Open-Source FPGA Implementation of Post-Quantum Cryptographic Hardware Primitives

Rashmi Agrawal, Bu Lake, Alan Ehret, and Michel Kinsky
Adaptive & Secure Computing Systems Lab
Department of Electrical & Computer Engineering
Boston University
Presentation Outline

- Motivation: why quantum-proof?
- NIST: steps towards standardization
- State of the Art: main algorithm
- FPGA-based Implementation: primitives
- Evaluation: cost and performance
- Key Contributions: conclusion
Presentation Outline

- **Motivation:** why quantum-proof?
- **NIST:** steps towards standardization
- **State of the Art:** main algorithm
- **FPGA-based Implementation:** primitives
- **Evaluation:** cost and performance
- **Key Contributions:** conclusion
Ongoing Development

IBM’s Q System 50 Qubits, 20 Qubits

Intel’s Tangle Lake 49 Qubits

Google’s Bristlecone – 72 Qubits

IonQ 160 Qubits
With Quantum Supremacy…

- What is *NOT* considered as post-quantum secure?

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Secure in Post-quantum Era?</th>
</tr>
</thead>
<tbody>
<tr>
<td>RSA-1024, -2048, -4096</td>
<td>No</td>
</tr>
<tr>
<td>Elliptic Curve Crypto (ECC)-256, -521</td>
<td>No</td>
</tr>
<tr>
<td>Diffie-Hellman</td>
<td>No</td>
</tr>
<tr>
<td>ECC Diffie-Hellman</td>
<td>No</td>
</tr>
<tr>
<td>AES-128, -192</td>
<td>No</td>
</tr>
</tbody>
</table>

How does this impacts us?
Can we increase the key size of some popular encryption schemes, so that they can be post-quantum secure?

• Maybe yes, maybe no

<table>
<thead>
<tr>
<th>Attack Platform</th>
<th>Symmetric Encryption</th>
<th>Asymmetric (Public-key) Encryption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Algorithm</td>
<td>Key Size</td>
</tr>
<tr>
<td><strong>Classic Computers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-128</td>
<td>128</td>
<td>128</td>
</tr>
<tr>
<td>AES-256</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td><strong>Quantum Computers</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AES-128</td>
<td>128</td>
<td>64</td>
</tr>
<tr>
<td>AES-256</td>
<td>256</td>
<td>128</td>
</tr>
</tbody>
</table>

* Grover’s algorithm
* Shor’s algorithm

* TechBeacon, Waiting for quantum computing: Why encryption has nothing to worry about, 2018
Quantum Computer-based Cryptography vs General Computer-based Quantum-proof Cryptography

Batman & Ironman Vs Spiderman
Quantum Computer-based Cryptography vs General Computer-based Quantum-proof Cryptography
Presentation Outline

- **Motivation:** why quantum-proof?
- **NIST:** steps towards standardization
- **State of the Art:** main algorithm
- **FPGA-based Implementation:** primitives
- **Evaluation:** cost and performance
- **Key Contributions:** conclusion
Post-Quantum Cryptography (PQC) Standardization (Round -1)

- **NIST**
  - Jan 2017 – Dec 2018
  - Evaluating 69 (5 withdrawn) submissions of PQC, to bring up a standard (just like AES or RSA):
    - 21 lattice-based
    - 18 code-based
    - Some hash-based
    - Some others

Post-Quantum Cryptography (PQC) Standardization (Round -1)

- NIST
  - Jan 2017 – Dec 2018
  - Evaluating 69 (5 withdrawn) submissions of PQC, to bring up a standard (just like AES or RSA):
    - 21 lattice-based
    - 18 code-based
    - Some hash-based
    - Some others

Post-Quantum Cryptography (PQC) Standardization (Round -2)

- **NIST**
  - Jan 30, 2019 published candidates of Round-2:
  - 26 candidates
  - Who survived?
    - 12 lattice-based
    - 8 code-based
    - some multivariate-based and hash based for digital signatures
# Post-Quantum Cryptography (PQC) Standardization (Round -2)

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Public-Key Encryption</th>
<th>Code-based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lattice-based/R-LWE</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NTRU Prime (R-lattice)</td>
<td>Classic McEliece (Binary Goppa)</td>
</tr>
<tr>
<td>2</td>
<td>NTRU (R-lattice)</td>
<td>HQC (BCH &amp; Cyclic)</td>
</tr>
<tr>
<td>3</td>
<td>LAC (R-LWE)</td>
<td>RQC (Cyclic)</td>
</tr>
<tr>
<td>4</td>
<td>SABER (Mod-LWR)</td>
<td>LEDA (LDPC)</td>
</tr>
<tr>
<td>5</td>
<td>Round5 (R-LWR)</td>
<td>ROLLO (LAKE &amp; LOCKER) (LRPC)</td>
</tr>
</tbody>
</table>
Post-Quantum Cryptography (PQC) Standardization (Round -2)

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Key Establishment/Encapsulation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Lattice-based/R-LWE</td>
</tr>
<tr>
<td>1</td>
<td>NewHope (R-LWE)</td>
</tr>
<tr>
<td>2</td>
<td>NTRU (R-lattice)</td>
</tr>
<tr>
<td>3</td>
<td>FrodoKEM (R-LWE)</td>
</tr>
<tr>
<td>4</td>
<td>CRYSRALS (R-LWE)</td>
</tr>
<tr>
<td>5</td>
<td>SABER (Mod-LWR)</td>
</tr>
<tr>
<td>6</td>
<td>Three Bears (Mod-LWR)</td>
</tr>
</tbody>
</table>
# Post-Quantum Cryptography (PQC) Standardization (Round -2)

<table>
<thead>
<tr>
<th>Sr. No.</th>
<th>Digital Signature</th>
<th>Lattice-based/R-LWE</th>
<th>Multivariate-based</th>
<th>Others</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>FALCON (NTRU R-lattice)</td>
<td>GeMSS</td>
<td>Picnic</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>qTESLA (R-LWE)</td>
<td>MQDSS</td>
<td>SPHINCS</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>CRYSTALS (R-LWE)</td>
<td>LUOV</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Rainbow</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why Ring-LWE?

- Advantages
  1) Based on LWE - a branch of lattice-based cryptosystem
Learning with Error (LWE)

- An arbitrary number of equations, each distorted up to $\pm \alpha q$,
- How to find $s$?

\[
\begin{align*}
(2s_1 + 13s_2 + 7s_3 + 3s_4) + e_1 &\approx 13 \pmod{q} \\
(4s_1 + 7s_2 + 9s_3 + 1s_4) + e_2 &\approx 12 \pmod{q} \\
(6s_1 + 14s_2 + 5s_3 + 11s_4) + e_3 &\approx 3 \pmod{q} \\
(5s_1 + 11s_2 + 13s_3 + 2s_4) + e_4 &\approx 9 \pmod{q}
\end{align*}
\]
Why Ring-LWE?

- **Advantages**
  1. Based on LWE - a branch of lattice-based cryptosystem
  2. Can perform
     - Public-key encryption
     - Key-exchange mechanism
     - Digital signature
  3. Can extend to somewhat homomorphic encryption (SHE)
  4. Smaller key size (7k~15k bits vs. 1MB for code-based & 1TB for “post-quantum RSA”)
  5. Simpler computation & circuits
Presentation Outline

- **Motivation:** why quantum-proof?
- **NIST:** steps towards standardization
- **State of the Art:** main algorithm
- **FPGA-based Implementation:** primitives
- **Evaluation:** cost and performance
- **Key Contributions:** conclusion
Ring-Learning with Error (R-LWE)

- Public-Key Cryptosystem

Key Generator Module

- TRNG
- Gaussian Noise Sampler

Encryption Module

- $r_0, r_1, r_2$

Decryption Module

- Gaussian Noise Sampler

Alice

Encryption Module

Bob
Ring-Learning with Error (Ring-LWE)

- Public-key Cryptosystem (PKC)\[1\]
  - Setup (Alice)
    - Let \( q \) be a prime. In a ring \( \mathbb{R}^q \), picks \( a, s, e \), where \( s, e \) are small polynomials
    - s.t. polynomial \( b = a \cdot s + e \) \hspace{1cm} (1)
    - Publishes \( \{a, b\} \) as the public key, as well as \( t = \left\lfloor \frac{q}{2} \right\rfloor \)
    - Keeps \( s \) as the private key

\[1\] Oded Regev. “On lattices, learning with errors, random linear codes, and cryptography”, 2005

Department of Electrical & Computer Engineering
Ring-Learning with Error (Ring-LWE)

- Public-key Cryptosystem (PKC)\[1\]

  - **Setup (Alice)**
    - Publishes \( \{a, b = a \cdot s + e\} \) as the public key, as well as \( t = \left\lfloor \frac{a}{2} \right\rfloor \).
    - Keeps \( s \) as the private key
  
  - **Encryption (Bob to Alice):**
    - Has a plaintext \( m \) (a binary string in \( R_q \))
    - Picks small \( r_0, r_1, r_2 \)
    - Encryption using public key:
      - \( c_0 = b \cdot r_0 + r_2 + t m; \)
      - \( c_1 = a \cdot r_0 + r_1 \)

---

\[1\] Oded Regev, “On lattices, learning with errors, random linear codes, and cryptography”, 2005
Ring-Learning with Error (Ring-LWE)

- Public-key Cryptosystem (PKC)\(^{[1]}\)
  - **Setup (Alice)**
    - Publishes \(\{a, b = a \cdot s + e\}\) as the public key, as well as \(t = \lfloor \frac{q}{2} \rfloor\)
    - Keeps \(s\) as the private key
  - **Encryption (Bob to Alice):**
    - Generates the cipher:
      - \(c_0 = b \cdot r_0 + r_2 + tm;\)
      - \(c_1 = a \cdot r_0 + r_1\)
  - **Decryption (Alice computes):**
    - \(c_0 - s \cdot c_1 = b \cdot r_0 + r_2 + tm - s \cdot a \cdot r_0 - s \cdot r_1\) \hspace{1cm} (2)
      \[= tm + e \cdot r_0 + r_2 - s \cdot r_1 = tm + \text{“small”}\]
    - \(m = \lfloor (c_0 - s \cdot c_1)/t \rfloor\)

---

\[^{[1]}\] Oded Regev, “On lattices, learning with errors, random linear codes, and cryptography”, 2005
R-LWE Public Key Encryption Co-processor

- Public-key Cryptosystem (PKC)
R-LWE Public Key Encryption Co-processor

- **Basic Operations**
  (Every operation is modular)
  - Random Number Generator
  - Gaussian Noise Sampler
  - Polynomial Addition/Subtraction
  - Scalar Multiplication with a Binary Polynomial
  - Scalar Division to the Nearest Binary Integer
  - Polynomial Multiplication

- **Size of the Polynomials/Vectors**
  - Length: 256, 512, or 1024
  - Coefficients: within the prime number 1,049,089
R-LWE Public Key Encryption Co-processor

- **Basic Operations**
  (Every operation is modular)
  - Random Number Generator ✓
  - Gaussian Noise Sampler ✓
  - Polynomial Addition/Subtraction ✓
  - Scalar Multiplication with a Binary Polynomial ✓
  - Scalar Division to the Nearest Binary Integer ✓
    - **Can be done by 2 subtractions**
  - Polynomial Multiplication (hard)

- **Size of the Polynomials/Vectors**
  - Length: 256, 512, or 1024
  - Symbol: within the prime number 1,049,089
Presentation Outline

- **Motivation:** why quantum-proof?
- **NIST:** steps towards standardization
- **State of the Art:** main algorithm
- **FPGA-based Implementation:** primitives
- **Evaluation:** cost and performance
- **Key Contributions:** conclusion
Key Design Features

- **Parameterized**
  - Fully configurable parameters
    - Enable deployment in small devices like IoT as well as large platforms like Homomorphic Encryption

- **Optimized**
  - Fully optimized for reconfigurable hardware implementation

- **Provides building blocks for other schemes**
  - With little modifications to implement R-LWE schemes in NIST standardization process
R-LWE Public Key Encryption Co-processor

- Polynomial Addition
  - If \( a = [a_0, a_1], b = [b_0, b_1], \) then:
    - \( c = a + b = [(a_0+b_0)\%q, (a_1+b_1)\%q] \)
R-LWE Public Key Encryption Co-processor

- Polynomial Subtraction
  - If \(a = [a_0, a_1]\), \(b = [b_0, b_1]\), then:
    - \(c = a - b\)
      - \(c_0 = (a_0 - b_0) \% q\)
      - \(c_0 = (a_0 \geq b_0) \, ? \, (a_0 - b_0) \, : \, (q - (b_0 - a_0))\)
R-LWE Public Key Encryption Co-processor

- Scalar Multiplication
  - $t$ is a constant and pre-computed, and
  - $m$ the plaintext is a binary vector
  - $c_0 = (m[0] \equiv 1) \ ? \ t : 0$
R-LWE Public Key Encryption Co-processor

- Scalar division to the nearest binary integer
  - Denote $u = (c_0 - s \cdot c_1)$
  - Compute $m = \lfloor u/t \rfloor$
R-LWE Public Key Encryption Co-processor

- Modular Polynomial Multiplication
  - Naïve Convolution then Polynomial Reduction
  - By FFT over finite field

```
Algorithm  Polynomial multiplication using FFT
Let \( \omega \) be a primitive \( n \)-th root of unity in \( \mathbb{Z}_p \) and \( \phi^2 \equiv \omega \mod p \). Let \( \mathbf{a} = (a_0, \ldots, a_{n-1}) \), \( \mathbf{b} = (b_0, \ldots, b_{n-1}) \) and \( \mathbf{c} = (c_0, \ldots, c_{n-1}) \) be the coefficient vectors of degree \( n \) polynomials \( a(x) \), \( b(x) \), and \( c(x) \), respectively, where \( a_i, b_i, c_i \in \mathbb{Z}_p \), \( i = 0, 1, \ldots, n-1 \).

Input: \( \mathbf{a}, \mathbf{b}, \omega, \omega^{-1}, \phi, \phi^{-1}, n, n^{-1}, p \).
Output: \( \mathbf{c} \) where \( \mathbf{c}(x) = \mathbf{a}(x) \cdot \mathbf{b}(x) \mod (x^n + 1) \).

1: Precompute: \( \omega^i, \omega^{-i}, \phi^i, \phi^{-i} \) where \( i = 0, 1, \ldots, n-1 \)
2: for \( i = 0 \) to \( n-1 \) do
3: \( \bar{a}_i \leftarrow a_i \phi^i \mod p \)
4: \( \bar{b}_i \leftarrow b_i \phi^i \mod p \)
5: end for
6: \( \bar{A} \leftarrow \text{FFT}_n^p(\bar{a}) \)
7: \( \bar{B} \leftarrow \text{FFT}_n^p(\bar{b}) \)
8: for \( i = 0 \) to \( n-1 \) do
9: \( \bar{C}_i \leftarrow \bar{A}_i \bar{B}_i \mod p \)
10: end for
11: \( \bar{c} \leftarrow \text{IFFT}_n^p(\bar{C}) \)
12: for \( i = 0 \) to \( n-1 \) do
13: \( c_i \leftarrow \bar{c}_i \phi^{-i} \mod p \)
14: end for
15: return \( \mathbf{c} \)
```
R-LWE Public Key Encryption Co-processor

- Modular Polynomial Multiplication
R-LWE Public Key Encryption Co-processor

- Modular Polynomial Multiplication
  - NTT Module
R-LWE Public Key Encryption Co-processor

- Public-key Cryptosystem (PKC)
Presentation Outline

- **Motivation:** why quantum-proof?
- **NIST:** steps towards standardization
- **State of the Art:** main algorithm
- **FPGA-based Implementation:** primitives
- **Evaluation:** cost and performance
- **Key Contributions:** conclusion
Performance Evaluation

- **Target Platform**
  - Xilinx Zynq-7000 FPGA

- **Hardware Description Language**
  - Verilog 2001

- **Design Tool**
  - Xilinx Vivado 2018.2 design suite
## Correlation Between \( \{q, n\} \) and \{Latency, Area\} 

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>KeyGen</td>
<td>(3n + \frac{3n}{2} \log_2 n)</td>
</tr>
<tr>
<td>Enc</td>
<td>(7n + 2n \log_2 n)</td>
</tr>
<tr>
<td>Dec</td>
<td>(4n + n \log_2 n)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resource</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>LUTs</td>
<td>(O(n \log_2 n \log_2 q))</td>
</tr>
<tr>
<td>Registers</td>
<td>(O(n \log_2 n \log_2 q))</td>
</tr>
</tbody>
</table>
Hardware Cost for PKC with $q = 12,289$

- **LUTs Only Implementation**

<table>
<thead>
<tr>
<th>Length (n)</th>
<th>LUTs</th>
<th>Registers</th>
<th>DSP</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>66251</td>
<td>16805</td>
<td>26</td>
</tr>
<tr>
<td>256</td>
<td>114900</td>
<td>33138</td>
<td>26</td>
</tr>
<tr>
<td>512</td>
<td>227458</td>
<td>65643</td>
<td>26</td>
</tr>
<tr>
<td>1024</td>
<td>426402</td>
<td>130540</td>
<td>26</td>
</tr>
</tbody>
</table>

- **BRAM Implementation**

<table>
<thead>
<tr>
<th>Length (n)</th>
<th>LUTs</th>
<th>Registers</th>
<th>DSP</th>
<th>BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>7376</td>
<td>221</td>
<td>26</td>
<td>3.5</td>
</tr>
<tr>
<td>256</td>
<td>9152</td>
<td>396</td>
<td>26</td>
<td>3.5</td>
</tr>
<tr>
<td>512</td>
<td>11504</td>
<td>674</td>
<td>26</td>
<td>3.5</td>
</tr>
<tr>
<td>1024</td>
<td>15717</td>
<td>1255</td>
<td>26</td>
<td>3.5</td>
</tr>
</tbody>
</table>
Hardware Cost: Varying q and n values

Different q values
- n = 64
- n = 128
- n = 256
- n = 512
- n = 1024
PKC System Total Latency

![Graph showing total latency vs. polynomial length](image)

- y-axis: Latency (cycles)
- x-axis: Polynomial length (n)
- Graph shows an increasing trend as polynomial length increases.
NTT Multiplier Latency Comparison

![Graph showing latency comparison for different designs.](image-url)
Presentation Outline

- **Motivation:** why quantum-proof?
- **NIST:** steps towards standardization
- **State of the Art:** main algorithm
- **FPGA-based Implementation:** primitives
- **Evaluation:** cost and performance
- **Key Contributions:** conclusion
Conclusion

- Implementation
  - FGPA-tailored implementation of primitives

- Optimization
  - Algorithmic optimizations to reduce hardware cost

- Open Source
  - Release of the synthesizable and fully verifiable Verilog code with following advantages:
    - Parameterization
      - Enable deployment in small devices as well as large platforms
    - Fast Polynomial Multiplier
      - Efficient n-point NTT multiplier
Acknowledgements

- All ASCS lab members
Thank you

- Code available at:
  http://ascslab.org/research/pqcp/index.html