

# Advances Towards Practical Implementations of Isogeny Based Signatures

---

Robert Gorrie

McMaster University – Department of Computing & Software

*gorrierw@mcmaster.ca*

November 16, 2018

# Concerns of Cryptography

---

There are five rudimentary concerns of information security:

*Confidentiality*: information must be kept private from unauthorized individuals

*Integrity*: information must not be altered by unauthorized individuals

*Availability*: information must be available for authorized individuals

*Authenticity*: information must have a verifiable source

*Non-repudiation*: the source of information must be publicly verifiable

# Public-key Cryptography

---

The goal of cryptography is to define mathematically precise means of ensuring these information security goals.

Cryptographic protocols can be either *private-key* or *public-key* systems.

Public-key systems require that every party takes ownership of both a public key ( $pk$ ), the value of which is known by everyone on the network, and a private key ( $sk$ ), known only to the owner.

# Quantum Cryptanalysis

---

Efficient large-scale quantum computing → breaking most modern public-key cryptosystems.

This has lead to the development of the field known as post-quantum cryptography – the aim of which is to develop cryptosystems resistant to quantum cryptanalysis.

# Post-quantum Cryptography

---

Common approaches to post-quantum cryptography include

- Lattice-based cryptography

- Hash-based cryptography

- Multivariate-based cryptography

- Code-based cryptography

- Isogeny-based cryptography

# Post-quantum Cryptography

	Key Gen	Sign	Verify
SIDH	84,499,270	4,950,023,141.65	3,466,703,991.09
Sphincs	17,535,886.94	653,013,784	27,732,049
qTESLA	1,059,388	460,592	66,491
Picnic	13,272	9,560,749	6,701,701
RSA	12,800,000	1,113,600	32400
ECDSA	1,470,000	128,928	140,869

# My Contributions

---

# Overview

---

## Introduction & Background

- Post-quantum Cryptography & Motivation

- Elliptic Curves & Isogenies

- Supersingular Isogeny Diffie-Hellman

- Isogeny-based Signatures

## Batching Field Element Inversions

- Batching Partial Inversions

- Implementing Batching in SIDH 2.0

- Performance of Inversion Batching

## Compressing Isogeny-based Signatures

- SIDH Public Key Compression

- Implementing in SIDH 2.0

- Advantage and Cost of Compressions

## Results

- Performance Measurements



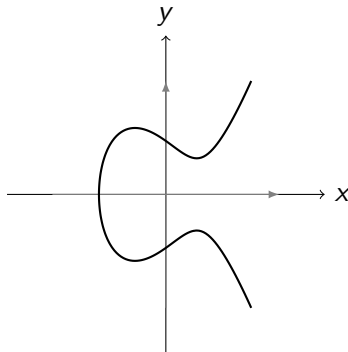
# Elliptic Curves as a Group

---

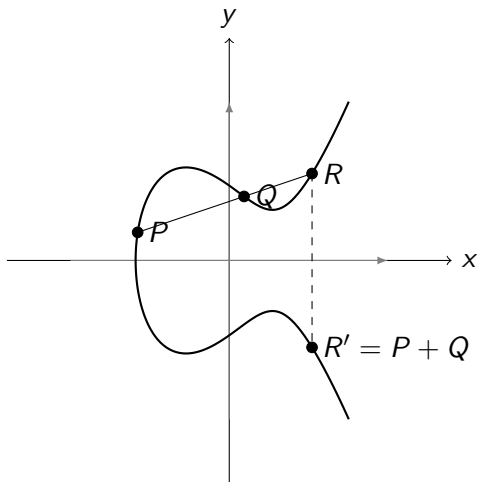
Elliptic curves are a class of algebraic curves satisfying

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

We can define a group  
composed of all the points  
 $P = (x, y)$  satisfying  $E$ .



# Elliptic Curves as a Group



# Torsion Subgroups

---

# Isogenies

---

Isogenies are maps that take a point on one elliptic curve to a point on another. For an isogeny  $\phi$  mapping from  $E_1$  to  $E_2$ , we can write

$$\phi : E_1 \rightarrow E_2$$

These maps have the following two properties

$$\phi(\mathcal{O}) = \mathcal{O}$$

$$\phi(P^{-1}) = (\phi(P))^{-1}$$

# Isogenies

---

Lemma (Uniquely identifying isogenies)

*Let  $E$  be an elliptic curve and let  $\Phi$  be a finite subgroup of  $E$ . There is a unique elliptic curve  $E'$  and a separable isogeny  $\phi : E \rightarrow E'$  satisfying  $\ker(\phi) = \Phi$ .*

# Key Exchange Protocols

---

Key exchange protocols are cryptographic schemes used to establish a shared secret between two party members

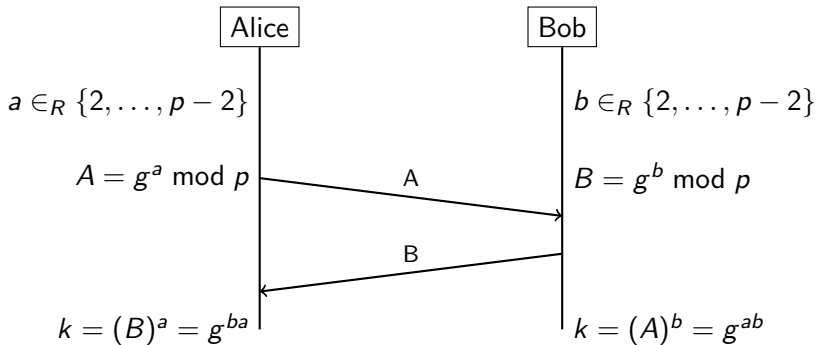
These can be defined by a tuple of algorithms

$$\Pi_{kex} = (\mathbf{KeyGen}, \mathbf{SecAgr}).$$

# Key Exchange Protocols

Public parameter:

$$g, p$$



# Supersingular Isogeny Diffie-Hellman

---

SIDH is a key-exchange protocol where **Alice** and **Bob** use Isogenies as their public keys and points on a curve as their private keys.

Here's how it works...



# Supersingular Isogeny Diffie-Hellman

We are concerned with curves over the field  $\mathbb{F}_{p^2}$  where

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

with  $f$  chosen such that  $p$  is prime.

We then choose a curve  $E$ , and bases  $\{P_A, Q_A\}$  and  $\{P_B, Q_B\}$  generating  $E[\ell_A^{e_A}]$  and  $E[\ell_B^{e_B}]$ .

And so, our set of public parameters is

$$\{p, E, \ell_A, \ell_B, e_A, e_B, P_A, Q_A, P_B, Q_B\}.$$

# Interactive Identification Schemes

Identification schemes are used to confirm the identity of a user on a network. These protocols are typically composed by the tuple of algorithms (**KeyGen**, **Commit**, **Prove**, **Verify**).

For **Bob** to prove his identity to **Alice**, a protocol of this type would run as follows:

- (i) **Bob** runs **KeyGen** ( $1^\lambda$ ) to generate his keypair  $(sk, pk)$ .
- (ii) **Bob** runs **Commit** () to generate  $com$  and sends it to **Alice**.
- (iii) **Alice** sends a randomly generated “challenge” value  $ch \in \omega$  and sends it to **Bob**.
- (iv) **Bob** runs **Prove** ( $sk, com, ch$ ) with output  $resp$ , the response to **Alice**’s challenge.
- (v) **Alice** runs **Verify** ( $pk, com, ch, resp$ ) with output  $b \in \{0, 1\}$ .  
**Bob** has successfully proven his identity to **Alice** if  $b = 1$ .

# Isogeny-based Proof of Identity

---

Signature schemes are used to prove that a particular party supplied a given message. These schemes consist of the algorithms **Sign**, **Verify**, and **Prove**.

# Signature Schemes

---

Signature schemes are used to prove that a particular party supplied a given message. These schemes consist of the algorithms **Sign**, **Verify**, and **Prove**.

# Fiat-Shamir Transform

---

The Fiat-Shamir transform is a process by which an interactive identification scheme can be turned into a signature scheme.

# Yoo Signatures

---

The signature scheme derived by Yoo et al. applies the Fiat-Shamir transform to the isogeny-based identification scheme to construct the first ever isogeny-based signature scheme.

# Recap

---

SIDH Key Exchange  $\rightarrow$  Isogeny-based Proof of Identity  $\rightarrow$  Yoo Signatures

# Partial $\mathbb{F}_{p^2}$ Inversions

---



# Batching Inversions

---

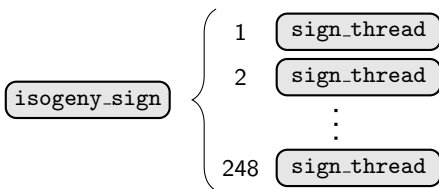
## Partial Batched Inversions

---

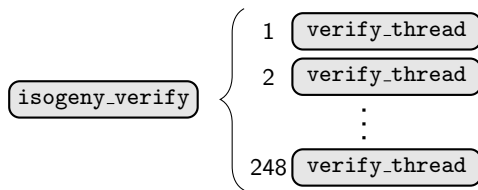
```
1: for  $i = 0 \dots (n-1)$  do
2:    $den_i \leftarrow (x_i)_a^2 + (x_i)_b^2 \pmod{p}$ 
3:  $a_0 \leftarrow den_0$ 
4: for  $i = 1 \dots (n-1)$  do
5:    $a_i \leftarrow a_{i-1} \cdot den_i \pmod{p}$ 
6:  $inv \leftarrow a_{n-1}^{-1} \pmod{p}$ 
7: for  $i = n-1 \dots 1$  do
8:    $a_i \leftarrow inv \cdot den_{i+1} \pmod{p}$ 
9:    $inv \leftarrow inv \cdot den_i \pmod{p}$ 
10:  $a_0 \leftarrow a_{inv}$ 
11: for  $i = 0 \dots (n-1)$  do
12:    $(x_{inv})_a \leftarrow a_i \cdot (x_i)_a \pmod{p}$ 
13:    $(x_{inv})_b \leftarrow a_i \cdot -(x_i)_b \pmod{p}$ 
14:    $x_i^{-1} \leftarrow \{(x_{inv})_a, (x_{inv})_b\}$ 
15: return  $\{x_0^{-1}, x_1^{-1}, \dots, x_{n-1}^{-1}\}$ 
```

# Structure of the Yoo Signature Implementation

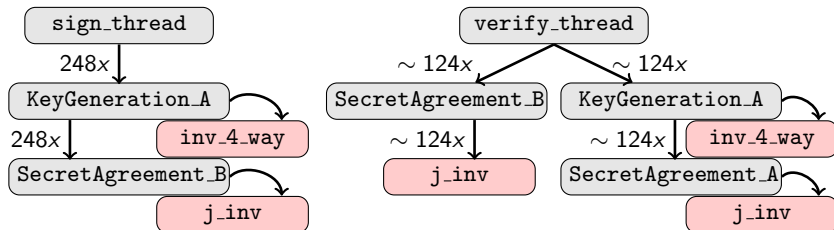
1. The signer executes **Sign** by spawning a thread running `sign_thread` for every iteration of the signing procedure



2. The verifier then executes **Verify** in a similar fashion.



## Batching Across Threads



# Performance Measurements for Partial Batched Inversions

---

# SIDH Public Key Compression

Azerderakhsh et al. showed that SIDH public keys can be compressed in the following way.

Take **Alice's** public key  $pk = (E_A, \phi_A(P_B), \phi_A(Q_B))$ .

By generating a basis  $\{R_1, R_2\}$  for  $E_A[\ell_B^{e_B}]$  we can represent the point components of  $pk$  as

$$P_A = \alpha_P R_1 + \beta_P R_2$$

$$Q_A = \alpha_Q R_1 + \beta_Q R_2$$

Giving  $pk = (E_A, \alpha_P, \beta_P, \alpha_Q, \beta_Q)$ . Public keys in this form are  $4 \log p$  bits compared to the usual  $6' \log p$  bits.

# SIDH Public Key Compression

---

Costello et al. then showed that  $(E_A, \alpha_P, \beta_P, \alpha_Q, \beta_Q)$  could be further compressed to  $(E_A, b, \zeta, \alpha, \beta)$ , where  $b \in \{1, 0\}$ .

## Citation

---

We use the SIDH public key compression technique developed by Azerderakhsh, Costello, and others to compress every  $\psi(S)$  component of a Yoo signature.

This statement requires citation [Smith, 2012].



# Citation

---

An example of the `\cite` command to cite within the presentation:

This statement requires citation [Smith, 2012].

# Questions?

---



*Journal Name* 12(3), 45 – 678