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

Abstract

¡Abstract here¿

Contents

List of Figures

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 re-

lated 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 serveral 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 has not been altered by

unauthorized individuals.

- *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. In public key primitives, each user possesses a *public* key (visible to every user in the network) as well as a *private* key, which only they have access to.

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 reffered 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 key exchange protocol, which we'll denote as Π_{kex}, can be represented in some contexts by a pair of polynomial time algorithms **KeyGen** and **SecAgr**: $\Pi_{kex} = (\mathbf{KeyGen}, \mathbf{SecAgr})$.

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 $KeyGen$.

Execution of Π_{kex} between Alice and Bob involves the following:

(i) Alice and Bob run $\mathbf{KeyGen}(1^\lambda)$: A probabilistic algorithm with input 1^λ and output (sk, pk). Typically pk is the image of $g(sk)$, where h is some *one-way* function.

(ii) Alice and Bob exchange (over an insecure channel) their public keys pk_{Alice} and pk_{Bob}.

(iii) Alice runs $\mathbf{SecAgr}(sk_{\text{Alice}}, pk_{\text{Bob}})$: A deterministic algorithm with input sk_{Alice} and pk_{Bob} and output $k_{\text{Alice}} \in \{0,1\}^\lambda$. Bob runs $\mathbf{SecAgr}(sk_{\text{Bob}}, pk_{\text{Alice}})$.

Π_{kex} is said to uphold *correctness* if $k_{Alice} = k_{Bob}$. Because we deal only with correct Π_{kex}, we refer to the output of Π_{kex} as simply k. Figure 2.1 illustrates an execution of the Diffie-Hellman key exchange protocol which relies on the difficulty of the *discrete logarithm* problem for its one-way function h.

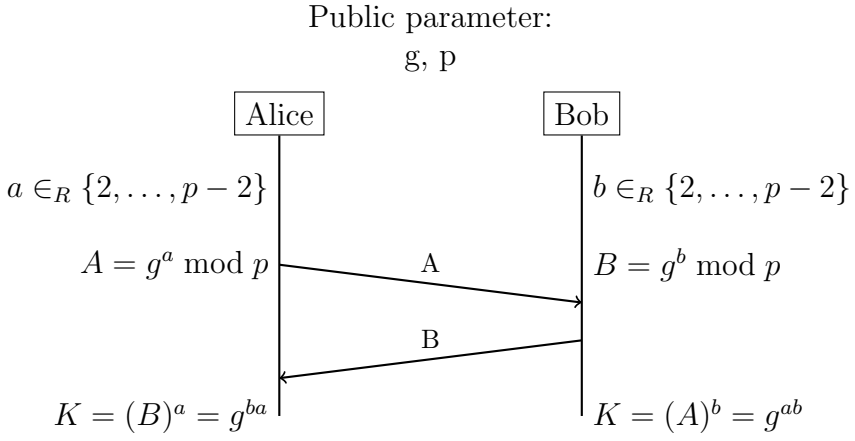


Figure 2.1: Alice and Bob's execution of Diffie-Hellman key exchange.

2.1.2 Interactive Identification Schemes

Imagine Alice wishes to confirm the identity of Bob. The idea behind interactive identification protocols is to offer Bob some way of proving to Alice (or anyone) that he has knowledge of some secret which **only** Bob could possess. The goal, of course, being to accomplish this without openly revealing the secret, so that it can continue to be used as an identifier for Bob.

An identification scheme (or otherwise "proof of knowledge" or "proof of identity") Π_{id} is composed of by the tuple of polynomial-time algorithms (**KeyGen**, **Prove**, **Verify**). Π_{id} is an interactive protocol, wherein the *prover* (Bob, for example) executes **Prove** and the *verifier* (Alice) executes **Verify**.

Execution of Π_{id} between Alice and Bob involves the following:

(i) Bob runs $\mathbf{KeyGen}(1^\lambda)$: A probabilistic algorithm with input 1^λ and output (sk, pk).

(ii) Bob sends to Alice his public key pk and a probabilistically generated initial commitment com. Alice will respond with a *challenge* value ch.

(iii) Bob runs $\mathbf{Prove}(sk, com, ch)$: A deterministic algorithm with input sk (Bob's secret key) and ch (challenge) and output $resp$.

(iv) Alice runs $\mathbf{Verify}(pk, com, ch, resp)$: A deterministic algorithm with input pk (Bob's public key), com, ch, and $resp$ and output $b \in 0, 1$. Bob has successfully proven his identity to Alice if $b = 1$.

If Alice accepts Bobs response, and $b = 1$, then we refer to the tuple $(com, ch, resp)$ as an *accepting transcript*. There exist variations upon this type of primitive wherein Alice is not required to send Bob a specific challenge value. These are known as *non-interactive* identification schemes, or non-interactive proofs of identity (NIPoI). These non-interactive approaches to solving the problem of identity and *authentication* further bridge the gap between identification protocols and signature schemes.

2.1.3 Signature Schemes

There are two main differences between signature schemes and interactive identification schemes. The first is equivalent to the difference between Second, because we wish to authenticate Bob on every message, our scheme needs to be run every time Bob wishes to send a message to Alice. We must also include Bob's message m in our execution of **Verify** (which in the context of signatures, we call **Sign**). And so, a signature scheme can be thought of as an NIPoI executed with respect to a particular message. We define a signature scheme as the tuple of algorithms $\Pi_{sig} = (\textbf{KeyGen}, \textbf{Sign}, \textbf{Verify})$.

Execution of Π_{sig} between Alice and Bob for a particular message m sent from Bob to Alice involves the following:

(i) Bob runs $\textbf{KeyGen}(1^\lambda)$: A probabilistic algorithm with input 1^λ and output (sk, pk).

(ii) Bob sends his public key to Alice.

(iii) Bob runs $\textbf{Sign}(sk, m)$: A deterministic algorithm with input sk (Bob's secret key) and m (the message Bob wishes to authorize) and output σ, known as a *signature*.

(iv) Bob sends m and σ to Alice.

(v) Alice runs $\textbf{Verify}(pk, m, \sigma)$: A deterministic algorithm with input pk (Bob's public key), m, and σ and output $b \in 0, 1$. Alice has confidence in the *integrity* and origin *authenticity* of m if $b = 1$.

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] $f_G : G \to G$, 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 commutitiviy. 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 $f_G : G \to G$, 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 $x \cdot y = z$.

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

Take also for example a second group $(H, *)$ and some map $\phi : G \to H$. 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 $h : G \to H$ 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 $h : G \to H$, 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. We denote the set of enomorphisms defineable over some group G as $\text{End}(G)$. The *kernel* of a particular homomorphism $h : G \to H$ is the set of elements in G that, when applied to h, map to the identity element of H. We write this set as $ker(h)$, and it is much analogous to the familiar concept from linear algebra, wherein the kernel denotes the set of elements mapped to the zero vector by some linear map.

Fields & Field Extensions. A **field** is a mathematical structure which, while being similar to a group, demands additional properties. Fields are defined by some set F and two operations: *addition* and *multiplication*. In order for some tuple $(F, +, \cdot)$ to constitute a field, it must satisfy an assortment of axioms:

addition axioms:

- If $x \in F$ and $y \in F$, then $x + y \in F$.
- $+$ is commutative.
- $+$ is associative.

- F contains an element 0 such that $\forall x \in F$ we have $0 + x = x$.
- $\forall x \in F$ there is a corresponding element $-x \in F$ such that $x + (-x) = 0$.

multiplication axioms:

- If $x \in F$ and $y \in F$, then $x \cdot y \in F$.
- $\cdot$ is commutative.
- $\cdot$ is associative.
- F contains an element $1 \neq 0$ such that $\forall x \in F$ we have $x \cdot 1 = x$.
- $\forall x \neq 0 \in F$ there is a corresponding element $1/x \in F$ such that $x \cdot (1/x) = 1$.

Additionally, a field $(F, +, \cdot)$ must uphold the *distributive law*, namely:

$$x \cdot (y + z) = x \cdot y + x \cdot y \text{ holds } \forall x, y, z \in F$$

While these axioms are known to be satisfied by the sets $\mathbb{Q}$, $\mathbb{R}$, and $\mathbb{C}$ with typically defined $+$ and $\cdot$, our focus will be on a particular class of field known as a *finite field*. Finite fields, as the name suggests, are fields in which the set F contains finitely many elements - we refer to the number of elements in F as the *order* of the field.

Let us take some prime number p. We can construct a finite field by taking F as the set of numbers $\{0, 1, ...p-1\}$ and defining $+$ and $\cdot$ as addition and multiplication *modulo* p. Finite fields defined in this fashion are denoted as $\mathbb{F}_p$, and have order p.

$$\forall x, y \in \mathbb{F}_p,\ x + y = (x + b) \mod p, \text{ and}$$
$$\forall x, y \in \mathbb{F}_p,\ x \cdot y = (x \cdot b) \mod p$$

For any given field K there exists a number q such that, for every $x \in K$, adding x to itself q times results in the additive identity 0. This number is referred to as the *characteristic* of K, for which we write $\text{char}(K)$. Finite fields are the only type of field for which $\text{char}(K) > 0$. Furthermore, if the field in question is finite and has prime order, then the order and the characteristic are equivalent.

A particular field K' is called an *extension field* of some other field K if $K \subseteq K'$. The complex numbers $\mathbb{C}$, for example, are an extension field of $\mathbb{R}$. A given field K is *algebraically closed* if there exists a root for every non-constant polynomial defined over K. If K itself is not algebraically closed, we denote the extension of K that is by $\overline{K}$.

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 $G_a(K)$. Naturally, in the case where our field is a finite field of order p, we write $G_a(\mathbb{F}_p)$.

These algebraic structures are all important for building up to the concept of an *isogeny*. The lowest-level object we will be concerned with when discussing the forthcoming isogeny-based protocols will typically be elements of 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 dicussing 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 (see figure 2.2 for a geometric visualization). 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.

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 2.2 for an illustrated representation of this process.

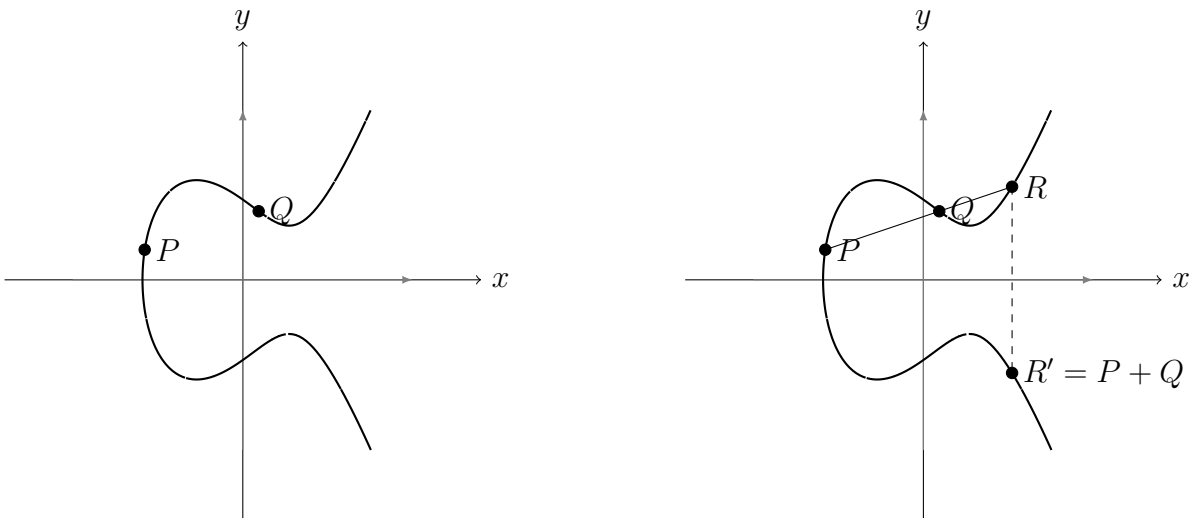


Figure 2.2: $+$ acting over points P and Q of $y^2 = x^3 - 2x + 2$.

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 2.3). 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 parrallel 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]

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

Consequently, because groups defined over elliptic curves in this fashion are commutitive, 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 ge-

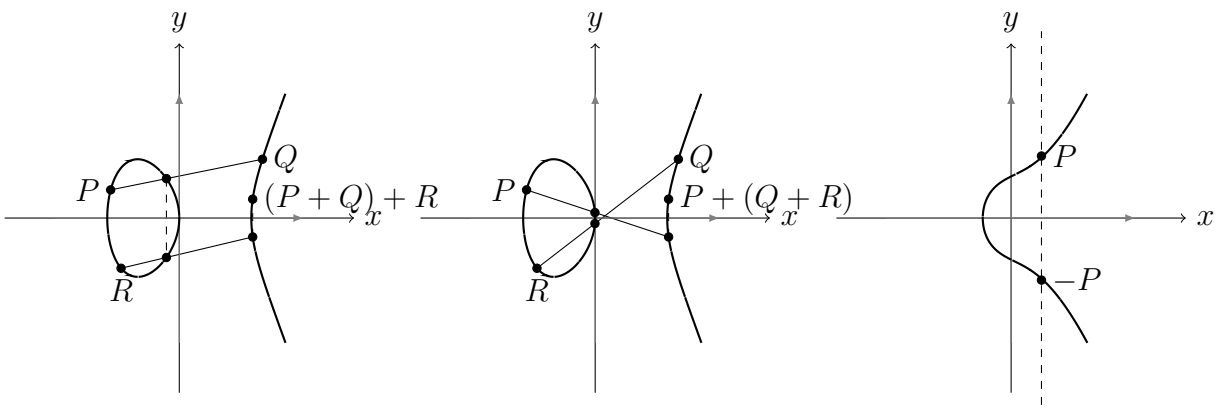


Figure 2.3: associativity illustrated on $y^2 = x^3 - 3x$ (left & center) and $P + (-P) = \mathcal{O}$ illustrated for $y^2 = x^3 + x + 1$ (right).

ometric 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 3 (Isogeny)**.** Let G and H be algebraic groups[ref]. An isogeny is a morphism[ref] $h : G \to H$ 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 $h : E_1 \to E_2$ 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 $[m] : E \to E$ 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 infor-

mation 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.

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:

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

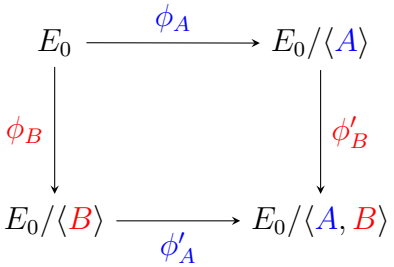
Wherein ℓ_A and ℓ_B are small primes (typically 2 & 3, respectively) and f is a cofactor ensuring the primality of p. We then define globally a supersingular curve E_0 defined over $\mathbb{F}_q$ with cardinality $(\ell_A^{e_A} \ell_B^{e_B} f)^2$. Consequently, the torsion group $E_0[\ell_A^{e_A}]$ is $\mathbb{F}_q$-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:

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

2.3.1 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.



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 $\phi_A : E_0 \to E/\langle A \rangle$ (or equivalent for B). Alice and Bob then exchange codomain curves and compute

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

To come to the shared secret agreement, the codomain curve of their composed isogenies, denoted E_{AB}. Below we've outlined the SIDH key exchange protocol $\Pi_{\text{SIDH}} = (\textbf{KeyGen}, \textbf{SecAgr})$ in a descriptive (though not yet algorithmic) manner.

KeyGen: 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 $\phi_A : E_0 \to E_A$ where $E_A = E_0/\langle [m_A]P_A, [n_A]Q_A \rangle$ (equivalently, $ker(\phi_A) = \langle [m_A]P_A, [n_A]Q_A \rangle$). 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:

$$(sk_A, pk_A) = (\{m_A, n_A\}, E_A)$$
$$(sk_B, pk_B) = (\{m_B, n_B\}, E_B)$$

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

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

- Alice sends Bob $(E_A, \{\phi_A(P_B), \phi_A(Q_B)\})$
- Bob sends Alice $(E_B, \{\phi_B(P_A), \phi_B(Q_A)\})$

SecAgr: After reception of Bob's tuple, Alice computes the isogeny $\phi'_A : E_B \to E_{AB}$ and Bob acts analogously. Alice and Bob then arrive at the equivalent image curve:

$$E_{AB} = \phi'_A(\phi_B(E_0)) = \phi'_B(\phi_A(E_0)) = E_0/\langle [m_A]P_A + [n_A]Q_A, [m_B]P_B + [n_B]Q_B \rangle$$

From which they can use the common j-invariant of as a shared secret k.

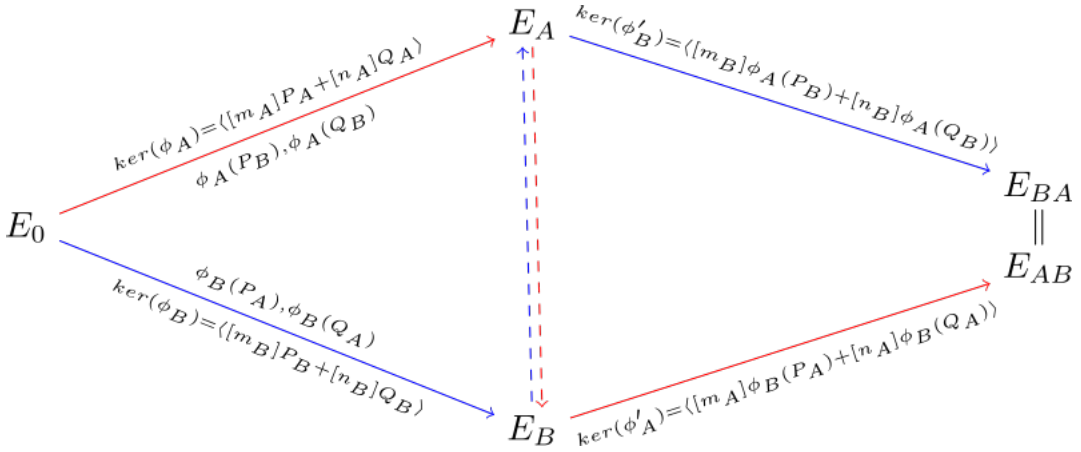


Figure 2.4: SIDH key exchange between Alice & Bob

2.3.2 Zero-Knowledge Proof of Identity

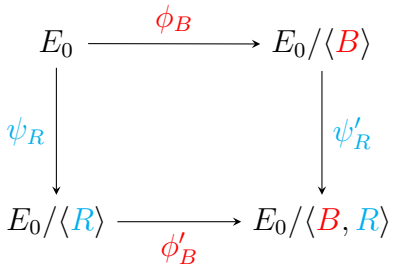
Recall the earlier discussed notion of an identification scheme. A canonical identification scheme $\Pi_{\text{SID}} = (\textbf{KeyGen}, \textbf{Prove}, \textbf{Verify})$ can be derived somewhat analogously to the SIDH protocol, and is outlined in the original work of De Feo et al.

Say Bob has derived for himself the key pair (sk_B, pk_B) with $sk_B = \{m_B, n_B\}$ and $pk_B = E_B = E_0/\langle [m_B]P_B + [n_B]Q_B \rangle$ in relation to the public parameters E_0 and $\ell_B^{e_B}$. With E_0 and E_B publicly known, Π_{ZKPoI} revolves around Bob trying to prove to Alice that he knows the generator for E_B without revealing it.

To achieve this, Bob internally mimicks an execution of the key exchange protocol Π_{SIDH} with an arbitrary "random" entity Randall.

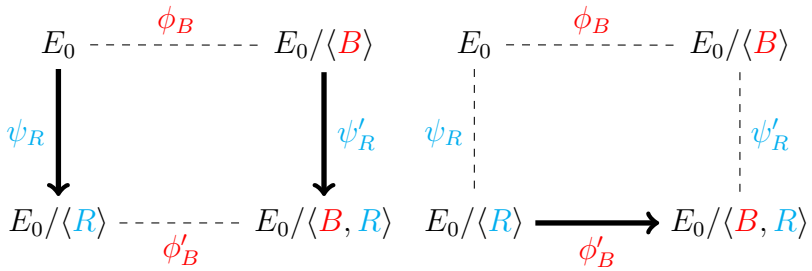
KeyGen: Key generation is performed exactly as in Π_{SIDH}, the only difference being that in Π_{ZKPoI} only the prover (Bob, in our example,) needs to generate a keypair.

Commitment: Bob generates a random point $R \in E_0[\ell_A^{e_A}]$ ($R = [m_R]P_A + [n_R]Q_A$) along with the corresponding isogenies necessary to compute the diagram below in full (if Alice were acting as the prover in Π_{ZKPoI}, then she would choose $R \in E_0[\ell_B^{e_B}]$). Bob sends his commitment com as $(com_1, com_2) = (E/\langle R\rangle, E/\langle B, R\rangle)$ to Alice.



Response: Alice chooses a bit b at random and sends her challenge $ch = b$ to Bob.

Prove: If Alice's response bit $ch = 0$ then Bob reveals the isogenies ψ_R and ψ'_R (to do this, he can simply reveal the kernels of ψ_R and ψ'_R; R and $\phi_B(R)$ respectively). This proves he knows the information necessary to form a shared secret with Randall *if and only if* he happens to know the private key $B = \{[m_B]P_B + [n_B]Q_B\}$. If $ch = 1$, Bob reveals the isogeny ϕ'_B. This proves that Bob knows the information necessary to form a shared secret with Randall *if and only if* he knows Randall's secret key R.



Note that Bob cannot at once reveal all of the information necessary to convince Alice that he knows B. If he reveals R, $\phi_B(R)$, and ϕ'_B all in one go, he incidentally reveals his secret key $B = [m_B]P_B + [n_B]Q_B$. This is because Bob reveals ϕ'_B by revealing the

generator of $ker(\phi'_B)$, namely:

$$(B, R) = [m_B]P_B + [n_B]Q_B, [m_R]P_A + [n_R]Q_A\rangle$$

How Π_{ZKPoK} handles this is by having Bob and Alice run **Prove()** and **Verify()** for λ iterations, with a different $(com, ch, resp)$ transcript generated for every instance. This way, if Bob is able to provide a $resp$ that satisfies Alice's ch for every iteration, she can be sufficiently confident that Bob has knowledge of B.

Verify(pk, com, ch): Like the proving procedure, verification is a conditional function depending on the value of b:

- if $ch = 0$: return 1 *if and only if* R and $\phi_B(R)$ have order $\ell_A^{e_A}$ and generate the kernels of isogenies from $E_0 \to E_0/\langle R\rangle$ and $E_0/\langle B\rangle \to E_0/\langle B, R\rangle$ respectively.
- if $ch = 1$: return 1 *if and only if* $\psi_R(B)$ has order $\ell_B^{e_B}$ and generates the kernel of an isogeny over $E_0/\langle R\rangle \to E_0/\langle B, R\rangle$.

This particular construction for an identification scheme is known in the literature as a *zero knowledge* proof of identity (or equivalently, a zero knowledge proof of *knowledge*).

2.4 Fiat-Shamir Construction

The Fiat-Shamir construction (also frequently referred to as the Fiat-Shamir transform, or Fiat-Shamir hueristic,) is a high-level technique for transforming a canonical identification scheme into a secure signature scheme. The construction relies heavily on the random oracle model

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 $\phi : E \to E/\langle S \rangle$. Return (pk, sk) where pk = $(E/\langle S \rangle, \phi(P_B), \phi(Q_B))$ and sk = S.

Sign():

Verify():

2.6 Implementations of Isogeny Based Schemes

2.6.1 Parameters

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

Key Representation.

2.6.2 Yoo et al. Signature Layer

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

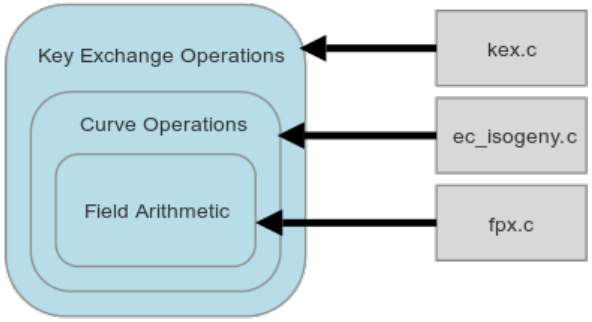


Figure 2.5: Relationship between SIDH key exchange & MR SIDH C library

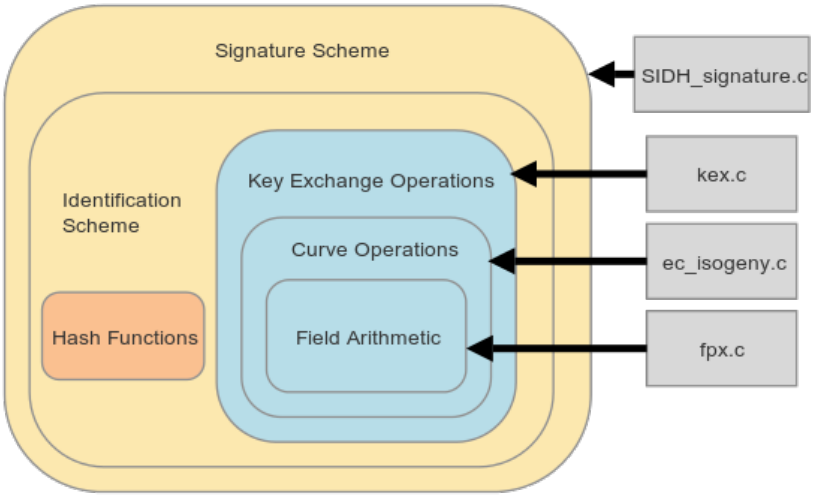


Figure 2.6: Relationship between SIDH based signatures & Our fork of the SIDH C library

Algorithm 3 Verify(pk, m, σ)

$J_1 \parallel ... \parallel J_{2\lambda} \leftarrow H(pk, m, (\texttt{com}_i)_i, (\texttt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j})$
for `i = 0..2`λ **do**
    check $h_{i,J_i} = G(\texttt{resp}_{i,J_i})$
    if $\texttt{ch}_{i,J_i} = 0$ **then**
        Parse $(R, \phi(R)) \leftarrow \texttt{resp}_{i,J_i}$
        check $(R, \phi(R))$ have order $\ell_B^{e_B}$
        check R generates the kernel of the isogeny $E \rightarrow E_1$
        check $\phi(R)$ generates the kernel of the isogeny $E/\langle S \rangle \rightarrow E_2$
    else
        Parse $\psi(S) \leftarrow \texttt{resp}_{i,J_i}$
        check $\psi(S)$ has order $\ell_A^{e_A}$
        check $\psi(S)$ generates the kernel of the isogeny $E_1 \rightarrow E_2$
if all checks succeed **then**
    return 1

Algorithm 1 KeyGen(λ)

Pick a random point S of order $\ell_A^{e_A}$
Compute the isogeny $\phi : E \to E/\langle S \rangle$
pk $\leftarrow (E/\langle S \rangle, \phi(P_B), \phi(Q_B))$
sk $\leftarrow S$
return (pk,sk)

Algorithm 2 Sign(sk, m)

for `i = 1..2`λ **do**
    Pick a random point R of order $\ell_B^{e_B}$
    Compute the isogeny $\psi : E \to E/\langle R \rangle$
    Compute either $\phi' : E/\langle R \rangle \to E/\langle R, S \rangle$ or $\psi' : E/\langle S \rangle \to E/\langle R, S \rangle$
    $(E_1, E_2) \leftarrow (E/\langle R \rangle, E/\langle R, S \rangle)$
    $\texttt{com}_i \leftarrow (E_1, E_2)$
    $\texttt{ch}_{i,0} \leftarrow_R \{0, 1\}$
    $(\texttt{resp}_{i,0}, \texttt{resp}_{i,1}) \leftarrow ((R, \phi(R)), \psi(S))$
    if $\texttt{ch}_{i,0} = 1$ **then**
        $\texttt{swap}(\texttt{resp}_{i,0}, \texttt{resp}_{i,1})$
    $h_{i,j} \leftarrow G(\texttt{resp}_{i,j})$
$J_1 \parallel ... \parallel J_{2\lambda} \leftarrow H(pk, m, (\texttt{com}_i)_i, (\texttt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j})$
return $\sigma \leftarrow ((\texttt{com}_i)_i, (\texttt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j}, (\texttt{resp}_{i,J_i})_i)$

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

Algorithm 4 KeyGen(λ)

(pk, sk) $\leftarrow$ `KeyGeneration_B()`
return (pk,sk)

2.6.3 Performance

Algorithm 5 Sign(sk, m)

for `i = 1..`2λ **do**
    $(, R, \psi) \leftarrow$ `KeyGeneration_A(E)`
    $E_1 \leftarrow E/\langle R \rangle$
    $(E_2, E/\langle R, S \rangle) \leftarrow$ `SecretAgreement_B()`
    $(E_1, E_2) \leftarrow (E/\langle R \rangle, E/\langle R, S \rangle)$
    $\texttt{com}[i] \leftarrow (E_1, E_2)$
    $\texttt{ch}[i] \leftarrow_R \{0, 1\}$
    $(\texttt{resp}[i]_0, \texttt{resp}[i]_1) \leftarrow ((R, \phi(R)), \psi(S))$
$J_1 \parallel ... \parallel J_{2\lambda} \leftarrow H(pk, m, (\texttt{com}_i)_i, (\texttt{ch}_i)_i, (h_{i,j})_{i,j})$
return $\sigma \leftarrow ((\texttt{com}_i)_i, (\texttt{ch}_{i,j})_{i,j}, (h_{i,j})_{i,j}, ((\texttt{resp})[J_i])$

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
- $3(n\text{-}1)$ $\mathbb{F}_p$ multiplications
- $2n$ $\mathbb{F}_p$ multiplications

The procedure is as follows:

Algorithm 6 Batched Partial-Inversion

```
1: procedure PARTIAL_BATCHED_INV(F_{p^2}[ ] VEC, F_{p^2}[ ] DEST, INT N)
2:     initialize F_p den[n]
3:     for i = 0..(n-1) do
4:         den[i] ← a[i][0]^2 + a[i][1]^2
5:     a[0] ← den[0]
6:     for i = 1..(n-1) do
7:         a[i] ← a[i-1]*den[i]
8:     a_inv ← inv(a[n-1])
9:     for i = n-1..1 do
10:        a[i] ← a_inv * dest[i − 1]
11:        a_inv ← a_inv * den[i]
12:    dest[0] ← a_inv
13:    for i = 0..(n-1) do
14:        dest[i][0] ← a[i] * vec[i][0]
15:        vec[i][1] ← −1 * vec[i][1]
16:        dest[i][1] ← a[i] * vec[i][1]
```

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 arithmitic 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.

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

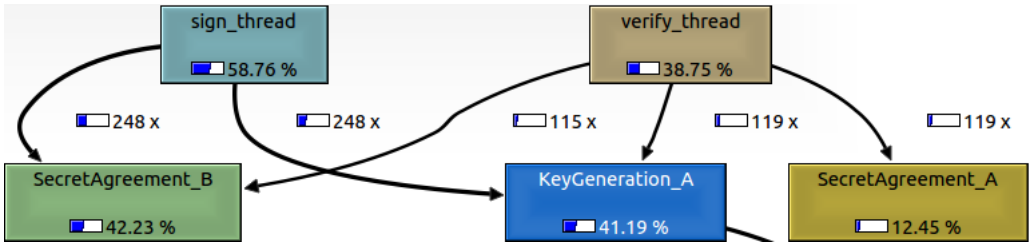


Figure 3.1: ¡Caption here¿

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 a 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 curvce isogenies.

[YY] Amir Jalali David Jao Vladimir Soukharev Youngho Yoo, Reza Azarderakhsh. A post-quantum digital signature scheme based on supersingular isogenies.