Implementing and Benchmarking Seven Round 2 Lattice-Based Key Encapsulation Mechanisms Using a Software/Hardware Codesign Approach

Farnoud Farahmand, Viet B. Dang, Michał Andrzejczak, Kris Gaj

George Mason University
Co-Authors

GMU PhD Students

Farnoud Farahmand

Viet Ba Dang

Visiting Scholar

Michał Andrzejczak

Military University of Technology in Warsaw, Poland
Hardware Benchmarking
<table>
<thead>
<tr>
<th>Candidates</th>
<th>#Round 2 candidates</th>
<th>Implemented in hardware</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES</td>
<td>5</td>
<td>5</td>
<td>100%</td>
</tr>
<tr>
<td>SHA-3</td>
<td>14</td>
<td>14</td>
<td>100%</td>
</tr>
<tr>
<td>CAESAR</td>
<td>29</td>
<td>28</td>
<td>97%</td>
</tr>
<tr>
<td>PQC</td>
<td>26</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
Software/Hardware Codesign
Software

Hardware

Most time-critical operation
SW/HW Codesign: Motivational Example 1

Software

- Major: 91%
- Other: 9%

Software/Hardware

- Major: ~1%
- Other: 9%

Time saved: 90%

speed-up ≥ 100

91% major operation(s)  9% other operations  →  ~1% major operation(s) in HW  9% other operations in SW

Total Speed-Up ≥ 10
SW/HW Codesign: Motivational Example 2

Software

- Major: 99%
- Other: 1%

Software/Hardware

- Major: ~1%
- Other: 1%
- Time saved: 98%
- Speed-up ≥ 100

99% major operation(s) → ~1% major operation(s) in HW
1% other operations → 1% other operations in SW

Total Speed-Up ≥ 50
SW/HW Codesign: Advantages

❖ Focus on a few major operations, known to be easily parallelizable
  ▪ much shorter development time (at least by a factor of 10)
  ▪ guaranteed substantial speed-up
  ▪ high-flexibility to changes in other operations (such as candidate tweaks)

❖ Insight regarding performance of future instruction set extensions of modern microprocessors

❖ Possibility of implementing multiple candidates by the same research group, eliminating the influence of different
  ▪ design skills
  ▪ operation subset (e.g., including or excluding key generation)
  ▪ interface & protocol
  ▪ optimization target
  ▪ platform
SW/HW Codesign: Potential Pitfalls

❖ Performance & ranking may strongly depend on

A. features of a particular platform
   - Software/hardware interface
   - Support for cache coherency
   - Differences in max. clock frequency

B. selected hardware/software partitioning

C. optimization of an underlying software implementation

❖ Limited insight on ranking of purely hardware implementations

First step, not the ultimate solution!
Two Major Types of Platforms

FPGA Fabric & Hard-core Processors
- Processor w/ Memory & I/O
- FPGA Fabric

Examples:
- Xilinx Zynq 7000 System on Chip (SoC)
- Xilinx Zynq UltraScale+ MPSoC
- Intel Arria 10 SoC FPGAs
- Intel Stratix 10 SoC FPGAs

FPGA Fabric, including Soft-core Processors
- Soft-core Processor
- FPGA Fabric

Examples:
- Xilinx Virtex UltraScale+ FPGAs
- Intel Stratix 10 FPGAs, including
  - Xilinx MicroBlaze
  - Intel Nios II
  - RISC-V, originally UC Berkeley
# Two Major Types of Platform

<table>
<thead>
<tr>
<th>Feature</th>
<th>FPGA Fabric and Hard-core Processor</th>
<th>FPGA Fabric with Soft-core Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processor</td>
<td>ARM</td>
<td>MicroBlaze, NIOS II, RISC-V, etc.</td>
</tr>
<tr>
<td>Clock frequency</td>
<td>&gt;1 GHz</td>
<td>max. 200-450 MHz</td>
</tr>
<tr>
<td>Portability</td>
<td>similar FPGA SoCs</td>
<td>various FPGAs, FPGA SoCs, and ASICs</td>
</tr>
<tr>
<td>Hardware accelerators</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Instruction set extensions</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Ease of design (methodology, tools, OS support)</td>
<td><strong>Easy</strong></td>
<td>Dependent on a particular soft-core processor and tool chain</td>
</tr>
</tbody>
</table>

**Xilinx Zynq UltraScale+ MPSoC**

1.2 GHz ARM Cortex-A53 + UltraScale+ FPGA logic
Choice of a Platform for Benchmarking

<table>
<thead>
<tr>
<th>Embedded Processor:</th>
<th>FPGA Architecture:</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM Cortex-M4</td>
<td>Artix-7</td>
</tr>
</tbody>
</table>

In NIST presentations to date:

Our recommendation:

<table>
<thead>
<tr>
<th>ARM Cortex-A53</th>
<th>UltraScale+</th>
</tr>
</thead>
</table>

- No FPGA SoC with ARM Cortex-M4 and Artix-7 on a single chip
- Cortex-M4 and Artix-7 more suitable for lightweight designs, Cortex-A53 and UltraScale+ for high performance
- Zynq UltraScale+:
  - capability to compare SW/HW implementations with fully-SW and fully-HW implementations realized using the same chip
  - likely in use in the first years of the new standard deployments
All elements located on a single chip
Code Release

• Full Code & Configuration of the Experimental Setup
• Software/Hardware Codesign of Round 1 NTRUEncrypt to be made available at
  https://cryptography.gmu.edu/athena
  under PQC
  by August 31, 2019
Our Case Study
SW/HW Codesign: Case Study

7 IND-CCA*-secure Lattice-Based Key Encapsulation Mechanisms (KEMs) representing
5 NIST PQC Round 2 Submissions

LWE (Learning with Error)-based:
  - FrodoKEM

RLWR (Ring Learning with Rounding)-based:
  - Round5

Module-LWR-based:
  - Saber

NTRU-based:
  - NTRU
    - NTRU-HPS
    - NTRU-HRSS
  - NTRU Prime
    - Streamlined NTRU Prime
    - NTRU LPRime

* IND-CCA = with Indistinguishability under Chosen Ciphertext Attack
SW/HW Partitioning

Top candidates for offloading to hardware

**From profiling:**
- Large percentage of the execution time
- Small number of function calls

**From manual analysis of the code:**
- Small size of inputs and outputs
- Potential for combining with neighboring functions

**From knowledge of operations and concurrent computing:**
- High potential for parallelization
Operations Offloaded to Hardware

- Major arithmetic operations
  - Polynomial multiplications
  - Matrix-by-vector multiplications
  - Vector-by-vector multiplications
- All hash-based operations
  - (c)SHAKE128, (c)SHAKE256
  - SHA3-256, SHA3-512
Example: LightSaber Decapsulation

- InnerProduct: 43.52%
- MatrixVectorMultiply: 43.44%
- Hash: 3.30%
- GenSecret: 2.30%
- GenMatrix: 5.03%
- Other: 2.40%
LightSaber Decapsulation

Execution time of functions to be moved to hardware 97.60%
Execution time of functions remaining in software 2.40%

Accelerator Speed-Up = 97.60/8.77 = 11.1
Total Speed-Up = 100/11.17 = 9.0
Tentative Results
Software Implementations Used

FrodoKEM, NTRU-HPS, NTRU-HRSS, Saber:

Round 2 submission packages – Optimized_Implementation

Round5:


Streamlined NTRU Prime, NTRU LPRime:

supercop-20190811 : factored

Changes made after the submission of the paper!

Results substantially different!

New version of the paper available on ePrint soon!
Total Execution Time in Software [$\mu$s]

Encapsulation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round5</td>
<td>16,192</td>
<td>34,609</td>
<td>62,076</td>
<td>16,192</td>
<td>34,609</td>
</tr>
<tr>
<td>Saber</td>
<td>1,000</td>
<td>1,500</td>
<td>2,000</td>
<td>1,000</td>
<td>1,500</td>
</tr>
<tr>
<td>Str NTRU Prime</td>
<td>1,500</td>
<td>2,000</td>
<td>2,500</td>
<td>1,500</td>
<td>2,000</td>
</tr>
<tr>
<td>NTRU LPrime</td>
<td>2,000</td>
<td>2,500</td>
<td>3,000</td>
<td>2,000</td>
<td>2,500</td>
</tr>
<tr>
<td>NTRU-HRSS</td>
<td>3,500</td>
<td>4,000</td>
<td>4,500</td>
<td>3,500</td>
<td>4,000</td>
</tr>
<tr>
<td>NTRU-HPS</td>
<td>4,500</td>
<td>5,000</td>
<td>5,500</td>
<td>4,500</td>
<td>5,000</td>
</tr>
<tr>
<td>FrodoKEM</td>
<td>6,000</td>
<td>6,500</td>
<td>7,000</td>
<td>6,000</td>
<td>6,500</td>
</tr>
</tbody>
</table>
### Total Execution Time in Software/Hardware [$\mu$s]

#### Encapsulation

<table>
<thead>
<tr>
<th>Round 5</th>
<th>Saber</th>
<th>NTRU-HRSS</th>
<th>Str NTRU Prime</th>
<th>NTRU-HPS</th>
<th>NTRU LPRime</th>
<th>FrodoKEM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 1</td>
<td>1</td>
<td>2</td>
<td>5⇒3</td>
<td>3⇒4</td>
<td>6⇒5</td>
<td>4⇒6</td>
</tr>
<tr>
<td>Level 2</td>
<td>1,223</td>
<td>1,642</td>
<td>2,186</td>
<td>7</td>
<td>1,223</td>
<td>1,642</td>
</tr>
</tbody>
</table>

- **Level 1**: Light green
- **Level 2**: Orange
- **Level 3**: Gray
- **Level 4**: Yellow
- **Level 5**: Blue
Total Speed-ups: Encapsulation
Accelerator Speed-ups: Encapsulation

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU-HRSS</td>
<td>146.4</td>
<td>140.4</td>
<td>192.2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTRU-HPS</td>
<td>43.5</td>
<td>44.3</td>
<td>46.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FrodoKEM</td>
<td></td>
<td></td>
<td></td>
<td>20.4</td>
<td>27.5</td>
</tr>
<tr>
<td>NTRU LPRime</td>
<td>12.3</td>
<td>15.3</td>
<td>17.7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Str NTRU Prime</td>
<td>8.7</td>
<td>10.6</td>
<td>11.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Round5</td>
<td>8.5</td>
<td>14.5</td>
<td>21.3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saber</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Legend:
- Green: Level 1
- Orange: Level 2
- Gray: Level 3
- Yellow: Level 4
- Blue: Level 5
SW Part Sped up by HW[%]: Encapsulation
Total Execution Time in Software [$\mu$s]
Decapsulation

Order reversed compared to encapsulation

Level 1  Level 2  Level 3  Level 4  Level 5
## Total Execution Time in Software/Hardware [μs]: Decapsulation

<table>
<thead>
<tr>
<th>System</th>
<th>Level 1</th>
<th>Level 2</th>
<th>Level 3</th>
<th>Level 4</th>
<th>Level 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round5</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Saber</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTRU-HPS</td>
<td>5⇒3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Str NTRU</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prime</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NTRU-HRSS</td>
<td></td>
<td></td>
<td>6⇒5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LPRime</td>
<td>3⇒6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FrodoKEM</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
</tr>
</tbody>
</table>

- Total Execution Time:
  - Round5: 1,319 μs
  - Saber: 1,866 μs
  - NTRU-HPS: 3,120 μs
  - Str NTRU: 1,866 μs
  - Prime: 3,120 μs
  - NTRU-HRSS: 3,120 μs
  - LPRime: 3,120 μs
  - FrodoKEM: 3,120 μs
Total Speed-ups: Decapsulation

<table>
<thead>
<tr>
<th>Method</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td>NTRU-HPS</td>
<td>119.3</td>
</tr>
<tr>
<td>NTRU-HRSS</td>
<td>77.4</td>
</tr>
<tr>
<td>Str NTRU Prime</td>
<td>74.1</td>
</tr>
<tr>
<td>FrodoKEM</td>
<td>45.5</td>
</tr>
<tr>
<td>Saber</td>
<td>20.0</td>
</tr>
<tr>
<td>Round5</td>
<td>17.9</td>
</tr>
<tr>
<td>NTRU LPrime</td>
<td>4.5</td>
</tr>
</tbody>
</table>

Level: 1, 2, 3, 4, 5
Conclusions

❖ Total speed-ups
  ▪ for encapsulation from 2.4 (Str NTRU Prime) to 28.4 (FrodoKEM)
  ▪ for decapsulation from 3.9 (NTRU LPRime) to 119.3 (NTRU-HPS)

❖ Total speed-up dependent on the percentage of the software execution time taken by functions offloaded to hardware and the amount of acceleration itself

❖ Hardware accelerators thoroughly optimized using Register-Transfer Level design methodology

❖ Determining optimal software/hardware partitioning requires more work

❖ Ranking of the investigated candidates affected, but not dramatically changed, by hardware acceleration

❖ It is possible to complete similar designs for all Round 2 candidates within the evaluation period (12-18 months)

❖ Additional benefit: Comprehensive library of major operations in hardware
Future Work

**Current work**

<table>
<thead>
<tr>
<th>Breadth</th>
<th>Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 Remaining Lattice-based KEMs</td>
<td>3 Lattice-based Digital Signatures</td>
</tr>
</tbody>
</table>

- More operations moved to hardware / C code optimized for ARM Cortex-A53*
- Algorithmic optimizations of software and hardware*

- Hardware library of basic operations of lattice-based candidates
- Hardware library of basic operations of code-based candidates
- Hardware library for PQC

Full hardware implementations

*collaboration with submission teams and other groups very welcome
Q&A

Thank You!

Questions?  Comments?

Suggestions?

CERG: http://cryptography.gmu.edu
ATHENa: http://cryptography.gmu.edu/athena
## Clock Frequency & Resource Utilization

<table>
<thead>
<tr>
<th>Level: Algorithm</th>
<th>Clock Freq [MHz]</th>
<th>#LUTs</th>
<th>#Slices</th>
<th>#FFs</th>
<th>#36kb BRAMs</th>
<th>#DSPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: FrodoKEM</td>
<td>402</td>
<td>7,213</td>
<td>1,186</td>
<td>6,647</td>
<td>13.5</td>
<td>32</td>
</tr>
<tr>
<td>1: Round5</td>
<td>260</td>
<td>55,442</td>
<td>10,381</td>
<td>82,341</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1: Saber</td>
<td>322</td>
<td>12,343</td>
<td>1,989</td>
<td>11,288</td>
<td>3.5</td>
<td>256</td>
</tr>
<tr>
<td>1: NTRU-HPS</td>
<td>200</td>
<td>24,328</td>
<td>4,972</td>
<td>19,244</td>
<td>2.5</td>
<td>677</td>
</tr>
<tr>
<td>1: NTRU-HRSS</td>
<td>200</td>
<td>27,218</td>
<td>5,770</td>
<td>21,410</td>
<td>2.5</td>
<td>701</td>
</tr>
<tr>
<td>2: Str NTRU Prime</td>
<td>244</td>
<td>55,843</td>
<td>8,134</td>
<td>28,143</td>
<td>3.0</td>
<td>0</td>
</tr>
<tr>
<td>2: NTRU LPrime</td>
<td>244</td>
<td>50,911</td>
<td>7,874</td>
<td>34,050</td>
<td>2.0</td>
<td>0</td>
</tr>
<tr>
<td>Device</td>
<td></td>
<td>274,080</td>
<td>34,260</td>
<td>548,160</td>
<td>912</td>
<td>2,520</td>
</tr>
</tbody>
</table>

Clock Frequency & Resource Utilization

- Clock Frequency & Resource Utilization
- < 21% of total resources of the given device
- < 31%
- < 15%
- < 2%
- < 28%
Minor Modifications to C code

Bare Metal vs. Linux

• No functions of OpenSSL – standalone implementations of:
  
  o **AES:** Optimized ANSI C code for the Rijndael cipher (T-box-based)
    by Vincent Rijmen, Antoon Bosselaers, and Paulo Barreto
    https://fastcrypto.org/front/misc/rijndael-alg-fst.c
  
  o **SHA-3:**
    
    ▪ fips202.c from SUPERCOP
      by Ronny Van Keer, Gilles Van Assche, Daniel J. Bernstein, and Peter Schwabe (for all candidates other than Round5)
    
    ▪ r5_xof_shake.c by Markku-Juhani O. Saarinen and
      keccak1600.c from SUPERCOP, by the same authors as fips202.c
      (for Round5)
  
  o **randombytes():** based on SHAKE rather than AES in NTRU-HPS,
    NTRU-HRSS, and Streamlined NTRU Prime

• No support for SUPERCOP scripts
randombytes()

- Function used for generating pseudorandom byte sequences
- The implementation vary among various benchmarking studies, depending on the mode of operation (Bare Metal vs. Operating System), and availability of libraries, such as OpenSSL
- Used to different extent by implementations of various candidates

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>#Calls</th>
<th>#Bytes (security category 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FrodoKEM</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Round5</td>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>Saber</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>NTRU-HPS</td>
<td>1</td>
<td>3211</td>
</tr>
<tr>
<td>NTRU-HRSS</td>
<td>1</td>
<td>1400</td>
</tr>
<tr>
<td>Str NTRU Prime</td>
<td>1</td>
<td>2612</td>
</tr>
<tr>
<td>NTRU LPRime</td>
<td>1</td>
<td>32</td>
</tr>
</tbody>
</table>

- For 3 algorithms was sped-up over 3 times by using SHAKE128