

Title Goes Here

Submitted by

Robert W.V. Gorrie
B.ASc. Computer Science (McMaster University)

Under the guidance of
Douglas Stebila

*Submitted in partial fulfillment of
the requirements for the award of the degree of*

**Masters of Science
in
Computer Science**



Department of Computing and Software
McMASTER UNIVERSITY
Hamilton, Ontario, Canada

Fall Semester 2017

Abstract

¡Abstract here¿

Contents

1	Introduction	1
1.1	Background and Recent Research	1
1.2	Motivation	1
1.2.1	Literature Survey	1
1.3	Layout of Paper	1
2	Technical Background	2
2.1	Cryptographic Primitives Background	3
2.1.1	Key Exchange	3
2.1.2	Identification Schemes	4
2.1.3	Signature Schemes	4
2.2	Algebraic Geometry & Isogenies	4
2.2.1	Elliptic Curves	5
2.2.2	Isogenies & Their Properties	8
2.3	Supersingular Isogeny Diffie-Hellman	8
2.3.1	Public Parameters	8
2.3.2	SIDH Key Exchange	9
2.3.3	Zero-Knowledge Proof of Identity	10
2.4	Fiat-Shamir Construction	10
2.4.1	Unruh’s Post-Quantum Adaptation	10
2.5	Isogeny Based Signatures	10
2.6	Implementations of Isogeny Based Schemes	11
2.6.1	Microsoft Research SIDH Library	11
2.6.2	Yoo et al. Signature Layer	11
3	Batching Operations for Isogenies	14
3.1	Batching Procedure in Detail	14
3.1.1	Projective Space	14
3.1.2	Remaining Opportunities	14
3.2	Implementation	15
3.3	Results	15
3.3.1	Analysis	16
4	Compressed Signatures	17
4.1	Compression of Public Keys	17
4.1.1	Sub-section title	17
4.1.2	Sub-section title	17
4.1.3	Sub-section title	17

4.1.4	Sub-section title	17
4.1.5	Sub-section title	17
4.2	Implementation	17
4.3	Results	17
5	Discussion & Conclusion	18
5.1	Results & Comparisons	18
5.2	Additional Opportunities for Batching	18
5.3	Future Work	18
	Acknowledgements	19

List of Figures

2.1	$+$ acting over points P and Q of $y^2 = x^3 - 2x + 2$	6
2.2	associativity illustrated on $y^2 = x^3 - 3x$ (left & center) and $P + (-P) = \mathcal{O}$ illustrated for $y^2 = x^3 + x + 1$ (right).	7
2.3	SIDH key exchange between Alice & Bob	10
2.4	Relationship between SIDH key exchange & MR SIDH C library	11
2.5	Relationship between SIDH based signatures & Our fork of the SIDH C library	11
3.1	¡Caption here!	15

Chapter 1

Introduction

1.1 Background and Recent Research

1.2 Motivation

1.2.1 Literature Survey

1.3 Layout of Paper

Over the course of the past decade, elliptic curve cryptography (ECC) has proven itself a mainstay in the wide world of applied cryptology. While isogeny based cryptography does build itself up from the same underlying field of mathematics as ECC, it simultaneously draws from a slightly more complicated space of algebraic notions. Much of this chapter will be dedicated to illuminating these notions in a manner that should be digestable for those without serious background in algebraic geometry, or abstract algebra in general.

Chapter 2

Technical Background

This chapter will cover the following preliminary topics: cryptographic primitives, isogenies and their relevant properties, supersingular isogeny Diffie-Hellman (SIDH), the Fiat-Shamir construction for digital signatures (and its quantum-safe adaptation), current landscape of isogeny based signature schemes, and finally the C implementations of isogeny based schemes with which we are concerned.

In the first section of this chapter we will introduce a few cryptographic schemes for the readers who may not be entirely familiar with modern cryptography. We will cover key exchange, identification schemes, and signature schemes - all at as high of an abstraction level as possible. Readers familiar with these topics can skip this section without harm.

Our discussion of isogenies will begin with some basic coverage of the underlying algebra. We will provide the material necessary for the remaining sections as we build up in the level of abstraction; working our way through groups, finite fields, elliptic curves, and finally isogenies and their properties.

Once we have presented the necessary algebra, we will illustrate the specifics of the supersingular isogeny Diffie-Hellman key-exchange protocol. We will spend most of this time dedicated to a modular deconstruction of the protocol, looking at the high-level procedures and algorithms which will be necessary for understanding in detail the signature protocol to come. This subsection will end with a briefing and analysis of the closely related zero-knowledge proof of identity (ZKPoI) isogeny protocol proposed in the original De Feo et al. paper[LDF], as it is the foundation for the isogeny based signature scheme presented by Yoo et al[YY].

In section 2.3 we will discuss the Fiat-Shamir transformation[?]; a technique which, given a secure interactive proof of knowledge (*identification scheme*), creates a secure digital signature scheme. We will also look at the quantum-secure adaptation published by Unruh[?], for applying a non-quantum-resistant transform to a quantum-resistant primitive would be rather frivolous.

Section 2.4 will be dedicated to covering current isogeny-based signature schemes - the topic of which this dissertation is mainly concerned. We will discuss the signature scheme of Yoo et al., which is a near direct application of Unruh's work to the SIDH zero-knowledge proof of identity.

Finally, the last section of this chapter will introduce the SIDH C library released by Microsoft Research, on top of which the core contributions of this thesis are implemented. We will also look at the implementation of the to-be-discussed signature scheme, which is a sort of proof-of-concept built ontop of the Microsoft API.

2.1 Cryptographic Primitives

Cryptographic primitives can be thought of as the basic building blocks used in the design of cryptographically secure applications. The idea of which being that if individual primitives are proven (or believeably) secure, we can be more confident in the security of the application as a whole.

To quickly recap some basic information security, there are several different security properties a cryptographic primitive may aim to offer:

- *Confidentiality*: The notion that the information in question is kept private from unauthorized individuals.
- *Integrity*: The notion that the information in question is accurate and complete.
- *Availability*: The notion that the information in question is available to authorized individuals when requested.
- *Authenticity*: The notion that the source of the information in question is verified.
- *Non-repudiation*: The notion that the source of the information in question **cannot** deny having originally provided the information.

Each of the primitives to come are designed to offer some utility in the communication between a given pair of entities. We will refer to these entities as Alice and Bob. The schemes we are concerned with in this paper are strictly *public key* (also known as *asymmetric key*) schemes.

The first class of primitives we will discuss, *key exchange* protocols, provide a means by which Alice and Bob can come to the agreement of some secret value. The goal of a key exchange protocol is for Alice & Bob, communicating over some open, insecure channel, to reach mutual agreement of the secret value while also ensuring the *confidentiality* of that value. The secret value is referred to as a *secret* or *shared* key and is intended for use in other cryptographic primitives.

Identification schemes are a class of primitives that aim to ensure *authenticity* of a given entity. If Alice is communicating with Bob and she wants to verify that Bob is who he claims to be, the two can utilize a secure identification scheme. After identification protocols we will look at signature schemes, which are somewhat of an extension of the former. Signature schemes aim to provide *authenticity* on every message sent from Bob to Alice, as well as *non-repudiability* & *integrity* of those messages.

2.1.1 Key Exchange

A given key exchange protocol, which we'll denote as Π , is specified by a pair of algorithms **KeyGen** and **SecAgr**: $\Pi = (\text{KeyGen}, \text{SecAgr})$. Alice and Bob will each run both of these procedures. The first they will run on the same input, 1^λ , a bit string of λ 1's. The second, short for "secret agreement", they will run on the output of *KeyGen*.

From the perspective of some entity X executing Π with Y , the procedure looks like the following:

KeyGen(1^λ): A probabilistic algorithm with input 1^λ and output (sk_X, pk_X) . Typically pk is the image of $g(sk)$, where g is some *one-way* function.

In between execution of **KeyGen** and **SecAgr**, Alice will send Bob her computed value pk_{Alice} and Bob will send Alice pk_{Bob} .

SecAgr(sk_X, pk_Y): A deterministic algorithm with input sk_X and pk_Y and output $k_X \in \{0, 1\}^\lambda$.

Π is said to uphold *correctness* if, for

2.1.2 Identification Schemes

2.1.3 Signature Schemes

2.2 Algebraic Geometry & Isogenies

Groups & Varieties. A **group** is a 2-tuple composed of a set of elements and a corresponding group operation (also referred to as the group *law*). Given some group defined by the set G and the operation \cdot (written as (G, \cdot)) it is typical to refer to the group simply as G . If \cdot is equivalent to some rational mapping[footnote about rational mappings] $f_G : G \rightarrow G$, then the group (G, \cdot) is said to form an *algebraic variety*[footnote about the inverse function]. A group which is also an algebraic variety is referred to as an **algebraic group**.

G is said to be an *abelian* group if, in addition to the four traditional group axioms (closure, associativity, existence of an identity, existence of an inverse), G satisfies the condition of commutativity. More formally, for some group G with group operation \cdot , we say G is an abelian group iff $x \cdot y = y \cdot x \ \forall x, y \in G$. An algebraic group which is also abelian is referred to as an **abelian variety**.

Definition 1 (Abelian Variety). for some algebraic group G with operation \cdot , we say G is an abelian variety iff $x \cdot y = y \cdot x \ \forall x, y \in G$.

For some group (G, \cdot) , some $x, y \in G$, and some rational mapping $f_G : G \rightarrow G$, let the following sequence of implications denote the classification of (G, \cdot) :

$$\text{group} \xrightarrow{x \cdot y = f_G(x, y)} \text{algebraic group} \xrightarrow{x \cdot y = y \cdot x} \text{abelian variety}$$

Morphisms. Let us again take for example some group (G, \cdot) . Let's also define some set $S_{(G, \cdot)}$ which contains every tuple (x, y, z) for group elements x, y, z which satisfy $x \cdot y = z$.

$$S_{(G, \cdot)} = \{x, y, z \in G \mid x \cdot y = z\}$$

Take also for example a second group $(H, *)$ and some map $\phi : G \rightarrow H$. ϕ is said to be *structure preserving* if the following implication holds:

$$(x, y, z) \in S_{(G, \cdot)} \Rightarrow (\phi(x), \phi(y), \phi(z)) \in S_{(H, *)}$$

A **morphism** is simply the most general notion of a structure preserving map. More specifically, in the domain of algebraic geometry, we will be dealing with the notion of a **group homomorphism**, defined as follows:

Definition 2 (Group Homomorphism). For two groups G and H with respective group operations \cdot and $*$, a group homomorphism is a structure preserving map $h : G \rightarrow H$ such that $\forall u, v \in G$ the following holds:

$$h(u \cdot v) = h(u) * h(v)$$

From this simple definition, two more properties of homomorphisms are easily deducible. Namely, for some homomorphism $h : G \rightarrow H$, the following properties hold:

- h maps the identity element of G onto the identity element of H , and
- $h(u^{-1}) = h(u)^{-1}, \forall u \in G$

Furthermore, an **endomorphism** is a special type of morphism in which the domain and the codomain are the same groups.

Definition 3 (Endomorphism). For two groups G and H , an endomorphism is a morphism $\psi : G \rightarrow H$ wherein $G = H$

Fields & Field Extensions. An algebraic group G_a is defined over a field K if each element $e \in G_a$ is defined over K and the corresponding f_{G_a} is also defined over K . To show that a particular algebraic group G_a is defined over some field K we will henceforth denote the group/field pairing as $G_a(K)$.

Quotient Groups.

These algebraic structures are all important for building up to the concept of an *isogeny*. The lowest-level structure we will be concerned with when discussing the forthcoming isogeny-based protocols will typically be abelian varieties. The lowest-level structure in the SIDH C codebase is a finite field element.

2.2.1 Elliptic Curves

An elliptic curve is an algebraic curve defined over some field K , the most general representation of which is given by

$$y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6.$$

This representation encapsulates elliptic curves defined over any field. If, however, we are discussing curves defined specifically over a field K such that $\text{char}(K) > 3$ [ref], then the more compact form $y^2 = x^3 + ax + b$ can be applied. In this dissertation we will default to this second representation, as the schemes with which we are concerned will always be defined over \mathbb{F}_p for some large prime p .

Within algebraic geometry, it is common practice to define a group structure over the points of a given elliptic curve (or any other smooth cubic curve). If we wish to define a group in accordance to a particular curve, we do so with the following notation:

$$E : y^2 = x^3 + ax + b$$

Wherein E denotes the group in question, the elements of which are all the points (solutions) of the curve. Throughout much of this section, the words *point* and *element* can be used interchangeably.

A given curve E has a *smooth order* if *math*.

The Group Law. The group operation we define for E , denoted $+$, is better understood geometrically than algebraically. Consider the following.

Given two elements P and Q of some arbitrary elliptic curve group E , we define $+$ geometrically as follows: drawing the line L formed by points P and Q , we follow L to its third intersection on the curve, which we will denote as $R = (x_R, y_R)$. We then set $P + Q = -R$, where $-R$ is the reflection of R over the x -axis: $(x_R, -y_R)$. See figure [fig] for an illustrated representation of this process.

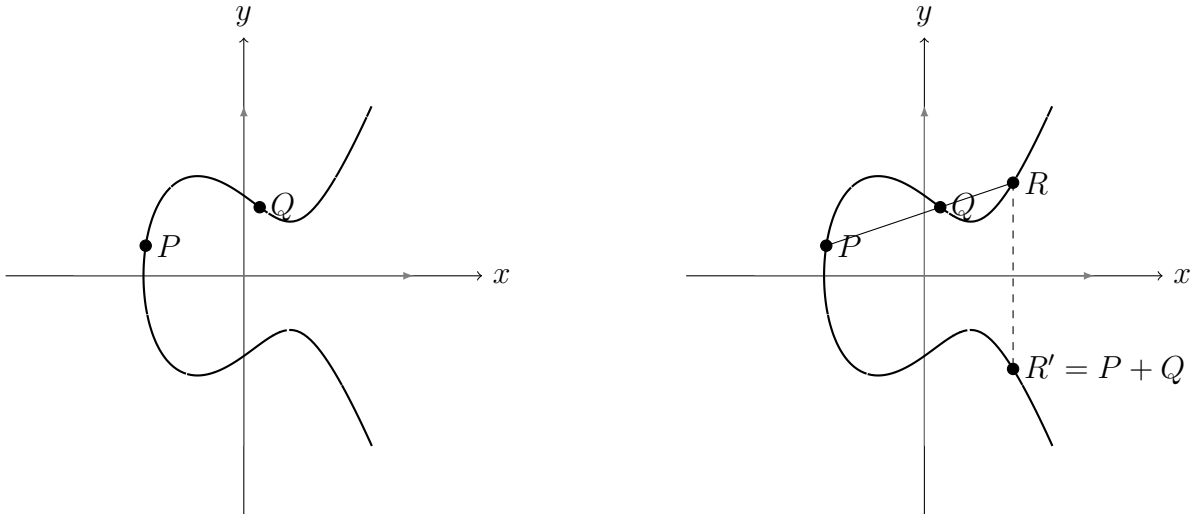


Figure 2.1: $+$ acting over points P and Q of $y^2 = x^3 - 2x + 2$.

The group operation $+$ is referred to as *pointwise addition*. In order for $(E, +)$ to properly form a group under pointwise addition, it must satisfy the four group axioms:

- *Closure*: Because elliptics curves are polynomials of degree of 3, we know any given line passing through two points P and Q of E will pass through a third point R . The exceptions here are twofold. First, when $P = Q$ and thus our line is tangent to E , and second, when $Q = -P$ and our line is parallel with the y -axis. We resolve the first case nicely by defining $P + P$ by means of taking L to be the line tangent to E at point P . In the second case, $P + (-P)$, by group axiom, should yield the identity element of the group. We will define this element and resolve this issue below.
- *Identity*: The identity element of elliptic curve groups, denoted as \mathcal{O} , is a specially defined point satisfying $P + \mathcal{O} = \mathcal{O} + P = P$, $\forall P \in E$. Because of the inclusion of this special element, we have that $\#(E(K))$ is equal to $1 +$ the number geometric points on E defined over K . This of course is only a noteworthy claim when K is a finite field (otherwise there are already infinitely many elements in E).
- *Associativity*: To show that associativity holds for geometrically defined points P , Q , and R in E ($(P + Q) + R = P + (Q + R)$) is rather simple (see figure [fig]). We

can trivially show that this holds when any combination of P , Q , and R are \mathcal{O} by applying the axiom of the identity.

- *Inverse*: Due to the x-symmetry of elliptic curves, every point $P = (x_P, y_P)$ of E has an associated point $-P = (x_P, -y_P)$. If we apply $+$ to P and $-P$, L assumes the line parallel to the y-axis at $x = x_P$. As discussed above, in this case there is no third intersection of L on E . In light of this, \mathcal{O} can be thought of as a point residing infinitely far in both the positive and negative directions of the y-axis. \mathcal{O} is equivalently referred to as the *point at infinity*. [footnote about whether $+$ actually constitutes a rational map due to this exception]

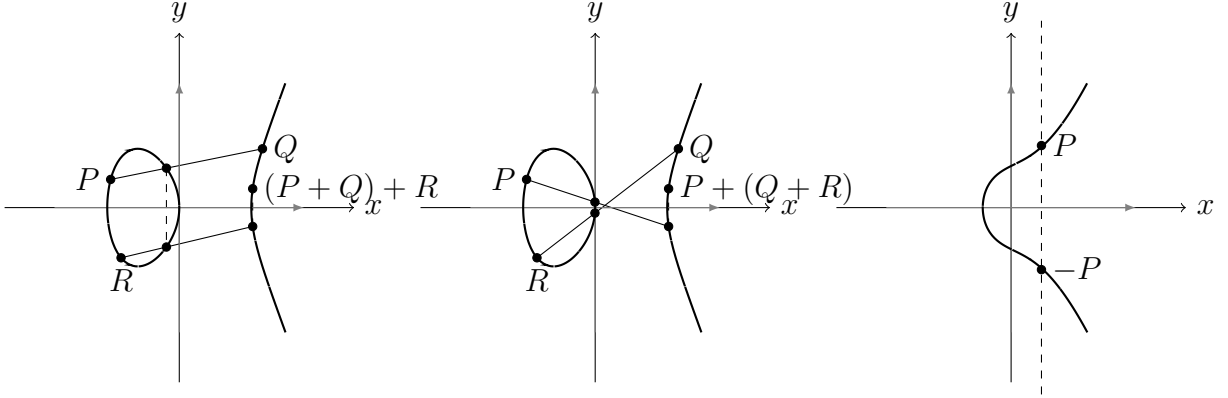


Figure 2.2: associativity illustrated on $y^2 = x^3 - 3x$ (left & center) and $P + (-P) = \mathcal{O}$ illustrated for $y^2 = x^3 + x + 1$ (right).

Additionally, we shorthand $\overbrace{P + P + \dots + P}^n$ as nP , analogous to scalar multiplication.

Consequently, because groups defined over elliptic curves in this fashion are commutative, they also constitute abelian varieties[ref].

When referring to curves as abelian varieties defined over a field, we will write them as $E_\alpha(K)$, for some curve E_α and some field K . If we are only concerned with the geometric properties of the curve, or curves as distinct elements of some group structure, it will suffice to write E_α . Moving forward from here, we will assume all general curves discussed are capable of definition over some finite field \mathbb{F}_p .

Torsion Groups. The r -torsion group of E is the set of all points $P \in E(\overline{\mathbb{F}}_q)$ such that $[r]P = \mathcal{O}$. We denote the r -torsion group of some curve as $E[r]$.

Supersingular Curves. An elliptic curve can be either *ordinary* or *supersingular*. There are several equivalent ways to define supersingular curves (and thus the distinction between them and ordinary curves.)

For the remainder of this paper, unless otherwise noted, all elliptic curves in discussion will be of the supersingular variety.

Montgomery Arithmetic.

2.2.2 Isogenies & Their Properties

Definition 4 (Isogeny). Let G and H be algebraic groups[ref]. An isogeny is a morphism[ref] $h : G \rightarrow H$ possessing a finite kernel.

In the case of the above definition where G and H are abelian varieties (such as elliptic curves,) the isogeny h is homomorphic[ref] between G and H . Because of this, isogenies over elliptic curves (and other abelian varieties) inherit certain characteristics. For an isogeny $h : E_1 \rightarrow E_2$ defined over elliptic curves E_1 and E_2 , the following holds:

- $h(\mathcal{O}) = \mathcal{O}$, and
- $h(u^{-1}) = h(u)^{-1}, \forall u \in G$

We write $\text{End}(E)$ to denote the ring formed by all the isogenies acting over E which are also endomorphisms. Note that m -repeated pointwise addition of a point with itself can equivalently be modelled by an endomorphism, we denote the application of such an endomorphism to a point P as $[m]P$, such that $[m] : E \rightarrow E$ and $[m]P = mP$.

2.3 Supersingular Isogeny Diffie-Hellman

This section will aim to accomplish two things. First, we will briefly explain the isogeny-level & key-exchange-level procedures of the SIDH protocol. Second, we will illuminate how these procedures map onto Microsoft Research's C implementation of SIDH. In this regard, this section can be considered an attempt to meld two domains of SIDH functions & procedures, in hopes of easing the navigation from the SIDH protocol to Microsoft's C implementation, and vice versa.

The original work of De Feo, Jao, and Plut outlines three different isogeny-based cryptographic primitives: Diffie-Hellman-esque key exchange, public key encryption, and the aforementioned zero-knowledge proof of identity. Because all three of these protocols require the same initialization and public parameters, we will begin by covering these parameters in detail. Immediately after, we will analyze the key exchange at a relatively high level. Our goal of this section is to explain in detail the algorithmic and cryptographic aspects of the ZKPoI scheme, as this forms the conceptual basis for the signature scheme we will be investigating. We begin with the key exchange protocol because its sub-routines are integral to the Yoo et al. signature implementation.

For the discussion that follows, we will assume every instance of an SIDH protocol occurs between two parties, **A** and **B** (eg. **Alice** & **Bob**,) for which we will colorize information particular to **A** in blue and **B** in red. This will include private keys & public keys as well as the variables and constants used in their generation.

2.3.1 Public Parameters

As the name suggests, SIDH protocols work over supersingular curves (with no singular points). Let $\mathbb{F}_q = \mathbb{F}_{p^2}$ be the finite field over which our curves are defined, \mathbb{F}_{p^2} denoting the quadratic extension field of \mathbb{F}_p . p is a prime defined as follows:

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

Wherein ℓ_A and ℓ_B are small primes (typically 2 & 3, respectively) and f is a cofactor ensuring the primality of p . We then define globally a supersingular curve E_0 defined over \mathbb{F}_q with cardinality $(\ell_A^{e_A} \ell_B^{e_B} f)^2$. Consequently, the torsion group $E_0[\ell_A^{e_A}]$ is \mathbb{F}_q -rational and has $\ell_A^{e_A-1}(\ell_A + 1)$ cyclic subgroups of order $\ell_A^{e_A}$, with the analogous statement being true for $E_0[\ell_B^{e_B}]$. Additionally, we include in the public parameters the bases $\{P_A, Q_A\}$ and $\{P_B, Q_B\}$, generating $E[\ell_A^{e_A}]$ and $E[\ell_B^{e_B}]$ respectively.

This brings our set of global parameters, G , to the following:

$$G = \{p, E_0, \ell_A, \ell_B, e_A, e_B, \{P_A, Q_A\}, \{P_B, Q_B\}\}$$

2.3.2 SIDH Key Exchange

This subsection will illustrate an SIDH key exchange run between party members **Alice** and **Bob**. The general idea of the protocol can be surmised by the diagram below. In the scheme, **private keys** take the form of isogenies[ref] defined with domain E , and **public keys** are the associated co-domain curve of said isogenies.

$$\begin{array}{ccc} E_0 & \xrightarrow{\phi_A} & E_0/\langle A \rangle \\ \downarrow \phi_B & & \downarrow \phi_{B'} \\ E_0/\langle B \rangle & \xrightarrow{\phi_{A'}} & E_0/\langle A, B \rangle \end{array}$$

The premise of the protocol is that both parties generate some random point (**A** or **B** in the diagram,) which, according to theorem [ref], indicates some distinct isogeny $\phi_A : E_0 \rightarrow E/\langle A \rangle$ (or equivalent for **B**). **Alice** and **Bob** then exchange codomain curves and compute

$$\begin{array}{c} \phi_A(E_0/\langle B \rangle) \\ \text{OR} \\ \phi_B(E_0/\langle A \rangle) \end{array}$$

To come to the shared secret agreement, the codomain curve of their composed isogenies, denoted E_{AB} . Below we've outli

Key Generation. **Alice** chooses two random numbers $m_A, n_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ such that $(\ell_A \nmid m_A) \vee (\ell_A \nmid n_A)$. **Alice** then computes the isogeny $\phi_A : E_0 \rightarrow E_{\langle [m_A]P_A, [n_A]Q_A \rangle}$ with kernel $\text{Ker}_A := \langle [m_A]P_A, [n_A]Q_A \rangle$. **Bob** undergoes the same procedure for random elements $m_B, n_B \in \mathbb{Z}/\ell_B^{e_B}\mathbb{Z}$. After completion, **Alice** and **Bob** hold their respective key pairs:

$$\begin{aligned} (sk_A, pk_A) &= (\{m_A, n_A\}, E_{\langle [m_A]P_A, [n_A]Q_A \rangle}) \\ (sk_B, pk_B) &= (\{m_B, n_B\},) \end{aligned}$$

Alice then applies her isogeny to the points which **Bob** will use in the creation of of his isogeny: $\{\phi_A(Q_B), \phi_A(Q_B)\}$.

PK Exchange. After **Alice** and **Bob** successfully complete their key generation, they perform the following over an insecure channel:

- Alice sends Bob
-

Secret Agreement.

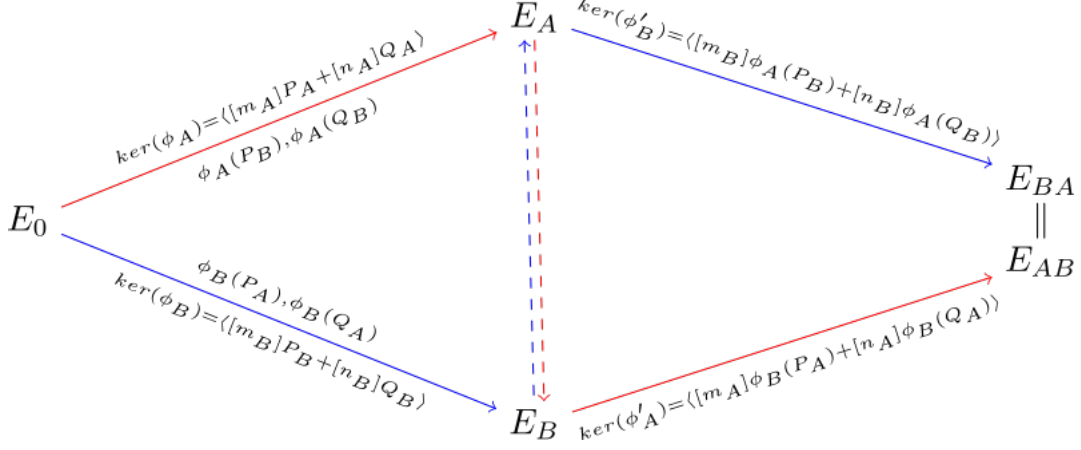


Figure 2.3: SIDH key exchange between Alice & Bob

2.3.3 Zero-Knowledge Proof of Identity

Recall the notion of a simple identification scheme:

2.4 Fiat-Shamir Construction

The Fiat-Shamir construction (sometimes referred to as the Fiat-Shamir heuristic or transform) is used

2.4.1 Unruh's Post-Quantum Adaptation

2.5 Isogeny Based Signatures

Now that we've introduced the zero-knowledge proof of identity scheme from [REFERENCE] as well as Unruh's quantum-safe Fiat-Shamir adaption, the isogeny based signature scheme presented by Yoo et. Al is a near-trivial application of the latter to the former.

The isogeny based signature scheme presented by Yoo et. Al is defined, in the traditional manner, by a tuple of algorithms. Namely, the scheme is defined by the tuple (KeyGen, Sign, Verify) with each algorithm loosely defined as follows:

KeyGen(): Select a random point S of order $\ell_A^{e_A}$, compute the isogeny $\phi : E \rightarrow E/\langle S \rangle$. Return (pk, sk) where $\text{pk} = (E/\langle S \rangle, \phi(P_B), \phi(Q_B))$ and $\text{sk} = S$.

Sign():

Verify():

2.6 Implementations of Isogeny Based Schemes

2.6.1 Microsoft Research SIDH Library

Figure [ref] illustrates the relationship between abstraction levels of the SIDH protocol and modules of the SIDH C library.

Key Representation.

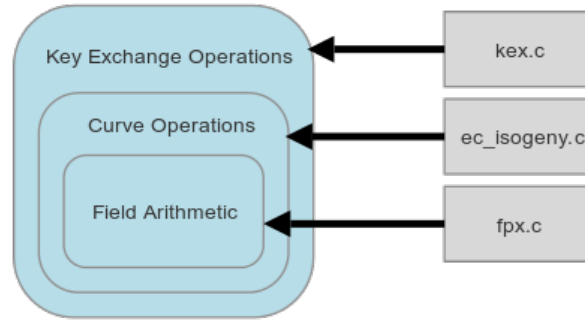


Figure 2.4: Relationship between SIDH key exchange & MR SIDH C library

2.6.2 Yoo et al. Signature Layer

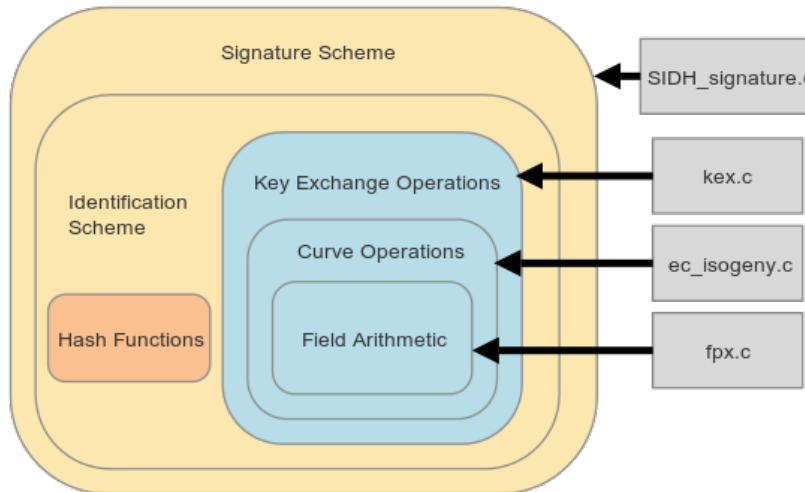


Figure 2.5: Relationship between SIDH based signatures & Our fork of the SIDH C library

Shortly after, the following, more in-depth algorithms are given as definitions:

Algorithm 1 KeyGen(λ)

- 1: Pick a random point S of order $\ell_A^{e_A}$
 - 2: Compute the isogeny $\phi : E \rightarrow E/\langle S \rangle$
 - 3: $\text{pk} \leftarrow (E/\langle S \rangle, \phi(P_B), \phi(Q_B))$
 - 4: $\text{sk} \leftarrow S$
 - 5: **return** (pk, sk)
-

Algorithm 2 Sign(sk, m)

- 1: **for** $i = 1 \dots 2\lambda$ **do**
 - 2: Pick a random point R of order $\ell_B^{e_B}$
 - 3: Compute the isogeny $\psi : E \rightarrow E/\langle R \rangle$
 - 4: Compute either $\phi' : E/\langle R \rangle \rightarrow E/\langle R, S \rangle$ or $\psi' : E/\langle S \rangle \rightarrow E/\langle R, S \rangle$
 - 5: $(E_1, E_2) \leftarrow (E/\langle R \rangle, E/\langle R, S \rangle)$
 - 6: $\text{com}_i \leftarrow (E_1, E_2)$
 - 7: $\text{ch}_{i,0} \leftarrow_R \{0, 1\}$
 - 8: $(\text{resp}_{i,0}, \text{resp}_{i,1}) \leftarrow ((R, \phi(R)), \psi(S))$
 - 9: **if** $\text{ch}_{i,0} = 1$ **then**
 - 10: $\text{swap}(\text{resp}_{i,0}, \text{resp}_{i,1})$
 - 11: $h_{i,j} \leftarrow G(\text{resp}_{i,j})$
 - 12: $J_1 \parallel \dots \parallel J_{2\lambda} \leftarrow H(\text{pk}, m, (\text{com}_i)_i, (\text{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j})$
 - 13: **return** $\sigma \leftarrow ((\text{com}_i)_i, (\text{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j}, (\text{resp}_{i,J_i})_i)$
-

Algorithm 3 Verify(pk, m, σ)

- 1: $J_1 \parallel \dots \parallel J_{2\lambda} \leftarrow H(\text{pk}, m, (\text{com}_i)_i, (\text{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j})$
 - 2: **for** $i = 0 \dots 2\lambda$ **do**
 - 3: **check** $h_{i,J_i} = G(\text{resp}_{i,J_i})$
 - 4: **if** $\text{ch}_{i,J_i} = 0$ **then**
 - 5: Parse $(R, \phi(R)) \leftarrow \text{resp}_{i,J_i}$
 - 6: **check** $(R, \phi(R))$ have order $\ell_B^{e_B}$
 - 7: **check** R generates the kernel of the isogeny $E \rightarrow E_1$
 - 8: **check** $\phi(R)$ generates the kernel of the isogeny $E/\langle S \rangle \rightarrow E_2$
 - 9: **else**
 - 10: Parse $\psi(S) \leftarrow \text{resp}_{i,J_i}$
 - 11: **check** $\psi(S)$ has order $\ell_A^{e_A}$
 - 12: **check** $\psi(S)$ generates the kernel of the isogeny $E_1 \rightarrow E_2$
 - 13: **if** all checks succeed **then**
 - 14: **return** 1
-

If we transcribe the above to the language of the Microsoft SIDH API, we have in essence the following:

Algorithm 4 KeyGen(λ)

- 1: $(\text{pk}, \text{sk}) \leftarrow \text{KeyGeneration.B}()$
 - 2: **return** (pk, sk)
-

Algorithm 5 $\text{Sign}(\text{sk}, m)$

```
1: for  $i = 1..2\lambda$  do
2:    $(, R, \psi) \leftarrow \text{KeyGeneration\_A}(\text{E})$ 
3:    $E_1 \leftarrow E / \langle R \rangle$ 
4:    $(E_2, E / \langle R, S \rangle) \leftarrow \text{SecretAgreement\_B}()$ 
5:    $(E_1, E_2) \leftarrow (E / \langle R \rangle, E / \langle R, S \rangle)$ 
6:    $\text{com}[i] \leftarrow (E_1, E_2)$ 
7:    $\text{ch}[i] \leftarrow_R \{0, 1\}$ 
8:    $(\text{resp}[i]_0, \text{resp}[i]_1) \leftarrow ((R, \phi(R)), \psi(S))$ 
9:  $J_1 \parallel \dots \parallel J_{2\lambda} \leftarrow H(pk, m, (\text{com}_i)_i, (\text{ch}_i)_i, (h_{i,j})_{i,j})$ 
10: return  $\sigma \leftarrow ((\text{com}_i)_i, (\text{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j}, ((\text{resp})[J_i])$ 
```

Chapter 3

Batching Operations for Isogenies

3.1 Batching Procedure in Detail

One of our main contributions is the embedding of a low-level \mathbb{F}_{p^2} procedure into Microsofts pre-existing SIDH library. The procedure in question reduces arbitrarily many unrelated/potentially parallel \mathbb{F}_{p^2} inversions to a sequence of \mathbb{F}_p multiplications & additions, as well as one \mathbb{F}_p inversion.

More specifically, the procedure takes us from n \mathbb{F}_{p^2} inversions to:

- $2n$ \mathbb{F}_p squarings
- n \mathbb{F}_p additions
- 1 \mathbb{F}_p inversion
- $3(n-1)$ \mathbb{F}_p multiplications
- $2n$ \mathbb{F}_p multiplications

The procedure is as follows:

3.1.1 Projective Space

Because the work of Yoo et al. was built on top of the original Microsoft SIDH library, all underlying field operations (and isogeny arithmetic) are performed in projective space. Doing field arithmetic in projective space allows us to avoid many inversion operations. The downside of this (for our work) is that the number opportunities for implementing the batched inversion algorithm becomes greatly limited.

3.1.2 Remaining Opportunities

There are two functions called in the isogeny signature system that perform a \mathbb{F}_{p^2} inversion: `j_inv` and `inv_4_way`. These functions are called once in `SecretAgreement` and `KeyGeneration` operations respectively. `SecretAgreement` and `KeyGeneration` are in turn called from each signing and verification thread.

Algorithm 6 Batched Partial-Inversion

```
1: procedure PARTIAL_BATCHED_INV( $\mathbb{F}_{p^2}[\ ]$  VEC,  $\mathbb{F}_{p^2}[\ ]$  DEST, INT N)
2:   initialize  $\mathbb{F}_p$  den[n]
3:   for i = 0..(n-1) do
4:     den[i]  $\leftarrow a[i][0]^2 + a[i][1]^2$ 
5:   a[0]  $\leftarrow$  den[0]
6:   for i = 1..(n-1) do
7:     a[i]  $\leftarrow a[i-1] * \text{den}[i]$ 
8:   ainv  $\leftarrow$  inv(a[n-1])
9:   for i = n-1..1 do
10:    a[i]  $\leftarrow a_{inv} * \text{dest}[i-1]$ 
11:    ainv  $\leftarrow a_{inv} * \text{den}[i]$ 
12:  dest[0]  $\leftarrow a_{inv}$ 
13:  for i = 0..(n-1) do
14:    dest[i][0]  $\leftarrow a[i] * \text{vec}[i][0]$ 
15:    vec[i][1]  $\leftarrow -1 * \text{vec}[i][1]$ 
16:    dest[i][1]  $\leftarrow a[i] * \text{vec}[i][1]$ 
```

This means that in the signing procedure there are 2 opportunities for implementing batched partial-inversion with a batch size of 248 elements. In the verify procedure, however, there are 3 opportunities for implementing batched inversion with a batch size of roughly 124 elements.

3.2 Implementation

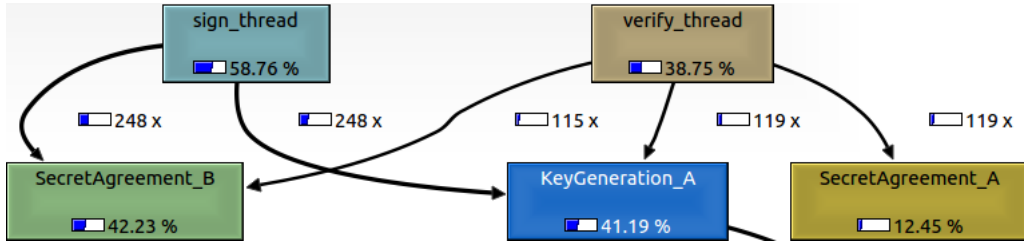


Figure 3.1: ;Caption here;

3.3 Results

Two different machines were used for benchmarking. System A denotes a single-core, 1.70 GHz Intel Celeron CPU. System B denotes a quad-core, 3.1 GHz AMD A8-7600.

The two figures below provide benchmarks for KeyGen, Sign, and Verify procedures with both batched partial inversion implemented (in the previously mentioned locations) and not implemented. All benchmarks are averages computed from 100 randomized sample runs. All results are measured in clock cycles.

Procedure	System A Without Batching	System A With Batching
KeyGen	68,881,331	68,881,331
Signature Sign	15,744,477,032	15,565,738,003
Signature Verify	11,183,112,648	10,800,158,871

Procedure	System B Without Batching	System B With Batching
KeyGen	84,499,270	84,499,270
Signature Sign	10,227,466,210	10,134,441,024
Signature Verify	7,268,804,442	7,106,663,106

System A: With inversion batching turned on we notice a 1.1 % performance increase for key signing and a 3.5 % performance increase for key verification.

System B: With inversion batching turned on we observe a 0.9 % performance increase for key signing and a 2.3 % performance increase for key verification.

3.3.1 Analysis

It should first be noted that, because our benchmarks are measured in terms of clock cycles, the difference between our two system clock speeds should be essentially ineffective.

In the following table, "Batched Inversion" signifies running the batched partial-inversion procedure on 248 \mathbb{F}_{p^2} elements. The procedure uses the binary GCD \mathbb{F}_p inversion function which, unlike regular \mathbb{F}_{p^2} montgomery inversion, is not constant time.

Procedure	Performance
Batched Inversion	1721718
\mathbb{F}_{p^2} Montgomery Inversion	874178

Do performance increases observed make sense?

Chapter 4

Compressed Signatures

4.1 Compression of Public Keys

We discussed rejection sampling A values from signature public keys until we found an A that was also the x-coord of a point. After some simple analysis, however, we found that it was extremely unlikely for A to be a point on the curve.

4.1.1 ;Sub-section title;

4.1.2 ;Sub-section title;

some text[?], some more text

4.1.3 ;Sub-section title;

4.1.4 ;Sub-section title;

Refer figure 3.1.

4.1.5 ;Sub-section title;

4.2 Implementation

4.3 Results

Chapter 5

Discussion & Conclusion

5.1 Results & Comparisons

5.2 Additional Opportunities for Batching

5.3 Future Work

¡Conclusion here¿

Acknowledgments

¡Acknowledgements here¿

¡Name here¿

¡Month and Year here¿

National Institute of Technology Calicut

References

- [LDF] Jerome Plut Luca De Feo, David Jao. Towards quantum-resistant cryptosystems from supersingular elliptic curve isogenies.
- [YY] Amir Jalali David Jao Vladimir Soukharev Youngho Yoo, Reza Azarderakhsh. A post-quantum digital signature scheme based on supersingular isogenies.