Recent Progress in Hardware Implementations of Post-Quantum Isogeny-Based Cryptography

Reza Azarderakhsh

Department of Computer and Electrical Engineering and Computer Science
Florida Atlantic University

ICMC 2018
Why Quantum Computing? Why now?

The history of Integrated Circuits (IC)
- 1958: First integrated circuit (1 cm², 2 transistors)
- 1971: Moore’s Law is born (2,300 transistors)
- 2014: IBM P8 Processor, 16 cores (650 mm², > 4.2 billion transistors)

Quantum Computers¹

2015: 4-Qbit
2016: 8-Qbit
2017: 16-Qbit
2018: 72-Qbit

¹Pictures are taken from IBM Q Project
Primary PQC Candidates

- Code-Based: McEliece
- Hash-Based: Lamport - Merkle Signatures
- Lattice-Based: NTRU - LWE
- Multivariate: Rainbow Signatures
- Isogeny-Based: SIDH - SIKE

Different degree isogeny maps between elliptic curves
Primary PQC Candidates

- Code-Based: McEliece
- Hash-Based: Lamport - Merkle Signatures
- Lattice-Based: NTRU - LWE
- Multivariate: Rainbow Signatures
- Isogeny-Based: SIDH - SIKE

Different degree isogeny maps between elliptic curves
Supersingular Isogeny-Based Cryptography History

- The first suggestions to use isogenies in crypto by Couveignes in 1997 (CRYPTO 2006)

- Supersingular isogeny hash function by Charles, Lauter and Goren in 2005 (Journal of Cryptology 2009)

- Isogeny-based public-key cryptosystems by Rostovtsev and Stolbunov in 2006

- The biggest impetus by Jao, De Feo (SIDH) in 2011 (PQCRYPTO).

- Supersingular Isogeny Key Encapsulation (SIKE) by Jao et al. submitted to NIST PQC competition 2017
Consider two supersingular elliptic curves defined over a large prime extension field: 
\( E_1 / \mathbb{F}_{p^2} \) and \( E_2 / \mathbb{F}_{p^2} \), where \( p \) is a large prime.

There exists some isogeny \( \phi : E_1 \rightarrow E_2 \) with a fixed, smooth degree \( \ell \) that is public which maps \( E_1 \) to \( E_2 \).

**Supersingular Isogeny Problem**

Given \( P, Q \in E_1 \) and \( \phi(P), \phi(Q) \in E_2 \), retrieve the secret isogeny map \( \phi \).

- The best known attack is based on **Claw finding algorithm**.
- **Claw finding algorithm complexity for SIKE and SIDH:**
  - \( \mathcal{O}(p^{1/4}) \rightarrow \text{Classical attacks} \)
  - \( \mathcal{O}(p^{1/6}) \rightarrow \text{Quantum attacks} \)
Public Parameters

\[ E_0/F_{p^2} \]
\[ p = ℓe_A^{e_B}e_B - 1 \]
\[ (P_A, Q_A) ∈ E_0[^e_A] \]
\[ (P_B, Q_B) ∈ E_0[^e_B] \]

\[ pk_A = [E_A, φ_A(P_B), φ_A(Q_B)] \]

\[ pk_B = [E_B, φ_B(P_A), φ_B(Q_A)] \]
Supersingular Isogeny-Based Cryptography Pros and Cons

- **Pros**
  - Very small public/private key size
  - Data-structure and implementation similar to ECC
  - Different security assumption compared to other candidates
  - No possibility of decryption error
  - No complicated error distribution, rejection sampling, etc.
  - Conservative security analysis on generic attacks

- **Cons**
  - Youngest PQC candidate
  - Slow
  - Security concerns when reuse keys
  - New schemes based on isogeny-based cryptography needs to be implemented on practical settings
Performance of SIDH on ARM Processors

- Benchmarks on Intel processors are reasonably practical.
- Performance on ARM processors is not efficient and requires more optimization.
- ARM processors are massively popular platform with significant market share among smartphones and IoT devices

ARmv7 Cortex-A15 (Jetson TK1 Board) and ARmv8 Cortex-A57 (Nexus smartphone)
Performance Comparison of SIDH and other PQC candidates on ARMv8

- Performance evaluation of fast PQC candidates compared to SIDH
- Generated by **Open Quantum-Safe (OQS) OpenSSL** library
- Single core ARMv8 Cortex-A57 processor

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Lang.</th>
<th>Alice0 (ms)</th>
<th>Bob (ms)</th>
<th>Alice1 (ms)</th>
<th>Communication (bytes)</th>
<th>PQ Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLWE BCNS</td>
<td>C</td>
<td>2.85</td>
<td>4.65</td>
<td>0.695</td>
<td>4,096</td>
<td>4,224</td>
</tr>
<tr>
<td>RLWE NewHope</td>
<td>C</td>
<td>0.284</td>
<td>0.442</td>
<td>0.106</td>
<td>1,824</td>
<td>2,048</td>
</tr>
<tr>
<td>RLWE MSR</td>
<td>C</td>
<td>0.199</td>
<td>0.361</td>
<td>0.065</td>
<td>1,824</td>
<td>2,048</td>
</tr>
<tr>
<td>LWE Frodo Recomm.</td>
<td>C</td>
<td>59.3</td>
<td>59.9</td>
<td>0.427</td>
<td>11,280</td>
<td>11,288</td>
</tr>
<tr>
<td>SIDH</td>
<td>C</td>
<td>497</td>
<td>1114</td>
<td>468</td>
<td>564</td>
<td>564</td>
</tr>
</tbody>
</table>
Supersingular Isogeny-based Schemes Software Implementations

- **SIDH:**
  - De Feo, Jao, and Plût (*J. of Mathematical Cryptology 2014*): Portable C
  - Azarderakhsh, Fishbein, and Jao (*Tech. report 2014*): Optimized AMD64
  - Costello, Longa, and Naehrig (*CRYPTO 2016*): Portable C, Optimized AMD64
  - Koziel, Jalali, Azarderakhsh, Jao, and Mozaffari-Kermani (*CANS 2016*): Optimized ARMv7
  - Jalali, Azarderakhsh, Mozaffari-Kermani, and Jao (*TDSC 2017*): Optimized ARMv8

- **Supersingular Isogeny Digital Signature:**
  - Yoo, Azarderakhsh, Jalali, Jao, Sooukharev (*FC 2017*): Portable C, Optimized AMD64, Optimized ARMv8

- **Supersingular Isogeny Undeniable Signature:**
  - Jalali, Azarderakhsh, Mozaffari-Kermani (*SAC 2017*): Portable C, Optimized ARMv8

- **SIKE:**
  - Jao, Azarderakhsh, Campagna, Costello, De Feo, Hess, Jalali, Koziel, LaMacchia, Longa, Naehrig, Renes, Soukharev, and Urbanik (*Submission to NIST PQC 2017*): Portable C, Optimized AMD64, ARMv8, and VHDL
- Cryptography protocols deal with **big integers** $\rightarrow$ field arithmetic
- From top to bottom, the number of operations increases
- Optimization on the lowest level operations

![Diagram showing different levels of operations in isogeny-based cryptography](image-url)
Isogeny-based cryptography includes an extensive number of field arithmetic:

Focus on optimized implementation on ARM-powered devices
- ARM hand-craft assembly
  - ARMv7: NEON vetorization
  - ARMv8: A64 ASM
- More optimization on ARMv8? Mixed ASIMD + A64 assembly

SIKE field operation counts over different parameter sets

<table>
<thead>
<tr>
<th>Scheme</th>
<th>mult.</th>
<th>red.</th>
<th>add.</th>
<th>sub.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIKEp503</td>
<td>195,889</td>
<td>149,138</td>
<td>56,978</td>
<td>83,142</td>
</tr>
<tr>
<td>SIKEp751</td>
<td>307,946</td>
<td>234,253</td>
<td>88,764</td>
<td>131,618</td>
</tr>
<tr>
<td>SIKEp964</td>
<td>408,786</td>
<td>310,707</td>
<td>117,666</td>
<td>172,910</td>
</tr>
</tbody>
</table>
Mixed ASIMD and A64 Assembly Multiplier on ARMv8

- Based on Karatsuba-multiplication algorithm

\[ a \times b = (a_h b_h)2^{2m} + (a_l b_h + a_h b_l)2^m + (a_l b_l) \]

- \((a_h b_h)\) and \((a_l b_l)\) are independent:
  - \((a_h b_h) \rightarrow \text{A64 ASM}\)
  - \((a_l b_l) \rightarrow \text{ASIMD ASM}\)
  - Fully-utilized pipeline
  - About 10% further improvement compared to pure A64 ASM
### SIDH performance evaluation on different families of ARM processors

- Different security levels

<table>
<thead>
<tr>
<th>Work</th>
<th>Lang.</th>
<th>Device</th>
<th>Field size</th>
<th>PQ Security</th>
<th>Total Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFJ14</td>
<td>C</td>
<td>Cortex-A15</td>
<td>771</td>
<td>128</td>
<td>1,308</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>1035</td>
<td>170</td>
<td>2,816</td>
</tr>
<tr>
<td>KJAJM16</td>
<td>ASM</td>
<td>Cortex-A15</td>
<td>1008</td>
<td>167</td>
<td>982</td>
</tr>
<tr>
<td>JAMJ17</td>
<td>ASM</td>
<td>Cortex-A57</td>
<td>751</td>
<td>125</td>
<td>331</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td>751</td>
<td>125</td>
<td>1,846</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>964</td>
<td>160</td>
<td>4,212</td>
</tr>
</tbody>
</table>
Hardware Implementation of SIDH

Xilinx Virtex-7 FPGA board

FPGA block diagram
All isogeny computations can be performed with finite-field addition and multiplication.

Addition/subtraction split into multiple 256-bit addition/subtraction units.

Choice of modular multiplier is crucial → we went with systolic high-radix Montgomery multiplier.

- Montgomery multiplier computes 16-bit products simultaneously.
- Each multiplier can support 2 simultaneous multiplications → even-odd multiplier.

Latency requirements evaluation:

<table>
<thead>
<tr>
<th>Prime</th>
<th>Read</th>
<th>Write</th>
<th>Add</th>
<th>Multiplication</th>
<th>Multiplication</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Interleaved</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Add</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_{503}$</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>100</td>
<td>69</td>
</tr>
<tr>
<td>$p_{751}$</td>
<td>3</td>
<td></td>
<td>3</td>
<td>148</td>
<td>101</td>
</tr>
<tr>
<td>$p_{1019}$</td>
<td>4</td>
<td></td>
<td>4</td>
<td>196</td>
<td>133</td>
</tr>
<tr>
<td>$p_{1533}$</td>
<td>6</td>
<td></td>
<td>6</td>
<td>292</td>
<td>197</td>
</tr>
<tr>
<td>$p_m$</td>
<td>$\left\lfloor \frac{m}{256} \right\rfloor$</td>
<td>$\left\lfloor \frac{m+2}{16} \right\rfloor + 4$</td>
<td>$2\left\lfloor \frac{m+2}{16} \right\rfloor + 5$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
High-Level Isogeny Accelerator Core

- Architecture centers on dual-port RAM block for 256 registers
- Fully-pipelined addition/subtraction unit
- Replicated multipliers since multiplication is much longer than addition
We used a greedy scheduling algorithm to generate program ROM. Optimizations based on data and output dependencies as well as available resources. Memory, addition, and multiplication controls were controlled for each cycle to allow for a high degree of parallelization. To satisfy the even-odd multiplier, we rescheduled the order of multiplications to be valid.

Figure: Scheduling an isogeny computation for high-performance parallelization
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(eloge)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(\ell \cdot e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(e \log e)$
Parallelizing Isogeny Evaluations

- Efficient large-degree isogeny computation → Get isogeny kernel, compute isogeny, push all stored points to new curve
- No data dependencies between isogeny evaluations in stored point queue → Do them all in parallel
- Requires many more resources to effectively parallelize
- Complexity of large-degree isogeny approaches $O(e)$ rather than $O(eloge)$
SIDH-FPGA Implementation Improvements

- Koziel, Azarderakhsh, Mozaffari-Kermani, Jao\(^2\) (TCAS published in 2017)
  - First implementation of SIDH in hardware
  - Utilizes Jao-DeFeo-Plût affine isogeny formulas
  - 84-bit quantum security SIDH in \(33.7\) ms (6 mults)

- Koziel, Azarderakhsh, Mozaffari-Kermani (Indocrypt 2016)
  - First constant-time implementation of SIDH in hardware
  - Utilizes Costello-Longa-Naehrig projective isogeny formulas
  - First use of a parallelized isogeny evaluation strategy
  - 83-bit quantum security SIDH in \(20.9\) ms (6 mults)

- Koziel, Azarderakhsh, Mozaffari-Kermani (TC SI PQC to appear in 2018)
  - Faster implementations over 83, 124, 168, and 252-bit quantum security
  - Improved FAU, scheduling, and formula choices
  - 83-bit quantum security SIDH in \(16.5\) ms (6 mults)

\(^2\)Submitted in 2016
SIDH-FPGA Implementation Results

- Timing results scale quadratically with prime size
- Benchmark on a Xilinx Virtex-7 FPGA

<table>
<thead>
<tr>
<th>Prime</th>
<th>Quantum Security (bits)</th>
<th>Area</th>
<th>Timing</th>
<th>SIDH/s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td># Mults.</td>
<td># Slices</td>
<td># DSPs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>7,491</td>
<td>192</td>
</tr>
<tr>
<td>p_{503}</td>
<td>83</td>
<td>12</td>
<td>13,203</td>
<td>384</td>
</tr>
<tr>
<td>p_{751}</td>
<td>124</td>
<td>6</td>
<td>11,277</td>
<td>288</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>19,892</td>
<td>576</td>
</tr>
<tr>
<td>p_{1019}</td>
<td>168</td>
<td>6</td>
<td>13,443</td>
<td>384</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>26,976</td>
<td>768</td>
</tr>
<tr>
<td>p_{1533}</td>
<td>252</td>
<td>6</td>
<td>20,559</td>
<td>576</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
<td>40,279</td>
<td>1152</td>
</tr>
</tbody>
</table>
FPGA Implementations of PQC

- Hardware comparison of PQC candidates and SIDH
- Isogeny-based cryptography features the smallest keys
- Remarks:
  - McBits scheme performs only key generation which has a 29% chance of success
  - In NewHope scheme, the first row is server-side and second row is client-side results

<table>
<thead>
<tr>
<th>Work</th>
<th>Scheme</th>
<th>Platform</th>
<th>Quan. Sec. (bits)</th>
<th>Public Key Size (Bytes)</th>
<th>Area</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>128</td>
<td></td>
<td># FFs</td>
<td># LUTs</td>
</tr>
<tr>
<td>WSN17</td>
<td>McBits</td>
<td>Ultrascale+</td>
<td></td>
<td>1,046,739</td>
<td>-</td>
<td>112,845</td>
</tr>
<tr>
<td>OG17</td>
<td>NewHope-Simple</td>
<td>Artix-7</td>
<td>128</td>
<td>2,176</td>
<td>4,452</td>
<td>5,142</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,635</td>
<td>4,498</td>
</tr>
<tr>
<td>KAM18</td>
<td>SIDH</td>
<td>Virtex-7</td>
<td>83</td>
<td>378</td>
<td>24,908</td>
<td>18,820</td>
</tr>
<tr>
<td>KAM18</td>
<td>SIDH</td>
<td>Virtex-7</td>
<td>124</td>
<td>564</td>
<td>38,489</td>
<td>27,713</td>
</tr>
</tbody>
</table>
Supersingular isogeny-based Cryptosystem is one of the PQC candidates. It has several advantages compared to its counterparts:

- Small key size
- Different security assumption
- No decryption error, rejection sampling, etc.
- Easier transition from ECC to post-quantum cryptography

It is the youngest PQC candidate and its performance and security need to be investigated further:

- Optimization on embedded devices
- Fast and parallel hardware implementation

The enhancements on the performance are promising.

Recent cryptanalysis by ADJ et al. shows the classical security levels can be achieved even with smaller prime bit-length → better performance.

Further optimization are coming! Stay tuned :)
B. Koziel, A. Jalali, R. Azarderakhsh, D. Jao, M. Mozaffari-Kermani
NEON-SIDH: efficient implementation of supersingular isogeny Diffie-Hellman key exchange protocol on ARM

B. Koziel, R. Azarderakhsh, D. Jao and M. Mozaffari Kermani
On Fast Calculation of Addition Chains for Isogeny-Based Cryptography
*in Proc. Inscrypt 2016*

B. Koziel, R. Azarderakhsh, and M. Mozaffari Kermani
Fast Hardware Architectures for Supersingular Isogeny Diffie-Hellman Key Exchange on FPGA
*in Proc. Inscrypt 2016*

R. Azarderakhsh, D. Jao, K. Kalach, B. Koziel and Ch. Leonardi
Key compression for isogeny-based cryptosystems
*in Proc. ASIAPKC 2016*

Y. Yoo, R. Azarderakhsh, A. Jalali, D. Jao, V. Soukharev
A post-quantum digital signature scheme based on supersingular isogenies
A. Jalali, R. Azarderakhsh, M. Mozaffari-Kermani, D. Jao
Supersingular isogeny Diffie-Hellman key exchange on 64-bit ARM

A. Jalali, R. Azarderakhsh, M. Mozaffari-Kermani
Efficient Post-Quantum Undeniable Signature on 64-Bit ARM

B. Koziel, R. Azarderakhsh and D. Jao
Side-Channel Attacks on Quantum-Resistant Supersingular Isogeny Diffie-Hellman

D. Jao, R. Azarderakhsh, M. Campagna, C. Costello, A. Jalali, B. Koziel, B. LaMacchia, P. Longa, M. Naehrig, J. Renes, D. Urbanik, V. Soukharev, B. Hess
Supersingular Isogeny Key Encapsulation
*Submission to NIST PQC*, 2017.

B. Koziel, R. Azarderakhsh, and M. Mozaffari Kermani, D. Jao
Post-quantum cryptography on FPGA based on Isogenies on elliptic curves
Bibliography III

B. Koziel, R. Azarderakhsh, M. Mozaffari Kermani
A High-Performance and Scalable Hardware Architecture for Isogeny-Based Cryptography
*IEEE Transactions on Computer (TC) Special Section on Cryptographic Engineering in a Post-Quantum World*, 2018

B. Koziel, R. Azarderakhsh and D. Jao
An Exposure Model for Supersingular Isogeny Diffie-Hellman Key Exchange
Thank you for your attention!
Any Questions?