SIKE in Hardware

Reza Azarderakhsh
Florida Atlantic University
CHES 2019

Atlanta, GA, USA
Quantum Threat to Information Security

- Large-scale quantum computers could break some encryption schemes
- Need to migrate encryption to quantum-resistant algorithms
- When we should start the process?
Timeline

Retroactive decryption: record encrypted data now, decrypt it once you have a quantum computer

2016
Start PQ Crypto

2017
Round 1

2019
Round 2

2022-23
Standardization draft

2030?
Quantum Computers
Post-Quantum Key-Exchange

- Lattice-based
- Code-based
- Isogeny-based

Post-Quantum Signatures

- Lattice-based
- Hash-based
- Multivariate-based
- Zero-Knowledge based
Open Questions about Post-Quantum Cryptography

• Design better post-quantum cryptosystems
• Improve classical and quantum attacks
• Pick parameter sizes
• Develop fast, efficient, and secure implementations
• Integrate them into the existing infrastructure
**Architecture Selection for Cryptographic Design**

**HW only**

- Highly **optimized** for dedicated purpose (power consumption, execution time, security)
- Extra HW costs
- Limited flexibility
- HW design effort/complexity

**HW/SW**

- Good **trade-off** between optimization/costs (still fast but less design effort/complexity easier to handle)
- Higher flexibility
- Not straight-forward to find optimal HW/SW partitioning
- Extra HW costs
- Less optimized than HW-only

**SW only**

- Limited HW costs (code/data storage)
- Highest **flexibility**
- Minimal HW design effort/eases handling of complexity (programming)
- Not optimized (energy, consumption, performance)
FPGAs: Field Programmable Gate Arrays

FPGAs are composed of:

- Programmable logic cells
- A configurable routing matrix
- Configurable input/output cells
- Embedded memory blocks
- Small embedded multipliers
- etc.

Inside a logic cell:

- Connections to the routing matrix
- Programmable lookup-tables
  - 4 inputs, 1 output
  - 6 inputs, 1 output
  - 6 inputs, 2 outputs

- Optional registers
  - Free pipelining
- More logic for fast carry-propagation

18-bit $\times$ 18-bit multiplier blocks
FPGAs vs. ASIC

+ prototyping
+ re-usability
+ short time to market
+ simpler design cycle
+ programmable in the field
+ hardware/software co-design

– speed
– silicon footprint
– power and energy consumption
– low cost for high volumes
– better performance
– reconfigurability and redundancy
A brief history of public key cryptography

<table>
<thead>
<tr>
<th>Cryptosystem</th>
<th>Hard Problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diffie-Hellman (1976)</td>
<td></td>
</tr>
<tr>
<td>Elliptic curve cryptography (1986)</td>
<td>Discrete logarithms</td>
</tr>
<tr>
<td>Pairing-based cryptography (2000)</td>
<td></td>
</tr>
<tr>
<td>RSA (1977)</td>
<td></td>
</tr>
<tr>
<td>Rabin (1978)</td>
<td>Factoring integers</td>
</tr>
<tr>
<td>Composite residues (1985)</td>
<td></td>
</tr>
<tr>
<td>Code-based cryptography (1979)</td>
<td>Decoding linear codes</td>
</tr>
<tr>
<td>Lattice-based / NTRU (1996)</td>
<td></td>
</tr>
<tr>
<td>Isogeny based / CRS (1996)</td>
<td>Finding Short lattice vectors</td>
</tr>
<tr>
<td>SIDH / SIKE (2011)</td>
<td>Computing isogenies</td>
</tr>
</tbody>
</table>
History of Supersingular Isogeny-based cryptography

• [2006]: Birth of Supersingular isogeny-based cryptosystem
  • Charles-Goren-Lauter
  • built hash function from supersingular isogeny graph

• [2011]: Supersingular isogeny Diffie-Hellman key exchange (SIDH)
  • Jao-De Feo

• [2017]: Supersingular isogeny key encapsulation (SIKE)
  • SIKE Team (https://sike.org)
Supersingular Isogeny Diffie-Hellman (Jao and De Feo, 2011):
- A key-exchange protocol, similar to Diffie-Hellman, using isogenies between supersingular elliptic curves

Why isogenies?
- Because they seem to be quantum-resistant

Why supersingular elliptic curves?
- There is a quantum subexponential attack for ordinary (i.e. non-supersingular) curves (Childs, Jao, and Soukharev 2014)

Supersingular Isogeny Key Encapsulation
- A more secure (and slower) version of SIDH, with random padding and protection against active attacks.
An elliptic curve over prime field: $E_W/F_p : y^2 = x^3 + ax + b$.

To avoid infinite sets, we choose $x, y, a, b \in F_p$.

Example: SAGE: $E=EllipticCurve(GF(11),[1,6])$

- Elliptic Curve defined by $y^2 = x^3 + x + 6$ over Finite Field of size 11

SAGE: $E.points()$


SAGE: $E.cardinality()$ or $E.order()=13$

- $E.is_ordinary(): True$

- $E.is_supersingular(): False$
Point Addition

- Let $E$ be an elliptic curve.
- Suppose $P, Q \in E$.
- We want to add $P$ and $Q$.
- Draw a line through $P$ and $Q$.
- Find where this line crosses $E$.
- Reflect around the $x$-axis.
- The reflected point is $P + Q$.
Point Doubling

- Let $E$ be an elliptic curve.
- Suppose $P, Q \in E$.
- To compute $P + Q$ when $P = Q$:
  - Draw the tangent line through $P$.
  - Find where this line crosses $E$.
  - Reflect around the $x$-axis.
  - The reflected point is $P + P$. 

\[
E/\mathbb{R} : y^2 = x^3 - 2: \text{doubling.}
\]
Montgomery Curves

- 1987 Montgomery: All Montgomery curves are elliptic curves.
- Not all elliptic curves can be written in Montgomery form.
\[ by^2 = x^3 + ax^2 + x \]

- It has been observed that the x-coordinate of \( R = P + Q \) depends only on the x-coordinates of \( P, Q, \) and \( P - Q \).

\[
\begin{align*}
(x_3 + y_3) - (x_2, y_2) &= (x_1, y_1) \\
(x_3 + y_3) + (x_2, y_2) &= (x_5, y_5)
\end{align*}
\]

\[ \Rightarrow x_5 = \frac{(x_2 x_3 - 1)^2}{x_1 (x_2 - x_3)^2} \]

- Similarly when \( P = Q \) it is true for doubling: x-only doubling.

\[ 2(x_2 + y_2) = (x_4, y_4) \Rightarrow x_4 = \frac{(x_2^2 - 1)^2}{4x_2(x_2^2 + ax_2 + 1)} \]

- Can compute \( P + Q \) from \( \{P, Q, P - Q\} \) without y-coordinates.
- Use projective coordinates: points \((X : Z)\) with \( x = X/Z \)
- Cheap differential addition \( 4M + 2S \) and doubling \( (2M + 2S) \)
Real-world usage: Curve25519

- Proposed by Dan Bernstien 2006:
  \[ E_M/F_p : y^2 = x^3 + 486662x^2 + x \]
- Over prime number \( p = 2^{255} - 19 \)
- Used for DH key exchange
Montgomery Ladder

- Point multiplication: \( Q = k \cdot P \)

```plaintext
function scalar-mult( k, P):
    T0 ← O
    T1 ← P
    for i ← n − 1 downto 0:
        if \( k_i = 1 \):
            T0 ← T0 + T1
            T1 ← 2T1
        else:
            T1 ← T0 + T1
            T0 ← 2T0
    return T0
```

- Properties:
  - perform one addition and one doubling at each step
  - ensure that both results are used in the next step
- Example: \( k = 26 = (11010)_2 \)
Supersingular Elliptic Curves

- Let $E/\mathbb{F}_q$ be an elliptic curve with $q = p^n$.
- $E$ is supersingular if $p \mid (q + 1 - \#E(\mathbb{F}_q))$. Otherwise, it is ordinary.
- Special cases:
  - When $E/\mathbb{F}_p$ supersingular and $\#E(\mathbb{F}_p) = P + 1$
  - When $E/\mathbb{F}_{p^2}$ supersingular and $\#E(\mathbb{F}_{p^2}) = (P + 1)^2$

Example: EllipticCurve(GF(11),[1,0]): $y^2 = x^3 + x$

- E.is_supersingular()
  - True, E.order()=12
- EllipticCurve(GF(11^2),[1,0]): $y^2 = x^3 + x$ is also supersingular.
  - E.order()=144
Supersingular Elliptic Curves

- There are only a finite number of supersingular elliptic curves.
- All supersingular curves can be defined over $\mathbb{F}_{p^2}$.
- Over an algebraically closed field an elliptic curve is determined by its $j$-invariant.
  - It can be viewed as a way to group multiple elliptic curves into disjoint sets.
- Example: $\text{EllipticCurve}(\mathbb{GF}(11),[1,0]): y^2 = x^3 + x$
  - $j(E) = j(a, b) = 1728 \frac{4a^3}{4a^3 + 27b^2}$
  - $E.j\_invariant=1$
- The $j$-invariant determines isomorphism class over the field.
- $E_1/\mathbb{F}_{13} : y^2 = x^3 + 9x + 8$, $E_1=\text{EllipticCurve}(\mathbb{GF}(13),[9,8])$
- $E_2/\mathbb{F}_{13} : y^2 = x^3 + 3x + 5$, $E_2=\text{EllipticCurve}(\mathbb{GF}(13),[3,5])$
  - $E_1.j\_invariant()=E_2.j\_invariant()=3$
  - $E_1.is\_isomorphic(E_2): True$
Isomorphisms and Isogenies

- \( E_1 \) and \( E_2 \) isomorphic iff \( j(E_1) = j(E_2) \).
- \( E_1 \) and \( E_2 \) isogenous iff \( \#E_1 = \#E_2 \).
- \( \#E(\mathbb{F}_q) \leq q + 1 \pm 2\sqrt{q} \) (Hasse theorem)

So
- isogeny classes: \( O(\sqrt{q}) \)
- isomorphism classes: \( O(q) \)
Isogenies of elliptic curves

Definition

An isogeny of elliptic curves over $k$ is a non-zero morphism $E \to E'$ with finite kernel.

Definition

Let $E, E'/\mathbb{F}_q$ be elliptic curves and let $\ell \in \mathbb{Z}_{>0}$ be coprime to $q$. An $\ell$-isogeny $f : E \to E'$ is an isogeny with #$\ker(f) = \ell$.

Fact

An isogeny is uniquely determined by its kernel.

- Write $\phi_G : E \to E/G$ for the isogeny from $E$ with kernel $G$.
- Vélu’s formulas [1971] compute the $\ell$-isogeny from its kernel in time $O(\ell)$.
Isogenies of elliptic curves

- We call an isogeny cyclic if its kernel is cyclic.
- The kernel of a cyclic $\ell$-isogeny is generated by an $\ell$-torsion point.
  - An $\ell$-torsion point is a point $P \in E(k)$ such that $[\ell]P = P_\infty$.
- We will work on isogenies with big kernels.
Isogeny graph: Wouter Castryck

2-isogenies

3-isogenies
SIDH overview

1. Public parameters: Supersingular elliptic curve \( E \) over \( \mathbb{F}_{p^2} \).

2. Alice chooses a kernel \( A \subset E(\mathbb{F}_{p^2}) \) and sends \( E/A \) to Bob.

3. Bob chooses a kernel \( B \subset E(\mathbb{F}_{p^2}) \) and sends \( E/B \) to Alice.

4. The shared secret is

\[
E/\langle A, B \rangle = (E/A)/\phi_A(B) = (E/B)/\phi_B(A).
\]

The core operation in SIDH is to compute \( \phi_A : E \rightarrow E/A \) given \( A \).
SIDH: from Cloudflare
SIDH Overview

• We are interested in the set of supersingular curves (up to isomorphism) over a specific field
• Prime $p = 2^{e_A} \cdot 3^{e_B} \cdot f \pm 1$
• $f = 1$ (Efficiency)
• $e_A$ is even (Efficiency)
• Balanced isogeny graph size, $2^{e_A} \approx 3^{e_B}$ (Security)
• Elliptic curves over $\mathbb{F}_{p^2}$, $\#E = (p \mp 1)^2$
• Supersingular $j$-invariants: $\#S_{p^2} \approx \lfloor p/12 \rfloor$ (isogenous elliptic curves)

Prime $p = 2^3 \cdot 3^2 - 1 = 71$, $\#E = 72^2$, $\#S_{p^2} = 7$
SIDH Parameter Selection

- Finite extension field formed as $\mathbb{F}_{p^2} = \mathbb{F}_p(i)$ with $i^2 + 1 = 0$ (Efficiency)

- Starting curve selection:
  - Montgomery curve: $E_0/\mathbb{F}_{p^2} : y^2 = x^3 + x$ (where $j(E_0) = 1728$)
  - SIKE round 2: $E_0/\mathbb{F}_{p^2} : y^2 = x^3 + 6x^2 + x$ (where $j(E_0) = 287496$) (Security)
  - Torsion points in $E_0[2^{e_A}]$ and $E_0[3^{e_B}]$
Isogeny Graphs

Vertices: All isogenous elliptic curves over $\mathbb{F}_{p^2}$.

Edges: Isogenies of degree $\ell$.

With Isogeny of degree $\ell$, we get a connected $(\ell+1)$-regular graph.

2-isogeny graph

3-isogeny graph
Public Parameters

$$E_0 / \mathbb{F}_{p^2}$$
$$\{P_A, Q_A\} \in E_0[2^{e_A}]$$
$$\{P_B, Q_B\} \in E_0[3^{e_B}]$$

$E_0 : y^2 = x^3 + x$

Alice

$$P_A = (53, 55)$$
$$Q_A = (18, 27w + 44)$$

Bob

$$P_A = (7w + 20, 31w + 50)$$
$$Q_A = (21w + 64, 38w + 13)$$
$s_A \in [0, 2^{e_A}]$
$s_B \in [0, 3^{e_B}]$

$\begin{align*}
\text{Alice} & : s_A = 6 \\
\text{Bob} & : s_B = 3
\end{align*}$
Public Key Generation

\[ E_A = E_0 / \langle A \rangle \]
\[ \{ R_A, S_A \} = \{ \phi_A(P_B), \phi_A(Q_B) \} \]
\[ E_B = E_0 / \langle B \rangle \]
\[ \{ R_B, S_B \} = \{ \phi_B(P_A), \phi_B(Q_A) \} \]
\[ E_{AB} = E_B / \langle A' \rangle \sim = E_A / \langle B' \rangle \]

Alice

Bob

\[ E_0 : y^2 = x^3 + x \]
Public Key Generation

$E_A = E_0 / \langle A \rangle$
$\{ R_A, S_A \} = \{ \phi_A(P_B), \phi_A(Q_B) \}$
$E_B = E_0 / \langle B \rangle$
$\{ R_B, S_B \} = \{ \phi_B(P_A), \phi_B(Q_A) \}$
$E_{AB} = E_B / \langle A' \rangle \sim E_A / \langle B' \rangle$

$\phi_A : E_0 \rightarrow E_A$

$\phi_B : E_0 \rightarrow E_B$

$E_0 : y^2 = x^3 + x$
Public Key Generation

\[
E_A = E_0 / \langle A \rangle \\
\{R_A, S_A\} = \{\phi_A(P_B), \phi_A(Q_B)\}
\]

\[
E_B = E_0 / \langle B \rangle \\
\{R_B, S_B\} = \{\phi_B(P_A), \phi_B(Q_A)\}
\]

\[
\phi_A = \ker(\phi_A) = \langle A \rangle = \langle P_A + [s_A]Q_B \rangle
\]

\[
\phi_B = \ker(\phi_B) = \langle B \rangle = \langle P_B + [s_B]Q_A \rangle
\]

\[
E_{AB} = E_B / \langle A' \rangle \approx E_A / \langle B' \rangle
\]

\[
E_0 : y^2 = x^3 + x
\]

\[
\phi_A : E_0 \rightarrow E_A \\
E_A : y^2 = x^3 + 22x + 35
\]

\[
\phi_B : E_0 \rightarrow E_B \\
E_B : y^2 = x^3 + 63x + (55w + 16)
\]
Key exchange

\[ E_A = E_0 / \langle A \rangle \]
\[ \{ R_A, S_A \} = \{ \phi_A(P_B), \phi_A(Q_B) \} \]

\[ E_B = E_0 / \langle B \rangle \]
\[ \{ R_B, S_B \} = \{ \phi_B(P_A), \phi_B(Q_A) \} \]

Alice

Bob

\[ \phi_A : E_0 \rightarrow E_A \]
\[ E_A : y^2 = x^3 + x \]
\[ E_B : y^2 = x^3 + 22x + 35 \]

\[ \phi_B : E_0 \rightarrow E_B \]
\[ E_B : y^2 = x^3 + 63x + (55w + 16) \]
Shared Secret Generation

\[ \Phi_A \ker(\phi_A) = \langle A \rangle = \langle P_A + [s_A]Q_A \rangle \]
\[ \Phi_B \ker(\phi_B) = \langle B \rangle = \langle P_B + [s_B]Q_B \rangle \]

\[ E_A = E_0/\langle A \rangle \]
\[ \{R_A, S_A\} = \{\phi_A(P_B), \phi_A(Q_B)\} \]

\[ E_B = E_0/\langle B \rangle \]
\[ \{R_B, S_B\} = \{\phi_B(P_A), \phi_B(Q_A)\} \]

Alice

Bob

\[ E_A : y^2 = x^3 + 63x + (55w + 16) \]

\[ E_B : y^2 = x^3 + 22x + 35 \]
Shared Secret Generation

\[
E_A = E_0/\langle A \rangle
\]
\[
\{ R_A, S_A \} = \{ \phi_A(P_B), \phi_A(Q_B) \}
\]

\[
E_B = E_0/\langle B \rangle
\]
\[
\{ R_B, S_B \} = \{ \phi_B(P_A), \phi_B(Q_A) \}
\]

**Alice**

\[
E_B: y^2 = x^3 + 63x + (55w + 16)
\]

\[
\phi_{AB}: E_B \rightarrow E_{AB}
\]

**Bob**

\[
E_B: y^2 = x^3 + 22x + 35
\]

\[
\phi_{BA}: E_A \rightarrow E_{BA}
\]

\[
\ker(\phi_A) = \langle A \rangle = \langle P_A + [5A]Q_A \rangle
\]

\[
\ker(\phi_B) = \langle B \rangle = \langle P_B + [8B]Q_B \rangle
\]

\[
\phi'_A: \ker(\phi'_A) = \langle A' \rangle = \langle R_B + [s_A]S_B \rangle
\]

\[
\phi'_B: \ker(\phi'_B) = \langle B' \rangle = \langle R_A + [s_B]S_A \rangle
\]
Shared Secret Generation

Alice

Bob

\[ E_A = E_0/\langle A \rangle \]
\[ \{ R_A, S_A \} = \{ \phi_A(P_B), \phi_A(Q_B) \} \]

\[ \phi_A : E_B \rightarrow E_{AB} \]
\[ E_B : y^2 = x^3 + 21w + 14x + (57w + 21) \]

\[ E_B : y^2 = x^3 + (21w + 14)x + (57w + 21) \]

\[ \phi_B : E_A \rightarrow E_{BA} \]
\[ E_B : y^2 = x^3 + 2x + 35 \]
Shared Secret Generation

\[ E_A = E_0 / \langle A \rangle \]
\[ \{ R_A, S_A \} = \{ \phi_A(P_B), \phi_A(Q_B) \} \]

\[ E_B = E_0 / \langle B \rangle \]
\[ \{ R_B, S_B \} = \{ \phi_B(P_A), \phi_B(Q_A) \} \]

\[ E_{AB} = E_B / \langle A' \rangle \cong E_A / \langle B' \rangle = E_{BA} \]
**SIDH: Security**

- **Hard problem:** Given $P, Q \in E$ and $\phi(P), \phi(Q) \in \phi(E)$ $\Rightarrow$ compute $\phi$.
- **Best known attack:** classical $O(p^{1/4})$ and quantum $O(p^{1/6})$. 
### SIKE Round 2 Key sizes

<table>
<thead>
<tr>
<th>NIST Level</th>
<th>Prime size (bits) Round 1</th>
<th>Prime size (bits) Round 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (AES128)</td>
<td>503</td>
<td>434</td>
</tr>
<tr>
<td>2 (SHA256)</td>
<td>—</td>
<td>503</td>
</tr>
<tr>
<td>3 (AES192)</td>
<td>751</td>
<td>610</td>
</tr>
<tr>
<td>5 (AES256)</td>
<td>964</td>
<td>751</td>
</tr>
</tbody>
</table>
## SIKE Round 2 Key sizes

<table>
<thead>
<tr>
<th>NIST Level</th>
<th>Prime size (bits)</th>
<th>Prime</th>
<th>Public key size (bytes)</th>
<th>Compressed PK size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>434</td>
<td>$2^{216} \cdot 3^{137} - 1$</td>
<td>330</td>
<td>196</td>
</tr>
<tr>
<td>2</td>
<td>503</td>
<td>$2^{250} \cdot 3^{159} - 1$</td>
<td>378</td>
<td>224</td>
</tr>
<tr>
<td>3</td>
<td>610</td>
<td>$2^{305} \cdot 3^{192} - 1$</td>
<td>462</td>
<td>273</td>
</tr>
<tr>
<td>5</td>
<td>751</td>
<td>$2^{372} \cdot 3^{239} - 1$</td>
<td>564</td>
<td>331</td>
</tr>
</tbody>
</table>
SIDH Computations

- $F_p$ Arithmetic
- $F_{p^2}$ Arithmetic
- Group Ops
- Extended group ops
- PQC protocols
- Addition
- Multiplication
- Inversion
- Point Addition
- Point Doubling
- Isogeny Evaluation and Computation
- Double Point Multiplication
- Large Degree Isogeny Computation

R. Azarderakhsh (Florida Atlantic University)
SIDH Computations

\[ F_p \text{ Arithmetic} \]
\[ F_{p^2} \text{ Arithmetic} \]

Group Ops

Point Addition
Point Doubling
Isogeny Evaluation and Computation

Addition     Mult.     Squaring     Inversion

Double Point Multiplication
Large Degree Isogeny Computation

SIDH
SIDH Computations

- Extended group ops
- Double Point Multiplication
- Large Degree Isogeny Comput.
- Point Addition
- Point Doubling
- Isogeny Evaluation and Computation
- Addition
- Mult.
- Squaring
- Inversion
- Addition
- Mult.
- Inversion
- Group Ops
- $F_p$ Arithmetic
- $F_{p^2}$ Arithmetic
SIDH Computations

- SIDH
  - PQC protocols
  - SIDH
  - Double Point Multiplication
  - Large Degree Isogeny Comput.
  - Point Addition
  - Point Doubling
  - Isogeny Evaluation and Computation
  - Double Point Multiplication
  - Large Degree Isogeny Comput.
  - Add. Mult.
  - Squaring
  - Inversion

- Extended group ops

- Group Ops

- $F_p$ Arithemtic
  - Addition
  - Mult.
  - Inversion

- $F_{p^2}$ Arithemtic
  - Addition
  - Mult.
  - Squaring
  - Inversion
Three-Point Differential Ladder

- Compute $R = P + [k]Q$
- Jao et al. (2014) - Cost: 2PA + 1PD
- Hernandez et al. (2017) - Cost: 1PA + 1PD
- Example: $k = 9 = 1001_b$
Large Degree Isogeny Computations

- Vélu’s formulas is only suitable for small degree isogenies: $\phi_G : E \to E/G$

- Evaluate $\phi_G$ as a chain of small-degree isogenies
- Complexity: $O(e^2 \cdot \ell)$. Exponentially smaller than $\ell^e$.
  - Jao and De Feo [2014]: Optimal strategy improves this to $O(e \log e \cdot \ell)$.
  - For SIDH we only use isogenies of degree $\ell^e$ for $\ell \in \{2, 3\}$. 
Large Degree Isogeny Computations

- Get isogeny Kernel $[\ell^{e-i-1}]R_i$
- Compute Isogenies $\phi_i := E_i/\langle[\ell^{e-i-1}]R_i\rangle$
- Compute $E_{i+1} = \phi_i(E_i)$
- Push points to new curve $R_{i+1} = \phi_i(R_i)$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$

- e.g., $\phi : E_0/\langle R_0 \rangle$, $\text{ord}(R_0) = \ell^7$
Large Degree Isogeny Computations

Base Curve

Point mult
by $\ell$

Apply $\ell$-isogeny

Point in queue

Get $\ell$-isogeny

$R_0 \ [\ell] R_0 R_1$

$[\ell^6] R_0 \ [\ell^3] R_3 R_6$

$E_0 \ \phi_0$

Order of $R_0$ is $\ell^7$

$e = 7$

$R_0 \ E_0$
Large Degree Isogeny Computations

e = 7

Order of $[\ell]R_0$ is $\ell^6$
Large Degree Isogeny Computations

\[ \ell \text{-isogeny} \]

Base Curve

Point mult by \( \ell \)

Apply \( \ell \)-isogeny

Point in queue

Get \( \ell \)-isogeny

\[ \ell \]

\[ \ell^2 \]

\[ \ell^3 \]

\[ \ell^4 \]

\[ \ell^5 \]

\[ \ell^6 \]

\[ \ell^7 \]

\[ E_0 \]

\[ R_0 \]

\[ \phi_0 \]

Order of \( [\ell^2]R_0 \) is \( \ell^5 \)

\( e = 7 \)
Large Degree Isogeny Computations

Order of \([\ell^3]R_0\) is \(\ell^4\)

\[\phi_0 \quad \text{Get } \ell \text{-isogeny} \]

Base Curve

Point in queue

Point mult by \(\ell\)

Apply \(\ell\)-isogeny

\(R_0\)

\([\ell]R_0\)

\([\ell^2]R_0\)

\([\ell^3]R_0\)

\([\ell^4]R_0\)

\([\ell^5]R_0\)

\([\ell^6]R_0\)

\(E_0\)

\(e = 7\)
Large Degree Isogeny Computations

Base Curve
Point mult by $\ell$
Apply $\ell$-isogeny
Point in queue
Get $\ell$-isogeny

$e = 7$

Order of $[\ell^4]R_0$ is $\ell^3$
Large Degree Isogeny Computations

Point mult by $\ell$
Apply $\ell$-isogeny
Point in queue
Get $\ell$-isogeny

Order of $[\ell^5]R_0$ is $\ell^2$

$e = 7$
Large Degree Isogeny Computations

\[ e = 7 \]

Order of \([\ell^6]R_0\) is \(\ell\)
Large Degree Isogeny Computations

\[ e = 7 \]

\[ \phi_0 := E_0 / \langle [\ell^6]R_0 \rangle \]

\[ E_i = \phi_0(E_0) \]
Large Degree Isogeny Computations

Base Curve

Point mult by $\ell$

Apply $\ell$-isogeny

Point in queue

Get $\ell$-isogeny

$\phi_0$

$R_0$

$[\ell]R_0$

$[\ell^3]R_0$

$[\ell^6]R_0$

$E_0$

$R_0$

$\phi_0$

$[\ell]R_0$

$[\ell^3]R_0$

$[\ell^6]R_0$

$E_1$

$R_1$

$\phi_0$

$[\ell^5]R_1$

$[\ell^3]R_1$

$E_2$

$[\ell^4]R_0$

$[\ell^3]R_0$

$[\ell^6]R_0$

$E_3$

$[\ell^5]R_0$

$[\ell^3]R_0$

$E_4$

$[\ell^5]R_0$

$[\ell^3]R_0$

$E_5$

$[\ell^5]R_0$

$[\ell^3]R_0$

$E_6$

$[\ell^5]R_0$

$[\ell^3]R_0$

$e = 7$

$R_1 = \phi_0(R_0)$

Order of $[\ell^5]R_1$ is $\ell$
Large Degree Isogeny Computations

\[ e = 7 \]

\[
\phi_1 := E_1 / \langle [\ell^5] R_1 \rangle \\
E_2 = \phi_1(E_1)
\]
Large Degree Isogeny Computations

\[ R_2 = \phi_1(R_1) \]
Order of \([\ell^3]R_2\) is \(\ell^2\)

\[ e = 7 \]
Large Degree Isogeny Computations

$e = 7$

Order of $[\ell^4]R_2$ is $\ell$
Large Degree Isogeny Computations

$$e = 7$$

$$\phi_2 := E_2 / \langle [\ell^4]R_2 \rangle$$

$$E_3 = \phi_2(E_2)$$

Base Curve
Point in queue
Point mult by $\ell$
Apply $\ell$-isogeny
Get $\ell$-isogeny

$$\phi_0$$

$$[\ell]R_0$$
$$[\ell^2]R_0$$
$$[\ell^3]R_0$$
$$[\ell^4]R_0$$
$$[\ell^5]R_0$$
$$[\ell^6]R_0$$

$$\phi_0$$

$$[\ell]R_0$$
$$[\ell^2]R_0$$
$$[\ell^3]R_0$$
$$[\ell^4]R_0$$
$$[\ell^5]R_0$$
$$[\ell^6]R_0$$

$$\phi_0$$

$$\phi_1$$

$$\phi_2$$

$$\phi_0$$

$$\phi_1$$

$$\phi_2$$
Large Degree Isogeny Computations

$e = 7$

$R_3 = \phi_2(R_2)$

Order of $[\ell^3]R_3$ is $\ell$

Base Curve

Point in queue

Point mult by $\ell$

Apply $\ell$-isogeny

Get $\ell$-isogeny

$R_0$

$[\ell]R_0$

$\phi_0$

$[\ell^2]R_0$

$[\ell^3]R_0$

$[\ell^4]R_0$

$[\ell^5]R_0$

$[\ell^6]R_0$

$R_1$

$[\ell]R_1$

$[\ell^3]R_1$

$[\ell^4]R_1$

$[\ell^5]R_1$

$\phi_1$

$[\ell^6]R_1$

$R_2$

$[\ell^2]R_2$

$[\ell^3]R_2$

$[\ell^4]R_2$

$[\ell^5]R_2$

$\phi_2$

$[\ell^6]R_2$

$R_3$

$\phi_3$

$[\ell]R_3$

$[\ell^2]R_3$

$[\ell^3]R_3$

$[\ell^4]R_3$

$[\ell^5]R_3$

$\phi_4$

$[\ell^6]R_3$

$R_4$

$[\ell]R_4$

$[\ell^2]R_4$

$[\ell^3]R_4$

$[\ell^4]R_4$

$\phi_5$

$[\ell^6]R_4$

$R_5$

$[\ell]R_5$

$[\ell^2]R_5$

$[\ell^3]R_5$

$\phi_6$

$[\ell^6]R_5$

$R_6$

$[\ell]R_6$
Large Degree Isogeny Computations

\[ \phi_3 := E_3 / \langle [\ell^3] R_3 \rangle \]
\[ E_4 = \phi_3(E_3) \]

\[ e = 7 \]
Large Degree Isogeny Computations

$e = 7$

$R_4 = \phi_3(R_3)$
Order of $R_4$ is $\ell^3$
Large Degree Isogeny Computations

$e = 7$

Order of $[\ell]R_4$ is $\ell^2$

Base Curve

Point in queue

Point mult by $\ell$

Apply $\ell$-isogeny

Get $\ell$-isogeny

$\phi_0$

$E_0$

$[\ell]R_0$ $[\ell^2]R_0$ $[\ell^3]R_0$ $[\ell^4]R_0$ $[\ell^5]R_0$ $[\ell^6]R_0$ $R_1$ $R_2$ $R_3$ $R_4$ $R_5$ $R_6$

$E_1$

$E_2$

$E_3$

$E_4$

$\phi_0$

$\phi_1$

$\phi_2$

$\phi_3$

$\phi_4$
Large Degree Isogeny Computations

$e = 7$

Order of $[\ell^2]R_4$ is $\ell$

$[\ell^6]R_0 \rightarrow [\ell^5]R_0 \rightarrow [\ell^4]R_0 \rightarrow [\ell^3]R_0 \rightarrow [\ell^2]R_0 \rightarrow [\ell]R_0 \rightarrow R_0$

$[\ell^6]R_0 \rightarrow [\ell^5]R_0 \rightarrow [\ell^4]R_0 \rightarrow [\ell^3]R_0 \rightarrow [\ell^2]R_4 \rightarrow E_0$

$E_0 \rightarrow R_0 \rightarrow \phi_0 \rightarrow \phi_0 \rightarrow \phi_0$
Large Degree Isogeny Computations

$e = 7$

$\phi_4 := E_4 / \langle [\ell^2] R_2 \rangle$

$E_5 = \phi_4(E_4)$

$[\ell^6] R_0 \xrightarrow{\phi_4} E_5$

$[\ell^5] R_0 \xrightarrow{\phi_4} [\ell^4] R_0 \xrightarrow{\phi_1} [\ell^3] R_0 \xrightarrow{\phi_0} R_0$

$[\ell^4] R_0 \xrightarrow{\phi_2} [\ell^3] R_0 \xrightarrow{\phi_1} R_0$

$[\ell^3] R_0 \xrightarrow{\phi_2} R_0$

$[\ell^2] R_0 \xrightarrow{\phi_2} R_0$

$[\ell] R_0 \xrightarrow{\phi_2} R_0$

$[\ell] R_0 \xrightarrow{\phi_0} R_0$

$R_0 \xrightarrow{\phi_0} E_0$

$R_1 \xrightarrow{\phi_0} E_1$

$R_2 \xrightarrow{\phi_0} E_2$

$R_3 \xrightarrow{\phi_0} E_3$

$R_4 \xrightarrow{\phi_0} E_4$

$R_5 \xrightarrow{\phi_0} E_5$

$R_6 \xrightarrow{\phi_0} E_6$
Large Degree Isogeny Computations

Point in queue

Base Curve

Point mult by \( \ell \)

Apply \( \ell \)-isogeny

Get \( \ell \)-isogeny

\[ \phi_0 \]

\[ \ell \]

\[ \ell^2 \]

\[ \ell^3 \]

\[ \ell^4 \]

\[ \ell^5 \]

\[ \ell^6 \]

\[ \phi_0 \]

\[ \phi_1 \]

\[ \phi_2 \]

\[ \phi_3 \]

\[ \phi_4 \]

\[ \phi_5 \]

\[ \phi_6 \]

\[ E_0 \]

\[ E_1 \]

\[ E_2 \]

\[ E_3 \]

\[ E_4 \]

\[ E_5 \]

\[ R_0 \]

\[ R_1 \]

\[ R_2 \]

\[ R_3 \]

\[ R_4 \]

\[ R_5 \]

\[ R_6 \]

\[ [\ell]R_0 \]

\[ [\ell^2]R_0 \]

\[ [\ell^3]R_0 \]

\[ [\ell^4]R_0 \]

\[ [\ell^5]R_0 \]

\[ [\ell^6]R_0 \]

\[ [\ell^3]R_1 \]

\[ [\ell^4]R_1 \]

\[ [\ell^5]R_1 \]

\[ [\ell^6]R_1 \]

\[ [\ell^3]R_2 \]

\[ [\ell^4]R_2 \]

\[ [\ell^5]R_2 \]

\[ [\ell^6]R_2 \]

\[ [\ell^3]R_3 \]

\[ [\ell^4]R_3 \]

\[ [\ell^5]R_3 \]

\[ [\ell^6]R_3 \]

\[ [\ell^3]R_4 \]

\[ [\ell^4]R_4 \]

\[ [\ell^5]R_4 \]

\[ [\ell^6]R_4 \]

\[ [\ell^3]R_5 \]

\[ [\ell^4]R_5 \]

\[ [\ell^5]R_5 \]

\[ [\ell^6]R_5 \]

\[ E_6 \]

\( e = 7 \)

\( R_5 = \phi_4(R_4) \)

Order of \([\ell]R_5\) is \( \ell \)
Large Degree Isogeny Computations

$e = 7$

$\phi_5 := E_5 / \langle [\ell] R_5 \rangle$

$E_6 = \phi_5(E_5)$
Large Degree Isogeny Computations

$e = 7$

$R_6 = \phi_5(R_5)$

Order of $R_6$ is $\ell$
Large Degree Isogeny Computations

\[ e = 7 \]

\[ \phi_6 := E_6 / \langle [\ell] R_6 \rangle \]

\[ E_7 = \phi_6(E_6) \]
High-level Hardware Architecture for SIDH

Public SIDH Parameters

Round 1

\[ E_0, \quad P_A, \quad Q_A, \quad P_B, \quad Q_B \]

Round 2

\[ E_B, \quad \phi_B(P_A), \quad \phi_B(Q_A) \]
Fast Kernel Computations

\[ R = \ker(\phi) = \langle P + [s]Q \rangle \]
Field Multiplication

- Field multiplication performs $C = A \times B \mod p$
- Choice of multar multiplier is crucial: Montgomery multiplication
- Systloic Montgomery multiplier
  - PEs process various chunks of the results in parallel
  - For SIKE primes $(2^{e_A} \cdot 3^{e_B} - 1)$, $p = 1 \ldots \overbrace{111 \ldots 111}^{e_A}$ and $p' = -p^{-1} = 1( \mod 2^w)$ where $w \leq e_A$

Coarsely Integrated Operand Scanning (CIOS):
- Alternate between multiplication and reduction
- Longer Critical Path: 1 Mult + 1 Additions
- More clock cycles ($4 \times$ Number of words)

Finely Integrated Operand Scanning (FIOS):
- Parallelize Multiplication and reduction
- Longer Critical Path: 1 Mult + 2 Additions
- Less clock cycles ($3 \times$ Number of words)
FIOS Design (Number of words = 4)
Arithmetic over $\mathbb{F}_{p^2}$

Each of the $\mathbb{F}_{p^2}$ arithmetic are built upon a series of $\mathbb{F}_p$ arithmetic

<table>
<thead>
<tr>
<th>$\mathbb{F}_{p^2}$</th>
<th>$\mathbb{F}_p$</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a + b$</td>
<td>$(a_0 + b_0, a_1 + b_1)$</td>
<td>$2A$</td>
</tr>
<tr>
<td>$a - b$</td>
<td>$(a_0 - b_0, a_1 - b_1)$</td>
<td>$2A$</td>
</tr>
<tr>
<td>$a \times b$</td>
<td>$(a_0 \cdot b_0 - a_1 \cdot b_1, (a_0 + a_1) \cdot (b_0 + b_1) - a_0 \cdot b_0 - a_1 \cdot b_1)$</td>
<td>$3M + 5A$</td>
</tr>
<tr>
<td>$a^2$</td>
<td>$((a_0 + a_1) \cdot (a_0 - a_1), 2a_0 \cdot a_1)$</td>
<td>$2M + 3A$</td>
</tr>
<tr>
<td>$a^{-1}$</td>
<td>$(a_0 \cdot (a_0^2 + a_1^2)^{-1}, -a_1 \cdot (a_0^2 + a_1^2)^{-1}$</td>
<td>$4M + 2A + 1I$</td>
</tr>
</tbody>
</table>
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

$E_B = E_0 / \langle P_B + [s_B] Q_B \rangle$

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

---

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

Bob’s public key $pk_B$

$r = \text{Keccak}(m, pk_B)$

$E_{AB} = E_B / \langle \phi_B(P_B) + [r] \phi_B(Q_B) \rangle$

$E_A = E_0 / \langle P_A + [r] Q_A \rangle$

$c = \text{Keccak}(j(E_{AB})) \oplus m$

Alice’s public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$

$ct = \{ pk_A, c \}$

Shared Secret $ss_A = \text{Keccak}(m, pk_A, c)$

---

**KEY DECAPSULATION (Bob)**

$ciphertext = \{ pk_A, c \}$

$E_{BA} = E_A / \langle \phi_A(P_B) + [r'] \phi_A(Q_B) \rangle$

$m' = \text{Keccak}(j(E_{BA})) \oplus c$

$r' = \text{Keccak}(m', pk_B)$

$E_A' = E_0 / \langle P_A + [r'] Q_A \rangle$

Alice’s public key $pk_A' = \{ E_A', \phi_A'(P_B), \phi_A'(Q_B) \}$

---

**Check**

Public Parameters
- Alice’s values
- Bob’s values
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$ →

Isogeny

\[
E_B = E_0 / \langle P_B + [s_B]Q_B \rangle
\]

Bob’s public key $pk_B = \{ E_B, \phi(B)(P_A), \phi(B)(Q_A) \}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

Bob’s public key $pk_B$

\[
r = Keccak(m, pk_B)
\]

\[
E_{AB} = E_B / \langle \phi(B)(P_B) + [r]_B \phi(B)(Q_B) \rangle
\]

\[
E_A = E_0 / \langle P_A + [r]_A Q_A \rangle
\]

\[
c = Keccak(j(E_{AB})) \oplus m
\]

Alice’s public key $pk_A = \{ E_A, \phi(A)(P_B), \phi(A)(Q_B) \}$

**KEY DECAPSULATION (Bob)**

ciphertext $ct$

\[
E_{BA} = E_A / \langle \phi(A)(P_B) + [s_B]A \phi(A)(Q_B) \rangle
\]

\[
m' = Keccak(j(E_{BA})) \oplus c
\]

\[
r' = Keccak(m', pk_B)
\]

\[
E'_A = E_0 / \langle P_A + [r']_A Q_A \rangle
\]

Alice’s public key $pk'_A = \{ E'_A, \phi'(A)(P_B), \phi'(A)(Q_B) \}$

Check $pk'_A = pk_A$

**Shared Secret**

\[
ss_A = Keccak(m, pk_A, c)
\]

**Isogeny Hash**

Public Parameters

Alice’s values
Bob’s values
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

$E_B = E_0 / \langle P_B + [s_B] Q_B \rangle$

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**Isogeny**
SIKE Control Flow

KEY GENERATION (BOB)

Bob’s secret key $s_B$  

Bob’s public key $pk_B$ = $\{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

KEY ENCAPSULATION (Alice)

Alice’s secret message $m$

Bob’s public key $pk_B$
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob's secret key $s_B$ is converted into Bob's public key $pk_B$.

- $E_B = E_0/\langle P_B + [s_B]Q_B \rangle$
- $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

**KEY ENCAPSULATION (Alice)**

Alice's secret message $m$ is encapsulated with Bob's public key.

- $r = \text{Keccak}(m, pk_B)$
- $c = \text{Keccak}(j(E_{AB})) \oplus m$

**Shared Secret (Bob)**

Bob's public key $pk_A$ is used to verify the shared secret.

- $\text{Shared Secret} = \text{Keccak}(m, pk_A, c)$
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$ → $E_B = E_0/\langle P_B + [s_B]Q_B \rangle$

Bob’s public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$ → $r = \text{Keccak}(m, pk_B)$

Bob’s public key $pk_B$ → $E_{AB} = E_B/\langle \phi_B(P_B) + [r]\phi_B(Q_B) \rangle$

Isogeny

Bob’s public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

Isogeny

**Public Parameters**

Alice’s values

Bob’s values

R. Azarderakhsh (Florida Atlantic University)

SIKE in Hardware
SIKE Control Flow

KEY GENERATION (BOB)

Bob’s secret key $s_B$ → $E_B = E_0/\langle P_B + [s_B]Q_B \rangle$ → Bob’s public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

KEY ENCAPSULATION (Alice)

Alice’s secret message $m$ → $r = \text{Keccak}(m, pk_B)$ → $E_{AB} = E_B/\langle \phi_B(P_B) + [r]\phi_B(Q_B) \rangle$ → $c = \text{Keccak}(j(E_{AB})) \oplus m$

Alice’s public key $pk_A = \{E_A, \phi_A(P_B), \phi_A(Q_B)\}$

Isogeny

Hash

Public Parameters

Alice’s values

Bob’s values
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

$E_B = E_0 \langle P_B + [s_B]Q_B \rangle$

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

$r = \text{Keccak}(m, pk_B)$

$E_{AB} = E_B \langle \phi_B(P_B) + [r]Q_B \rangle$

$c = \text{Keccak}(j(E_{AB})) \oplus m$

$ciphertext( ct ) \{ pk_A, c \}$

Alice’s public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$

$ciphertext( ct ) \{ pk_A, c \}$

**Public Parameters**

Alice’s values
Bob’s values
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$ → $E_B = E_0/\langle P_B + [s_B]Q_B \rangle$ → Bob’s public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$ → $r = \text{Keccak}(m, pk_B)$ → $E_{AB} = E_B/\langle \phi_B(P_B) + [r] \phi_B(Q_B) \rangle$

Bob’s public key $pk_B$ → $c = \text{Keccak}(j(E_{AB})) \oplus m$ → ciphertext($ct$) = $\{pk_A, c\}$

Alice’s public key $pk_A$ = $\{E_A, \phi_A(P_B), \phi_A(Q_B)\}$ → Shared Secret($ss_A$) = $\text{Keccak}(m, pk_A, c)$
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

$E_B = E_0 / \langle P_B + [s_B]Q_B \rangle$

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

$r = \text{Keccak}(m, pk_B)$

$c = \text{Keccak}(j(E_{AB})) \oplus m$

$ciphertext(ct) = \{ pk_A, c \}$

**KEY DECAPSULATION (Bob)**

ciphertext($ct$)

$E_{AB} = E_B / \langle \phi_B(P_B) + [r]\phi_B(Q_B) \rangle$

$E_A = E_0 / \langle P_A + [r]Q_A \rangle$

Alice’s public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$

Shared Secret($ss_B$) = $\text{Keccak}(m, pk_A, c)$

Public Parameters

Alice’s values

Bob’s values
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$:

$E_B = E_0 / \langle P_B + [s_B]Q_B \rangle$

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$:

$E_A = E_0 / \langle P_A + [r]Q_A \rangle$

Alice’s public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$

Bob’s public key $pk_B$:

$c = \text{Keccak}(j(E_{AB})) \oplus m$

$ciphertext(ct) = \{ pk_A, c \}$

Shared Secret $ss_A = \text{Keccak}(m, pk_A, c)$

**KEY DECAPSULATION (Bob)**

$ciphertext(ct)$:

$E_{BA} = E_A / \langle \phi_A(P_B) + [s_B]\phi_A(Q_B) \rangle$
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

\[ E_B = E_0 / \langle P_B + [s_B]Q_B \rangle \]

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

\[ r = \text{Keccak}(m, pk_B) \]

Bob’s public key $pk_B$

\[ E_{AB} = E_B / \langle \phi_B(P_B) + [r] \phi_B(Q_B) \rangle \]

Alice’s public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$

\[ c = \text{Keccak}(j(E_{AB})) \oplus m \]

\[ \text{ciphertext}(ct) = \{ pk_A, c \} \]

**KEY DECAPSULATION (Bob)**

ciphertext($ct$)

\[ E_{BA} = E_A / \langle \phi_A(P_B) + [s_B] \phi_A(Q_B) \rangle \]

\[ m' = \text{Keccak}(j(E_{BA})) \oplus c \]

Shared Secret($ss_A$) = Keccak($m, pk_A, c$)

Public Parameters

- Alice’s values
- Bob’s values

---

R. Azarderakhsh (Florida Atlantic University)
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

Bob’s public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

Alice’s public key $pk_A = \{E_A, \phi_A(P_B), \phi_A(Q_B)\}$

**KEY DECAPSULATION (Bob)**

Check $pk_A' == pk_A$

Shared Secret $ss_A = \text{Keccak}(m, pk_A, c)$
### KEY GENERATION (BOB)

Bob’s secret key $s_B$

$E_B = E_0 / (P_B + [s_B]Q_B)$

Bob’s public key $pk_B = \{E_B, \phi_B(P_A), \phi_B(Q_A)\}$

### KEY ENCAPSULATION (Alice)

Alice’s secret message $m$

$r = \text{Keccak}(m, pk_B)$

$c = \text{Keccak}(j(E_{AB})) \oplus m$

Alice’s public key $pk_A = \{E_A, \phi_A(P_B), \phi_A(Q_B)\}$

### KEY DECAPSULATION (Bob)

ciphertext $ct$

$m' = \text{Keccak}(j(E_{BA})) \oplus c$

$s_{ss_A} = \text{Keccak}(m, pk_A, c)$
SIKE Control Flow

**KEY GENERATION (BOB)**
- Bob’s secret key $s_B$
  
  
  $E_B = E_0 / \langle P_B + [s_B]Q_B \rangle$

  Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**
- Alice’s secret message $m$
  
  
  $r = \text{Keccak}(m, pk_B)$

  $E_{AB} = E_B / \langle \phi_B(P_B) + [r] \phi_B(Q_B) \rangle$

  $c = \text{Keccak}(j(E_{AB})) \oplus m$

  ciphertext($ct$) = $\{ pk_A, c \}$

  Shared Secret($ss_A$) = $\text{Keccak}(m, pk_A, c)$

**KEY DECAPSULATION (Bob)**
- ciphertext($ct$)
  
  
  $E_{BA} = E_A / \langle \phi_A(P_B) + [s_B] \phi_A(Q_B) \rangle$

  $m' = \text{Keccak}(j(E_{BA})) \oplus c$

  $E' = E_0 / \langle P_A + [r'] Q_A \rangle$

  Alice’s public key $pk'_A = \{ E'_A, \phi'_A(P_B), \phi'_A(Q_B) \}$

  $r' = \text{Keccak}(m', pk_B)$

  $m' = \text{Keccak}(m', pk_B)$
SIKE Control Flow

**KEY GENERATION (BOB)**

Bob’s secret key $s_B$

$$E_B = E_0 / \langle P_B + [s_B] Q_B \rangle$$

Bob’s public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**

Alice’s secret message $m$

$$r = \text{Keccak}(m, pk_B)$$

Bob’s public key $pk_B$

$$E_{AB} = E_B / \langle \phi_B(P_B) + [r] \phi_B(Q_B) \rangle$$

Alice’s public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$

$$E_A = E_0 / \langle P_A + [r] Q_A \rangle$$

$$c = \text{Keccak}(j(E_{AB})) \oplus m$$

$ciphertext(\text{ct}) = \{ pk_A, c \}$

Shared Secret $ss_A = \text{Keccak}(m, pk_A, c)$

**KEY DECAPSULATION (Bob)**

ciphertext(\text{ct})

$$E_{BA} = E_A / \langle \phi_A(P_B) + [s_B] \phi_A(Q_B) \rangle$$

$$m' = \text{Keccak}(j(E_{BA})) \oplus c$$

Alice’s public key $pk_A' = \{ E_A', \phi_A'(P_B), \phi_A'(Q_B) \}$

$$E_A' = E_0 / \langle P_A + [r'] Q_A \rangle$$

$$r' = \text{Keccak}(m', pk_B)$$

$$ciphertext(\text{ct}) = \{ pk_A', c \}$$

Check $pk_A' == pk_A$
SIKE Control Flow

**KEY GENERATION (BOB)**
- Bob's secret key $s_B$
- $E_B = E_0 / \langle P_B + [s_B]Q_B \rangle$
- Bob's public key $pk_B = \{ E_B, \phi_B(P_A), \phi_B(Q_A) \}$

**KEY ENCAPSULATION (Alice)**
- Alice's secret message $m$
- $r = \text{Keccak}(m, pk_B)$
- $E_{AB} = E_B / \langle \phi_B(P_B) + [r]\phi_B(Q_B) \rangle$
- $c = \text{Keccak}(j(E_{AB})) \oplus m$
- $ciphertext(ct) = \{ pk_A, c \}$
- Alice's public key $pk_A = \{ E_A, \phi_A(P_B), \phi_A(Q_B) \}$
- Shared Secret $(ss_A) = \text{Keccak}(m, pk_A, c)$

**KEY DECAPSULATION (Bob)**
- $r' = \text{Keccak}(m', pk_B)$
- $m' = \text{Keccak}(j(E_{BA})) \oplus c$
- $E_{BA} = E_A / \langle \phi_A(P_B) + [s_B]\phi_A(Q_B) \rangle$
- $E'_A = E_0 / \langle P_A + [r']Q_A \rangle$
- Alice's public key $pk'_A = \{ E_A, \phi'_A(P_B), \phi'_A(Q_B) \}$
- Shared Secret $(ss_B) = \text{Keccak}(m, pk_A, c)$

**Public Parameters**
- Alice's values
- Bob's values
The host initializes any isogeny inputs $x(P), x(Q), x(Q - P)$ and key $k$. 

The diagram illustrates the interaction between the Host CPU and the SIKE Accelerator. The Host CPU provides the isogeny inputs and key $k$ to the SIKE Accelerator through the Program ROM and TRNG modules. The Accelerator processes these inputs and outputs data through the SIKE mux selects, Keccak-1088, and Isogeny Controller modules.
Isogeny Operations

Total number of $\mathbb{F}_{p^2}$ arithmetic operations in fastest known isogeny formulas (from SIKE submission)

<table>
<thead>
<tr>
<th>Isogeny Operation</th>
<th>$\mathbb{F}_{p^2}$ Mult.</th>
<th>$\mathbb{F}_{p^2}$ Squaring</th>
<th>$\mathbb{F}_{p^2}$ Addition</th>
</tr>
</thead>
<tbody>
<tr>
<td>xDBL</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>get_2_isog</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>eval_2_isog</td>
<td>4</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>xTPL</td>
<td>7</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>get_3_isog</td>
<td>2</td>
<td>3</td>
<td>12</td>
</tr>
<tr>
<td>eval_3_isog</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
</tbody>
</table>
Total number of $\mathbb{F}_p$ arithmetic operations in SIKEp503

<table>
<thead>
<tr>
<th>$\mathbb{F}_p$</th>
<th>Keygen</th>
<th>Encapsulation</th>
<th>Decapsulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>31,882</td>
<td>43,127</td>
<td>51,620</td>
</tr>
<tr>
<td>Multiplication</td>
<td>40,107</td>
<td>64,372</td>
<td>69,550</td>
</tr>
<tr>
<td>Inversion</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
## SIDH and SIKE in PC

<table>
<thead>
<tr>
<th></th>
<th>SIDH</th>
<th>SIKE</th>
<th>CSIDH</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key size (quantum 128-bit)</td>
<td>330 bytes</td>
<td>330 bytes</td>
<td>64 bytes</td>
</tr>
<tr>
<td>Running time (x86-64)</td>
<td>2.5 ms</td>
<td>4.5 ms</td>
<td>50 ms</td>
</tr>
<tr>
<td>Compressed SIDH/SIKE</td>
<td>196 bytes</td>
<td>195 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6.5 ms</td>
<td>8.6 ms</td>
<td></td>
</tr>
<tr>
<td>Exponential quantum security</td>
<td>✓</td>
<td>✓</td>
<td>X</td>
</tr>
<tr>
<td>Active attack security</td>
<td>X</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Direct key validation</td>
<td>X</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Digital signatures</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>NIST candidate</td>
<td>X</td>
<td>✓</td>
<td>X</td>
</tr>
</tbody>
</table>
SIKE Performance in FPGA

NIST-Round 1 Submission and updated in Round 2.

<table>
<thead>
<tr>
<th>NIST</th>
<th>SIKE</th>
<th>Area</th>
<th>Freq</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level</td>
<td>Prime</td>
<td>#FFs</td>
<td>LUTs</td>
<td>#Slices</td>
</tr>
<tr>
<td>2</td>
<td>SIKEp503</td>
<td>26,971</td>
<td>25,094</td>
<td>9,514</td>
</tr>
<tr>
<td>5</td>
<td>SIKEp751</td>
<td>50,390</td>
<td>45,893</td>
<td>17,530</td>
</tr>
</tbody>
</table>
SIKE in FPGA Area Results

Area distribution of NIST level 5 SIKEp751 on Virtex-7 FPGA xc7vx690tfg1157-3

- FFs: 5.82%
- LUTs: 10.59%
- Slices: 16.19%
- DSPs: 14.22%
- BRAMs: 2.96%
SIKE: Results for NIST level 1

**Target: High Performance Edge**

<table>
<thead>
<tr>
<th></th>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM Cortex-A53</td>
<td></td>
<td>~10K Slices</td>
</tr>
<tr>
<td>Artix-7 FPGA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Target: Resource-constrained IoT**

<table>
<thead>
<tr>
<th></th>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM Cortex-M4</td>
<td></td>
<td>~3K Slices only</td>
</tr>
<tr>
<td>Artix-7 FPGA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

R. Azarderakhsh (Florida Atlantic University)
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$
Side Channel Attacks on SIDH/SIKE

- Attacking isogeny computations: recover steps $\phi_i$ in secret walk $\phi$
- Loop abort attack $\rightarrow$ Fault an implementation to stop isogeny operation early and give $E_i$
- Refined Power Analysis $\rightarrow$ Force zero-values to divulge $\phi_i$

$$\phi = \phi_6 \cdot \phi_5 \cdot \phi_4 \cdot \phi_3 \cdot \phi_2 \cdot \phi_1 \cdot \phi_0$$
The case for SIKE

• The post-quantum landscape is uncharted territory:
  • The smallest scheme is the slowest, and the fastest scheme is the largest.
  • Compare with traditional cryptography, where the fastest scheme (ECC) is also the smallest.

• This situation introduces a new set of tradeoffs.
  • SIKE’s advantages will become more pronounced over time.
  • SIKE’s disadvantages will become less pronounced over time.

• Why not CSIDH?
  • CSIDH has sub-exponential quantum security, compared to SIDH/SIKE which has exponential quantum security.
  • Over time, CSIDH becomes less attractive compared to SIKE.
The future of SIKE: Computational Costs

- Hardware gets faster over time.
- Software also gets faster over time.
- The above happens naturally, without effort or expenditure.
- An across-the-board performance increase reduces the performance penalty of SIKE (in absolute terms).
- We can also spend more money for faster hardware.
- Certain expenditures (e.g. hardware acceleration) provide good value per unit cost.
• As hardware and software gets faster, attacks get faster.

• Faster attacks require larger keys to counteract.

• An across-the-board key size increase enlarges the communication cost benefits of SIKE (in absolute terms).

• Variance in communication channels is much higher than variance in cycle counts. SIKE already wins today on desktop browsers when including variance.
Open Research Directions

• How to make isogenies **FASTER**?
  • Different curves, formulas, algorithms, isogeny base degrees, etc.
• How to design high-performance field arithmetic operators?
  • Addition, multiplication, inversion, etc.
• How can we **attack** and **defend** isogenies?
  • Side-channels, countermeasures, security analysis, etc.
• How to create efficient isogeny cryptosystems?
  • Signatures, hashing, PAKE, etc.
• How can we apply isogenies?
  • Blockchain, OT, broadcast encryption, etc.
Questions?

Thanks for your attention!