Advances Towards Practical Implementations of Isogeny Based Signatures

Robert Gorrie

McMaster University – Department of Computing & Software

gorrierw@mcmaster.ca

November 18, 2018

Concerns of Cryptography

There are five rudimentary concerns in 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. Proofs of cryptographic security depend, in many cases, on assumptions about the difficulty of solving some problem.

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 of the keypair.

Quantum Cryptanalysis

A large-scale quantum computer will have the capability of 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.

Performance of Post-quantum Cryptosystems

Common approaches to post-quantum cryptography include

- Lattice-based cryptography,
- Hash-based cryptography,
- Multivariate-based cryptography,
- Code-based cryptography, and
- Isogeny-based cryptography.

Post-quantum Cryptography

	Key Gen	Sign	Verify
Yoo	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

Post-quantum Cryptography

	Public Key	Private Key	Signature
SIDH	768	48	88,064
Sphincs	32	64	8,080 - 16,976
Rainbow	152,097 - 192,241	100,209 - 114,308	64 - 104
qTESLA	4,128	2,112	3,104
Picnic	33	49	34,004 - 53,933
RSA	384	256	384
ECDSA	32	32	32

Contributions

I outline two ways in which performance improvements can be made to a state-of-the-art isogeny-based signature scheme. We show how

1. certain operations in the Yoo signature scheme can be batched together, improving runtime by roughly 8%, and that
2. by adopting public key compression techniques from the literature, Yoo signatures can be compressed by 144λ bytes.

Overview

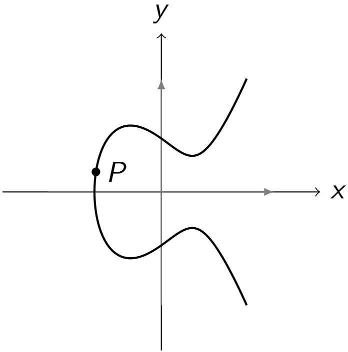
Overview

Elliptic Curves as a Group

Elliptic curves are a class of algebraic curves satisfying

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

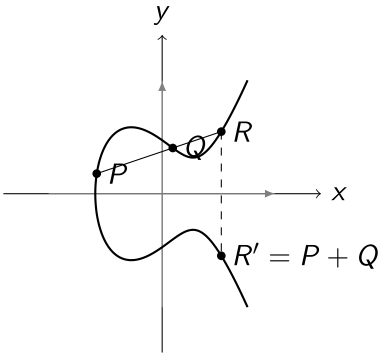
We can define a *group* of elements composed of all the points $P = (x_P, y_P)$ satisfying E.



Elliptic Curves as a Group

We can define the group operation $+$ of an elliptic curve group by computing $P + Q$ for $P, Q \in E$ as shown to the right.

We write $[m]P$ to denote m repeated applications of this operation to the point P.

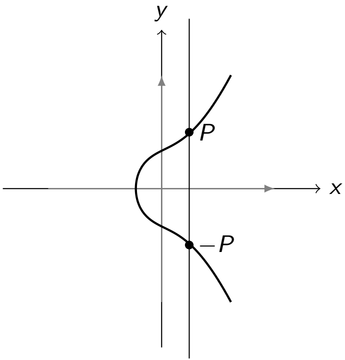


Elliptic Curves as a Group

We write $\mathcal{O}$ to denote the identity of this group, which is conceptualized as a point residing at positive and negative inifity.

Note to the right how $R + -(R) = \mathcal{O}$ on this elliptic curve.

This example also illustrates the existence of an inverse element, which for $P = (x, y)$ will always be $-P = (x, -y)$.



Torsion Subgroups

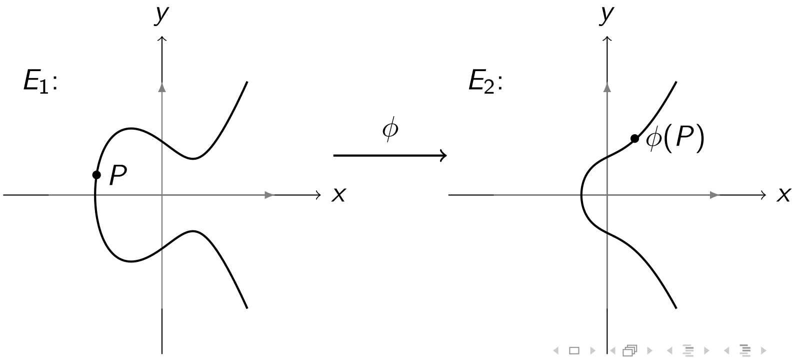
We write $E[r]$ to denote the set of all points on a curve E with order r, e.g;

$$E[r] = \{P \in E : [r]P = \mathcal{O}\}$$

Isogenies

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

$$\phi : E_1 \to E_2$$



Isogenies

Isogenies have the following two properties

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

They can also be uniquely identified by finite subgroups of E.

Lemma (Uniquely identifying isogenies)

Let E be an elliptic curve and let S be a finite subgroup of E. There is a unique elliptic curve E' and a seperable isogeny $\phi : E \to E'$ satisfying $\mathit{ker}(\phi) = S$.

Key Exchange Protocols

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

These protocols can be defined by a tuple of algorithms $(\textbf{KeyGen}, \textbf{SecAgr})$.

Generally, Alice and Bob will both run **KeyGen** to generate their keypairs (pk_A, sk_A) and (pk_B, sk_B), respectively.

Then each will run **SecAgr** with their own private key and the others public key to generate an equivalent, shared secret key.

Concerns of Cryptography

There are five rudimentary concerns in 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.

Supersingular Isogeny Diffie-Hellman

SIDH is a *role-based* key-exchange protocol where Alice and Bob use elliptic curves as their public keys and isogenies 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

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

Supersingular Isogeny Diffie-Hellman

KeyGen: Alice chooses a random number $m_A \in \mathbb{Z}/\ell_A^{e_A}\mathbb{Z}$ and computes the isogeny $\phi_A : E \to E_A$ where $E_A = E/\langle P_A, [m_A]Q_A \rangle$. Bob does the same for his parameters. The resulting private and secret keys are:

$$\begin{aligned} sk_A &= (m_A, n_A) \\ pk_A &= (E_A, \phi_A(P_B), \phi_A(Q_B)) \\ sk_B &= (m_B, n_B) \\ pk_B &= (E_B, \phi_B(P_A), \phi_B(Q_A)) \end{aligned}$$

SecAgr: Alice computes the isogeny $\phi'_A : E_B \to E_{AB}$ and Bob computes ϕ'_B analogously. The resulting shared secret value is some representation of

$$E_{AB} = E/\langle P_A + [m_A]Q_A, P_B + [m_B]Q_B \rangle$$

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:

1. Bob runs **KeyGen** (1^{λ}) to generate his keypair (pk_B, sk_B).
2. Bob runs **Commit** $()$ to generate *com* and sends it to Alice.
3. Alice sends a randomly generated "challenge" value $ch \in \omega$ and sends it to Bob.
4. Bob runs **Prove** (sk, com, ch) with output *resp*, the response to Alice's challenge.
5. Alice runs **Verify** $(pk, com, ch, resp)$ with output $b \in 0, 1$. Bob has successfully proven his identity to Alice if $b = 1$.

Concerns of Cryptography

There are five rudimentary concerns in 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.

Isogeny-based Proof of Identity

The isogeny-based proof of identity protocol outlined by De Feo et al. also consists of algorithms **KeyGen**, **Commit**, **Prove**, and **Verify**.

In the scheme, Bob can prove to Alice (or vice-versa) that he is the owner of the keypair (pk_B, sk_B) by forming many SIDH secret agreements with an arbitrary, random entity Randall, and showing that he can always come to a valid shared secret with Randall.

Isogeny-based Proof of Identity

Commit: Bob generates Randall's random point $R \in E[\ell_A^{e_A}]$ and the corresponding isogeny ψ_R. Bob then performs **SecAgr** with Randall and sets $com = (E/\langle R \rangle, E/\langle B, R \rangle)$.

Challenge: Alice responds with a random challenge bit $ch \in \{0, 1\}$.

Prove: If $ch = 0$ then Bob reveals isogenies ψ_R and $\psi'_R : E/\langle B \rangle \rightarrow E/\langle B, R \rangle$. If $ch = 1$ then Bob reveals $\phi'_B : E/\langle R \rangle \rightarrow E/\langle B, R \rangle$.

Verify: If $ch = 0$ return 1 iff R and $\phi_B(R)$ have order $\ell_A^{e_A}$ and generate the kernels of isogenies from $E \rightarrow E/\langle R \rangle$ and $E \rightarrow E/\rangle B, R \rangle$ respectively. If $ch = 0$ return 1 iff $\psi_R(B)$ has order $\ell_B^{e_B}$ and generates the kernel of an isogeny over $E/\langle R \rangle \rightarrow E/\langle B, R \rangle$.

Signature Schemes

Signature schemes are used to prove that a particular party supplied and authenticated a given message m, and that m had not been edited by an unauthorized party. These schemes consist of the algorithms **KeyGen**, **Sign**, and **Verify**.

If Bob wishes to authenticate a message m and send it to Alice, the following will take place:

1. Bob runs **KeyGen** to produce his keypair (pk_B, sk_B).
2. Bob runs **Sign** (m, sk_B) to produce a signature σ, and sends (m, σ) to Alice.
3. Alice runs **Verify** (m, σ, pk_B) returning 1 if Bob's message was successfully verified and 0 otherwise.

Concerns of Cryptography

There are five rudimentary concerns in 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.

Fiat-Shamir Transform

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

The transform works by first circumventing the need for challenge input from the verifier, turning it into a non-interactive identification scheme.

Then, the message m is factored into the identification protocol, allowing the prover to verify not just their own identity, but the authenticity of a message as well.

Applying Fiat-Shamir To an IPoI

Returning to our outline for a generic, interactive identification protocol, we would make the following changes:

1. Bob runs **KeyGen** (1^λ) to generate his keypair (sk, pk).
2. Bob runs **Commit** () to generate *com* and sends it to Alice.
3. ~~Alice sends a randomly generated "challenge" value~~ $\cancel{ch \in \omega}$ ~~and sends it to Bob.~~
3. Bob computes the challenge value as $ch = H(com, m)$ for some cryptographically secure hash function H and for message m.
4. Bob runs **Prove** (sk, com, ch) with output *resp*, the response to Alice's challenge.
5. Alice runs **Verify** $(pk, com, ch, resp)$ with output $b \in 0, 1$. Bob has successfully proven his identity to Alice if $b = 1$.

Yoo Signatures

The work of Yoo et al. applies the Fiat-Shamir transform to the isogeny-based identification scheme turning it into a general purpose isogeny-based signature scheme.

To generate a signature σ with λ-bit security requires running 2λ SIDH key-exchanges.

Yoo Signatures

A Yoo signature σ has the following components:

- *com* – A list of 2λ pairs of curves,
- *ch* – 2λ random bits derived from H, and
- *resp* – A list of size 2λ where the entry is either $\psi_R(B)$ or $(R, \phi_B(R))$.

Recap

- Isogenies are maps between elliptic curves, and can be uniquely identified by a subgroup of the domain curve.
- A key-exchange protocol can be derived with isogenies as private keys and the corresponding codomain curves as public keys.
- An isogeny-based identity proving scheme with n-bit security can be constructed by performing $2n$ SIDH key-exchanges with a random entity.
- Yoo signatures are derived by applying the fiat-shamir transform to this proof of identity protocol.

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

A $\mathbb{F}_{p^2}$ element $q = (q_a, q_b)$ can be inverted using only $\mathbb{F}_p$ operations like so:

$$q^{-1} = \left(\frac{q_a}{q_a^2 + q_b^2}, \frac{-q_b}{q_a^2 + q_b^2}\right)$$

Batching Inversions

Also, n many $\mathbb{F}_{p^2}$ inversions can be reduced to one $\mathbb{F}_{p^2}$ inversion and $3(n-1)$ $\mathbb{F}_{p^2}$ multiplications.

Input: $\{x_0, x_1, ..., x_{n-1}\}$

$a_0 \leftarrow x_0$
for `i = 1..(n-1)` **do**
  $a_i \leftarrow a_{i-1} \cdot x_i$

upward-percolation

$inv \leftarrow a_{n-1}^{-1}$ $\longrightarrow$ inversion phase

for `i = (n-1)..1` **do**
  $x_i^{-1} \leftarrow a_{i-1} \cdot inv$
  $inv \leftarrow inv \cdot x_i$

downward-percolation

$x_0^{-1} = inv$
return $\{x_0^{-1}, x_1^{-1}, ..., x_{n-1}^{-1}\}$

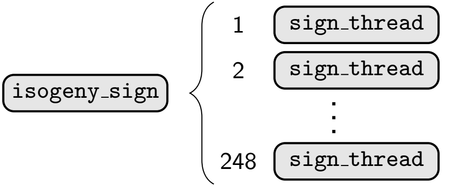
Partial Batched Inversions

These two techniques can be combined in the following way, yielding the fastest method of inverting n-many arbitrary $\mathbb{F}_{p^2}$ elements.

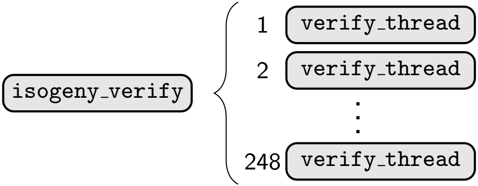
for `i = 0..(n-1)` **do**
    $den_i \leftarrow (x_i)_a^2 + (x_i)_b^2 \pmod p$
$a_0 \leftarrow den_0$
for `i = 1..(n-1)` **do**
    $a_i \leftarrow a_{i-1} \cdot den_i \pmod p$
upward-percolation
$inv \leftarrow a_{n-1}^{-1} \pmod p$ ⟶ inversion phase
for `i = n-1..1` **do**
    $a_i \leftarrow inv \cdot dest_{i-1} \pmod p$
    $inv \leftarrow inv \cdot den_i \pmod p$
downward-percolation
$a_0 \leftarrow a_{inv}$
for `i = 0..(n-1)` **do**
    $(xinv_i)_a \leftarrow a_i \cdot (x_i)_a \pmod p$
    $(xinv_i)_b \leftarrow a_i \cdot -(x_i)_b \pmod p$
    $x_i^{-1} \leftarrow \{(xinv_i)_a, (xinv_i)_b\}$
return $\{x_0^{-1}, x_1^{-1}, ..., x_{n-1}^{-1}\}$

Structure of the Yoo Signature Implementation

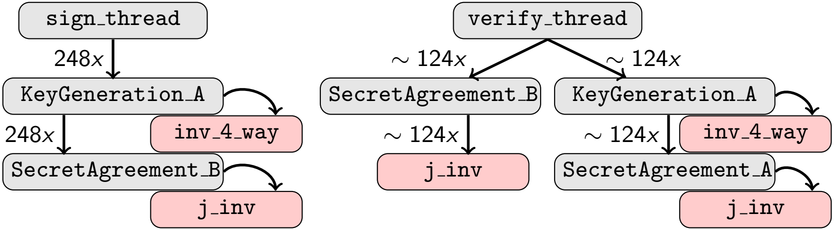
1. The signer executes **Sign** by spawning a thread running `sign_thread` for every iteration of the signing procedure



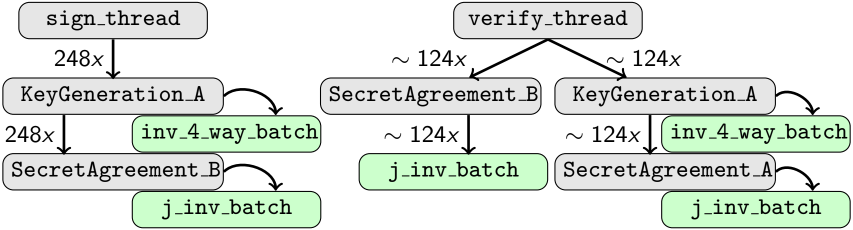
2. The verifier then executes **Verify** in a similar fashion.



Batching Across Threads



Batching Across Threads



Performance Measurements for Partial Batched Inversions

	Without Batching	With Batching
KeyGen	84,499,270	84,499,270
Sign	4,950,023,141.65	4,552,062,482.52
Verify	3,466,703,991.09	3,173,340,239.46

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

$$\phi_A(P_B) = \alpha_P R_1 + \beta_P R_2$$

$$\phi_A(Q_B) = \alpha_Q R_1 + \beta_Q R_2$$

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 $6'logp$ bits.

SIDH Public Key Compression

Costello et al. note that in public key compression, because one of α_P and β_P will be non-zero, one will be able to be normalized out.

This notion of normalizing $P = \alpha R_1 + \beta R_2$ to $P = R_1 + \alpha^{-1}\beta R_2$ is also utilized in our implementation.

Compressing $\psi(S)$

We use these SIDH public key compression techniques to compress every EC point component of a Yoo signature which is the result of applying Randall's isogeny to the signers private key.

A Yoo signature σ has the following components:

- *com* – A list of 2λ pairs of curves,
- *ch* – 2λ random bits derived from H, and
- *resp* – A list of size 2λ where the entry is either $\psi_R(B)$ or $(R, \phi_B(R))$.

Compressing $\psi(S)$

This allows us to transmit every $\psi(S)$ as two elements of $\mathbb{Z}/n\mathbb{Z}$ (with $n = \ell_B^{e_B}$ when Bob is signing and $\ell_A^{e_A}$ when Alice is signing): $\psi(S) = \alpha R_1 + \beta R_2$.
We then normalize to $\psi(S) = R_1 + \alpha^{-1}\beta R_2$ and are able to send $\alpha^{-1}\beta$ in the place of $\psi(S)$. For each of these we also send a bit b denoting whether α or β was inverted.

Final Results

	Key Gen	Sign	Verify
SIDH	84,499,270	4,950,023,142	3,466,703,991
SIDH Batched	84,499,270	4,552,062,483	3,173,340,239
SIDH Compressed	84,499,270	10,224,610,997	4,472,444,450
SIDH C+B	84,499,270	10,016,427,840	4,326,294,568
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

Final Results

	Public Key	Private Key	Signature
SIDH	768	48	88,064
SIDH Compressed	768	48	69,632
Sphincs	32	64	8,080 - 16,976
Rainbow	152,097 - 192,241	100,209 - 114,308	64 - 104
qTESLA	4,128	2,112	3,104
Picnic	33	49	34,004 - 53,933
RSA	384	256	384
ECDSA	32	32	32

Next Steps for Implementation

- Continue improving code performance: 1 inversion call in signing and verifying paths that has yet to be processed for batched inversions.
- Continue efforts to test modifications for correctness and application security.
- Work to introduce Yoo signatures and our improved implementations into the world of open source isogeny-based cryptosystems.

Questions?

References

Reza Azerderalhsh, David Jao, Kassem Kalach, Brian Koziel, and Christopher Leonardi (2016)
Key Compression for Isogeny-Based Cryptosystems

Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Rene, and David Urbanik (2016)
Efficient Compression of SIDH Public Keys
IACR-CRYPTO-2016

Craig Costello, David Jao, Patrick Longa, Michael Naehrig, Joost Rene, and David Urbanik (2016)
Efficient Compression of SIDH Public Keys
IACR-CRYPTO-2016