HQC: Hamming Quasi-Cyclic
An IND-CCA2 Code-based Public Key Encryption Scheme

June the 8th, 2021
NIST 3rd PQC Standardization Conference
https://pqc-hqc.org

C. Aguilar Melchor
ISAE-Supaéro, Univ. Toulouse

N. Aragon
University of Limoges

S. Bettaieb
Worldline

L. Bidoux
Worldline

O. Blazy
University of Limoges

J. Bos
Wordline

J.-C. Deneuville
ENAC, University of Toulouse

A. Dion
ISAE-Supaéro, Univ. Toulouse

P. Gaborit
University of Limoges

J. Lacan
ISAE-Supaéro, Univ. Toulouse

E. Persichetti
Florida Atlantic University

J.-M. Robert
University of Toulon

P. Véron
University of Toulon

G. Zémor
IMB, University of Bordeaux
Outline

1. HQC design rationale and recap
2. Third round tweaks
3. Hardware implementation
HQC Classification / Design Rationale

Important features:
- IND-CPA code-based PKE
- Reduction to a well-known and difficult problem:
  Decoding random quasi-cyclic codes
- No hidden trap in the code
- Efficient decoding
- Precise DFR analysis
HQC Encryption Scheme

Encryption scheme in Hamming metric, using Quasi-Cyclic Codes

- Notation: Secret data - Public data - One-time Randomness
- $G$ is the generator matrix of some public code $C$
- $S_w^n(\mathbb{F}_2) = \{x \in \mathbb{F}_2^n \text{ such that } \omega(x) = w\}$

\[
\begin{align*}
\text{Alice} & : \\
\text{Bob} & : \\
seed_h & \leftarrow \{0, 1\}^\lambda, \quad h \leftarrow F_2^n \\
x, y & \leftarrow S_w^n(\mathbb{F}_2), \quad s \leftarrow x + hy \\
m & \leftarrow C.\text{Decode}(v - uy) \\
\end{align*}
\]
We provided a better decryption failure analysis that allows to decrease the size of our public keys.

We switched from the BCH-repetition decoder to a concatenated Reed-Muller and Reed-Solomon (RMRS) decoder.

The size of the decoded messages are set to the security level (i.e., dimension 128 instead of 256 for level 1), thus improving the decoding capability of the code.
### 3rd round parameters and timings

#### Sizes in bytes

<table>
<thead>
<tr>
<th></th>
<th>pk size</th>
<th>ct size</th>
<th>Improvement wrt. 2nd round</th>
</tr>
</thead>
<tbody>
<tr>
<td>hqc-128</td>
<td>2,249</td>
<td>4,481</td>
<td>28%</td>
</tr>
<tr>
<td>hqc-192</td>
<td>4,522</td>
<td>9,026</td>
<td>23%</td>
</tr>
<tr>
<td>hqc-256</td>
<td>7,245</td>
<td>14,469</td>
<td>18%</td>
</tr>
</tbody>
</table>

#### Timings in kilocycles

<table>
<thead>
<tr>
<th></th>
<th>AVX2 Implementation</th>
<th>Improvement wrt. 2nd round</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Keygen</td>
<td>Encaps</td>
</tr>
<tr>
<td>hqc-128</td>
<td>83</td>
<td>197</td>
</tr>
<tr>
<td>hqc-192</td>
<td>200</td>
<td>456</td>
</tr>
<tr>
<td>hqc-256</td>
<td>400</td>
<td>887</td>
</tr>
</tbody>
</table>
Hardware implementation

- We now only use **KECCAK-based random oracles** in order to limit software footprint.

- **HLS implementation** of HQC: C translated into VHDL by Xilinx tools.
  - Easy to modify, good for quick tests.
  - Compatible with the software KATs.
  - Improvable VHDL by tweaking/replacing modules → there is room for improvement.

### Hardware performances

<table>
<thead>
<tr>
<th>Function</th>
<th>Frequency (MHz)</th>
<th>Slices</th>
<th>LUT</th>
<th>FF</th>
<th>BRAM</th>
<th>Cycles</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keygen</td>
<td>150</td>
<td>3.9k</td>
<td>12k</td>
<td>9k</td>
<td>3</td>
<td>40k</td>
<td>0.27</td>
</tr>
<tr>
<td>Encaps</td>
<td>151</td>
<td>5.5k</td>
<td>16k</td>
<td>13k</td>
<td>5</td>
<td>89k</td>
<td>0.59</td>
</tr>
<tr>
<td>Decaps</td>
<td>152</td>
<td>6.2k</td>
<td>19k</td>
<td>15k</td>
<td>9</td>
<td>190k</td>
<td>1.2</td>
</tr>
</tbody>
</table>

- **Compact version**: 2 times smaller and 10 times slower.
Questions?

HQC official website and updates:
https://pqc-hqc.org/