal{G}_{\ell_A}) \approx p/12$). However, it surprised this author that we see so strong simulation results even for small examples of p. The number of collisions are (on average) as few as if we were to choose the vertices $j(E_A)$ uniformly at random (with repetition) from $V(\mathcal{G}_{\ell_A})$. In some sense this is the best possible mixing one could possibly hope for given the length of the walks. We ask whether the fact that the walks are non-backtracking in combination with theoretical results for such walks in expander graphs (for example [ABLS07]) could possibly be used to explain the strong simulation results that we see? Although every simulation result this author has seen indicates that the distribution of $j(E_A)$ is very similar to that of a randomly chosen element of the set S in Estimate 7.3.1, we note that strictly speaking our simulations discussed here only compare the number of distinct elements. We leave a more thorough study of the two distributions to future work.

Table 1: Results for $p = 2^8 3^5 - 1$

	Distinct $j(E_A)$ mean	Distinct j in S mean	Distinct j in S theoretical expectation
Alice	370.69	369.92	370.16
Bob	313.70	313.88	314.12

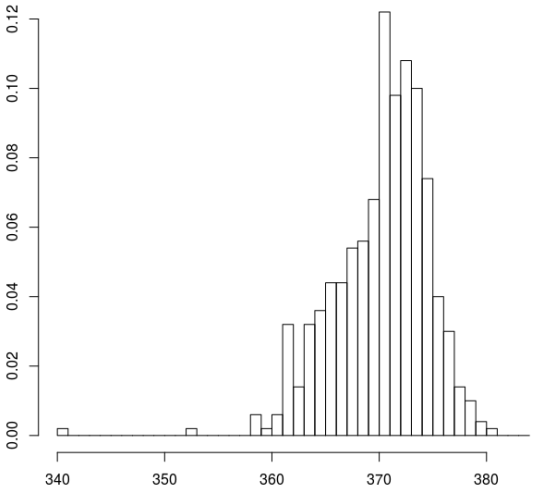


Figure 2: Number of distinct j-invariants $j(E_A)$ for Alice ($p = 2^8 3^5 - 1$).

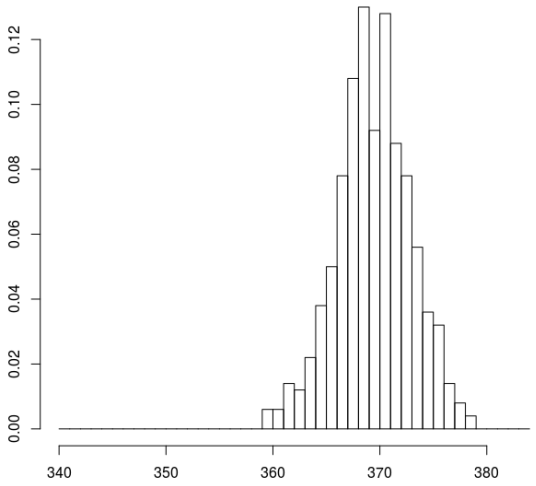


Figure 3: number of distinct elements j in S for Alice ($p = 2^8 3^5 - 1$).

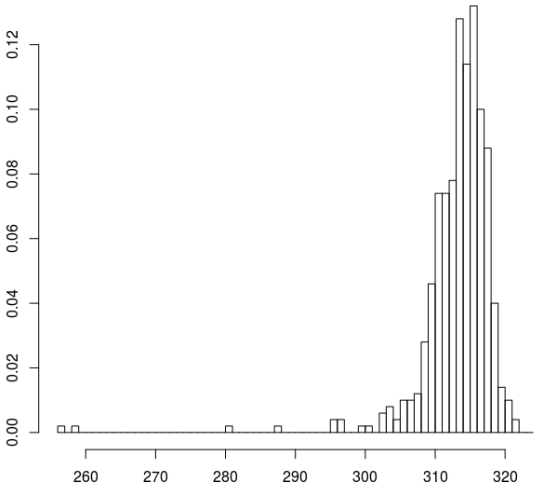


Figure 4: number of distinct j-invariants $j(E_B)$ for Bob ($p = 2^8 3^5 - 1$).

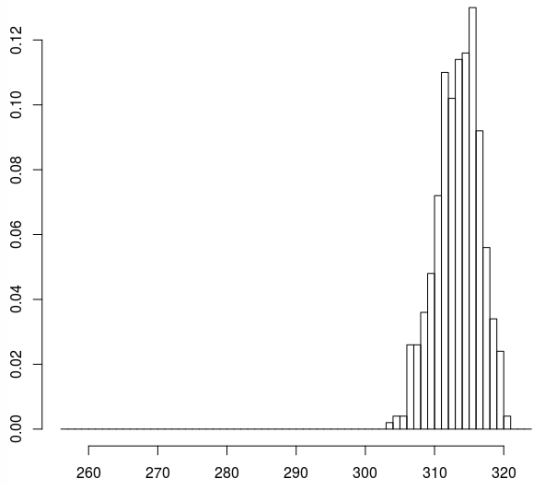


Figure 5: number of distinct elements j in S for Bob ($p = 2^8 3^5 - 1$).

Table 2: Results for $p = 2^9 3^6 5 - 1$

	Distinct $j(E_A)$ mean	Distinct j in S mean	Distinct j in S theoretical expectation
Alice	766.12	766.22	766.11
Bob	968.58	968.91	968.97

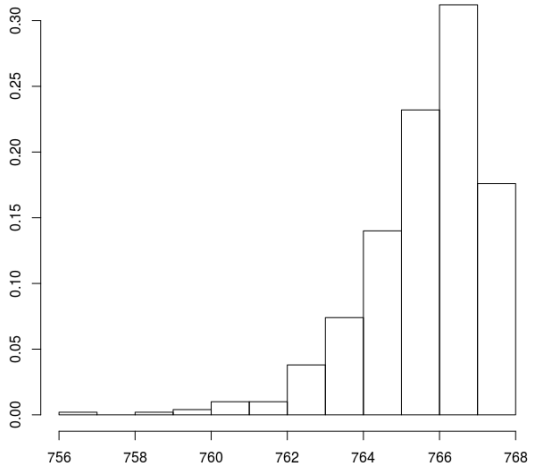


Figure 6: number of distinct j-invariants $j(E_A)$ for Alice ($p = 2^9 3^6 5 - 1$).

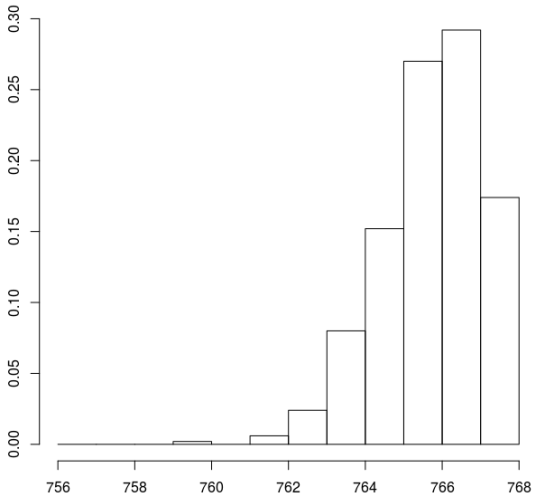


Figure 7: number of distinct elements j in S for Alice ($p = 2^9 3^6 5 - 1$).

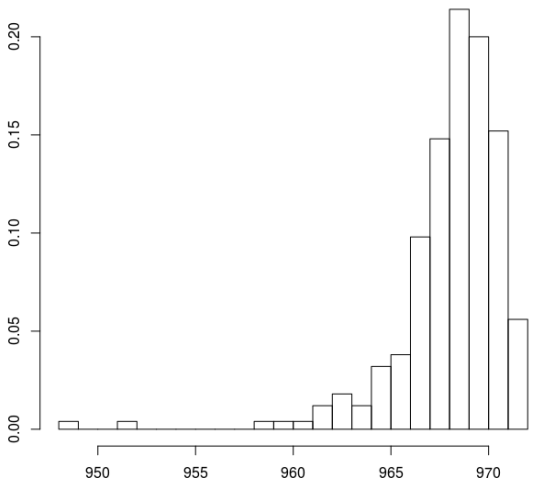


Figure 8: number of distinct j-invariants $j(E_B)$ for Bob ($p = 2^9 3^6 5 - 1$).

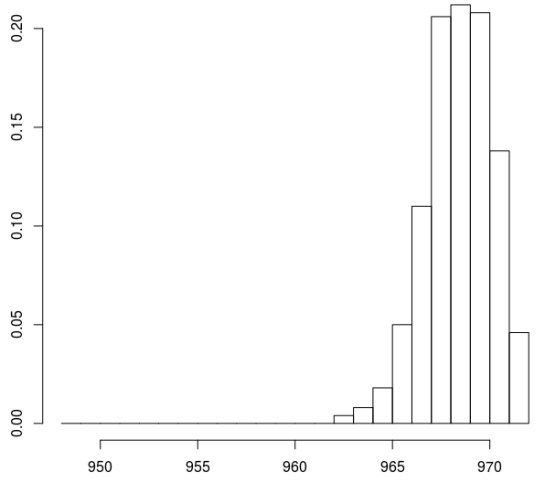


Figure 9: number of distinct elements j in S for Bob ($p = 2^9 3^6 5 - 1$).

Table 3: Results for $p = 2^{10}3^67 - 1$

	Distinct $j(E_A)$ mean	Distinct j in S mean	Distinct j in S theoretical expectation
Alice	1533.44	1533.24	1533.30
Bob	970.88	970.90	970.92

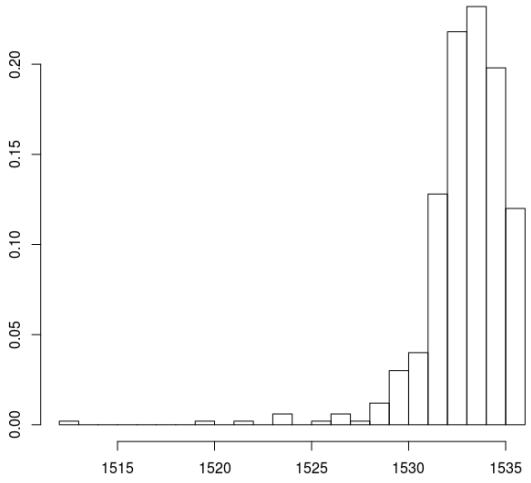


Figure 10: number of distinct j-invariants $j(E_A)$ for Alice ($p = 2^{10}3^67 - 1$).

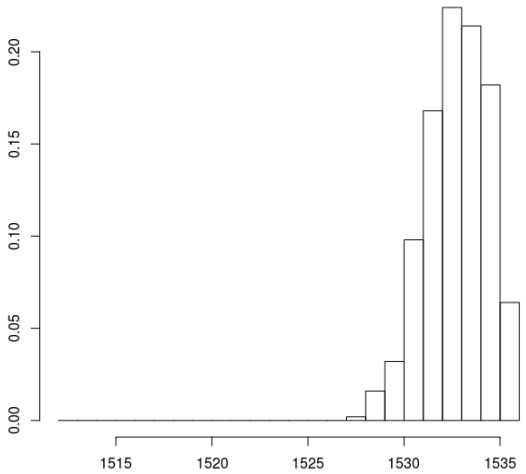


Figure 11: number of distinct elements j in S for Alice ($p = 2^{10}3^67 - 1$).

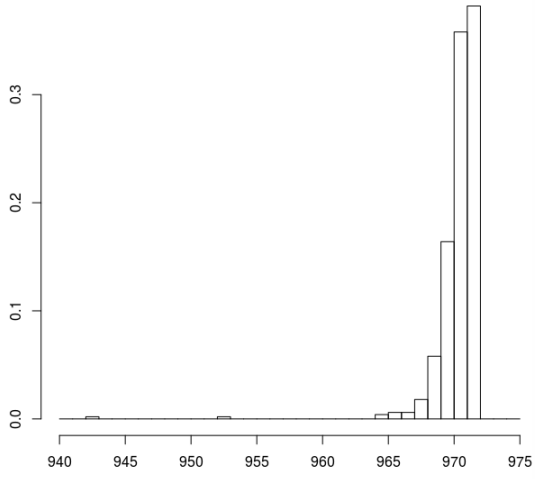


Figure 12: number of distinct j-invariants $j(E_B)$ for Bob ($p = 2^{10}3^67 - 1$).

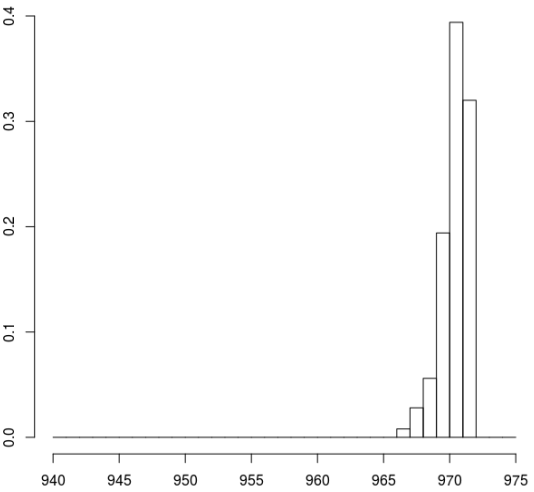


Figure 13: number of distinct elements j in S for Bob ($p = 2^{10}3^67 - 1$).

7.4 Estimating the distribution of $j(E_{AB})$

In the SIDH protocol we first walk e_A non-backtracking steps in $\mathcal{G}_{\ell_A}$ (the walk given by ϕ_A) to arrive at $[E_A]$. Then we walk e_B non-backtracking steps in $\mathcal{G}_{\ell_B}$ (the walk given by ϕ'_B) from $[E_A]$ to $[E_{AB}]$. We know that we arrive at the same vertex (j-invariant) if we instead follow the walks given by ϕ_B and ϕ'_A. Much like the question about the distribution of $j(E_A)$ that we studied in Section 7.3, to this author's knowledge there are no theoretical results on the distribution of $j(E_{AB})$ when assuming that Alice and Bob generate their secret keys honestly. It should be "good" in some sense, since $\mathcal{G}_{\ell_A}$ and $\mathcal{G}_{\ell_B}$ are strong expander graphs, but how good? By the construction of SIDH, the possible j-invariants $[E_A]$ are all connected by a non-backtracking walk of length e_A to $[E_0]$ in $\mathcal{G}_{\ell_A}$. Thus, they are close to each other in $\mathcal{G}_{\ell_A}$. But how close are they to each other in $\mathcal{G}_{\ell_B}$? To this author's knowledge there is no known useful relation between $\mathcal{G}_{\ell_A}$ and $\mathcal{G}_{\ell_B}$ that we can use to answer this question. Let us explore the reasonable heuristic assumption that they are as far apart from each other in $\mathcal{G}_{\ell_B}$ as the same number of randomly picked vertices in $\mathcal{G}_{\ell_B}$ are expected to be. Based on this and our simulation results in the previous section we make the following heuristic estimation.

Estimation 7.4.1. Suppose that Alice and Bob generate their secret keys honestly in the SIDH key exchange. We estimate that $j(E_{AB})$ is distributed as when we pick a random j' from the multiset S that we construct in the following way.

1. Pick $\ell_A^{e_A-1}(\ell_A+1)$ vertices of $\mathcal{G}_{\ell_A}$ uniformly at random with repetition.
2. For each distinct element j picked in the previous step, let z be the number of times it was picked and do:
   (a) Pick $\ell_B^{e_B-1}(\ell_B+1)$ vertices of $\mathcal{G}_{\ell_B}$ uniformly at random with repetition. Each time a vertex j' is picked, put z copies of it in S.

To provide an indication that our estimation works well we run simulations for three small values of p. The Sage code is available for download[5]. We start from a fixed vertex $[E]$ of $\mathcal{G}_{\ell_A}$. We then walk enough random steps to ensure mixing under Theorem 7.1.4 and obtain a vertex $[E_0]$. By iterating over all possible secret keys for Alice and Bob we record how often each supersingular j-invariant (with respect to p) occurs as the shared key $j(E_{AB})$. Using Theorem 7.1.3 we also know how many supersingular j-invariants it is that do not occur as $j(E_{AB})$. In Figures 14, 16 and 18 we show the fraction of j-invariants that occur x times each, where x is the value of the horizontal axis. In Figures 15, 17 and 19 we show the fraction of j-invariants that occur x times, where x is the value of the horizontal axis, in the set S when it is constructed as described above. Note that in Figures 16, 15, 18 and 19, some j-invariants appear $x = 0$ times.

The simulation results show that, even for small examples of p, the distribution of $j(E_{AB})$ and that of a randomly chosen element of the set S appears to be very similar. The results indicate that the distribution of $j(E_{AB})$ is essentially the best we could have hoped for.

[5] https://github.com/eriktho/thesis-sage-code

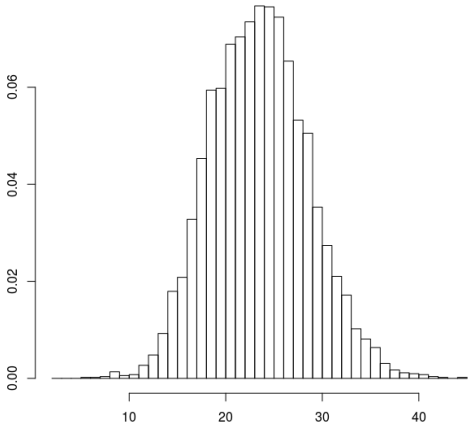


Figure 14: results for $j(E_{AB})$ for $p = 2^8 3^5 - 1$.

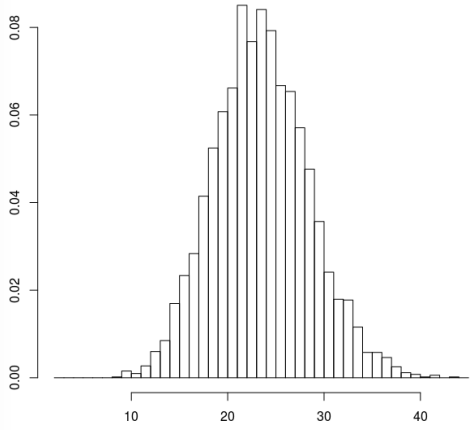


Figure 15: results for the set S for $p = 2^8 3^5 - 1$.

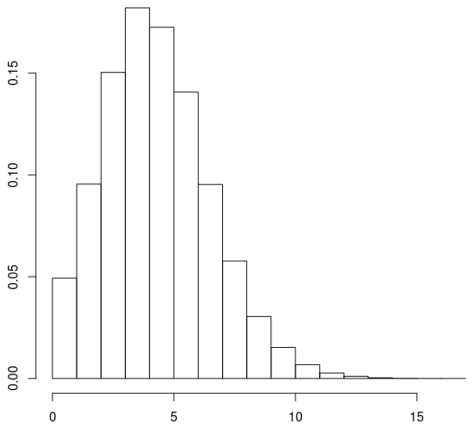


Figure 16: results for $j(E_{AB})$ for $p = 2^9 3^6 5 - 1$.

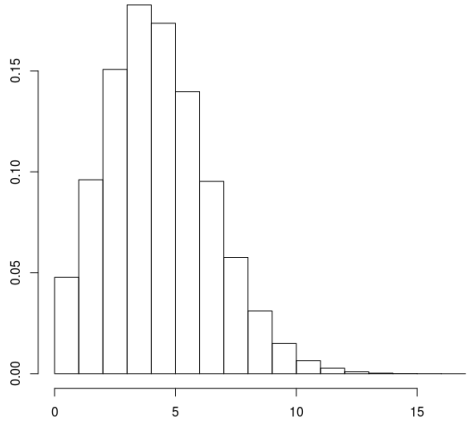


Figure 17: results for the set S for $p = 2^9 3^6 5 - 1$.

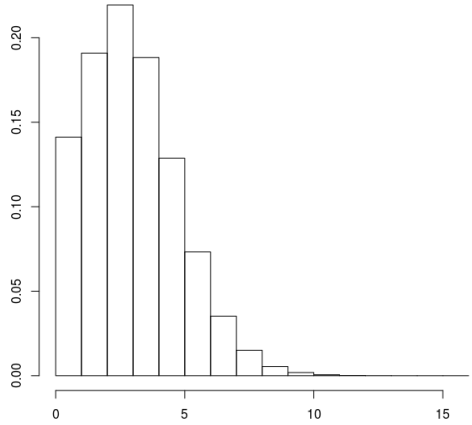


Figure 18: results for $j(E_{AB})$ for $p = 2^{10} 3^6 7 - 1$.

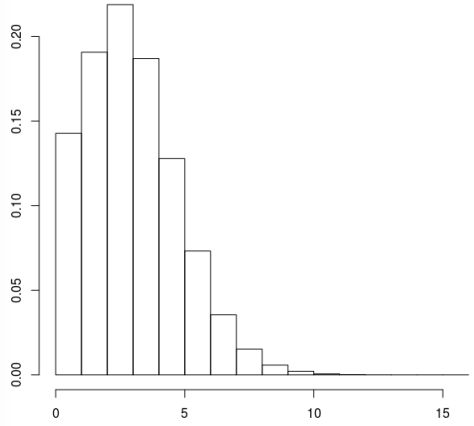


Figure 19: results for the set S for $p = 2^{10} 3^6 7 - 1$.

7.5 SIDH computational problems

As in SIDH, let $p = \ell_A^{e_A}\ell_B^{e_B}f - 1$ be a prime, E_0 a supersingular elliptic curve defined over $\mathbb{F}_{p^2}$ such that

$$E_0(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}f} \oplus \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B}f}$$

and P_A, Q_A, P_B, Q_B points such that

$$\langle P_A, Q_A\rangle = E_0(\mathbb{F}_{p^2})[\ell_A^{e_A}] = E_0[\ell_A^{e_A}]$$
$$\langle P_B, Q_B\rangle = E_0(\mathbb{F}_{p^2})[\ell_B^{e_B}] = E_0[\ell_B^{e_B}]$$

The following is [DFJP14, Problem 5.2].

Problem 7.5.1 (Computational supersingular isogeny (**CSSI**) problem)**.** Let ϕ be an $\ell_A^{e_A}$-degree isogeny from E_0 to another supersingular elliptic curve E_A defined over $\mathbb{F}_{p^2}$ with $\ker\phi = \langle m_A P_A + n_A Q_A\rangle$, where $m_A, n_A \in_R \mathbb{Z}_{\ell_A^{e_A}}$ are not both divisible by ℓ_A. Given E_1, $\phi(P_B)$ and $\phi(Q_B)$, find a generator of $\langle m_A P_A + n_A Q_A\rangle$.

Remark 12. There are several ways to retrieve the (normalised) secret key $sk_A = (m_A, n_A)$ from a generator of $\langle m_A P_A + n_A Q_A\rangle$.

- One way is to use the generator to compute $\phi(P_A)$ and $\phi(Q_A)$. As discussed in Section 6.3, we may assume that either $(m_A, n_A) = (1, \alpha)$ for some $\alpha \in \mathbb{Z}_{\ell_A^{e_A}}$, or $(m_A, n_A) = (\beta, 1)$ for some $\beta \in \ell_A\mathbb{Z}_{\ell_A^{e_A-1}}$. Suppose without loss of generality that we are in the former case, then

  $$\phi(P_A) = -\alpha\phi(Q_A) \tag{26}$$

  Since Q_A has smooth order $\ell_A^{e_A}$, $\phi(Q_A)$ also has smooth order. So (26) is a DLP instance that is easily solved with the Pohlig-Hellman algorithm of Section 3.2.1. Hence we can retrieve $(1, \alpha)$.

- Another way to find (m_A, n_A) is to note that a generator of $\langle m_A P_A + n_A Q_A\rangle$ is an element $k(m_A P_A + n_A Q_A)$ with $k \in \mathbb{Z}^*_{\ell_A^{e_A}}$, where P_A and Q_A are known. Then, assuming $(m_A, n_A) = (1, \alpha)$,

  $$e_{\ell_A^{e_A}}(Q_A, k(P_A + \alpha Q_A) = e_{\ell_A^{e_A}}(Q_A, P_A)^k$$
  $$e_{\ell_A^{e_A}}(P_A, k(P_A + \alpha Q_A) = e_{\ell_A^{e_A}}(P_A, Q_A)^{k\alpha}$$

  Since $e_{\ell_A^{e_A}}(Q_A, P_A)$ and $e_{\ell_A^{e_A}}(P_A, Q_A)$ are known, these are DLP instances in the group $\mu_{\ell_A^{e_A}}$. Here e denotes the Weil pairing of Section 2. As discussed on [Was08, p. 86], the group $\mu_{\ell_A^{e_A}}$ has smooth order $\ell_A^{e_A}$, so these DLP instances are easily solved. This means that we can retrieve $(1, \alpha)$ from the generator $k(P_A + \alpha Q_A)$.

The following is [DFJP14, Problem 5.1].

Problem 7.5.2 (Decisional Supersingular Isogeny (**DSSI**) problem)**.** Given two supersingular elliptic curves E_0 and E_1 defined over $\mathbb{F}_{p^2}$. Decide whether there is an $\ell_A^{e_A}$-degree isogeny $\phi : E_0 \to E_1$.

Note that the DSSI problem says nothing about the isogeny ϕ having a cyclic kernel. But we note that our reduction in the next section would still work if the DSSI problem was only concerned with isogenies ϕ with cyclic kernel. This would only help the reduction. The DSSI problem is discussed in [San15, Section 4.2.5] and [DFJP14, p. 17].

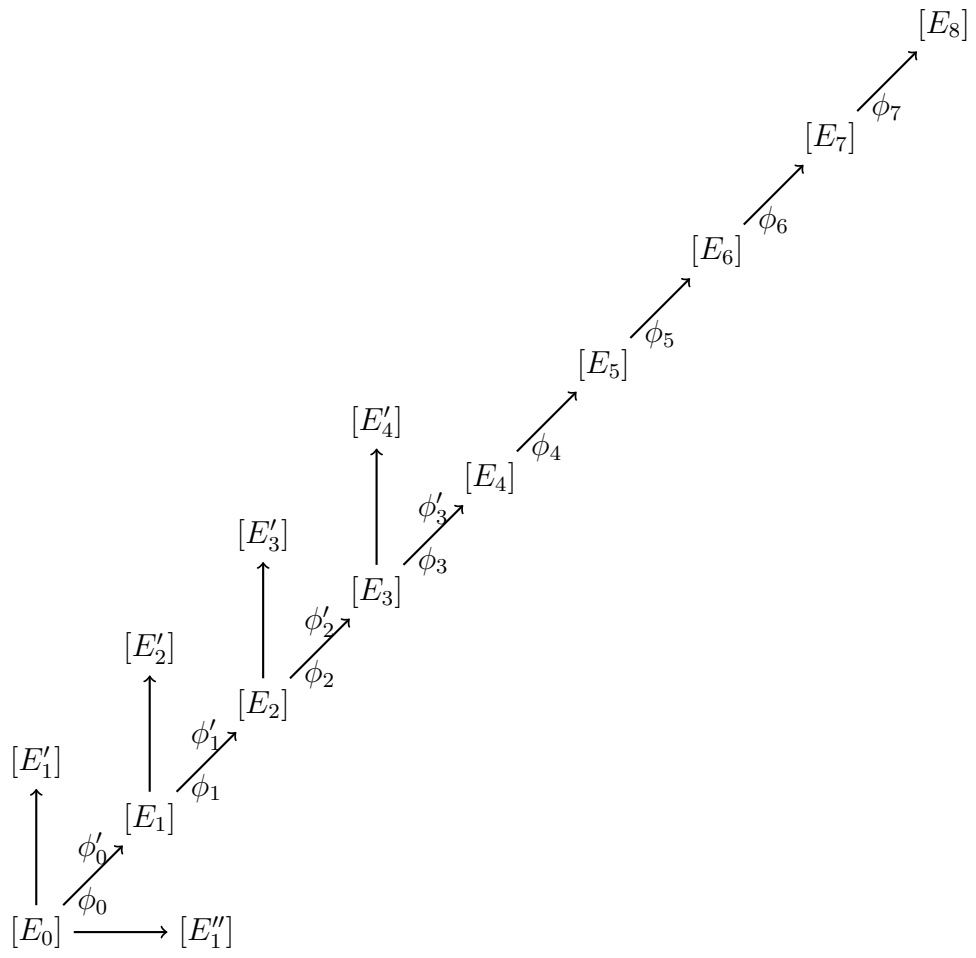
In Section 9.8.2 we state another computational problem related to SIDH that is a direct analogue of the decisional Diffie-Hellman problem.

7.6 A simple heuristic reduction from the CSSI problem to the DSSI problem

Suppose that we have an algorithm A that solves the DSSI problem with probability 1, then we can use it to heuristically solve the CSSI problem. We assume that A outputs 1 when an isogeny of correct degree exists and 0 otherwise. Let E_0, P_A and Q_A be as in Section 7.5. Let E'_A, $\phi'(P_B)$ and $\phi'(Q_B)$ be the curve and the points we are given in the CSSI problem. So our job is to find a generator of $\ker \phi' = \langle m_A P_A + n_A Q_A \rangle$.

We will use A to find an isogeny ϕ such that $\ker \phi = \ker \phi'$. Then it is easy to find a generator of $\ker \phi$. For an isogeny ψ, let $\text{domain}(\psi)$ and $\text{codomain}(\psi)$ be the domain and codomain of ψ, respectively. Just like in Section 6.4 we will compute ϕ as a composition $\phi = \phi_{e_A-1} \circ ... \circ \phi_0$, where each ϕ_i is an ℓ_A-isogeny and $\text{domain}(\phi_0) = E_0$. The codomain of ϕ_{e_A-1} will be isomorphic to E'_A. Figure 20 shows the idea behind the reduction.

Figure 20: Edges of $\mathcal{G}_{\ell_A}$ showing the idea behind the reduction. In this small example $\ell_A = 2$ and $e_A = 4$. The CSSI instance isogeny $\phi' = \phi'_3 \circ ... \circ \phi'_0$ gives a non-backtracking walk in $\mathcal{G}_{\ell_A}$ from $[E_0]$ to $[E'_A] = [E_4]$. Our reduction finds an isogeny $\phi = \phi_3 \circ ... \circ \phi_0$ that gives the same walk from $[E_0]$ to $[E_4]$. The reduction relies on the heuristic assumption that there will be no $\ell_A^{e_A}$-isogeny giving a walk from $[E'_i]$ (or $[E''_i]$) to $[E_{e_A+i}]$ for any $1 \leq i \leq e_A$. At the same time there is clearly an $\ell_A^{e_A}$-isogeny giving a walk from $[E_i]$ to $[E_{e_A+i}]$ for any $1 \leq i \leq e_A$. Since we can ask A whether such an $\ell_A^{e_A}$-isogeny exists, we can use A to find ϕ_0, ϕ_1, ..., ϕ_4, starting with ϕ_0.



7.6.1 Description of the reduction

1. Pick one of the ℓ_A+1 many ℓ_A-order subgroups $H_k \subseteq E'_A[\ell_A]$ at random. Use Vélu's formulae to compute a separable isogeny ϕ_{e_A} that has kernel equal to the picked subgroup. Let
$$E_{e_A+1} = \text{codomain}(\phi_{e_A})$$
In the example of Figure 20, E'_A would be E_4 and E_{e_A+1} would be E_5.

2. For each of the ℓ_A+1 many ℓ_A-order subgroups $G_k \subseteq E_0[\ell_A]$, let ψ_k be the separable isogeny with $\ker \psi_k = G_k$ that Vélu's formulae give. We will assume that there is exactly one k_0 such that
$$A(\text{codomain}(\psi_{k_0}), E_{e_A+1}) = 1 \tag{27}$$
We will discuss this assumption afterwards. In the example of Figure 20, $\text{codomain}(\psi_{k_0})$ would be E_1. The idea here is that ψ_{k_0} is the first edge in the walk given by ϕ' in $\mathcal{G}_{\ell_A}$. Let
$$\phi_0 = \psi_{k_0}, \quad E_1 = \text{codomain}(\phi_0)$$

3. For $1 \leq i \leq e_A - 1$:

   (a) Pick one of the ℓ_A many ℓ_A-order subgroups
   $$H_k \subseteq E_{e_A+i}[\ell_A] \setminus \ker \widehat{\phi_{e_A+i-1}} \tag{28}$$
   at random. We need to exclude $\ker \widehat{\phi_{e_A+i-1}}$ for assumption (29) to hold. Use Vélu's formulae to compute a separable isogeny ϕ_{e_A+i} that has kernel equal to the picked subgroup H_k. Let
   $$E_{e_A+i+1} = \text{codomain}(\phi_{e_A+i})$$

   (b) For each of the ℓ_A many ℓ_A-order subgroups $G_k \subseteq E_i[\ell_A] \setminus \ker \widehat{\phi_{i-1}}$, let ψ_k be the separable isogeny with $\ker \psi_k = G_k$ that Vélu's formulae give. We will assume that there is exactly one k_0 such that
   $$A(\text{codomain}(\psi_{k_0}), E_{e_A+i+1}) = 1 \tag{29}$$
   Let
   $$\phi_i = \psi_{k_0}, \quad E_{i+1} = \text{codomain}(\phi_i)$$

4. Let $\phi = \phi_{e_A-1} \circ ... \circ \phi_0$.

5. As discussed in Remark 12, it is easy to retrieve (m_A, n_A) from $\phi(P_A)$ and $\phi(Q_A)$, assuming $\ker \phi = \ker \phi'$. Then $m_A P_A + n_A Q_A$ is a solution to the given CSSI instance.

7.6.2 Analysis of the reduction

The following analysis is built on a heuristic assumption. The reader could therefore choose to skip it and instead take our simulation results in the next section as heuristic evidence that the reduction works.

Suppose we have written ϕ' as a composition of ℓ_A-isogenies

$$\phi' = \phi'_{e_A-1} \circ ... \circ \phi'_0$$

One problem that could occur in step 1 is if $\ker \phi_{e_A} = \ker \widehat{\phi'_{e_A-1}}$. It is clear from Figure 20 that we will then have

$$A(\text{codomain}(\psi_k), E_{e_A+1}) = 1$$

for more than one k in step 2. This happens with probability $\frac{1}{\ell_A+1}$ and if it happens we can just run the algorithm again. We therefore now assume that we have chosen $\phi_{e_A} \neq \ker \widehat{\phi'_{e_A-1}}$ in step 1.

We have made the assumption that there is only one k_0 such that

$$A(\text{codomain}(\psi_{k_0}), E_{e_A+i+1}) = 1$$

in steps 2 and 3a. In step 2, for the example in Figure 20 this means that there is no $\ell_A^{e_A}$-isogeny from E'_1 to E_{e_A+1} (or from E''_1 to E_{e_A+1}). As discussed in Section 7.6.3, the probability that a pair E'_{i+1} and E_{e_A+i+1} is connected by an $\ell_A^{e_A}$-isogeny is negligible in $\log p$[6] . This is the heuristic assumption mentioned in the beginning of this section. By the union bound, the probability that there is an $\ell_A^{e_A}$-isogeny between any of the $\mathrm{O}\big(\log p\big)$ many E'_{i+1} (or E''_{i+1}) and E_{e_A+i+1} tested in (27) and (29) is still negligible in $\log p$. Hence we can assume that there is exactly one k_0 such that

$$A(\text{codomain}(\psi_{k_0}), E_{e_A+i+1}) = 1$$

in steps 2 and 3a. That also means that there is only one non-backtracking e_A-walk from $[E_0]$ to $[E'_A]$ in $\mathcal{G}_{\ell_A}$, so necessarily

$$\ker \phi = \ker \phi'$$

since they both give such a walk.

7.6.3 On the probability that there is an $\ell_A^{e_A}$-isogeny from E'_{i+1} to E_{e_A+i+1}

We now explain why, heuristically, the probability that there is an $\ell_A^{e_A}$-isogeny from E'_{i+1} to E_{e_A+i+1} in the previous section is negligible in $\log p$.

We start with the following observation. Suppose ψ is an $\ell_A^{e_A}$-isogeny with domain E. By [Sut15, Corollary 6.9], ψ splits as

$$\psi = \psi_{e_A-1} \circ ... \circ \psi_0 \tag{30}$$

where each ψ_i is an ℓ_A-isogeny. If there is an $1 \leq i \leq e_A - 1$ such that

$$\ker \widehat{\psi_{i-1}} = \ker \psi_i,$$

[6]Recall that a function $f(x) : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is **negligible** if for any positive constant c there is an $x_0 \in \mathbb{R}_{\geq 0}$ such that $f(x) < \frac{1}{x^c}$ for all $x > x_0$. We sometimes simplify notation and say that a function is negligible in x if it is a negligible function of x. We call a function that is not negligible (in x) **non-negligible** (in x).

then $\psi_i = \theta \circ \widehat{\psi_{i-1}}$ for some isomorphism θ by Theorem 5.1.9. Thus,

$$\psi = \psi_{e_A-1} \circ ... \circ \psi_i \circ \psi_{i-1} \circ \psi_{i-2} \circ ... \circ \psi_0 =$$

$$\psi_{e_A-1} \circ ... \circ \theta \circ \widehat{\psi_{i-1}} \circ \psi_{i-1} \circ \psi_{i-2} \circ ... \circ \psi_0 =$$

$$\psi_{e_A-1} \circ ... \circ \theta \circ \psi_{i-2} \circ ... \circ \psi_0 \circ [\ell_A] =$$

$$\psi_{e_A-1} \circ ... \circ \psi'_{i-2} \circ ... \circ \psi_0 \circ [\ell_A] =$$

where $\psi'_{i-2} = \theta \circ \psi_{i-2}$. Continuing in this manner we can write ψ as a composition

$$\psi = \gamma_{e_A-1-2k} \circ ... \circ \gamma_0 \circ [\ell_A^k]$$

for some integer $0 \leq k \leq \lfloor e_A/2 \rfloor$ and ℓ_A-isogenies γ_i such that there is no $1 \leq i \leq e_A - 1 - 2k$ with

$$\ker \widehat{\gamma_{i-1}} = \ker \gamma_i$$

Note that in the notation of Section 7.2 this means that ψ gives a non-backtracking walk of length $e_A - 2k$ in $\mathcal{G}_{\ell_A}$ from $[E_0]$ to $[\text{codomain}(\gamma_{e_A-1-2k})]$. Thus, if there is an $\ell_A^{e_A}$-isogeny from E_0 to another supersingular curve E_A, then there is a non-backtracking walk of length $e_A - 2k$, for some non-negative integer k, in $\mathcal{G}_{\ell_A}$ from $[E_0]$ to $[E_A]$. It is easy to see that the converse holds as well.

When e_A is odd there are

$$\sum_{k=0}^{\lfloor e_A/2 \rfloor} (\ell_A + 1)\ell_A^{2k} = \frac{(\ell_A + 1)(\ell_A^{2(\lfloor e_A/2 \rfloor + 1)} - 1)}{\ell_A^2 - 1}$$

many non-backtracking walks of length j from $[E_0]$ in $\mathcal{G}_{\ell_A}$, for odd $1 \leq j \leq e_A$. Thus, for a randomly chosen vertex v of $\mathcal{G}_{\ell_A}$ we can upper bound the probability that there is such a non-backtracking walk from $[E_0]$ to v in $\mathcal{G}_{\ell_A}$ by

$$\frac{(\ell_A + 1)(\ell_A^{2(\lfloor e_A/2 \rfloor + 1)} - 1)}{(\ell_A^2 - 1)\#V(\mathcal{G}_{\ell_A})} \tag{31}$$

Since $\#V(\mathcal{G}_{\ell_A}) \approx p/12 \approx \ell_A^{e_A}\ell_B^{e_B} f/12$ by Theorem 7.1.3, this probability is negligible in $\log p$. Similarly, one sees that the corresponding upper bound is negligible in $\log p$ when e_A is even. Our assumption is that (31) also works to heuristically approximate the probability that the curves E'_i and E_{e_A+i+1} in our reduction are connected by an $\ell_A^{e_A}$-isogeny. This is in line with our related simulation results in Section 7.3 and is similar to the assumption in [GPST16, p. 12 footnote 1]. We do not include simulation results that justify this assumption since the simulation results of the next section show that the whole reduction works.

7.6.4 Simulation results providing heuristic proof that the reduction works

We implement our reduction and run simulations for small examples. The Sage code is available for download[7]. We do the following for fixed values of $p = \ell_A^{e_A}\ell_B^{e_B} f - 1$. Pick a fixed supersingular curve E such that

$$E(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B} f} \oplus \mathbb{Z}_{\ell_A^{e_A}\ell_B^{e_B} f}$$

[7] https://github.com/eriktho/thesis-sage-code

1. Walk enough steps in $\mathcal{G}_{\ell_B}$ from E to ensure mixing under Theorem 7.1.4. Let E_0 be the supersingular elliptic curve we end up at. Let A be an algorithm that solves the DSSI problem by brute force.

2. Generate an instance of the CSSI problem for E_0 and attempt to solve it using A as described in our reduction.

We repeat steps 1-2 100 times and count the total number of failures. As p grows we expect that all the failures we see are due to accidentally choosing

$$\ker \phi_{e_A} = \ker \widehat{\phi'_{e_A-1}}$$

in step 1 of the reduction, as discussed in Section 7.6.2. This happens with probability $\frac{1}{\ell_A+1}$. So we expect the number of failures to behave like a random variable with distribution $Bin(100, \frac{1}{\ell_A+1})$ as p grows. This agrees well with our results in the top three rows of Table 4. The bottom three rows are the corresponding results when we work with ℓ_B-isogenies in $\mathcal{G}_{\ell_B}$ instead. Then we expect that the number of failures behave like a random variable with distribution $Bin(100, \frac{1}{\ell_B+1})$ as p grows. Note that we could achieve a failure percentage that tends to 0 as p grows by choosing another kernel for ϕ_{e_A}, instead of outputting *fail*, when we accidentally choose

$$\ker \phi_{e_A} = \ker \widehat{\phi'_{e_A-1}}$$

We have omitted this improvement for a simpler reduction.

Table 4: Simulation results for three different choices of $p = \ell_A^{e_A}\ell_B^{e_B} f - 1$.

	Number of failures
$\mathcal{G}_{\ell_A}$, $p = 2^8 3^5 - 1$, $\ell_A = 2$	61
$\mathcal{G}_{\ell_A}$, $p = 2^9 3^6 5 - 1$, $\ell_A = 2$	35
$\mathcal{G}_{\ell_A}$, $p = 2^{10} 3^6 7 - 1$, $\ell_A = 2$	27
$\mathcal{G}_{\ell_B}$, $p = 2^8 3^5 - 1$, $\ell_B = 3$	59
$\mathcal{G}_{\ell_B}$, $p = 2^9 3^6 5 - 1$, $\ell_B = 3$	33
$\mathcal{G}_{\ell_B}$, $p = 2^{10} 3^6 7 - 1$, $\ell_B = 3$	23

7.6.5 Relaxing our assumption on the DSSI-algorithm A

It is not strictly necessary that A always answers correctly. What is needed for our algorithm to work with non-negligible probability in $\log p$ is that A answers all of our $\mathrm{O}\big(\log p\big)$ queries correctly with non-negligible probability in $\log p$.

7.7 A hash function on $\mathcal{G}_2$

In the article [CLG09] the idea of of using non-backtracking walks on $\mathcal{G}_2$ as a hash function was introduced. This hash function is sometimes referred to as the Pizer hash function in honour of Arnold K. Pizer who studied the expansion properties of supersingular isogeny graphs [CLG09, Section 1]. The idea is to fix a prime p (in general not of the same form as in SIDH) and a supersingular curve

$$E : y^2 = f(x)$$

defined over $\mathbb{F}_{p^2}$. We then look at the supersingular isogeny graph $\mathcal{G}_2$ and take $[E]$ as a starting vertex in $\mathcal{G}_2$. The input to the hash function is a bit string x as usual and we let the bits of x determine a non-backtracking walk in $\mathcal{G}_2$ starting at $[E]$ as follows:

1. Start by walking one step along some predetermined edge $([E], [E_0])$.
2. For each bit x_i in the input $x = x_0...x_n$:
   (a) Order the outgoing edges at $[E_i]$ by some canonical ordering $\mathcal{O}$ and discard the edge that is the dual of the previous edge. Among the two remaining edges (recall that each vertex in $\mathcal{G}_2$ has three outgoing edges by Definition 7.1.1) choose the edge $([E_i], [E_{i+1}])$ that corresponds to the value of x_i under $\mathcal{O}$.
3. Let the j-invariant of the vertex $[E_{n+1}]$ be the output of the hash function.

To find the outgoing edges at $[E_i]$ at step 2(a) above, the method of [CLG09, Section 5] factors the cubic $f(x)$ defining the curve $E_i : y^2 = f(x)$. This requires roughly $2 \log p$ field multiplications and is also the dominating operation. Thus, the total cost is roughly $2 \log p$ field multiplications per bit of input [CLG09, Section 5]. According to [Pet09], a C-implementation of the hash function for a fixed p with bit size 256 can process 13kb of input data per second. This is a low throughput compared to other constructions considered by Petit in [Pet09, Section 7.3]. [Pet09, Section 7.4] concludes that the hash function will most likely only ever be of theoretical interest.

The problem of finding a collision for the Pizer hash function is analysed in, for example, [Pet09, Problem 7.1].

Problem 7.7.1. Given two supersingular curves E_0 and E_{n+1}, find two distinct isogenies $\phi : E_0 \to E_{n+1}$ and $\phi' : E_0 \to E_{n+1}$ with cyclic kernels of degree 2^e and $2^{e'}$ for positive integers e and e'.

7.8 Algorithms for solving isogeny problems

In this section we survey some algorithms for solving the problem of finding isogenies between given supersingular elliptic curves.

In the context of SIDH, [GPST16, Algorithm 2] gives a (classical) algorithm that recovers Alice's secret isogeny ϕ_A (from her public key) in time polynomial in $\log p$ [GPST16, Theorem 4.2]. The problem is that the algorithm requires explicit descriptions of $\mathrm{End}(E_0)$ and $\mathrm{End}(E_A)$, where E_0 and E_A are the curves from the SIDH protocol. Computing the endomorphism ring of a supersingular elliptic curve is an exponentially hard problem. As explained in [GPST16, p. 3]: building on work by Kohel [Koh96], Galbraith [Gal99] gives an algorithm with complexity $\tilde{\mathrm{O}}(p^{1/2})$.

Delfs and Galbraith [DG16] give an algorithm to compute an isogeny between two supersingular curves E and E' in time $\mathrm{O}\big(p^{1/4}\big)$. The catch is that both curves must be defined over $\mathbb{F}_p$. In general, the supersingular curves we work with are defined over $\mathbb{F}_{p^2}$ and only at most $\tilde{\mathrm{O}}(\sqrt{p})$[8] of them are defined over $\mathbb{F}_p$ [DG16, p. 2]. In the case we want to compute an isogeny between E and E' when the curves are not both defined over $\mathbb{F}_p$, [DG16] suggest first walking randomly to find curves $\tilde{E}$ and $\tilde{E}'$, defined over $\mathbb{F}_p$, that are connected through isogenies to E and E', respectively. Then we use the $\mathbb{F}_p$-algorithm to

[8] Recall that a function $f(n)$ is $\tilde{\mathrm{O}}(g(n))$ if it is $\mathrm{O}(g(n) \log^k(g(n)))$ for some positive constant k.

find an isogeny between $\tilde{E}$ and $\tilde{E}'$. In the more general case of the curves being defined over $\mathbb{F}_{p^2}$, the running time of the algorithm is $\mathrm{O}\big(p^{1/2}\big)$. The problem with both cases is that the final isogeny we get is not necessarily of degree ℓ^e for some prime ℓ and integer e. Instead the factorisation of the degree may contain powers of several distinct prime factors. There are no known efficient methods for transforming such isogenies into ones of degree ℓ^e. Regardless, we already have a classical attack against SIDH that runs in time $\mathrm{O}\big(p^{1/4}\big)$, see Section 8.1.

[BJS14] uses methods from [DG16] and [CJS14] to give a quantum algorithm for constructing isogenies between supersingular curves defined over $\mathbb{F}_{p^2}$. They do it in a way that is similar to how [DG16] classically constructed isogenies in the general case when the two supersingular curves were defined over $\mathbb{F}_{p^2}$. Let E and E' be the two curves defined over $\mathbb{F}_{p^2}$ that we wish to find an isogeny between. Like [DG16], [BJS14] first use random walks to find $\tilde{E}$ and $\tilde{E}'$, defined over $\mathbb{F}_p$, that are connected through isogenies to E and E', respectively. But now on a quantum computer, [BJS14] can do this in time $\tilde{\mathrm{O}}(p^{1/4})$ with Grover's algorithm, instead of $\tilde{\mathrm{O}}(p^{1/2})$ as in [DG16]. The logarithmic terms that are hidden by the $\sim$ are from isogeny computations. Then [BJS14] use observations from [DG16] to apply the quantum algorithm from [CJS14] to compute an isogeny between $\tilde{E}$ and $\tilde{E}'$ in subexponential time. The fact that both $\tilde{E}$ and $\tilde{E}'$ are defined over $\mathbb{F}_p$ is crucial here. Their full algorithm solves the general problem of finding an isogeny between E and E' in time $\tilde{\mathrm{O}}(p^{1/4})$. As discussed in [BJS14, Section 6] and [San15, p. 55], this attack is not relevant to SIDH as it produces an isogeny of arbitrary degree. Also, the current best quantum attack against SIDH runs in time $\tilde{\mathrm{O}}(p^{1/6})$, see Section 8.1. Still, [BJS14, Section 6] advises implementers of SIDH to choose a starting curve E_0 that is not defined over $\mathbb{F}_p$. [CLN16, Remark 2] discuss why they have chosen a starting curve E_0 defined over $\mathbb{F}_p$ anyway.

Informally speaking, it appears to be hard for a general isogeny algorithm to do a better job attacking SIDH than the claw algorithm. The reason is that the complexity of the general algorithms we surveyed above runs in exponential time in the bit size of p. Meanwhile, the complexity of the claw algorithm depends mainly on $\ell_A^{e_A}$ which has half the bit size compared to that of p. This shows that the claw algorithm has a significant head start compared to other, more general, algorithms when attacking SIDH.

8 Attacks on SIDH

The best known classical and quantum attacks against SIDH are algorithms for solving the claw problem, see [DFJP14, Section 5.1] and [CLN16, SIDH history and security]. We discuss the claw problem in Section 8.1. In Section 8.2 we discuss an attack that Bob can carry out against Alice when she is using a static key. Passive attacks are discussed in Section 8.3. These are the attacks that anyone with access to the public information of the key exchange can carry out. Note that the claw attack is a passive attack.

8.1 The claw-problem

In the claw problem we are given black box-access to two functions $f : A \to C$ and $g : B \to C$ with $\#A = N \leq M = \#B$. We are then asked to find a *claw*, that is, a pair (a, b) such that $f(a) = g(b)$, or determine that no such pair exists.

Breaking SIDH can be approached as a claw problem in a natural way. An adversary takes E_A from Alice's public key $pk_A = (E_A, \phi_A(P_B), \phi_B(Q_B))$ and looks at all the non-backtracking walks of length $\frac{e_A}{2}$ going out from $[E_0]$ and $[E_A]$ in $\mathcal{G}_{\ell_A}$. Note that the attacker could just as well choose to attack Bob's public key. This is the reason we choose $\ell_A^{e_A} \approx \ell_B^{e_B}$ in the SIDH protocol. By our heuristic estimation in Section 7.3, the walk given by ϕ_A is the only non-backtracking e_A-walk from $[E_0]$ to $[E_A]$ in $\mathcal{G}_{\ell_A}$ with very high probability. This walk gives a claw (a, b), where $f(a)$ is taken to be the vertex our walk a from $[E_0]$ ends at and $g(b)$ is taken to be the vertex our walk b from $[E_A]$ ends at. Then

$$N = M \approx \ell_A^{e_A/2} \approx p^{1/4}$$

in the claw problem above. Note that we can change the cardinalities N and M. We could, for instance, choose to look at walks of length $\frac{e_A}{3}$ from $[E_0]$ and walks of length $\frac{2e_A}{3}$ from $[E_A]$. Regardless of how we choose N and M, the size of NM is constantly $\approx \ell_A^{e_A}$.

As described in [DFJP14, Section 5.1], the natural classical approach to solving a claw problem is to build a hash table of $(f(x), x)$ for all $x \in A$ and then look through all $y \in B$ checking for a match in the table. This gives a $\mathrm{O}\big(N\big)$ space and $\mathrm{O}\big(N + M\big)$ time classical algorithm. When stating the classical security of SIDH it is often assumed that $N = M$ and the security is stated as $\log M \approx \frac{1}{4} \log p$ bits[9]. Note that this is somewhat of an underestimate as the attack involves building a hash table with $p^{1/4}$ entries, something which is not necessary when for example finding a $\frac{1}{4} \log p$ bit symmetric encryption key by brute force.

Assuming $N = M$, a naive parallelisation of the classical algorithm is to divide both A and B in K many disjoint and equally sized parts $(A_1, ..., A_K)$ and $(B_1, ..., B_K)$, respectively. We can then run all pairs (A_i, B_j), with $i \leq j$, in parallel as separate claw problem instances. However, this is not an efficient parallelisation as it requires $\binom{K}{2}$ computers and only gives a factor K speed-up in time. A more efficient parallelisation would be to let K computers build and query the hash table in parallel. This approach has time complexity $\mathrm{O}\big(\frac{N+M}{K}\big)$.

With a quantum computer, one can do better. The query complexity of a quantum claw algorithm is the total number of queries it makes to f and g together. By Tani [Tan09] the problem can be solved in $\mathrm{O}\big((NM)^{1/3}\big)$ queries when $M < N^2$, and $\mathrm{O}\big(M^{1/2}\big)$

[9] As defined in [CCJ+16, p. 6]: *"an algorithm is said to have n bits of [classical] security if the difficulty of attacking it with a classical computer is comparable to the time and resources required to brute-force search for a n-bit cryptographic key".*

queries when $M \geq N^2$. These quantum query complexities are optimal when f and g are regarded as black boxes [Tan09, p. 2]. Thus, the quantum complexity when attacking SIDH is $\mathrm{O}\big(\ell_A^{e_A/3}\big) = \mathrm{O}\big(p^{1/6}\big)$ queries. For the rest of this thesis we let λ be the SIDH quantum security in bits[10]. That is,

$$\lambda = \frac{1}{6}\log p$$

Note that by the discussion above the classical security in bits is 1.5λ. The naive parallelisation described above applies to the quantum setting as well. It is even less efficient here as parallelising over $\binom{K}{2}$ quantum computers when $N = M$ gives a $\mathrm{O}\big((N^2/K^2)^{1/3}\big)$ query complexity per computer, which is only a factor $K^{2/3}$ improvement. This author has not been able to find a more efficient parallelisation in relevant literature [Tan09][Zha05][BHT98] [BDH+01][MSS07]. We will discuss this further in Section 9.12.

Remark 13. In the claw problem it is assumed that the domains of f and g are fixed. As noted above, when attacking SIDH we can dynamically change these at any time. There does not appear to be any way to use this to do better than the general claw problem complexities.

8.2 An attack on static keys

[GPST16] gives a powerful attack on SIDH when Alice is using a static secret key. The attack is based on Bob's ability to send a corrupt public key pk_B to Alice. In an initial key exchange Bob generates a public key in the standard honest way. He then sends a corrupt (but related) public key to Alice in a second key exchange. This enables Bob to extract one bit of information from Alice's secret key by checking whether Alice arrives at the same shared key in both key exchanges. Note that it is a very reasonable assumption that Bob can check this since he and Alice will likely use the shared key as for example a symmetric encryption key for future communication. It turns out that this attack can be generalised and done in a systematic way to very efficiently retrieve Alice's secret key bit by bit. When for instance $\ell_A = 2$, Bob only needs to engage in about e_A many key exchanges with Alice. We refer to [GPST16, Secion 3] for more details. We stress that standard SIDH key exchange with a static key is completely broken because of this attack. We discuss possible counter-measures in Sections 9.8.1 and 9.8.4.

8.3 The torsion points in SIDH

As discussed in [DFJP14, p. 18] the torsion points $\phi_A(P_B)$ and $\phi_A(Q_B)$ in Alice's public key do not seem to give any information about the kernel $\langle m_A P_A + n_A Q_A \rangle$ of ϕ_A. There does not seem to be any efficient way to use them to say something about for instance $\phi_A(P_A)$ and $\phi_A(Q_A)$. However, in [Pet17] Petit studies (passive) attacks using the torsion points in settings related to that in SIDH. The author describes heuristic polynomial time algorithms for finding the kernel of an N_1-degree isogeny $\phi : E_0 \to E_1$ when the image of ϕ on the N_2-torsion subgroup of E_0 is revealed, if also: The N_2-torsion subgroup, which has

[10] Here we simply follow the rest of the literature on SIDH and let the quantum security in bits of SIDH be the logarithm of the number of queries in the quantum claw attack. See for example [CCJ+16, Section 4] and Section 4.A.5 in http://csrc.nist.gov/groups/ST/post-quantum-crypto/evaluation-criteria.html for a discussion about quantum security strength in bits.

trivial intersection with the N_1-torsion subgroup, has significantly larger order than N_1. In SIDH we have $N_1 = \ell_A^{e_A} \approx \ell_B^{e_B} = N_2$. Note that the attacks have other requirements as well, we refer to [Pet17] for details. None of his attacks are currently applicable to SIDH, but as they are helped by when the starting curve E_0 has certain properties (see [Pet17, p. 12]), the author suggests avoiding such starting curves as a precaution. The starting curve of [CLN16] falls into this category of curves [Pet17, p. 2]. See [Pet17, Section 5] for a discussion about choosing a starting curve E_0.

In the previous section we mentioned an adaptive attack that uses the torsion points in Bob's public key in a crucial way. [Pet17] is an interesting paper as it describes the first passive attacks that use the torsion points in a setting similar to that in SIDH. It is very interesting for future work to study if any of the attacks can possibly be carried over to the setting of SIDH. Along with the short walks in the isogeny graphs, discussed in Section 7.3, the torsion points are what makes the isogeny problems that SIDH give rise to special compared to the more general isogeny problems that we discussed in Section 7.8.

9 SIDH implementation status

In this section we give an overview of the current implementation status of SIDH. There is a lot of research currently being done on SIDH. Most of the results that we will discuss are from 2016 and 2017.

9.1 Optimising performance

Several authors have implemented SIDH [DFJP14] [AFJ14] [KAJMK16a] [CLN16]. In their original paper, [DFJP14] discuss several components of an efficient SIDH implementation:

i) In Section 6.4 we mentioned that they discuss a "balance" between scalar ℓ-multiplications and ℓ-degree isogeny evaluations in implementing ℓ^e-degree isogeny computations [DFJP14, Section 4.2.2]. The balance depends on the cost ratio between the two operations. As noted in [AFJ14, p. 6], an optimal strategy with regards to this ratio can be computed once and for all for a specific platform and then reused for all key exchanges. Thus, for example on a limited device, computing the optimal strategy is not a problem and the computation does not have to be performed on the device. The actual computation of the optimal strategy can be done efficiently with dynamic programming [CLN16, Strategies for isogeny computation and evaluation].

ii) [DFJP14, Algorithm 1] computes a generator R of the kernel of Alice and Bob's secret isogenies efficiently.

iii) [DFJP14] suggests working on Montgomery curves when performing isogeny computations and evaluations. See [CLN16, Section 3] for more details.

Other authors discuss optimising finite field arithmetic for SIDH [CLN16, Section 5] [KAJMK16b] [BF16]. The article [CLN16], by Costello, Longa and Naehrig, improves the performance of SIDH significantly. Their implementation is 2.9 times faster than the previously fastest implementation ([AFJ14]). The perhaps most important improvement is working not only with projective coordinates, but working with the Montgomery curve coefficients projectively (see [CLN16, Section 3] for details). This allows them to reduce the number of field inversions that each party needs to perform to three. Their implementation is also constant-time and therefore protected against timing attacks [Koc95]. The implementation is tailor-made for a prime p offering (about) 128 bits of quantum security.

9.2 Implementations in hardware

The first hardware implementation of SIDH was [KAKJ16]. A faster implementation is given in [KAMK16]. It implements the projective isogeny formulas of [CLN16] on Field programmable gate array (FPGA) hardware. Like the one in [CLN16], the implementation of [KAMK16] runs in constant time. At 128 bits of quantum security the implementation of [CLN16] runs Alice's computations in 28.8 ms (i7 3.4 GHz), the corresponding computations in [KAMK16] take 20.1 ms (Virtex-7 FPGA) [KAMK16, Table 6]. The corresponding timings for Bob's computations are 34.1 ms (i7 3.4 GHz) and 22.4 ms (Virtex-7 FPGA), respectively.

9.3 Perfect forward secrecy

Ephemeral SIDH offers perfect forward secrecy. Generating new secret keys for Alice and Bob just amounts to generating new pairs (m_A, n_A) and (m_B, n_B), respectively. Here we assume that we let the curve E_0 and the points P_A, Q_A, P_B and Q_B of the SIDH protocol stay the same when we generate fresh pairs of secret keys. This is how it is done in state-of-the-art implementations such as [CLN16]. This is similar to how the elliptic curve and generator point remains fixed in standard implementations of ephemeral ECDH.

9.4 Key compression

Recall from Section 8 that $\log p = 6\lambda$, where p is the prime of SIDH key exchange and λ is the quantum security in bits.

Alice's public key in SIDH is

$$(E_A, \phi_A(P_B), \phi_A(Q_B)) \tag{32}$$

Note that what we will discuss here holds equally well for Bob's key. The curve E_A is given by a short Weierstrass equation

$$y^2 = x^3 + Ax + B, \quad A, B \in \mathbb{F}_{p^2} \tag{33}$$

As usual, the point $\phi_A(P_B) = (x, y) \in E_A(\mathbb{F}_{p^2})$ can be transmitted as its x-coordinate along with one bit signalling which square root of $x^3 + Ax + B$ that is equal to y. The same holds for $\phi_A(Q_B)$. In a naive implementation, (32) is therefore an element of

$$\mathbb{F}_{p^2}^2 \times (\mathbb{F}_{p^2} \times \mathbb{Z}_2) \times (\mathbb{F}_{p^2} \times \mathbb{Z}_2)$$

which has size $8 \log p = 48\lambda$. However, the state-of-the-art implementation [CLN16] does not send

$$(A, B, \phi_A(P_B), \phi_A(Q_B)),$$

where A and B are from (33). Instead

$$(x_{\phi_A(P_B)}, x_{\phi_A(Q_B)}, x_{\phi_A(P_B)-\phi_A(Q_B)}) \in \mathbb{F}_{p^2}^3 \tag{34}$$

are sent. These are the x-coordinates of three points on a Montgomery curve [CLN16, Remark 4]. The curve can be recovered from the three points. This means that an (uncompressed) SIDH key only has size $6 \log p = 6 \cdot 6\lambda = 36\lambda$ bits.

In [AJK+16] it is described how the key size can be reduced further. First, instead of sending E_A, Alice sends $j(E_A) \in \mathbb{F}_{p^2}$. From $j(E_A)$, Bob constructs a curve $E'_A \cong E_A$ in a deterministic canonical way. We refer to [AJK+16, Section 3.1, Solutions 1] for details. Obviously, Alice also needs to update the points $\phi_A(P_B)$ and $\phi_A(Q_B)$, so that they are on E'_A and not E_A. To do this Alice computes an isomorphism θ defined over $\mathbb{F}_{p^2}$ from E_A to E'_A, see [AJK+16, Section 3.1, Solutions 2].

Second, Alice does not send $\theta(\phi_A(P_B))$ and $\theta(\phi_A(P_B))$. Instead she notes that

$$\theta(\phi_A(P_B)), \theta(\phi_A(Q_B)) \in E'_A[\ell_B^{e_B}]$$

Then she, in a deterministic canonical way, finds R_1 and R_2 such that

$$\langle R_1, R_2 \rangle = E'_A[\ell_B^{e_B}]$$

Using the Weil pairing it is easy to find $\alpha_1, \beta_1, \alpha_2, \beta_2 \in \mathbb{Z}_{\ell_A^{e_A}}$ such that

$$\alpha_1 R_1 + \beta_1 R_2 = \theta(\phi_A(P_B)), \quad \alpha_2 R_1 + \beta_2 R_2 = \theta(\phi_A(Q_B))$$

Hence Alice can send $\alpha_1, \beta_1, \alpha_2$ and β_2 to Bob instead of $\theta(\phi_A(P_B))$ and $\theta(\phi_A(Q_B))$. Bob finds the same R_1 and R_2 on his end and uses $\alpha_1, \beta_1, \alpha_2$ and β_2 to get $\theta(\phi_A(P_B))$ and $\theta(\phi_A(Q_B))$.

Since $j(E_A) \in \mathbb{F}_{p^2}$ and $\log(\ell_A^{e_A}) \approx \frac{1}{2}\log p$, Alice's public key is

$$2\log p + 4 \cdot \frac{1}{2}\log p = 4\log p$$

under this compression. The compression only operates on data from Alice's original public key and hence it does not effect the security of the key exchange. For the same reason an implementation of the compression does not need to run in constant time [CJL+16b, p. 4].

The authors of [CJL+16b] optimise the performance of the key compression described above significantly. The authors also introduce another tweak to the compressed key size. One can show that $\theta(\phi_A(P_B)) = \alpha_1 R_1 + \beta_1 R_2$ has order $\ell_B^{e_B}$. Thus, α_1 or β_1 is invertible modulo $\ell_B^{e_B}$. If α_1 is invertible, then Alice sends a bit flag set to 1 and

$$\alpha_1^{-1}\beta_1, \alpha_1^{-1}\alpha_2, \alpha_1^{-1}\beta_2$$

Otherwise, β_1 is invertible and she sends a bit flag set to 0 and

$$\beta_1^{-1}\alpha_1, \beta_1^{-1}\alpha_2, \beta_1^{-1}\beta_2$$

Depending on the bit flag, Bob computes

$$\alpha_1^{-1}\theta(\phi_A(P_B) = R_1 + \alpha_1^{-1}\beta_1 R_2, \quad \alpha_1^{-1}\theta(\phi_A(Q_B) = \alpha_1^{-1}\alpha_2 R_1 + \alpha_1^{-1}\beta_2 R_2$$

or

$$\beta_1^{-1}\theta(\phi_A(P_B) = \beta_1^{-1}\alpha_1 R_1 + R_2, \quad \beta_1^{-1}\theta(\phi_A(Q_B) = \beta_1^{-1}\alpha_2 R_1 + \beta_1^{-1}\beta_2 R_2$$

Regardless,

$$\langle m_B\alpha_1^{-1}\theta(\phi_A(Q_B) + n_B\alpha_1^{-1}\theta(\phi_A(Q_B)\rangle =$$
$$\langle m_B\theta(\phi_A(Q_B) + n_B\theta(\phi_A(Q_B))\rangle =$$
$$\langle m_B\beta_1^{-1}\theta(\phi_A(Q_B) + n_B\beta_1^{-1}\theta(\phi_A(Q_B)\rangle$$

where (m_B, n_B) is Bob's secret key. Hence the change does not effect the kernel of Bob's isogeny ϕ'_B. This brings the compressed key size down to $\frac{7}{2}\log p = 21\lambda$ bits.

9.5 Protocol support

There is a patch for using the ephemeral SIDH implementation from [CLN16] in OpenSSL 1.0.2g[11].

9.6 Patents

To this author's knowledge, SIDH is not patented. This is also stated on Wikipedia[12].

[11] https://www.microsoft.com/en-us/download/details.aspx?id=54053

[12] https://en.wikipedia.org/wiki/Supersingular_isogeny_key_exchange#Introduction, access date 2017-05-12.

9.7 Hybrid schemes

No post-quantum alternative is as well understood and reviewed by the research community as our current RSA and DLP-based cryptosystems. Some researchers have discussed the possibility of combining a thoroughly reviewed cryptosystem such as Elliptic curve Diffie-Hellman (ECDH) with a PQ-cryptosystem until we feel sufficiently confident in the security of one of the PQ-cryptosystems. The authors of [CLN16] have implemented a hybrid cryptosystem combining ECDH with SIDH. They discuss their results in [CLN16, Section 8].

9.8 Using a static key in SIDH

As mentioned in Section 8.2, static key SIDH is currently broken due to the attack given in [GPST16]. In Section 9.8.1 we discuss a counter-measure that was suggested in [KLM+15]. In Section 9.8.4 we propose a slightly different counter-measure in the form of an IND-CCA2 secure KEM built on SIDH. Sections 9.8.2 and 9.8.3 contains some preliminary results needed for our KEM in Section 9.8.4. In Section 9.8.5 we briefly discuss two recent papers on fault attacks against SIDH. For the rest of this section we assume that Alice is using a static key.

9.8.1 A suggested counter-measure to the static key attack

In this section we follow [GPST16, Section 2.5] and [KLM+15, p. 14]. This counter-measure was proposed in [KLM+15]. As noted in [GPST16, Section 2.5] there is no formal security proof for it (such as IND-CCA2 security). We stress that Bob must use an ephemeral secret key sk_B that he reveals to Alice as part of the key exchange. This is the idea of the counter-measure, that Alice verifies that Bob's public key is honestly generated from sk_B. Thereby stopping Bob from fooling Alice into processing corrupt public keys pk_B as in [GPST16, Section 3].

At some earlier point in time, Alice has chosen a secret key $sk_A = (m_A, n_A)$ and published her public key $pk_A = (E_A, \phi_A(P_B), \phi_A(Q_B))$. The suggested updated static SIDH key exchange is the following. We let PRF be a suitable pseudorandom function and KDF be a suitable key derivation function.

1. Bob generates a random bitstring r_B.
2. Bob derives
$$sk_B = (m_B, n_B) = \mathrm{PRF}(r_B)$$
3. Bob computes $j(E_{AB})$ from sk_B and pk_A. He also computes $pk_B = (E_B, \phi_B(P_A), \phi_B(Q_A))$ from sk_B. He then computes a shared key SK and a verification key VK
$$SK||VK = \mathrm{KDF}(j(E_{AB}))$$
He uses symmetric encryption to encrypt $r_B \oplus SK$
$$c_B = \mathrm{Enc}_{VK}(r_B \oplus SK) \tag{35}$$
and sends (pk_B, c_B) to Alice.

4. Alice computes $j(E'_{BA})$ from sk_A and pk_B. She then computes

$$SK'||VK' = \mathrm{KDF}(j(E'_{BA}))$$

$$r'_B = SK' \oplus \mathrm{Dec}_{VK'}(c_B)$$

$$sk'_B = \mathrm{PRF}(r'_B)$$

To verify that Bob is honest she now computes pk'_B from sk'_B. Alice terminates the protocol in an accepting state with shared key $SK' = SK$ if and only if

$$pk'_B = pk_B$$

As explained earlier, the idea is that Bob can not tamper with his public key, all he can do is to choose the seed r_B for the PRF.

With Grover's algorithm in mind we suggest using a seed r_B of bit size 2λ. Note that the bit size $|r_B|$ is not discussed in [GPST16] or [KLM$^+$15]. There are normally $\approx 2^{3\lambda}$ possible secret keys (m_B, n_B) for Bob. But since r_B is inputted to a PRF, using a seed r_B of bit size 2λ does not appear to effect the security.

We see that the only extra information that needs to be sent from Bob to Alice, compared to a naive implementation of static key SIDH, is c_B which Bob sends together with his public key pk_B. The ciphertext c_B is a symmetric encryption of $r_B \oplus VK$ and we approximate its length with that of r_B. If we consider c_B part of Bob's public key, then his public key size is now

$$21\lambda + 2\lambda = 23\lambda$$

with key compression and $(36 + 2)\lambda$ without compression.

For each key exchange, Alice needs to perform roughly twice the work compared to a naive implementation of static key SIDH. She needs to compute $\phi'_A : E_B \to E_{BA}$ as usual, and then also $\phi_B : E_0 \to E_B$ when verifying Bob's public key. Bob's workload is roughly the same. The only extra work for both parties are KDF-computations, PRF-computations and symmetric encryptions. These are very fast operations compared to the isogeny computations in SIDH. But we also note that when using key compression Alice receives pk_B compressed. She may therefore need to do some extra work compared to when validating an uncompressed pk_B. It is an interesting idea for future work to investigate exactly what (if any) extra work she needs to do.

9.8.2 IND-CCA2 secure public key encryption from SIDH

As before we assume that Alice is using a static key. Let

$$\mathcal{H} = \{H_w : \mathbb{F}_{p^2} \to \{0,1\}^k \mid w \in W\} \tag{36}$$

be a family of cryptographic hash functions indexed a finite set W (for Theorem 9.8.2 we also assume that the hash functions are *entropy-smoothing*, see [Sto10, Definition 3]). Alice picks $w \in_R W$ and lets w be a part of her public key. From now on we assume that such a w has been fixed and write H instead of H_w.

We choose $k = 4\lambda$ in (36) in the following discussion. As usual, the security parameter is $\lambda = \frac{1}{6}\log p$ where p is one of the public parameters of SIDH. Other choices of k are also possible, we refer the reader to [FO00] for more details. Suppose that Bob wants to encrypt a message $x \in \{0,1\}^k$ using Alice's public key pk_A. A SIDH public key encryption

scheme $\Pi = (\mathcal{E}, \mathcal{D})$ that accomplishes this is given in [JDF11, Section 3.2]. See Appendix A for the definition of a public key encryption scheme. The public parameters of Π are the same as those in SIDH (with the addition of $\mathcal{H}$) and Alice's key generation works just as in SIDH (with the addition of choosing w). Let Q_B be the set of possible normalised secret keys for Bob in SIDH. For each message $x \in \{0,1\}^k$ that Bob wants to encrypt he picks $r \in_R Q_B$ and does:

$$c = (c_1, c_2) = \mathcal{E}_{pk_A}(x, r) = (pk_B, H(j(E_{AB})) \oplus x) \tag{37}$$

Here $pk_B = (E_B, \phi_B(P_A), \phi_B(Q_A))$ is the public key that is associated to the secret key $sk_B = (m_B, n_B) = r$ in SIDH. Alice can decrypt c using

$$x = \mathcal{D}_{sk_A}(c) = H(j(E_{BA})) \oplus c_2 = H(j(E_{BA})) \oplus (H(j(E_{AB})) \oplus x)$$

The attack against static key SIDH shows that this scheme is not secure in the sense of IND-CCA. We refer to Appendix A for definitions of the standard notions of security in the sense of IND-CPA, IND-CCA and IND-CCA2.

The counter-measure of Section 9.8.1 can be applied to Π as well. However, as noted in [GPST16, Section 2.5] there is no formal security proof for the counter measure. We note that we have the following IND-CCA2 secure public key encryption scheme $\bar{\Pi} = (\bar{\mathcal{E}}, \bar{\mathcal{D}})$ from the transformation in [FO00]. Let

$$G_1 : \{0,1\}^k \to Q_B$$

be a random oracle (see Appendix A). The message space of $\bar{\Pi}$ will be $\{0,1\}^{k-k_0}$, where we let $k_0 = 2\lambda$. To encrypt a message $x \in \{0,1\}^{k-k_0}$, Bob picks $s \in_R \{0,1\}^{k_0}$ and encrypts using

$$c = (c_1, c_2) = \bar{\mathcal{E}}_{pk_A}(x, s) = \mathcal{E}_{pk_A}(x||s, G_1(x||s)) = (pk_B, H(j(E_{AB})) \oplus x||s)$$

Here pk_B is the public key that is associated to the secret key $sk_B = (m_B, n_B) = G(x||s)$ in SIDH. Let $\perp$ denote decryption failure. For a ciphertext c, let $*$ be the condition that

i) $x||s = \mathcal{D}_{sk_A}(c)$ exists (i.e., $\mathcal{D}_{sk_A}(c) \neq \perp$), and

ii) $\mathcal{E}_{pk_A}(x||s, G_1(x||s)) = c$.

Then

$$\bar{\mathcal{D}}_{sk_A}(c) = \begin{cases} x, & \text{if } * \text{ is satisfied} \\ \perp, & \text{otherwise} \end{cases}$$

Since the isogeny computations in SIDH are expensive we stress that in $\bar{\mathcal{D}}$, when Alice is verifying that ii) holds, she does not need to compute $j(E_{AB})$ from pk_A and sk_B like Bob did when he encrypted. She first verifies that $sk_B = (m_B, n_B) = G_1(x||s)$ gives the public key $pk_B = c_1$. We then know that pk_B and sk_A give rise to $j(E_{BA}) = j(E_{AB})$. Since Alice has already computed $j(E_{BA})$ in i) she therefore just checks that

$$H(j(E_{BA})) \oplus x||s = c_2$$

Remark 14. Note that the proposal for a static key exchange in [KLM+15] is based on another transformation by Fujisaki-Okamoto [FO99], according to [KLM+15, p. 12]. [KLM+15] does not discuss a secure public key encryption scheme. It is noted in [GPST16, Section 2.5] that the proposal of [KLM+15] can be applied to Π as well.

The following is [DFJP14, Problem 5.4]. We assume that public parameters p, E_0, (P_A, Q_A) and (P_B, Q_B) such as those in SIDH have been chosen.

Problem 9.8.1 (Supersingular Decision Diffie-Hellman (**SSDDH**) problem)**.** Given a tuple chosen with probability 1/2 from one of the following distributions:

1. (pk_A, pk_B, E_{AB}), where pk_A and pk_B are public keys that the randomly chosen normalised keys $sk_A = (m_A, n_A)$ and $sk_B = (m_B, n_B)$ give rise to, and
$$E_{AB} \cong E_0/\langle m_A P_A + n_A Q_A, m_B P_B + n_B Q_B \rangle$$
2. (pk'_A, pk'_B, E_{AB}), where pk'_A and pk'_B are public keys that the randomly chosen normalised keys $sk'_A = (m'_A, n'_A)$ and $sk'_B = (m'_B, n'_B)$ give rise to,

determine from which distribution the tuple is chosen.

The **SSDDH assumption** [DFJP14, Section 5] says that for any probabilistic polynomial time (in $\log p$) algorithm, the probability of correctly solving the SSDDH problem is at most
$$\frac{1}{2} + \mu$$
for a negligible (in $\log p$) function μ.

Theorem 9.8.2. *If the SSDDH assumption holds, then* $\Pi = (\mathcal{E}, \mathcal{D})$ *is secure in the sense of IND-CPA.*

Proof. See [DFJP14, Theorem 6.2]. □

Let x and c be any elements of the message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$ of Π. We define
$$\gamma(x, c) = \Pr_{r \in_R Q_B}[\mathcal{E}_{pk_A}(x, r) = c]$$
Following [FO00, Definition 3.9] we say that Π is γ**-uniform** if
$$\max_{x \in \mathcal{P}, c \in \mathcal{C}} \gamma(x, c) \leq \gamma$$
For some cryptosystems γ-uniformity with γ a negligible function of the security parameter is a trivial mathematical fact, one such example is the ElGamal encryption scheme [FO00, Section 3]. To this author's knowledge, this is not the case for our scheme Π.

Assumption 9.8.3. The public key encryption scheme $\Pi = (\mathcal{E}, \mathcal{D})$ is γ-uniform, where γ is a negligible function of $\log p$.

Heuristically, Assumption 9.8.3 is very plausible. Suppose that $r \neq r'$ are such that $r, r' \in Q_B$ and
$$\mathcal{E}_{pk_A}(x, r) = c = \mathcal{E}_{pk_A}(x, r')$$
Not only must r and r' give rise to the same curve E_B in $pk_B = c_1$, they must also map P_A and Q_A to the same points in E_B. Note that this means that the isogenies $\phi_B : E_0 \to E_B$ and $\phi'_B : E_0 \to E'_B$ that r and r' give rise to respectively are identical on all of $E_0[\ell_A^{e_A}]$. Also note that we only need Π to be secure in the sense of IND-CPA so we can assume that Bob does not compute his isogeny $\phi_B : E_0 \to E_B$ in some malicious way. In fact, our discussion above about r and r' immediately yields a proof that the SSDDH assumption implies that Assumption 9.8.3 holds.

Lemma 9.8.4. *If the SSDDH assumption holds, then Assumption 9.8.3 holds.*

Proof. Suppose that Assumption 9.8.3 does not hold. Then, in particular, a fraction μ of the $r \in Q_B$ give rise to the same public key pk_B with μ a non-negligible function of $\log p$. Then Bob picks a secret key $sk_B = r$ in this fraction with probability μ. At the same time an adversary can also pick a secret key r' in this fraction with probability μ. If both Bob and the adversary pick secret keys from this fraction, then the adversary can play the role of Bob in the key exchange and compute a curve that is isomorphic to E_{AB}. It follows that the adversary can distinguish between the two tuples in the SSDDH problem with probability $\frac{1}{2} + \mu'$, where μ' is a non-negligible function of $\log p$. Hence the SSDDH assumption does not hold. □

Theorem 9.8.5. *If the SSDDH assumption holds, then the public key encryption scheme $\bar{\Pi} = (\bar{\mathcal{E}}, \bar{\mathcal{D}})$ is secure in the sense of IND-CCA2 in the random oracle model.*

Proof. Note that k_0 is polynomial in k. If the SSDDH assumption holds, then Π is γ-uniform with γ a negligible function of $\log p = 6\lambda = \frac{6}{4}k$ by Lemma 9.8.4. Thus, Π is also γ'-uniform with γ' a negligible function of k. The theorem now follows from Theorem 9.8.2 by the discussion in [FO00, Section 5] preceding Theorem 5.4. □

Remark 15. [FO00, Theorem 5.4] also derive the concrete security bounds of their transformation. We leave a more detailed study of the concrete security bounds of our scheme $\bar{\Pi}$ to future work.

9.8.3 SIDH as a KEM

A **Key Encapsulation Mechanism** (KEM) (see Appendix A) is a set of algorithms that allows Bob to send an ephemeral key K to Alice. When Alice is using a static key only one message in total needs to be transmitted between the two parties: the message from Bob to Alice containing an encapsulation of K that has been created using Alice's public key. SIDH can be seen as a KEM in the following way.

- $pp \leftarrow \text{Setup}(1^\lambda)$. We generate the public parameters of the SIDH key exchange with $\log p = 6\lambda$.
- $(sk_A, pk_A) \leftarrow \text{Gen}(pp)$. Alice computes a public key pk_A from her secret key sk_A.
- $(K, c) \leftarrow \text{Encaps}(pk_A)$. Bob computes $K = j(E_{AB})$ from pk_A and lets $c = pk_B$.
- $K \leftarrow \text{Decaps}(sk_A, c)$. Alice computes $K = j(E_{BA})$ from c.

This scheme is not IND-CCA secure due to the static key attack discussed in Section 8.2. Our goal in the next section is to describe how to build an IND-CCA2 secure KEM from SIDH. See Appendix A for the standard definition of an IND-CCA2 secure KEM.

9.8.4 An IND-CCA2 secure KEM from SIDH

As noted by [Pei14, Section 5] we could build an IND-CCA2 secure KEM by letting the encapsulation of the KEM encrypt an ephemeral key $K \in_R \{0,1\}^{2\lambda}$ using our IND-CCA2 secure encryption scheme $\bar{\Pi}$ from Section 9.8.2. This is not satisfactory since the encapsulation created by Bob already contains an ephemeral key $j(E_{AB})$. Our goal now is therefore to save those 2λ bits and still have an IND-CCA2 secure KEM. By doing so

we answer an open question from [KLM+15, p. 18] in the context of SIDH. We prove that: yes, the security proof for the Fujisaki-Okamoto transform can be carried over to the KEM. But we use the transform from [FO00] instead of the one from [FO99] and our KEM is different from the one proposed in [KLM+15]. We also note that [DF17, p. 37] mentions, without giving any details, that it is possible to obtain an IND-CCA2 secure KEM from SIDH through a generic transformation. Security in the sense of IND-CCA2 is considered by NIST to be a relevant security notion for a (non-ephemeral) KEM[13].

We start by changing the parameters k and k_0 of Section 9.8.2 to 2λ and $2\lambda - 1$, respectively. This means that $\bar{\mathcal{E}}$ in $\bar{\Pi} = (\bar{\mathcal{E}}, \bar{\mathcal{D}})$ now only encrypts a single bit. The arguments used in proving Theorem 9.8.5 still hold, so this variant of the public key encryption scheme is secure (in the security parameter λ) in the sense of IND-CCA2 under the SSDDH assumption.

Let

$$G_2 : \mathbb{F}_{p^2}^3 \times \mathbb{F}_{p^2} \to \{0,1\}^{2\lambda}$$

be a random oracle. Our KEM is defined by

- $pp \leftarrow \text{Setup}(1^\lambda)$. The public parameters of the KEM are the same as the public parameters of $\bar{\Pi}$.
- $(sk_A, pk_A) \leftarrow \text{Gen}(pp)$. Alice computes a public key pk_A from her secret key sk_A.
- $(K, c) \leftarrow \text{Encaps}(pk_A)$. Bob computes

$$K = G_2(pk_B, j(E_{AB})) \tag{38}$$

$$c = (c_1, c_2) = \bar{\mathcal{E}}_{pk_A}(x, s) = \mathcal{E}_{pk_A}(x||s, G_1(x||s)) = (pk_B, H(j(E_{AB})) \oplus x||s) \tag{39}$$

  with $x||s \in_R \{0,1\}^{2\lambda}$.

- $K \leftarrow \text{Decaps}(sk_A, c)$. Alice terminates the KEM protocol with shared key

$$K = G_2(pk_B, j(E_{BA}))$$

  if and only if $\bar{D}_{sk_A}(c) \neq \perp$. We write $\perp \leftarrow \text{Decaps}(sk_A, c)$ when $\bar{D}_{sk_A}(c) = \perp$. We say an encapsulation c is **valid** if $\perp \not\leftarrow \text{Decaps}(sk_A, c)$

Lemma 9.8.6. *Suppose that an algorithm A breaks the KEM in the sense of IND-CCA2. Let*

$$c^* = (c_1^*, c_2^*) = (pk_B, H(j(E_{AB})) \oplus x||s) \tag{40}$$

be the challenge encapsulation in the IND-CCA2 game. Then A does at least one of the following with probability μ where μ is a non-negligible function of λ.

1. *The algorithm A queries G_2 on $(pk_B, j(E_{AB}))$.*
2. *The algorithm A queries the decapsulation oracle (that A has access to in the IND-CCA2 game) on a valid encapsulation $c = (c_1, c_2)$ such that $c_1 = c_1^*$.*

Proof. The challenge key K^* in the IND-CCA2 game is either chosen randomly in $\{0,1\}^{2\lambda}$ or it is equal to $G_2(pk_B, j(E_{AB}))$. In both cases it is a random string that is chosen independently of pk_B and $j(E_{AB})$). There are therefore only two ways in which A can learn anything about whether K^* is the key associated to c^* or not:

[13] Section 4.A.2 of http://csrc.nist.gov/groups/ST/post-quantum-crypto/documents/call-for-proposals-final-dec-2016.pdf, access date 31-07-2017.

1. The algorithm A queries G_2 on $(pk_B, j(E_{AB}))$.

2. The algorithm A queries the decapsulation oracle (that A has access to in the IND-CCA2 game) on a valid encapsulation $c = (c_1, c_2)$ such that $c_1 = c_1^*$. Note that $j(E_{AB})$ is completely determined by $c_1^* = pk_B$ in a valid encapsulation since sk_A is fixed, and (pk_B, sk_A) completely determines $j(E_{BA}) = j(E_{AB})$.

Let "A queries" be the event that 1 or 2 happens. Let "A wins" be the event that A wins in the IND-CCA2 game for a KEM. Then

$$\Pr[A \text{ wins } | \neg A \text{ queries }] \leq \frac{1}{2}$$

Thus,

$$\frac{1}{2} + \mu' =: \Pr[A \text{ wins }] =$$
$$\Pr[A \text{ wins } | \neg A \text{ queries }]\Pr[\neg A \text{ queries }] +$$
$$\Pr[A \text{ wins } | A \text{ queries }]\Pr[A \text{ queries }] \leq$$
$$\frac{1}{2} \cdot \Pr[\neg A \text{ queries }] + 1 \cdot \Pr[A \text{ queries }] \leq$$
$$\frac{1}{2} + \cdot \Pr[A \text{ queries }] \Rightarrow$$
$$\mu' \leq \Pr[A \text{ queries }] =: \mu$$

By assumption μ' is a non-negligible function of λ and therefore μ is a non-negligible function of λ. □

Theorem 9.8.7. *Suppose that the SSDDH assumption holds. Then the KEM is secure in the sense of IND-CCA2 in the random oracle model.*

Proof. We will sketch the proof that is standard. Given an algorithm A that breaks the KEM in the sense of IND-CCA2, we will describe an algorithm A' that breaks $\bar{\Pi}$ in the sense of IND-CCA2.

The algorithm A has oracle access to a decapsulation oracle and the random oracle G_2 in the IND-CCA2 game (in the context of a KEM scheme). The algorithm A' has access to a decryption oracle for $\bar{\Pi}$ in the IND-CCA2 game (in the context of a public key encrytion scheme). We need to argue that A' can simulate the decapsulation oracle and G_2 in a way that is indistinguishable from the real thing to A. The following two lemmas show that this is possible. In particular, we need to be careful that the following situation is handled correctly.

- The algorithm A queries G_2 on a pair $(a_1, a_2) \in \mathbb{F}_{p^2}^3 \times \mathbb{F}_{p^2}$ and receives an answer K.

- At some other point in time (not necessarily afterwards) A queries the decapsulation oracle on a valid encapsulation

  $$c = (pk_B, H(j(E_{AB})) \oplus x||s),$$

  such that $(a_1, a_2) = (pk_B, j(E_{AB}))$, and receives an answer K'.

We need to make sure that $K = K'$. To this end we keep a list L of valid encapsulations c that A has queried its decapsulation oracle on, but for which we do not know the value of $j(E_{AB})$. The algorithm A' will query its decryption oracle on each such c and we also save the message bit x that is returned and associates it to c. We also associate a randomly chosen key K to each such c. As discussed in Lemma 9.8.6, $j(E_{AB})$ is completely determined by c_1 in a valid encapsulation

$$c = (c_1, c_2) = (pk_B, H(j(E_{AB})) \oplus x||s),$$

we therefore only save one c in L for each distinct c_1.

For simplicity, we call the random oracle that A' simulates G_2 as well. We say $G_2(a_1, a_2)$ is defined or undefined depending on if A' has chosen an output or not for input (a_1, a_2).

Lemma 9.8.8. *The algorithm A can not distinguish the real decapsulation oracle from the one that A' simulates.*

Proof. Suppose that A queries the decapsulation oracle on an encapsulation c.

1. If the decryption oracle returns $\perp$ on input c, then A' returns $\perp$.
2. Otherwise, the decryption oracle returns an x such that

   $$c = (c_1, c_2) = \bar{\mathcal{E}}_{pk_A}(x, s) = \mathcal{E}_{pk_A}(x||s, G_1(x||s)) = (pk_B, H(j(E_{AB})) \oplus x||s)$$

   for some $s \in \{0,1\}^{2\lambda-1}$.
3. The algorithm A' checks if G_2 is defined for any (a_1, a_2) such that
   (a) $a_1 = c_1$, and
   (b) $H(a_2) \oplus c_2 =: x'||s'$ is such that $\mathcal{E}_{pk_A}(x'||s', G_1(x'||s')) = c$, and
   (c) $a_2 = j(E_{AB})$ where $j(E_{AB})$ is the j-invariant associated to pk_A and $sk_B = G_1(x'||s') = G_1(x||s)$.

   If A' finds such a pair (a_1, a_2), then it returns

   $$K = G_2(a_1, a_2)$$

4. Otherwise, if there is a $c' = (c'_1, c'_2)$ in L such that $c_1 = c'_1$, then A' returns the K associated to c'.
5. Otherwise, A' adds c to L, associates the message bit x and a string $K \in_R \{0,1\}^{2\lambda}$ to c, and returns K.

Using also how random oracle queries are handled in the proof of Lemma 9.8.9 one can verify that this is the behaviour A expects. □

Lemma 9.8.9. *The algorithm A can not distinguish the real random oracle G_2 from the G_2 that A' simulates.*

Proof. Suppose that A queries G_2 on a pair (a_1, a_2).

1. If $G_2(a_1, a_2)$ is defined, then A' returns $G_2(a_1, a_2)$.

2. Otherwise, for each c in L, A' checks if

   (a) $c_1 = a_1$, and

   (b) $H(a_2) \oplus c_2 =: x||s$ is such that $\mathcal{E}_{pk_A}(x||s, G_1(x||s)) = c$, and

   (c) $a_2 = j(E_{AB})$ where $j(E_{AB})$ is the j-invariant associated to pk_A and $sk_B = G_1(x||s)$.

   If A' finds such a c, then it defines

   $$G_2(a_1, a_2) = K$$

   where K is the key that is associated to c. The algorithm A' then deletes c from L and returns $G_2(a_1, a_2) = K$.

3. Otherwise, A' defines

   $$G_2(a_1, a_2) \in_R \{0,1\}^{2\lambda}$$

   and returns $G_2(a_1, a_2)$.

Using also how decapsulation queries are handled in the proof of Lemma 9.8.8 one can verify that this is the behaviour A expects. □

If A signals that it wishes to receive its challenge (K^*, c^*), then A' signals that it wishes to receive its challenge c^* by outputting $(x_0, x_1) = (0, 1)$. By definition of the IND-CCA2 game (in the context of a public key encryption scheme), A' then receives

$$c^* = (c_1^*, c_2^*) = (pk_B, H(j(E_{AB})) \oplus x_b||s) \tag{41}$$

where b is randomly chosen in secret. The algorithm A' wins the IND-CCA2 game if it outputs b (which in this case is the same as outputting x_b).

1. For every pair (a_1, a_2) such that $G_2(a_1, a_2)$ is defined, A' checks if

   $$H(a_2) \oplus c_2^* =: x_b'||s' \tag{42}$$

   is such that

   $$\bar{\mathcal{E}}_{pk_A}(x_b', s') = \mathcal{E}_{pk_A}(x_b'||s', G_1(x_b'||s')) = c^* \tag{43}$$

   If (43) holds, then A' wins by outputting $x_b' = x_b$.

2. If A' can not find a pair (a_1, a_2) such that (43) holds, then A' checks if there is a $c = (c_1, c_2)$ in L such that $c_1 = c_1^*$.

   (a) If there is such a c, then we let x be the message bit that is associated to c. We know that $j(E_{AB})$ in (41) is completely determined by $c_1^* = c_1$. Let $(y)_i$ be the ith leftmost bit of a bitstring y. But then

   $$(c_2)_1 \oplus x$$

   is the first bit of $H(j(E_{AB}))$. Hence

   $$x_b = (c_2^*)_1 \oplus ((c_2)_1 \oplus x)$$

   So A' wins by outputting x_b.

(b) If there is no such c, then A' picks $K^* \in_R \{0,1\}^{2\lambda}$ and sends the challenge pair (K^*, c^*) to A.

The algorithm A' now continues to simulate A. By definition of the IND-CCA2 game, A will not query the decapsulation oracle on c^*. Any queries to the decapsulation oracle or to the random oracle G_2 that A makes are handled as before the challenge was given to A. The only modifications are the following.

1. If A queries the random oracle G_2 on a pair (a_1, a_2), then the first thing A' does is to check whether (a_1, a_2) lets it decrypt c^* as in (42) and (43). If this is the case then A' decrypts the ciphertext c^* and outputs b. Otherwise, A' proceeds to handle the query as in Lemma 9.8.9.

2. If A queries the decapsulation oracle on a ciphertext $c = (c_1, c_2)$, then the first thing A' does is to check if $c_1 = c_1^*$ and if its decryption oracle decrypts c successfully. If this is the case, then A' decrypts x_b as in 2(a) above and wins the IND-CCA2 game by outputting it. Otherwise, A' proceeds to handle the query as in Lemma 9.8.8.

If A queries G_2 on a pair (a_1, a_2) that lets A' decrypt c^* or queries the decapsulation oracle on a ciphertext c that lets A' decrypt x_b, then A' always outputs the correct value b from (41). Otherwise, A' guesses the value of b and is correct with probability $\frac{1}{2}$. By Lemma 9.8.6, A' wins in the IND-CCA2 game with probability

$$\frac{1}{2} \cdot (1 - \mu) + 1 \cdot \mu = \frac{1 + \mu}{2}$$

where μ is a non-negligible function of λ. By Theorem 9.8.2, this means that the SSDDH assumption does not hold. □

This author believes that the arguments and constructions we have studied in these last sections may possibly be generalised and applied to other DH-like key exchanges, besides SIDH, to construct IND-CCA2 secure KEMs.

Remark 16. Note that the proof of Lemma 9.8.8 shows why it was necessary define the key in the KEM as $K = G_2(pk_B, j(E_{AB})$ and not simply $K = G_2(j(E_{AB}))$. If we defined the key as $K = G_2(j(E_{AB}))$, then A could possibly query the decapsulation oracle on two encapsulations

$$c = (pk_B, H(j(E_{AB})) \oplus x||s)$$
$$c = (pk'_B, H(j(E_{AB})') \oplus x'||s')$$

such that $pk_B \neq pk'_B$ and $j(E_{AB}) = j(E_{AB})'$. Then it would not necessarily be possible for A' to tell that A expects the same decapsulated key K in return for both queries. The underlying problem is that, in SIDH, distinct public keys pk_B for Bob can give rise to the same j-invariant $j(E_{AB})$ (even if Alice's secret key is fixed).

Remark 17. The Fujisaki-Okamoto transform in [FO99] starts from a public key encryption scheme that is one-way secure (as opposed to secure in the sense of IND-CPA as in [FO00]). It is an interesting idea for future work to derive a one-way secure encryption system from SIDH under a weaker assumption than the SSDDH assumption and then study if our KEM construction can be easily adapted to the secure public key encryption constructed in [FO99] instead.

9.8.5 Fault attacks

Ti [Ti17] describes a fault attack against supersingular isogeny cryptosystems. We assume that an attacker has the capability to introduce errors in Alice's public key computation and makes Alice publish

$$(E_A, \phi_A(X), \phi_A(Q_B) \tag{44}$$

for a random point X, instead of the correct triple

$$(E_A, \phi_A(P_B), \phi_A(Q_B))$$

[Ti17] argues that this allows the attacker to retrieve Alice's private key with high probability. As discussed in [Ti17, Section 4.1] this attack is more suited for some of the supersingular isogeny signature schemes that have been proposed, rather than for SIDH. In SIDH, when Alice uses a static key she only computes her public key once for the duration of her private key. When both Alice and Bob use ephemeral keys in SIDH, they will not reach the same shared key $j(E_{AB})$ if an attacker causes Alice to publish an incorrect public key (44).

Gélin and Wesolowski [GW17] describe a fault attack in the following situation. We assume that Alice is using a static secret key $(m_A, n_A) = (1, \alpha)$ in SIDH. We also assume that Bob has the capability to introduce loop-abort faults in Alice's iterative computation of the isogeny $\phi'_A : E_B \to E_{BA}$. This means that Bob can choose a k and make Alice compute an ℓ_A^k-isogeny with kernel

$$\langle \ell_A^{e_A - k}(\phi_B(P_A) + \alpha\phi_B(Q_A)) \rangle$$

instead of the $\ell_A^{e_A}$-degree isogeny ϕ'_A with kernel

$$\langle \phi_B(P_A) + \alpha\phi_B(Q_A) \rangle$$

that Alice normally computes. By choosing different values of k over a series of key exchanges with Alice, Bob can retrieve Alice's static key $(1, \alpha)$ [GW17, Section 4.2]. In situations where this kind of attack is relevant the authors suggest that Alice uses counters as a cheap countermeasure to make sure that her iterative isogeny computation runs the expected number of iterations [GW17, Section 5].

9.9 SIDH benchmarks

By public key computation we mean that a party computes its public key pk from its secret key sk. By shared key computation we mean that a party computes $j(E_{AB})$ from the other party's pk. In Tables 5, 6, 7 and 8 we give benchmarks and estimates of several variants of SIDH. The benchmarked implementations in Tables 5 and 6 are those of [CLN16] and [CJL+16b] in the SIDH library v2.0[14]. These implementations are for ephemeral SIDH only. We use these benchmarks of ephemeral implementations to give estimates of running times for static variants of SIDH in Tables 7 and 8. The platform in all four tables is i7-4700MQ 2.4 GHz with turbo boost disabled running ubuntu 16.04 LTS. Clock cycle (cc) measurements are rounded off to the nearest million. The runtime estimates in milliseconds (ms) are obtained by dividing the clock cycle measurements by the frequency of the platform and rounding off to the nearest millisecond.

[14] https://www.microsoft.com/en-us/research/project/sidh-library/

[15] This estimation is based on, among other things, an assumption about the time complexity of $\mathbb{F}_{p^2}$-arithmetic, see Section 9.10.5 for details.

Table 5: **Ephemeral SIDH (without key compression)**. On a faster desktop PC running at 3.4-4 GHz the total running time for Alice is closer to 25-30 ms.
†) The clock cycle measurement for $\lambda = 85$ is not a benchmark, but a very rough estimate that has been obtained by dividing the clock cycle measurements for $\lambda = 128$ by 2.9. In Section 9.10.5 we will estimate[15] that going from $\lambda = 128$ to $\lambda = 85$ speeds up computation times in a state-of-the-art implementation of SIDH by roughly a factor 2.9.

Quantum security λ (bits)	85		128	
Classical security (bits)	128		192	
Key size = $36\lambda/8$ (bytes)	383		576	
Alice public key computation (cc | ms)	-	-	46M	19
Bob public key computation (cc | ms)	-	-	52M	22
Alice shared key computation (cc | ms)	-	-	44M	18
Bob shared key computation (cc | ms)	-	-	50M	21
Alice total (cc | ms)	-	-	90M	38
Bob total (cc | ms)	-	-	102M	43
Both parties total (cc | ms)	≈66M†	≈28	192M	56

Table 6: **Compressed ephemeral SIDH**.
†) The clock cycle measurement for $\lambda = 85$ is not a benchmark, but a very rough estimate that is obtained by multiplying † in Table 5 by 2.4. [CJL+16b, p. 19] estimates that key compression introduce a factor 2.4-slowdown of SIDH.

Quantum security λ (bits)	85		128	
Classical security (bits)	128		192	
Key size = $21\lambda/8$ (bytes)	224		336	
Alice public and shared key computation (cc | ms)	-	-	78M	33
Alice key compression (cc | ms)	-	-	107M	46
Alice decompression of Bob's key (cc | ms)	-	-	42M	18
Bob public and shared key computation (cc | ms)	-	-	91M	38
Bob key compression (cc | ms)	-	-	111M	46
Bob decompression of Alice's key (cc | ms)	-	-	33M	14
Alice total (cc | ms)	-	-	231M	96
Bob total (cc | ms)	-	-	240M	100
Both parties total (cc | ms)	≈158M†	≈66	471M	196

In Tables 7 and 8 we give estimates of running times for static variants of SIDH using the benchmarks for the ephemeral variants of SIDH in Tables 5 and 6. As Alice's rounds in SIDH are faster than Bob's in Table 5 **we assume that Bob is using the static key in Tables 7 and 8. We stress that there is no existing implementation of static key SIDH**. We also asumme that the counter-measure of Section 9.8.1 is used (the estimates would be the same if the IND-CCA2 secure KEM of Section 9.8.4 was used). As discussed in Section 9.8.1, ignoring symmetric encryptions/decryptions and key derivation functions (which are fast compared to the operations of SIDH), Alice and Bob's work in static key SIDH is roughly equal to their work in ephemeral SIDH. For each key exchange execution: instead of computing his own public key (which is static and already computed), Bob must recompute Alice's public key to assert that it is honestly generated (step 4 in Section 9.8.1). Alice's public key is larger because of the ciphertext c_A from (35) in Section 9.8.1. We assume that a seed r_A of size 2λ is used in step 1 of Section 9.8.1.

Table 7: **Estimates for static key SIDH (without key compression)**. We approximate the work in uncompressed static key SIDH with the measurements from Table 5 for uncompressed ephemeral SIDH.
†) Is a very rough estimate that is obtained by just taking the estimate † from Table 5.

Quantum security λ (bits)	85		128	
Classical security (bits)	128		192	
Bob (static) key size = $36\lambda/8$ (bytes)	383		576	
Alice (ephemeral) key size = $(36\lambda + 2\lambda)/8$ (bytes)	404		608	
Alice public key computation (cc | ms)	-	-	46M	19
Alice shared key computation (cc | ms)	-	-	44M	18
Bob shared key computation (cc | ms)	-	-	50M	21
Bob recomputing Alice's public key (cc | ms)	-	-	46M	19
Alice total (cc | ms)	-	-	90M	38
Bob total (cc | ms)	-	-	96M	40
Both parties total (cc | ms)	$\approx$66M†	$\approx$28	186M	78

Table 8: **Estimates for compressed static key SIDH**. We assume that Bob's static key is either not compressed or has already been decompressed before the execution of the static key exchange.
†) Is a very rough estimate obtained by taking the estimate † in Table 6 times $(1 - 0.09)$. We do this because Alice's decompression of Bob's public key make up 9% of both parties total computation cost in Table 6.
††) Since Bob's static public key is not compressed, this is Alice public and shared key computation from Table 5. Alice's shared key computation differs slightly depending on if Bob's public key is compressed or not in the SIDH library v.2.0.
†††) Note that Bob received Alice's public key compressed. When validating Alice's public key, he must therefore do some extra work compared to when recomputing Alice's public key in Table 7. We assume here that he must perform the entire compression of Alice's public key. The work accounted for here is Bob's shared key computation (Table 6), Alice's public key computation (Table 5) and Alice's public key compression (Table 6).

Quantum security λ (bits)	85		128	
Classical security (bits)	128		192	
Bob (uncompressed static) key size = $36\lambda/8$ (bytes)	383		576	
Alice compressed (ephemeral) key size = $(21\lambda + 2\lambda)/8$ (bytes)	244		368	
Alice public and shared key computation†† (cc | ms)	-	-	90M	38
Alice key compression (cc | ms)	-	-	107M	46
Bob shared key computation and recomputation of $pk_A^{\dagger\dagger\dagger}$ (cc | ms)	-	-	192M	80
Bob decompression of Alice's key (cc | ms)	-	-	33M	14
Alice total (cc | ms)	-	-	197M	82
Bob total (cc | ms)	-	-	225M	94
Both parties total (cc | ms)	$\approx$144M†	$\approx$60	422M	176

9.9.1 Memory usage

We employ the same method of measuring memory usage as that in [Kin17, Section 4.3]; the Valgrind tool Massif with the argument "--stacks=yes". This gives us a peak memory usage when measuring both heap and stack size. Measuring the ephemeral uncompressed SIDH implementation of [CLN16] gives a maximum memory usage of 15,904 bytes (11,800 bytes being stack memory) for any party, while measuring the compressed implementation of [CJL$^+$16b] gives a maximum memory usage of 27,808 bytes (22,424 bytes being stack memory) for any party[16].

By Kindberg's comparison in [Kin17, Table 5.4] the memory usage of SIDH is low compared to many other PQ-cryptosystems[17].

9.9.2 Several PQ-cryptosystems benchmarked as KEM schemes

In Section 9.8.3 we discussed how SIDH can be used as a KEM. Any public key encryption system can be used in a KEM in a natural way: Bob uses Alice's public key to encrypt a randomly generated ephemeral key K. So in the KEM framework we can compare post-quantum public key encryption systems to post-quantum key exchange systems. We did this using the Open Quantum Safe project [SM16]. The results are shown in Table 9. Unfortunately, to this author's knowledge, this comparison is rough in the sense that the different KEM implementations are not benchmarked at the exact same claimed quantum security level. The implementation using SIDH is that of [CLN16], which has a claimed quantum security level of (about) 128 bits and uses ephemeral keys. Note that the [CLN16] implementation uses uncompressed public keys. Some other examples of varying security levels include the following. The implementation using R-LWE BCNS15 appears to have a claimed quantum security level of 78 bits[18]. The implementation using LWE Frodo appears to have a claimed quantum security level of 130 bits[19].

Table 9: Comparison of several different PQ-cryptosystems within the KEM framework. The platform is i7-4700MQ 2.4GHz with turbo boost disabled running ubuntu 16.04 LTS.

	A Gen cc	μs	B Encaps cc	μs	A Decaps cc	μs	A $\rightarrow$ B	B $\rightarrow$ A
R-LWE BCNS15	3.5M	1473	5.6M	2334	0.6M	240	4096	4224
R-LWE NewHope	0.2M	100	0.4M	154	0.1M	27	1824	2048
R-LWE MSR LN16	0.2M	84	0.3M	143	0.1M	26	1824	2048
LWE Frodo	12.3M	5144	14.1M	5871	0.4M	151	11280	11288
SIDH CLN16	52.8M	22053	118.9M	49657	50.0M	20876	576	576
Code McBits	822.6M	343537	0.2M	71	0.5M	199	311736	141
NTRU EES743EP1	5.1M	2144	0.5M	227	0.4M	162	1027	1022

We see in Table 9 that McBits can be an interesting alternative to SIDH when Alice is using a static key. The drawback of McBits is Alice's very large public key that must be distributed to Bob.

[16]We measure the implementations in the SIDH library v2.0 compiled with "make ARCH=x64" and debug information. The platform is i7-4700MQ 2.4 GHz running ubuntu 16.04 LTS.

[17]SIDH is represented in the table by the implementations of [CLN16] and [CJL$^+$16b]

[18]https://github.com/open-quantum-safe/liboqs/blob/master/docs/Algorithm%20data%20sheets/kex_rlwe_bcns15.md, access date 2017-06-13.

[19]https://github.com/open-quantum-safe/liboqs/blob/master/docs/Algorithm%20data%20sheets/kex_lwe_frodo.md, access date 2017-06-13.

9.10 Estimations of work, memory usage and key sizes as functions of λ

Our end goal in this section is to estimate the computational work, memory usage and key sizes in SIDH as functions of λ, the security parameter of SIDH. Unfortunately, our analysis can not be self-contained due to space constraints. We will make references to algorithms and analyses in [CLN16], [CJL+16b], [CH17] and [DFJP14]. To this author's knowledge, no total estimates, such as the ones we will do in the coming sections, currently exist the literature. However, as noted in Section 1.1, the work we do here is straightforward thanks to earlier analyses in the papers mentioned above. Our estimations are done by studying the state-of-the-art implementations of [CLN16] (uncompressed ephemeral SIDH) and (compressed ephemeral SIDH) [CJL+16b]. These are included in the SIDH library v2.0[20] and any reference hereafter to the SIDH library means specifically v2.0. Sometimes we will need to refer to code in the SIDH library. As is standard when estimating the time complexity of functions in finite fields, we will give our estimates in the number of finite field arithmetic operations. In general we ignore any constant number of arithmetic operations, i.e., when the number of times the operation is performed does not depend on λ. The exceptions are the inversions done in [CLN16]. These are implemented in constant-time and are fairly expensive. Note that we ignore any constant number of inversions in the compression and decompression of public keys in [CJL+16b]. As discussed in [CJL+16b, p. 4], since compression and decompression is done on public data, the inversions there can be implemented in faster non-constant-time ways. Since [CLN16] uses a prime $p = \ell_A^{e_A}\ell_B^{e_B}f - 1$ such that $\ell_A = 2$, $\ell_B = 3$ and $f = 1$, our estimates here also assume that $\ell_A = 2$, $\ell_B = 3$ and $f = 1$. By the discussion in Section 8.1 we have the relations

$$e_A \approx 3\lambda, \quad e_B \approx e_A/\log 3 \approx 1.89\lambda$$

To verify that our estimations are close to the true costs we do the following. We benchmark the $\mathbb{F}_{p^2}$ arithmetic operations **M** (multiplication), **S** (squaring), addition (**a**) and inversion (**I**) in the SIDH library.

$$\mathbf{M} = 2027, \quad \mathbf{S} = 1458, \quad \mathbf{a} = 103, \quad \mathbf{I} = 612086 \tag{45}$$

These costs, which are in clock cycles (cc), are for the optimised implementation in the SIDH library[21] for the prime $p = 2^{372}3^{239} - 1$. By running kex_tests.c we can also benchmark the ephemeral uncompressed and compressed SIDH implementations of [CLN16] and [CJL+16b] for the same prime p. The output gives us detailed benchmarks of the same computations that we are estimating. This gives us a way to check how close our estimations are to the true costs for this specific prime p. Whenever we compare our estimates to benchmarks in the coming sections we use the arithmetic costs in (45) to convert our estimations to clock cycles, and then we compare against the output of kex_tests.c.

9.10.1 Estimating the computational work in uncompressed SIDH

The operation costs in [CLN16, Table 1] and [CH17, Table 3] provide a basis for estimating the computational work in ephemeral uncompressed SIDH. As explained in [CH17,

[20] https://www.microsoft.com/en-us/research/project/sidh-library/

[21] The library is compiled with "make ARCH=x64". The platform is i7-4700MQ 2.4 GHz with turbo boost disabled running ubuntu 16.04 LTS.

Appendix A], [CH17, Table 3] provide optimisations of some of the operations in [CLN16, Table 1] that can be plugged into the implementation of [CLN16].

We want to estimate the cost of the four main computations in uncompressed ephemeral SIDH: Alice's public key generation ([CLN16, Algorithm 3]), Alice's shared key computation ([CLN16, Algorithm 7]), Bob's public key generation ([CLN16, Algorithm 5]) and Bob's shared key computation ([CLN16, Algorithm 9]). Given the operation costs in [CLN16, Table 1] and [CH17, Table 3] this is fairly straightforward except for that any competitive implementation of SIDH must currently use a platform specific strategy (see i) in Section 9.1) for computing isogenies that is computed through dynamic programming. The strategy depends on the operation cost ratio between computing a scalar ℓ-multiplication (on an elliptic curve) and evaluating an ℓ-degree isogeny. Fortunately, it turns out that using a balanced strategy, where we assume that computing an ℓ-degree multiplication is exactly as expensive as computing an ℓ-degree isogeny, gives a very good estimation in practice. For the cost ratio discussed in [DFJP14, p. 12] the optimal strategy is only 2.1% cheaper than the balanced strategy. We refer the reader to [DFJP14, Table 2] for other such comparisons. There is a detailed theoretical study of the difference between balanced and optimal strategies in [DFJP14, Section 4.2.2] and the authors note on p. 16 that the theoretical predictions are line with the the measurements of [DFJP14, Table 2].

[CLN16] works with 4-multiplications and 4-degree isogenies instead of 2-multiplications and 2-degree isogenies for Alice's computations. We let x_A be the number of 4-multiplications in the balanced strategy in Alice's computations. Note that the number of 4-isogenies in the strategy is also x_A. Similarly, we let x_B be the number of 3-multiplications (and 3-isogenies) in the balanced strategy in Bob's computations. As noted on [DFJP14, p. 9], we have that

$$x_A = \frac{1}{2}\frac{e_A}{2}\log(\frac{e_A}{2}) \approx 0.75\lambda\log(1.5\lambda), \quad x_B \approx \frac{1}{2}\frac{e_A}{\log 3}\log(\frac{e_A}{\log 3}) \approx 0.95\lambda\log(1.89\lambda)$$

Our estimates in Tables 10, 11, 12 and 13 are straightforward using the balanced strategy estimation. Line numbers refer to the respective computation's pseudocode in [CLN16]. The main reason our estimations are lower than the benchmarked costs is that we use the optimised operation costs from [CH17, Table 3].

Table 10: Alice public key generation pkGen_A ([CLN16, Algorithm 3]). pkGen_A sums to 45M cc using the arithmetic benchmarks in (45). Benchmarking Alice's public key generation in the implementation of [CLN16] gives 46M cc.

Line(s)	**M**	**S**	**a**	**I**	Source
5	$5e_A$	$4e_A$	$9e_A$	-	[CLN16, Table 1]
16	$8x_A$	$4x_A$	$8x_A$	-	[CLN16, Table 1]
19	-	$2e_A$	$2e_A$	-	[CH17, Table 3]
21	$6x_A$	$2x_A$	$6x_A$	-	[CH17, Table 3]
23-25	$9e_A$	$3e_A$	$9e_A$	-	[CH17, Table 3]
33	-	-	-	1	[CLN16, Table 1]
pkGen_A (total)	$(42 + 10.5\log(1.5\lambda))\lambda$	$(27 + 4.5\log(1.5\lambda))\lambda$	$(60 + 10.5\log(1.5\lambda))\lambda$	1	

Table 11: Alice shared key computation SKGen_A ([CLN16, Algorithm 7]). SKGen_A sums to 41M cc using the arithmetic benchmarks in (45). Benchmarking Alice's public key generation in the implementation of [CLN16] gives 44M cc.

Line(s)	**M**	**S**	**a**	**I**	Source
1	-	-	-	1	[CLN16, Table 1]
2	$9e_A$	$6e_A$	$14e_A$	-	[CLN16, Table 1]
9	$8x_A$	$4x_A$	$8x_A$	-	[CLN16, Table 1]
12	-	$2e_A$	$2e_A$	-	[CH17, Table 3]
14	$6x_A$	$2x_A$	$6x_A$	-	[CH17, Table 3]
20	-	-	-	1	[CLN16, Table 1]
SKGen_A (total)	$(27 + 10.5\log(1.5\lambda))\lambda$	$(24 + 4.5\log(1.5\lambda))\lambda$	$(48 + 10.5\log(1.5\lambda))\lambda$	2	

Table 12: Bob public key computation pkGen_B ([CLN16, Algorithm 5]). pkGen_B sums to 48M cc using the arithmetic benchmarks in (45). Benchmarking Alice's public key generation in the implementation of [CLN16] gives 52M cc.

Line(s)	**M**	**S**	**a**	**I**	Source
5	$5e_B$	$4e_B$	$9e_B$	-	[CLN16, Table 1]
12	$8x_B$	$4x_B$	$8x_B$	-	[CLN16, Table 1]
15	$2e_B$	$3e_B$	$14e_B$	-	[CH17, Table 3]
17	$4x_B$	$2x_B$	$4x_B$	-	[CH17, Table 3]
19-21	$12e_B$	$6e_B$	$12e_B$	-	[CH17, Table 3]
29	-	-	-	1	[CLN16, Table 1]
pkGen_B (total)	$(35.9 + 11.4\log(1.89\lambda))\lambda$	$(24.6 + 5.7\log(1.89\lambda))\lambda$	$(66.2 + 11.4\log(1.89\lambda))\lambda$	1	

Table 13: Bob shared key computation SKGen_B ([CLN16, Algorithm 9]). SKGen_B sums to 44M cc using the arithmetic benchmarks in (45). Benchmarking Bob's shared key computation in the implementation of [CLN16] gives 50M cc.

Line(s)	**M**	**S**	**a**	**I**	Source
1	-	-	-	1	[CLN16, Table 1]
2	$9e_B$	$6e_B$	$14e_B$	-	[CLN16, Table 1]
8	$8x_B$	$4x_B$	$8x_B$	-	[CLN16, Table 1]
11	$2e_B$	$3e_B$	$14e_B$	-	[CH17, Table 3]
13	$4x_B$	$2x_B$	$4x_B$	-	[CH17, Table 3]
19	-	-	-	1	[CLN16, Table 1]
SKGen_B (total)	$(20.8 + 11.4\log(1.89\lambda))\lambda$	$(17 + 5.7\log(1.89\lambda))\lambda$	$(52.9 + 11.4\log(1.89\lambda))\lambda$	2	

9.10.2 Estimating the memory usage in uncompressed SIDH

We start with estimating the size of the list pts used in Alice's public key generation ([CLN16, Algorithm 3]) and Alice's shared key computation ([CLN16, Algorithm 7]). Using the balanced strategy estimation it is easy to see that the length the list will never be larger than its length after the first iteration, which is $\log \frac{e_A}{2}$. We note that the actual maximum number of elements in pts is 8 in the SIDH library[22] (where $e_A = 372$), which is in line with our estimation. Each element in pts consists of two $\mathbb{F}_{p^2}$ elements. Hence we estimate the bit size of pts by

$$2\log\left(\frac{e_A}{2}\right)12\lambda \approx 2\log(e_A)12\lambda \approx 2\log(3\lambda)12\lambda$$

Besides the list pts, we need at any stage of Alice's computations a constant number of $\mathbb{F}_{p^2}$-variables for temporary variables, the public parameters of SIDH, public keys, shared keys, etc. Since the implementation of [CLN16], where p is 768 bits, used ≈ 16000 bytes of memory in Section 9.9.1, we know that this constant number of variables is always less than

$$\frac{16000 \cdot 8}{768 \cdot 2} \approx 84$$

We note that this is an overestimate, but it is sufficient for our purposes here. We therefore estimate that Alice's memory usage is less than

$$(84 + 2\log(3\lambda))12\lambda$$

bits. The corresponding analysis for Bob gives that his estimated memory usage is also less than

$$(84 + 2\log(3\lambda))12\lambda$$

9.10.3 Estimating the computational work in compression and decompression of public keys

Alice's compression of her public key pk_A consists of three main parts:

1. Find a 3^{e_B}-torsion basis $\{R_1, R_2\}$ for $E_A[3^{e_B}]$ ([CJL+16b, Section 3]).
2. Express $\phi_A(P_B)$ and $\phi_A(Q_B)$ in the basis $\{R_1, R_2\}$:
   (a) Use the Tate-pairing [Was08, Section 3.4] to find DLP-instances related to the coefficients used for expressing $\phi_A(P_B)$ and $\phi_A(Q_B)$ in the basis $\{R_1, R_2\}$ ([CJL+16b, Section 4]).
   (b) Use the Pohlig-Hellman algorithm to solve the DLP-instances from (a) ([CJL+16b, Section 5]).

Bob's compression consists of the corresponding three main parts.

As explained in [CJL+16b], the torsion basis generation in step 1 above is probabilistic. We start with analysing Bob's 2^{e_A}-torsion basis generation, which is done in the function ec_isogeny.generate_2_torsion_basis.c in the SIDH library. As discussed in [CJL+16b, Remark 3] and [CLN16, Section 9], when we try to find the second 2^{e_A}-torsion basis element R_2 we require that

$$x(3^{e_B-1}R_1) \neq x(3^{e_B-1}R_2) \tag{46}$$

[22] MAX_INT_POINTS_ALICE in SIDH_internal.h

Since there are three non-trivial elements in $E(\mathbb{F}_{p^2})[2]$ that together have three distinct x-coordinates, we would expect that (46) holds with probability $\frac{2}{3}$. This probability appears to be in line with what we observe if we let the seed of the random generator[23] in the SIDH library vary[24]. We therefore estimate that the loop (which tries a candidate R_2 in each iteration) in generate_2_torsion_basis executes 1.5 times in expectation. We present our estimations of the computational work in generate_2_torsion_basis in Table 14.

Table 14: generate_2_torsion_basis

Function (line)	**M**	**S**	**a**	**I**
xTPLe (688)	$8e_B$	$4e_B$	$8e_B$	-
xDBLe (689)	$4e_A$	$2e_A$	$4e_A$	-
xTPLe (700)	$1,5 \cdot 8e_B$	$1,5 \cdot 4e_B$	$1,5 \cdot 8e_B$	-
xDBLe (701)	$1,5 \cdot 4e_A$	$1,5 \cdot 2e_A$	$1,5 \cdot 4e_A$	-
Total	$10e_A + 20e_B$	$5e_A + 10e_B$	$10e_A + 20e_B$	-

We now analyse Alice's 3^{e_B}-torsion basis generation, which is done in the function ec_isogeny.generate_3_torsion_basis.c in the SIDH library.

1. As discussed in [CJL+16b, Section 3.3], the function get_3_torsion_elt may or may not find the first 3^{e_B}-torsion basis element by chance. We estimate the probability that get_3_torsion_elt does not find the first 3^{e_B}-torsion basis element with $\frac{1}{9} \approx 0.1$. This is justified in the following way. Let $\{R, S\}$ be a basis for $E_A(\mathbb{F}_{p^2})$, i.e., $\langle R, S\rangle = E_A(\mathbb{F}_{p^2})$[25]. Then picking a random point on $E_A(\mathbb{F}_{p^2})$ is the same thing as picking random integer coefficients α and β and letting the random point be $\alpha R + \beta S$. The function get_3_torsion_elt does not find the first torsion basis point R_1 if the random point it finds is in $[3](E(\mathbb{F}_{p^2}))$ (the image of the multiplication by 3-map restricted to $E(\mathbb{F}_p^2)$). The probability that the random element is in $[3](E(\mathbb{F}_{p^2}))$ is the same as the probability that both α and β are divisible by 3, which is $\frac{1}{3^2} = \frac{1}{9}$. Thus, with probability 0.1 we must still find the first 3^{e_B}-torsion basis element in the first iteration of the loop in generate_3_torsion_basis. This estimation appears to be in line with what we observe if we let the seed of the random generator in the SIDH library vary.

2. As discussed in [CJL+16b, Remark 3] and [CLN16, Section 9], when we try to find the second 3^{e_B}-torsion basis element R_2 we require that

$$x(3^{e_B-1}R_1) \neq x(3^{e_B-1}R_2) \tag{47}$$

   Since there are eight non-trivial elements in $E_A(\mathbb{F}_{p^2})[3]$ that together have four distinct x-coordinates, we would expect that (47) holds with probability $\frac{3}{4}$. However, if we let the seed of the random generator in the SIDH library vary, then the probability appears to be significantly higher than $\frac{3}{4}$. It is not clear to this author why this is the case. In our estimates in Table 14 we simply assume that (47) always holds. We also note that the points R_1 and R_2 are generated differently when searching for a 3^{e_B}-torsion basis compared to when searching for a 2^{e_A}-torsion basis.

We present our estimations of the computational work in generate_3_torsion_basis in Table 15.

[23]The C-library function rand() is used by default in the SIDH library for testing purposes.
[24]We use srand(time(NULL)).
[25]Such points exist since $E_A(\mathbb{F}_{p^2}) \cong \mathbb{Z}_{p+1} \oplus \mathbb{Z}_{p+1}$.

Table 15: generate_3_torsion_basis

Function (line)	**M**	**S**	**a**	**I**
get_3_torsion_elt (891)	$4e_A + 8e_B$	$2e_A + 4e_B$	$4e_A + 8e_B$	-
xDBLe (952)	$1.1 \cdot 4e_A$	$1.1 \cdot 2e_A$	$1.1 \cdot 4e_A$	-
xTPLe (967)	$1.1 \cdot 8e_B$	$1.1 \cdot 4e_B$	$1.1 \cdot 8e_B$	-
Total	$8.4e_A + 16.8e_B$	$4.2e_A + 8.4e_B$	$8.4e_A + 16.8e_B$	-

As explained in [CJL+16b, Section 4], Alice uses the Tate pairing as the first step in computing the coefficients of $\phi_A(P_B)$ and $\phi_A(Q_B)$ in the 3^{e_B}-torsion basis which she found above. This is done in the function ec_isogeny.Tate_pairings_3_torsion.c which contains e_B iterations of a loop with each iteration containing two calls to tpl_and_parabola and five calls to cube_and_absorb_parab. By [CJL+16b, Tripling-and-parabola operations], tpl_and_parabola costs $19\mathbf{M} + 6\mathbf{S} + 21\mathbf{a}$ (we count subtractions as additions) and cube_and_absorb_parab costs $10\mathbf{M} + 2\mathbf{S} + 4\mathbf{a}$. The total estimated cost for Tate_pairings_3_torsion is therefore

$$e_B(2(19\mathbf{M} + 6\mathbf{S} + 21\mathbf{a}) + 5(10\mathbf{M} + 2\mathbf{S} + 4\mathbf{a})) = 86e_B\mathbf{M} + 22e_B\mathbf{S} + 62e_B\mathbf{a}$$

The corresponding analysis for Bob's Tate-pairing computations in ec_isogeny.Tate_pairings_2_torsion.c gives the total estimated cost

$$e_A(2(9\mathbf{M} + 5\mathbf{S} + 8\mathbf{a}) + 5(5\mathbf{M} + 2\mathbf{S} + 3\mathbf{a})) = 43e_A\mathbf{M} + 20e_A\mathbf{S} + 31e_A\mathbf{a}$$

[CJL+16b, Remark 2] notes that this part is the main bottleneck of the entire compression algorithm for their parameters.

The second step in computing coefficients in the torsion basis is to use the Pohlig-Hellman algorithm to solve the DLP instances in μ_{ℓ^e} that arise from the Tate-pairing computations [CJL+16b, Section 5]. We let $\mathbf{L}$ denote the cost of an exponentiation with ℓ in μ_{ℓ^e}, i.e., a squaring in the case $\mu_{2^{e_A}}$ and a cubing in the case $\mu_{3^{e_B}}$. [CJL+16b] use a "windowed" variant of the Pohlig-Hellman algorithm, we refer to [CJL+16b, Section 5.4] for details. By the analysis of the windowed Pohlig-Hellman algorithm in [CJL+16b, Section 5.4] we know that the estimated cost when using n windows is

$$F_{A,n} \approx \frac{1}{2}e(n+1)\left(e^{\frac{1}{n+1}} - 1\right)\mathbf{L} + \frac{n+1}{2}e\log(\ell)\mathbf{M} + (n+1)e\log(\ell)\mathbf{S}$$

In Alice's case, where we are solving DLP instances in $\mu_{3^{e_B}}$, we get (using that $\log 3 \approx 1.54$ and $\mathbf{L} = 0.67\mathbf{M} + 0.5\mathbf{S}$ [CJL+16b, Sections 5.1 and 5.6])

$$F_{A,n} \approx \frac{1}{2}e_B(n+1)\left(e_B^{\frac{1}{n+1}} - 1\right)(0.67\mathbf{M} + 0.5\mathbf{S}) + \frac{n+1}{2}e_B 1.54\mathbf{M} + (n+1)e_B 1.54\mathbf{S} \approx$$

$$\frac{1}{2}(1.89\lambda)(n+1)\left((1.89\lambda)^{\frac{1}{n+1}} - 1\right)(0.67\mathbf{M}+0.5\mathbf{S}) + \frac{n+1}{2}(1.89\lambda)1.54\mathbf{M} + (n+1)(1.89\lambda)1.54\mathbf{S}$$

We let $n_{A,0}$ be the non-negative integer n that minimises $F_{A,n}$ for given parameters λ, $\mathbf{M}$ and $\mathbf{S}$. For the parameters in [CJL+16b], this value is approximately 3 [CJL+16b, Section 5.5-5.6]. Similarly, we let $n_{B,0}$ be the non-negative integer n that minimises the corresponding expression for Bob (who is solving DLP instances in $\mu_{2^{e_A}}$)

$$F_{B,n} \approx \frac{1}{2}(3\lambda)(n+1)\left((3\lambda)^{\frac{1}{n+1}} - 1\right)\mathbf{S} + \frac{n+1}{2}(3\lambda)\mathbf{M} + (n+1)(3\lambda)\mathbf{S}$$

Alice and Bob each solve four DLP instances.

It follows from the discussion above that the estimated total cost of Alice's public key compression Compr_A is

$$(8.4e_A{+}16.8e_B)\mathbf{M}{+}(4.2e_A{+}8.4e_B)\mathbf{S}{+}(8.4e_A{+}16.8e_B)\mathbf{a}{+}86e_B\mathbf{M}{+}22e_B\mathbf{S}{+}62e_B\mathbf{a}{+}4F_{A,n_{A,0}} \approx$$

$$(220\mathbf{M} + 70\mathbf{S} + 174\mathbf{a})\lambda + 4F_{A,n_{A,0}}$$

The estimate Compr_A gives 99M cc for the parameters in [CJL+16b] where we used $n_{A,0} = 3$ as in [CJL+16b, Section 5.6]. Benchmarking the same computation in the SIDH library gives 107M cc. The main costs in the compression that are ignored in our estimate are the functions (get_X_on_curve, sqrt_Fp2_frac, is_cube_Fp2, etc) in the torsion basis generation that consist of operations in $\mathbb{F}_p$. The number of times these functions are executed in the torsion basis generation are random variables that have some small and constant expected value. In the SIDH library these functions costed roughly 6M cc in one compression this author observed. The total estimated cost of Bob's compression Compr_B is

$$(10e_A{+}20e_B)\mathbf{M}{+}(5e_A{+}10e_B)\mathbf{S}{+}(10e_A{+}20e_B)\mathbf{a}{+}43e_A\mathbf{M}{+}20e_A\mathbf{S}{+}31e_A\mathbf{a}{+}4F_{B,n_{B,0}} \approx$$

$$(197\mathbf{M} + 94\mathbf{S} + 161\mathbf{a})\lambda + 4F_{B,n_{B,0}}$$

The estimate Compr_B gives 101M cc for the parameters in [CJL+16b] where we used $n_{B,0} = 3$ as in [CJL+16b, Section 5.5]. Benchmarking the same computation in the SIDH library gives 110M cc.

Alice's decompression of Bob's public key starts with the same 2^{e_A}-torsion basis generation as Bob performed in his compression above. In Table 14 we estimated this work with

$$(10e_A + 20e_B)\mathbf{M} + (5e_A + 10e_B)\mathbf{S} + (10e_A + 20e_B)\mathbf{a} \tag{48}$$

After that Alice performs a ladder function in mont_twodim_scalarmult to compute a generator of the kernel of the isogeny in her shared key computation. Work similar to this is done in the shared key computation in uncompressed ephemeral SIDH. [CJL+16b] instead counts this as part of the decompression in compressed ephemeral SIDH. As a result the shared key computation is faster in compressed ephemeral SIDH than it is in the uncompressed variant. To make our presentation in Section 9.10.5 simpler, we estimate Alice's work Decompr_A when decompressing Bob's key with only that in (48) and we let the estimate of the work in Alice's shared key computation be the same in compressed ephemeral SIDH and uncompressed ephemeral SIDH. Hence

$$\text{Decompr}_A \approx (67.8\mathbf{M} + 33.9\mathbf{S} + 67.8\mathbf{a})\lambda$$

For the parameters of [CJL+16b] our estimate gives $\text{Decompr}_A + \text{SKGen}_A = 65$M cc. Benchmarking the decompression and shared key computation in the implementation of [CJL+16b] gives 73.9M cc. We make the corresponding choices in the estimation of the work in Bob's decompression of Alice's key and get the estimate

$$\text{Decompr}_B = (8.4e_A + 16.8e_B)\mathbf{M} + (4.2e_A + 8.4e_B)\mathbf{S} + (8.4e_A + 16.8e_B)\mathbf{a} \approx$$

$$(57\mathbf{M} + 28.5\mathbf{S} + 57\mathbf{a})\lambda$$

For the parameters of [CJL+16b] our estimate gives $\text{Decompr}_B + \text{SKGen}_B = 62$M cc. Benchmarking the decompression and shared key computation in the implementation of [CJL+16b] gives 72M cc.

9.10.4 Estimating the memory usage in compression and decompression of public keys

We start with estimating the size of the lookup tables that Alice needs for solving the DLP instances in $\mu_{3^{e_B}}$. These contain precomputations of the g_i used for each window in [CJL+16b, Section 5.3]. Using (7) in [CJL+16b], the lookup table for window w_j has size

$$\frac{w_{j-1}}{w_j} = e_B^{\frac{1}{n_{A,0}+1}}$$

for $1 \leq j \leq n_{A,0}$ where $w_0 = e_B$. Besides these lookup tables we need a constant number of $\mathbb{F}_{p^2}$-variables at any point of Alice's computations. Since the implementation of [CJL+16b], where p is 768 bits, used ≈ 27000 bytes of memory in Section 9.9.1, we know that this constant number of variables is always less than

$$\frac{27000 \cdot 8}{768 \cdot 2} \approx 140$$

We note that this is an overestimate, but it is sufficient for our purposes here. The total estimated memory usage of Alice is therefore less than

$$(140 + n_{A,0} e_B^{\frac{1}{n_{A,0}+1}})12\lambda \approx (140 + n_{A,0}(1.89\lambda)^{\frac{1}{n_{A,0}+1}})12\lambda$$

bits, where $n_{A,0}$ will be some small non-negative integer. The corresponding analysis for Bob gives that his estimated memory usage is less than

$$(140 + n_{B,0}(3\lambda)^{\frac{1}{n_{B,0}+1}})12\lambda$$

9.10.5 Overview of estimates

In this section we use the estimations from the last sections to give estimates of computational work, memory usage and key size in several variants of SIDH as a function of the quantum security in bits λ. For the static key variants we assume that one of the proposals from Section 9.8.1 or Section 9.8.4 is used. We present our results in Table 16.

Table 16: Overview of estimates for different variants of SIDH. The cost of the operations in the column Computational work are given in Table 17.

		Computational work	Memory usage	Key size
Ephemeral	Alice	pkGen_A+SKGen_A	$(84+2\log(3\lambda))12\lambda$	36λ
	Bob	pkGen_B+SKGen_B	$(84+2\log(3\lambda))12\lambda$	36λ
Static	Alice	pkGen_A+SKGen_A	$(84+2\log(3\lambda))12\lambda$	$(36+2)\lambda$
	Bob	SKGen_B+pkGen_A	$(84+2\log(3\lambda))12\lambda$	36λ
Compressed ephemeral	Alice	pkGen_A+Compr_A+Decompr_B+SKGen_A	$(140+n_{A,0}(1.89\lambda)^{\frac{1}{n_{A,0}+1}}+2\log(3\lambda))12\lambda$	21λ
	Bob	pkGen_B+Compr_B+Decompr_B+SKGen_B	$(140+n_{B,0}(3\lambda)^{\frac{1}{n_{B,0}+1}}+2\log(3\lambda))12\lambda$	21λ
Compressed static	Alice	pkGen_A+Compr_A+SKGen_A	$(140+n_{A,0}(1.89\lambda)^{\frac{1}{n_{A,0}+1}}+2\log(3\lambda))12\lambda$	$(21+2)\lambda$
	Bob	Decompr_B+SKGen_B+pkGen_A+Compr_A	$(140+n_{A,0}(1.89\lambda)^{\frac{1}{n_{A,0}+1}}+2\log(3\lambda))12\lambda$	36λ

Table 17: Cost of operations.

pkGen_A	$((42+10.5\log(1.5\lambda))\mathbf{M}+(27+4.5\log(1.5\lambda))\mathbf{S}+(60+10.5\log(1.5\lambda))\mathbf{a})\lambda$
SKGen_A	$((27+10.5\log(1.5\lambda))\mathbf{M}+(24+4.5\log(1.5\lambda))\mathbf{S}+(48+10.5\log(1.5\lambda))\mathbf{a})\lambda$
pkGen_B	$((35.9+11.4\log(1.89\lambda))\mathbf{M}+(24.6+5.7\log(1.89\lambda))\mathbf{S}+(66.2+11.4\log(1.89\lambda))\mathbf{a})\lambda$
SKGen_B	$((20.8+11.4\log(1.89\lambda))\mathbf{M}+(17+5.7\log(1.89\lambda))\mathbf{S}+(52.9+11.4\log(1.89\lambda))\mathbf{a})\lambda$
$F_{A,n}$	$\frac{1}{2}(1.89\lambda)(n+1)((1.89\lambda)^{\frac{1}{n+1}}-1)(0.67\mathbf{M}+0.5\mathbf{S})+\frac{n+1}{2}(1.89\lambda)1.54\mathbf{M}+(n+1)(1.89\lambda)1.54\mathbf{S}$
$F_{B,n}$	$\frac{1}{2}(3\lambda)(n+1)((3\lambda)^{\frac{1}{n+1}}-1)\mathbf{S}+\frac{n+1}{2}(3\lambda)\mathbf{M}+(n+1)(3\lambda)\mathbf{S}$
Compr_A	$(220\mathbf{M}+70\mathbf{S}+174\mathbf{a})\lambda+4F_{A,n_{A,0}}$
Compr_B	$(197\mathbf{M}+94\mathbf{S}+161\mathbf{a})\lambda+4F_{B,n_{B,0}}$
Decompr_A	$(67.8\mathbf{M}+33.9\mathbf{S}+67.8\mathbf{a})\lambda$
Decompr_B	$(57\mathbf{M}+28.5\mathbf{S}+57\mathbf{a})\lambda$

Under the assumption that the $\mathbb{F}_{p^2}$-arithmetic operations $\mathbf{M}$ and $\mathbf{S}$ run in time $\Theta(\lambda^{1.6})$[26], the time complexity of all SIDH variants grows roughly proportional to $\lambda^{2.6}$. Here we ignore the factors $\log\lambda$ and $(n+1)\lambda^{\frac{1}{n+1}}$ (keeping in mind that $n\in\{n_{A,0},n_{B,0}\}$ grows as λ grows). By similar reasoning, the memory grows roughly proportional to λ. We note that under the assumption that the computational work grows proportional to $\lambda^{2.6}$, then going, for example, from $\lambda=128$ to $\lambda=85$ speeds up the computations with roughly a factor 2.9.

9.11 Random bit usage

The number of random bits used in any variant SIDH (compressed, ephemeral, static, etc) is at most the number of bits needed to randomly pick a normalised secret key in Section 6.3. As explained in that section there are $(\ell_A+1)\ell_A^{e_A-1}\approx 2^{3\lambda}$ normalised secret keys for Alice and $(\ell_B+1)\ell_B^{e_B-1}\approx 2^{3\lambda}$ normalised secret keys for Bob. Thus, each party needs approximately 3λ random bits in ephemeral SIDH. By inspecting the implementation in the SIDH library v2.0, we see that the compression implemented in [CJL+16b] requires no extra random bits. In static key SIDH Bob will only need as many random bits as the bit length of his seed $x||s$ in Section 9.8.4 (that seed has the same bit length as the seed r_B of Section 9.8.1). In Section 9.8.4 we suggested using a seed of bit length 2λ with the asymptotical query complexity of Grover's algorithm in mind. As discussed in the next section, the bit length of the seed could possibly be lowered to 1.5λ.

[26]Here we assume that numbers are kept in Montgomery representation and that Karatsuba's method is used for multiplication. Note that which multiplication method that should be used depends on λ.

We note that the random bit usage in SIDH is low compared to many other PQ-cryptosystems by [Kin17, Table 5.8]. Note that [Kin17, Table 5.8] lists the total random bit usage by both parties together.

9.12 How small can we make λ in practice?

Our main interest in SIDH is in its small public key sizes. Since these grow linearly in λ in all SIDH variants we have studied it is interesting to discuss how small we can make λ in practice to get the smallest possible SIDH keys. When discussing the claw attack from Section 8.1, we as usual assume that it is Alice's public key that is being attacked. Everything we say holds equally well for Bob's public key as well. From our discussion in Section 8.1 we know the classical claw algorithm must either build a hash table of size at least

$$\sqrt{(\ell_A+1)\ell_A^{e_A-1}} \approx 2^{1.5\lambda}$$

or query a hash table at least

$$\frac{\sqrt{(\ell_A+1)\ell_A^{e_A-1}}}{2} \approx 2^{1.5\lambda}$$

times in expectation. Thus, without loss of generality we can assume that the concrete classical claw attack time complexity is greater than $2^{1.5\lambda}$. We also saw that it is reasonable to assume that the classical claw attack is efficiently parallelisable. Setting $\lambda = 85$ then gives 128 bits of classical security which is secure even against an efficiently parallelisable attack. According to [CLN16, SIDH history and security] the classical complexity $\mathrm{O}\big(2^{1.5\lambda}\big)$ is optimal for solving a black box claw problem of this size. Assuming there is a single claw (which is the situation we will be in in SIDH with very high probability), one way to understand that this complexity is indeed optimal is to note that the function with the largest domain in our claw problem has domain of size at least $2^{1.5\lambda}$. We then need at least $2^{1.5\lambda-1}$ queries in expectation to find the claw. This is similar to the situation for classical algorithms for solving the problem solved by Grover's algorithm in Section 4.2. We note however that this analysis is based on black box functions, future cryptanalysis of SIDH or insights about the isogenies used in SIDH may change the classical security. A related remark is the following. In Section 8.1 we saw that queries to the function f (g) of the claw problem corresponded to computing end-nodes of non-backtracking $e_A/2$-walks in $\mathcal{G}_{\ell_A}$ from $[E_0]$ ($[E_A]$). We note that a classical claw attack does not require an attacker to compute a complete $e_A/2$-walk in $\mathcal{G}_{\ell_A}$ for each query. One efficient way appears to be to explore the end-nodes of all non-backtracking $e_A/2$-walks from $[E_0]$ (or $[E_A]$) in a depth-first-search manner. For example, suppose that $\ell_A = 2$. Then $3(2^{e_A/2}-1)$ edges needs to be computed to explore all $3 \cdot 2^{e_A/2-1}$ end-nodes. This gives an average cost of computing two edges in $\mathcal{G}_{\ell_A}$ per end-node (query).

As defined in [CCJ+16, p. 6]: *"an algorithm is said to have 128 bits of security if the difficulty of attacking it with a classical computer is comparable to the time and resources required to brute-force search for a 128-bit cryptographic key"*. We note that when using $\lambda = 85$ we have more than 128 bits of classical security since the attack involves building a very large hash table which is not necessary in an ordinary brute-force search of a 128-bit cryptographic key. However, since the attacker can choose the size of the hash table and since keys of size 112 bits (which is the standard size below 128) will likely be phased out around 2031 in NIST recommendations [CCJ+16, p. 6] this author believes that $\lambda = 85$ is

perhaps the most interesting security level below the standard $\lambda = 128$ that is considered in many implementations of SIDH. If one estimates the largest size of a hash table that an attacker could possibly build, then it is possible that security levels below $\lambda = 85$ could be interesting as well. Assuming the largest hash table an attacker can build has size 2^{80} bits (assuming that each entry only takes one bit of storage)[27], then we have about 128 bits of classical security at $\lambda = 70$[28]. Going from $\lambda = 85$ to $\lambda = 70$ would reduce key sizes with 18%. Of course, before making λ smaller, learning more about the concrete complexity of the quantum claw attack is crucial.

With respect to quantum attacks and the quantum claw attack in particular, we would like to know more. It is an interesting idea for future work to try to determine the concrete time complexity of the (best) quantum claw algorithm at, say, $\lambda = 85$ bits. Ideally, we would like to have estimates of the number of gates in a concrete quantum circuit for solving the problem as well as the depth of such a circuit. Another approach could be to determine the constant in the asymptotic lower bounds for the quantum query complexity of the claw problem. However, that does not necessarily say anything about quantum circuit depth or whether any of the queries can be run in parallel. Can anything be said in particular in the special case of the claw problem that we are interested in? That is, when we know that there is a unique claw and we just want to find it. Regardless, any approach most likely include estimating the size and depth of quantum circuits for computing walks in supersingular isogeny graphs. It would also be very interesting to know if the quantum claw attack can be efficiently parallelised. The naive parallelisation that we discussed in Section 8.1 is highly inefficient.

We finally note that the bit size of the seeds used in the static key SIDH key exchanges of Sections 9.8.1 and 9.8.4 could maybe be lowered from 2λ to 1.5λ if the concrete complexity (and not only the asymptotic complexity) of Grover's attack against the seed can be estimated. This is not a significant optimisation since 0.5λ is less than 3% of Bobs $(21 + 2)\lambda$ bit public key in compressed static key SIDH (and less than 2% of Bobs $(36 + 2)\lambda$ bit public key in uncompressed static key SIDH). Going lower than 1.5λ would effect the classical security since the classical attacker could then choose to attack Bob's seed instead of trying to solve the claw problem that Bob's public key gives rise to.

[27] This is roughly one million times the estimated data content of the deep web (https://www.wolframalpha.com/input/?i=2%5E80+bits+in+exabytes, access date 25-07-2017).

[28] Since $3\lambda - 80 = 210 - 80 \approx 128$.

10 Conclusions

In our simulations in Section 7.3 we saw that the number of distinct $j(E_A)$ that Alice can get to by following non-backtracking walks given by isogenies $\phi_A : E_0 \to E_A$ is very close to the number of distinct elements we expect to get when choosing $(\ell_A + 1)\ell_A^{e_A - 1}$ vertices uniformly at random (with repetition) from the vertex set $V(\mathcal{G}_{\ell_A})$. A similar statement was true for Bob. Since $(\ell_A + 1)\ell_A^{e_A - 1}$ is much smaller than $\#V(\mathcal{G}_{\ell_A}) \approx \ell_A^{2e_A}/12$ we could never expect uniform mixing as in Theorem 7.1.4. However our simulation results show that, even for very small examples of p, the distribution of $j(E_A)$ appears to be essentially the best possible. In Section 7.4 we saw that the distribution of $j(E_{AB})$ also appears to be essentially the best possible, even for small examples of p. The shared key $j(E_{AB})$ appears to follow the same distribution as a randomly chosen element from the set S in Estimation 7.4.1. We are not the first to make the simple observation that the isogenies of SIDH are non-backtracking. But we ask if it could possibly be used in combination with results for non-backtracking random walks in expander graphs such as those in [ABLS07] to explain the strong results we see in our simulations? The fact that the isogenies in SIDH are non-backtracking inspired us to describe a heuristic reduction from the CSSI-problem to the DSSI-problem in Section 7.6. We gave convincing simulation results as evidence that the reduction works. It is interesting to note that the isogeny graphs $\mathcal{G}_{\ell_A}$ and $\mathcal{G}_{\ell_B}$ behave "sufficiently random" for the reduction to work, see Section 7.6.3. This adds further to our confidence in assumptions such as the one made in [GPST16, p. 12 footnote 1].

Impressive work in speeding up SIDH has been done in for example [CLN16]. However, there is still a gap to many PQ-cryptosystems as we observed in our comparison in Section 9.9.2. SIDH is still roughly somewhere between 10-500 times slower than many competing PQ-cryptosystems. A single server-thread can perform roughly 30-40 ephemeral key exchanges at 128-bits of security per second. It is up to the server administrator if the savings in bandwidth that SIDH offer are worth it. Given the structure of SIDH computations seen in Section 9.10: a large number of small degree (elliptic curve) scalar multiplications and isogeny evaluations that each contain some constant number of field arithmetic operations; it is plausible that these benchmarks will remain fairly stable unless significant theoretical breakthroughs in isogeny computations are made. Of course any such breakthrough could also possibly challenge our belief in the security of SIDH. This author believes that future speedups in SIDH computation times will be modest. Another way to say this is: the server administrator who would not be interested in SIDH even if computation times could be speed up by, say, a factor 10 will likely never be interested in SIDH.

In Sections 9.9.1 and 9.11 we saw that memory usage and random bit usage is modest in all the variants of SIDH. This makes SIDH a good candidate for devices where such resources are limited. However, since SIDH is relatively slow on a desktop PC it will also be fairly slow on any such limited device. Thus, the critical question will likely be how often the limited device needs to engage in a key exchange with the server.

Due to its slow computations, SIDH will likely remain a special-purpose cryptosystem. Key compression for SIDH is therefore very interesting since it was the small keys of the original system that made it interesting to begin with. For the same reason static key SIDH is also very interesting since it roughly cuts communication between the parties in the key exchange in half. [KLM+15] gave a proposal for a static key variant of SIDH and in Section 9.8.4 we gave a similar proposal that comes with a formal IND-CCA2 security proof in the random oracle model. While both proposals increase Alice's computation

time (since she needs to recompute Bob's public key), seeing how SIDH will likely be a special-purpose system the big drawback of both proposals might be the 2λ bit encrypted seed sent alongside Bob's public key to Alice. On the other hand, the size of the encrypted seed is only 5.6% of the 36λ bit public key sent by Bob to Alice in uncompressed SIDH and only 9.5% of the 21λ bit public key by him in compressed SIDH. Maybe this will be an acceptable price to pay for IND-CCA2 security.

For the setting where we really want to keep bandwidth to a minimum it is a tempting option to lower the quantum security λ from 128 bits to, say, 85 bits. Suppose that we use compressed static key SIDH at this security level. [CJL+16b, p. 19] estimate that compression gives a 2.4 factor slowdown of SIDH, at the same time we estimated[29] in Section 9.10.5 that going from $\lambda = 128$ to $\lambda = 85$ speeds up SIDH with roughly a factor 2.9. Assuming that we also use one of the proposals for static key SIDH that we studied in Sections 9.8.1 and 9.8.4, this would mean that our compressed static key SIDH with $\lambda = 85$ runs at roughly the same speed as uncompressed ephemeral SIDH with $\lambda = 128$. As we saw in Section 5, each party's computations in uncompressed ephemeral SIDH with $\lambda = 128$ currently take roughly 25-30 ms on a modern desktop PC. Ignoring the communication cost when Alice publishes her static key, the only communication needed for one key exchange would be the $\frac{(21+2)\lambda}{8} \approx 244$ bytes large key encapsulation sent from Bob to Alice. With our current knowledge in SIDH cryptanalysis, $\lambda = 85$ means that we have 128 bits of classical security. As is well known this is completely secure in the classical setting even though the best known classical attack appears to be efficiently parallelisable. However, with regards to quantum attackers at security levels below $\lambda = 128$ we would like to know more. Ideally we would like something similar to the analysis for AES done in [GLRS16]: if we set $\lambda = 85$, then what is the concrete number of quantum operations necessary to attack us by solving the claw problem on a quantum computer? We would also like to know the depth of such a circuit. SIDH might gain from its slow computations when the black box-functions in the claw problem are replaced with quantum circuits for computing walks in the isogeny graphs $\mathcal{G}_{\ell_A}$ and $\mathcal{G}_{\ell_B}$. It would also be very interesting to know if the quantum attack can be efficiently parallelised. The naive method of running the quantum attack in parallel is highly inefficient as we saw in Section 8.1.

It appears that the most important future work is still building confidence in the security of SIDH. SIDH is still often considered to be a young cryptosystem in the literature [BL17, p. 5][CCJ+16, p. 4]. However, we saw in Sections 7.7 and 7.8 that the problem of computing isogenies between supersingular elliptic curves is older than SIDH. Regardless, the isogenies in SIDH are special cases, as mentioned in for example [DFJP14, Section 5.1] and [GPST16, Section 6], since they give relatively short walks in the isogeny graphs $\mathcal{G}_{\ell_A}$ and $\mathcal{G}_{\ell_B}$. There is also the open question of how the torsion points $\phi_A(P_B)$, $\phi_A(Q_B)$, $\phi_B(P_A)$ and $\phi_B(Q_A)$ effect the hardness of finding the parties secret isogenies. The recent contribution of [Pet17] that we mentioned in Section 8.3 is very interesting, it is the first passive attack that utilises the torsion points to attack certain "unbalanced" variants of SIDH. Whether these attacks can ever be generalised and applied to SIDH remains to be seen. One can safely say that more cryptanalysis is needed before considering using SIDH in a production environment. However, with regards to performance, SIDH is already today a good alternative for a special purpose low-bandwith PQ-KEM.

[29] This estimation was based on, among other things, an assumption about the time complexity of $\mathbb{F}_{p^2}$-arithmetic, see Section 9.10.5 for details.

10.1 Future work

Besides building confidence in the security of SIDH the most important ideas for future work that we have highlighted in this thesis are the following.

- **Estimating the computational work in the quantum claw attack against SIDH.** As discussed in the beginning of Section 10 it may be interesting to know how small we can make the quantum bit security λ. Hence we would like to have concrete estimates of the computational work in the quantum claw attack, and not just asymptotic query bounds. Two possible approaches to start with were mentioned in Section 9.12. Both approaches likely involve estimating the size of a quantum circuit for computing walks in $\mathcal{G}_{\ell_A}$ and $\mathcal{G}_{\ell_B}$. As explained in the beginning of Section 10, making λ as small as possible may be very interesting as the public key sizes grow linearly with λ in all variants of SIDH. A related question is whether the quantum claw attack can be efficiently parallelised.

- **Implementing compressed static key SIDH.** As explained in the beginning of Section 10, compressed static key SIDH is very interesting due to the minimal amount of data that needs to be sent between the parties of the key exchange. In Section 9.8.4 we proposed a KEM scheme built on SIDH that is secure in the sense of IND-CCA2 in the random oracle model. Given the complexity of an efficient compressed SIDH implementation such a compressed static key SIDH implementation should perhaps be built on top of an existing implementation of compressed ephemeral SIDH. Questions to answer include the following.

  - Which shortcuts can Alice take when compressing Bob's public key (as part of her validation of his public key)? One shortcut appears to be that she has already found the torsion basis. She did that when she decompressed Bob's public key.
  - It appears that an efficient SIDH implementation needs to target a specific prime (and therefore also a specific security level). Another question is therefore which security level(s) to target.
  - To this author's knowledge, the question posed on [KLM+15, p. 18] is still open: is there some better solution to the key validation problem than Alice recomputing Bob's public key? After the powerful attack in [GPST16], any such proposal may need to come with some sort of security proof.

  Several authors [BJS14, Section 6][Pet17, Section 5] discuss avoiding certain starting curves E_0 in SIDH. We note that in static key SIDH, Alice could generate a random curve E_0 using Theorem 7.1.4 and then make the curve E_0 (and the torsion points P_A, Q_A, P_B and Q_B on it) part of her public key pk_B. The secret random walk that generated E_0 would be discarded after E_0 has been generated. Note that this approach increases the size of Alice's static public key.

References

[ABB+] Daniel Augot, Lejla Batina, Daniel J Bernstein, Joppe Bos, Johannes Buchmann, Wouter Castryck, Orr Dunkelman, Tim Güneysu, Shay Gueron, Andreas Hülsing, et al. Initial recommendations of long-term secure post-quantum systems (2015). *URL: https://pqcrypto. eu. org/docs/initial-recommendations. pdf. Citations in this document*, 16.

[ABLS07] Noga Alon, Itai Benjamini, Eyal Lubetzky, and Sasha Sodin. Non-backtracking random walks mix faster. *Communications in Contemporary Mathematics*, 9(04):585–603, 2007.

[AFJ14] Reza Azarderakhsh, Dieter Fishbein, and David Jao. Efficient implementations of a quantum-resistant key-exchange protocol on embedded systems. Technical report, Technical report, 2014, URL: http://cacr. uwaterloo. ca/techreports/2014/cacr2014-20. pdf, 2014.

[AJK+16] Reza Azarderakhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi. Key compression for isogeny-based cryptosystems. In *Proceedings of the 3rd ACM International Workshop on ASIA Public-Key Cryptography*, pages 1–10. ACM, 2016.

[BB06] John A Beachy and William D Blair. *Abstract algebra*. Waveland Press, 2006.

[BBD09] Daniel J Bernstein, Johannes Buchmann, and Erik Dahmen. *Post-quantum cryptography*. Springer Science & Business Media, 2009.

[BDH+01] Harry Buhrman, Christoph Durr, Mark Heiligman, Peter Hoyer, Frédéric Magniez, Miklos Santha, and Ronald De Wolf. Quantum algorithms for element distinctness. In *Computational Complexity, 16th Annual IEEE Conference on, 2001.*, pages 131–137. IEEE, 2001.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss-a practical forward secure signature scheme based on minimal security assumptions. In *International Workshop on Post-Quantum Cryptography*, pages 117–129. Springer, 2011.

[BDPR98] Mihir Bellare, Anand Desai, David Pointcheval, and Phillip Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology—CRYPTO'98*, pages 26–45. Springer, 1998.

[Bea02] Stephane Beauregard. Circuit for shor's algorithm using 2n+ 3 qubits. *arXiv preprint quant-ph/0205095*, 2002.

[BF16] Joppe W. Bos and Simon Friedberger. Fast arithmetic modulo $2^x p^y \pm 1$. Cryptology ePrint Archive, Report 2016/986, 2016. `http://eprint.iacr.org/2016/986`.

[BGJGP05] Matthew H Baker, Enrique González-Jiménez, Josep González, and Bjorn Poonen. Finiteness results for modular curves of genus at least 2. *American Journal of Mathematics*, 127(6):1325–1387, 2005.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. *LATIN'98: Theoretical Informatics*, pages 163–169, 1998.

[BJS14] Jean-François Biasse, David Jao, and Anirudh Sankar. A quantum algorithm for computing isogenies between supersingular elliptic curves. In *International Conference in Cryptology in India*, pages 428–442. Springer, 2014.

[BL17] Daniel J. Bernstein and Tanja Lange. Post-quantum cryptography—dealing with the fallout of physics success. Cryptology ePrint Archive, Report 2017/314, 2017. `http://eprint.iacr.org/2017/314`.

[BLP08] Daniel J Bernstein, Tanja Lange, and Christiane Peters. Attacking and defending the mceliece cryptosystem. In *International Workshop on Post-Quantum Cryptography*, pages 31–46. Springer, 2008.

[Brö09] Reinier Bröker. Constructing supersingular elliptic curves. *J. Comb. Number Theory*, 1(3):269–273, 2009.

[BSS99] Ian F Blake, Gadiel Seroussi, and Nigel Smart. *Elliptic curves in cryptography*, volume 265. Cambridge university press, 1999.

[CCJ+16] Lily Chen, Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. *Report on post-quantum cryptography*. US Department of Commerce, National Institute of Standards and Technology, 2016.

[CH17] Craig Costello and Huseyin Hisil. A simple and compact algorithm for sidh with arbitrary degree isogenies. Cryptology ePrint Archive, Report 2017/504, 2017. `http://eprint.iacr.org/2017/504`.

[CJL+16a] Lily Chen, Stephen Jordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, and Daniel Smith-Tone. Report on post-quantum cryptography. *National Institute of Standards and Technology Internal Report*, 8105, 2016.

[CJL+16b] Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Renes, and David Urbanik. Efficient compression of sidh public keys. Cryptology ePrint Archive, Report 2016/963, 2016. `http://eprint.iacr.org/2016/963`.

[CJS14] Andrew Childs, David Jao, and Vladimir Soukharev. Constructing elliptic curve isogenies in quantum subexponential time. *Journal of Mathematical Cryptology*, 8(1):1–29, 2014.

[CLG09] Denis X Charles, Kristin E Lauter, and Eyal Z Goren. Cryptographic hash functions from expander graphs. *Journal of CRYPTOLOGY*, 22(1):93–113, 2009.

[CLN16] Craig Costello, Patrick Longa, and Michael Naehrig. Efficient algorithms for supersingular isogeny diffie-hellman. In *Annual Cryptology Conference*, pages 572–601. Springer, 2016.

[CS03] Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.

[DDLL13] Léo Ducas, Alain Durmus, Tancrède Lepoint, and Vadim Lyubashevsky. Lattice signatures and bimodal gaussians. In *Advances in Cryptology–CRYPTO 2013*, pages 40–56. Springer, 2013.

[DF17] Luca De Feo. Mathematics of isogeny based cryptography. 2017.

[DFJP14] Luca De Feo, David Jao, and Jérôme Plût. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. *Journal of Mathematical Cryptology*, 8(3):209–247, 2014.

[DG16] Christina Delfs and Steven D Galbraith. Computing isogenies between supersingular elliptic curves over {\ mathbb {F}} _p. *Designs, Codes and Cryptography*, 78(2):425–440, 2016.

[DH76] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[DPV06] Sanjoy Dasgupta, Christos H Papadimitriou, and Umesh Vazirani. *Algorithms*. McGraw-Hill, Inc., 2006.

[DS05] Jintai Ding and Dieter Schmidt. Rainbow, a new multivariable polynomial signature scheme. In *International Conference on Applied Cryptography and Network Security*, pages 164–175. Springer, 2005.

[FO99] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Crypto*, volume 99, pages 537–554. Springer, 1999.

[FO00] Eiichiro Fujisaki and Tatsuaki Okamoto. How to enhance the security of public-key encryption at minimum cost. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 83(1):24–32, 2000.

[Gal99] Steven D Galbraith. Constructing isogenies between elliptic curves over finite fields. *LMS Journal of Computation and Mathematics*, 2:118–138, 1999.

[Gal12] Steven D Galbraith. *Mathematics of public key cryptography*. Cambridge University Press, 2012.

[GGH97] Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In *Advances in Cryptology-CRYPTO'97: 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 1997. Proceedings*, page 112. Springer, 1997.

[GLRS16] Markus Grassl, Brandon Langenberg, Martin Roetteler, and Rainer Steinwandt. Applying grover's algorithm to aes: quantum resource estimates. In *International Workshop on Post-Quantum Cryptography*, pages 29–43. Springer, 2016.

[GPST16] Steven D Galbraith, Christophe Petit, Barak Shani, and Yan Bo Ti. On the security of supersingular isogeny cryptosystems. In *Advances in Cryptology–ASIACRYPT 2016: 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part I 22*, pages 63–91. Springer, 2016.

[Gro96] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[GV17] Steven D. Galbraith and Frederik Vercauteren. Computational problems in supersingular elliptic curve isogenies. Cryptology ePrint Archive, Report 2017/774, 2017. `http://eprint.iacr.org/2017/774`.

[GW17] Alexandre Gélin and Benjamin Wesolowski. Loop-abort faults on supersingular isogeny cryptosystems. Cryptology ePrint Archive, Report 2017/374, 2017. `http://eprint.iacr.org/2017/374`.

[HHGP+] J Hoffstein, N Howgrave-Graham, J Pipher, JH Silverman, and W Whyte NTRUSign. Digital signatures using the ntru lattice. *Preliminary draft*, 2.

[HPS98] Jeffrey Hoffstein, Jill Pipher, and Joseph Silverman. Ntru: A ring-based public key cryptosystem. *Algorithmic number theory*, pages 267–288, 1998.

[HPSS08] Jeffrey Hoffstein, Jill Pipher, Joseph H Silverman, and Joseph H Silverman. *An introduction to mathematical cryptography*, volume 1. Springer, 2008.

[JDF11] David Jao and Luca De Feo. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies. In *International Workshop on Post-Quantum Cryptography*, pages 19–34. Springer, 2011.

[KAJMK16a] Brian Koziel, Reza Azarderakhsh, David Jao, and Mehran Mozaffari-Kermani. On fast calculation of addition chains for isogeny-based cryptography. *Proc. eprint 2016*, 1045:1–20, 2016.

[KAJMK16b] Brian Koziel, Reza Azarderakhsh, David Jao, and Mehran Mozaffari-Kermani. On fast calculation of addition chains for isogeny-based cryptography. *Proc. eprint 2016*, 1045:1–20, 2016.

[KAKJ16] Brian Koziel, Reza Azarderakhsh, Mehran Mozaffari Kermani, and David Jao. Post-quantum cryptography on fpga based on isogenies on elliptic curves. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2016.

[KAMK16] Brian Koziel, Reza Azarderakhsh, and Mehran Mozaffari-Kermani. Fast hardware architectures for supersingular isogeny diffie-hellman key exchange on fpga. In *Progress in Cryptology–INDOCRYPT 2016: 17th International Conference on Cryptology in India, Kolkata, India, December 11-14, 2016, Proceedings 17*, pages 191–206. Springer, 2016.

[Kin17] Marcus Kindberg. A usability study of post-quantum algorithms. 2017.

[KLM+15] Daniel Kirkwood, Bradley C Lackey, John McVey, Mark Motley, Jerome A Solinas, and David Tuller. Failure is not an option: Standardization issues for post-quantum key agreement. In *Talk at NIST workshop on Cybersecurity in a Post-Quantum World: http://www. nist. gov/itl/csd/ct/post-quantum-crypto-workshop-2015. cfm*, 2015.

[Koc95] Paul C Kocher. Cryptanalysis of diffie-hellman, rsa, dss, and other systems using timing attacks, 1995.

[Koh96] David Kohel. *Endomorphism rings of elliptic curves over finite fields*. PhD thesis, University of California at Berkeley, 1996.

[McE78] Robert J McEliece. A public-key cryptosystem based on algebraic. *Coding Thv*, 4244:114–116, 1978.

[Mes86] Jean-Francois Mestre. La méthode des graphes. exemples et applications. In *Proceedings of the international conference on class numbers and fundamental units of algebraic number fields (Katata)*, pages 217–242, 1986.

[MSS07] Frédéric Magniez, Miklos Santha, and Mario Szegedy. Quantum algorithms for the triangle problem. *SIAM Journal on Computing*, 37(2):413–424, 2007.

[NR06] Phong Q Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of ggh and ntru signatures. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 271–288. Springer, 2006.

[NX09] Harald Niederreiter and Chaoping Xing. *Algebraic geometry in coding theory and cryptography*. Princeton University Press, 2009.

[Pat96] Jacques Patarin. Hidden fields equations (hfe) and isomorphisms of polynomials (ip): Two new families of asymmetric algorithms. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–48. Springer, 1996.

[Pei14] Chris Peikert. Lattice cryptography for the internet. In *International Workshop on Post-Quantum Cryptography*, pages 197–219. Springer, 2014.

[Pet09] Christophe Petit. Cryptographic hash functions from expander graphs, phd thesis. 2009.

[Pet17] Christophe Petit. Faster algorithms for isogeny problems using torsion point images. Cryptology ePrint Archive, Report 2017/571, 2017. `http://eprint.iacr.org/2017/571`.

[PZ03] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):34, 2009.

[San15] Anirudh Sankar. Classical and quantum algorithms for isogeny-based cryptography. 2015.

[Sil09] Joseph H Silverman. *The arithmetic of elliptic curves*, volume 106. Springer Science & Business Media, 2009.

[SM16] Douglas Stebila and Michele Mosca. Post-quantum key exchange for the internet and the open quantum safe project. Technical report, Cryptology ePrint Archive, Report 2016/1017, 2016. http://eprint. iacr. org/2016/1017, 2016.

[Sto10] Anton Stolbunov. Constructing public-key cryptographic schemes based on class group action on a set of isogenous elliptic curves. *Adv. in Math. of Comm.*, 4(2):215–235, 2010.

[Sut15] Andrew Sutherland. *18.783 Elliptic Curves (Spring 2015) Lecture Notes*. Massachusetts Institute of Technology: MIT OpenCourseWare, 2015.

[Tan09] Seiichiro Tani. Claw finding algorithms using quantum walk. *Theoretical Computer Science*, 410(50):5285–5297, 2009.

[Tat66] John Tate. Endomorphisms of abelian varieties over finite fields. *Inventiones mathematicae*, 2(2):134–144, 1966.

[Ti17] Yan Bo Ti. Fault attack on supersingular isogeny cryptosystems. Cryptology ePrint Archive, Report 2017/379, 2017. `http://eprint.iacr.org/2017/379`.

[Vaz05] UV Vazirani. lecture notes on qubits, quantum mechanics, and computers for chem/cs/phys191. *University of California, Berkeley*, 2005.

[Vél71] Jacques Vélu. Isogénies entre courbes elliptiques. *CR Acad. Sci. Paris Sér. AB*, 273:A238–A241, 1971.

[Was08] Lawrence C Washington. *Elliptic curves: number theory and cryptography*. CRC press, 2008.

[Zal99] Christof Zalka. Grover's quantum searching algorithm is optimal. *Physical Review A*, 60(4):2746, 1999.

[Zha05] Shengyu Zhang. Promised and distributed quantum search. In *International Computing and Combinatorics Conference*, pages 430–439. Springer, 2005.

Appendix

A Notions of formal security

In this section we review the notions of security in the sense of IND-CPA, IND-CCA1 and IND-CCA2 for a public key encryption scheme and for a key encapsulation mechanism (KEM). First we recall the definition of a random oracle.

Definition A.1. Let A and B be finite subsets of $\{0,1\}^*$. **A random oracle** $H : A \to B$ is a function such that for each $a \in A$, $H(a) \in_R B$. Note that H is a function in the sense that $H(a)$ for an input a is fixed, it is chosen when H is instantiated.

Let $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a public key encryption scheme:

- The probabilistic algorithm $\mathcal{K}(1^\lambda)$ generates public parameters pp of the scheme, a public key pk_A for Alice and a secret key sk_A for Alice, in polynomial time in the security parameter λ.
- The (possibly probabilistic) algorithm $\mathcal{E}_{pk_A}$ encrypts an element in the message space of the scheme and outputs an element c in the ciphertext space of the scheme.
- The deterministic algorithm $\mathcal{D}_{sk_A}$ decrypts an element in the ciphertext space and outputs the decrypted message.

Sometimes when $\mathcal{K}$ is clear from the context we simplify the notation and write $\Pi = (\mathcal{E}, \mathcal{D})$. In the **random oracle model**, each party that is using Π is assumed to have access to the same random oracle H (hence the algorithms in Π may use H). The following definition of the IND-CPA game, IND-CCA1 game and IND-CCA2 game for a public key encryption scheme is from [BDPR98, Definition 2.1].

Definition A.2. The **IND-ATK game**, with ATK $\in$ {CPA, CCA1, CCA2}, for $\Pi = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ is the following. Let $A = (A_1, A_2)$ be a pair of probabilistic polynomial time algorithms. Let ε be the oracle that returns the empty string on any query. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be oracles such that

1. If ATK = CPA, then $\mathcal{O}_1 = \mathcal{O}_2 = \varepsilon$.
2. If ATK = CCA1, then $\mathcal{O}_1 = \mathcal{D}_{sk_A}$ and $\mathcal{O}_2 = \varepsilon$.
3. If ATK = CCA2, then $\mathcal{O}_1 = \mathcal{O}_2 = \mathcal{D}_{sk_A}$.

In the random oracle model, A is also given oracle access to H. Let

1. $(pp, pk_A, pk_B) \leftarrow \mathcal{K}(1^\lambda)$.
2. $(x_0, x_1, s) \leftarrow A_1(\mathcal{O}_1, pp, pk_A)$. The variable s is a state variable that A_1 can use to transfer its state to A_2.
3. $c \leftarrow \mathcal{E}_{pk_A}(x_b)$ with $b \in_R \{0,1\}$.

Note that A may not query $\mathcal{O}_2$ on c and that we require $|x_0| = |x_1|$. The algorithm-pair A **wins** in the IND-ATK game if

$$b \leftarrow A_2(\mathcal{O}_2, c_b, s)$$

Because of the state variable s we sometimes for simplicity refer to $A = (A_1, A_2)$ as just a single algorithm A.

Definition A.3. A public key encryption scheme Π is **secure in the sense of IND-ATK** if for every polynomial time (in λ) algorithm A there is a negligible (in λ) function μ such that

$$\Pr[A \text{ wins in the IND-ATK game}] \leq \frac{1}{2} + \mu$$

For any public key encryption scheme to be even IND-CPA secure, the encryption function $\mathcal{E}_{pk_A}$ must be probabilistic.

Following [Pei14, p. 4], a **Key encapsulation mechanism (KEM)** is defined by four algorithms.

1. The probabilistic algorithm Setup(1^λ) generates public parameters of the mechanism in polynomial time in the security parameter λ.
2. The probabilistic algorithm Gen(pp) generates a public key pk_A and a secret key sk_A for Alice.
3. The probabilistic algorithm Encaps(pp, pk_A) generates an ephemeral key K. It then outputs K and the encapsulation c of K.
4. The deterministic algorithm Decaps(sk_A, c) decapsulates c and outputs K.

In the **random oracle model**, each party that is using the KEM is assumed to have access to the same random oracle H (hence the algorithms in the KEM may use H). We may choose to let (pk_A, sk_A) be static, like in a public key encryption scheme. Following [CS03, Section 7.1.2], the IND-CPA game, IND-CCA1 game and IND-CCA2 game in the context of a KEM can be defined in the following way.

Definition A.4. The **IND-ATK game**, with ATK $\in$ {CPA, CCA1, CCA2}, for a given KEM is the following. Let $A = (A_1, A_2)$ be a pair of probabilistic polynomial time algorithms. Let ε be the oracle that returns the empty string on any query. Let $\mathcal{O}_1$ and $\mathcal{O}_2$ be oracles such that

1. If ATK = CPA, then $\mathcal{O}_1 = \mathcal{O}_2 = \varepsilon$.
2. If ATK = CCA1, then $\mathcal{O}_1 = \text{Decaps}(sk_A, \cdot)$ and $\mathcal{O}_2 = \varepsilon$.
3. If ATK = CCA2, then $\mathcal{O}_1 = \mathcal{O}_2 = \text{Decaps}(sk_A, \cdot)$.

In the random oracle model, A is also given oracle access to H. Let

1. $pp \leftarrow \text{Setup}(1^\lambda)$.
2. $(pk_A, sk_A) \leftarrow \text{Gen}(pp)$.
3. $s \leftarrow A_1(\mathcal{O}_1, pp, pk_A)$.
4. $(c, K) \leftarrow \text{Encaps}(pp, pk_A)$.
5. $b \in_R \{0, 1\}$.
6. $K' = K$ if $b = 0$. If $b = 1$, then let $K' \in_R \{0, 1\}^{|K|}$.

Note that A may not query $\mathcal{O}_2$ on c. The algorithm-pair A **wins** in the IND-ATK game if

$$b \leftarrow A_2(\mathcal{O}_2, c, K', s)$$

Because of the state variable s we sometimes for simplicity refer to $A = (A_1, A_2)$ as just a single algorithm A.

Definition A.5. A KEM is **secure in the sense of IND-ATK** if for every polynomial time (in λ) algorithm A there is a negligible (in λ) function μ such that

$$\Pr[A \text{ wins in the IND-ATK game}] \leq \frac{1}{2} + \mu$$