LEDAkem/LEDApkc

Marco Baldi\textsuperscript{\textcopyright}, Alessandro Barenghi\textsuperscript{■}, Franco Chiaraluce\textsuperscript{\textcopyright}, Gerardo Pelosi\textsuperscript{■}, Paolo Santini\textsuperscript{\textcopyright}

\textsuperscript{\textcopyright}Università Politecnica delle Marche \textsuperscript{■}Politecnico di Milano

April 13, 2018
Two proposals

- LEDA kem (Low-dEnsity parity-check coDe-bAsed key encapsulation mechanism)
  - IND-CPA key encapsulation mechanism, built on Niederreiter cryptosystem
- LEDApkc (Low-dEnsity parity-check coDe-bAsed public-key cryptosystem)
  - IND-CCA2 public-key cryptosystem, built on McEliece + Kobara-Imai Conversion
Underlying hard problems

General binary code decoding problem

- Given a $k \times n$ random binary matrix $G$ and a $n$-bit vector $\tilde{c} = c + e$, $wt(e) < t$, find $c$. Proven to be NP-Complete.

Syndrome decoding problem

- Given an $r \times n$ random binary matrix $H$ and a $r$-bit vector $s$, find the (unique) $n$-bit vector $e$ s.t. $He^T = s$, $wt(e) < t$. Proven to be NP-Complete.
Quasi-Cyclic Low-Density Parity-Check codes (QC-LDPC)

- Proposed in 2008 as a code family to instantiate McEliece/Niederreiter
- Low-Density Parity-Check: Secret code representation is a sparse matrix
  - Small size for private keys
  - Efficient representation/arithmetic during decoding
    - Parameter design must not allow to guess codewords
- Quasi-cyclic: $H$ and $G$ constituent blocks are circulant, hence fully defined by their first row
  - Smaller public keys
  - Reduction in arithmetic complexity in encoding/keygen
**Key Generation**

1. Generate a random $r \times n$ binary block circulant matrix $\mathbf{H} = [\mathbf{H}_0, \ldots, \mathbf{H}_{n_0-1}]$ made of $n_0$ circulant blocks, each with column weight $d_v \ll n$, $n = n_0p$, $p$ prime.

2. Generate a random, non-singular, $n \times n$ binary block circulant matrix $\mathbf{Q}$ made of $n_0 \times n_0$ circulant blocks, with total column weight $m \ll n$.

3. Store private key: $\mathbf{H}, \mathbf{Q}$.

4. Compute $\mathbf{L} = \mathbf{HQ} = [\mathbf{L}_0, \ldots, \mathbf{L}_{n_0-1}]$.

5. Store public key: $\mathbf{M} = (\mathbf{L}_{n_0-1})^{-1}[\mathbf{L}_0, \ldots, \mathbf{L}_{n_0-2}]$. 
**LEDAkem**

**Key Encapsulation**

1. Generate a random $n$-bit error vector $e$ with weight $t$
2. Compute the ciphertext (syndrome) $s = Me^T$
3. Derive the shared secret $x = KDF(e)$

**Key Decapsulation**

1. Obtain $e$ as $Q$-$\text{DECODER}(s, H, Q)$
   - $Q$-$\text{DECODER}$ exploits the fact that the parity matrix is built as $HQ$
2. Derive the shared secret $x = KDF(e)$
LEDApkc

- Built as a McEliece cryptosystem based on QC-LDPC codes
- Employs conversion by Kobara and Imai to achieve IND-CCA2 and allow using a systematic generator matrix $G$
  - Reduces the size of the public key
  - Speeds up the encryption process overall (K-I conversion is less computationally expensive than encoding with a non-systematic $G$)
- Decoding done via efficient syndrome decoding taking into account the matrix $Q$ (reuse decoder from LEDAkem)
  - Saves object code size/silicon area in implementations
Parameter sizing

Parameter design strategy

- Prevent message recovery attacks.
  - Choice of the number of errors $t$, code size $n$ and rate $\frac{k}{n}$ such that ISD of the public code is not feasible.

- Prevent key recovery ("structural") attacks.
  - Density of HQ sufficiently high that retrieving a low-weight codeword of the dual code is not feasible.

- Provide a good DFR (hinder reaction attacks against LEDApkc).
  - $n$ large enough to provide a satisfactory DFR ($\leq 10^{-8}$).

Parameter design was done conservatively, targeting $2^\lambda$, $\lambda \in \{128, 192, 256\}$, taking into account attackers provided with quantum computers.

- Ephemeral keys for LEDAkm, keys reusable up to $10^4 DFR^{-1}$ for LEDApkc.
## Proposed parameters for LEDA\textit{k}em/LEDA\textit{pkc}

<table>
<thead>
<tr>
<th>$\lambda$</th>
<th>$n_0$</th>
<th>$p$</th>
<th>$d_\nu$</th>
<th>$m$</th>
<th>$t$</th>
<th>DFR</th>
<th>Size K\textsubscript{pub} (B)</th>
<th>Size K\textsubscript{pri} (B)</th>
<th>Size K\textsubscript{pri} (at rest) (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>2</td>
<td>27,779</td>
<td>17</td>
<td>7</td>
<td>224</td>
<td>$\approx 8.3 \times 10^{-9}$</td>
<td>3,480</td>
<td>668</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>18,701</td>
<td>19</td>
<td>7</td>
<td>141</td>
<td>$\gtrsim 10^{-9}$</td>
<td>4,688</td>
<td>844</td>
<td>24</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>17,027</td>
<td>21</td>
<td>7</td>
<td>112</td>
<td>$\gtrsim 10^{-9}$</td>
<td>6,408</td>
<td>1,036</td>
<td>24</td>
</tr>
<tr>
<td>192</td>
<td>2</td>
<td>57,557</td>
<td>17</td>
<td>11</td>
<td>349</td>
<td>$\gtrsim 10^{-9}$</td>
<td>7,200</td>
<td>972</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>41,507</td>
<td>19</td>
<td>11</td>
<td>220</td>
<td>$\gtrsim 10^{-9}$</td>
<td>10,384</td>
<td>1,196</td>
<td>32</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>35,027</td>
<td>17</td>
<td>13</td>
<td>175</td>
<td>$\gtrsim 10^{-9}$</td>
<td>13,152</td>
<td>1,364</td>
<td>32</td>
</tr>
<tr>
<td>256</td>
<td>2</td>
<td>99,053</td>
<td>19</td>
<td>13</td>
<td>474</td>
<td>$\gtrsim 5.8 \times 10^{-8}$</td>
<td>12,384</td>
<td>1,244</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>72,019</td>
<td>19</td>
<td>15</td>
<td>301</td>
<td>$\gtrsim 5.8 \times 10^{-8}$</td>
<td>18,016</td>
<td>1,548</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>60,509</td>
<td>23</td>
<td>13</td>
<td>239</td>
<td>$\gtrsim 5.8 \times 10^{-8}$</td>
<td>22,704</td>
<td>1,772</td>
<td>40</td>
</tr>
</tbody>
</table>
Efficient implementation

Circulant matrix representation/arithmetics
- Represent circulant blocks as elements of $\mathbb{F}_2[x]/\langle x^p + 1 \rangle$
  - Reduces both time and space complexity for arithmetics
  - Bit packed representation for dense polynomials, sparse for sparse ones
- High sparsity of $H$ and $Q$ yields small (cache friendly) working set

Removed non-singularity check for $Q$
- $\text{ord}_2(p) = p - 1$, $\text{Perm}(\text{wt}(Q))$ is odd and $< p \Rightarrow Q$ is non-singular

Possible further optimizations
- Sub-quadratic polynomial multiplication
- Good fit for x86-64/Aarch64 ISA extensions (e.g. CLMUL/vector units).
## Running times for LEDA$kem$

Portable C99 implementation, on x86-64 nocona gcc target (no HW popcnt,pclmul*)

<table>
<thead>
<tr>
<th>Category</th>
<th>$n_0$</th>
<th>KeyGen (ms)</th>
<th>Encrypt (ms)</th>
<th>Decrypt (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>45.91 (± 0.95)</td>
<td>1.94 (± 0.09)</td>
<td>21.69 (± 1.39)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>24.70 (± 0.44)</td>
<td>2.13 (± 0.09)</td>
<td>25.34 (± 2.00)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>22.55 (± 0.30)</td>
<td>2.72 (± 0.12)</td>
<td>27.24 (± 1.77)</td>
</tr>
<tr>
<td>2–3</td>
<td>2</td>
<td>215.35 (± 3.42)</td>
<td>8.61 (± 0.28)</td>
<td>61.74 (± 4.95)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>118.93 (± 1.57)</td>
<td>9.09 (± 0.23)</td>
<td>54.12 (± 1.79)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>90.74 (± 1.12)</td>
<td>9.83 (± 0.20)</td>
<td>56.79 (± 2.21)</td>
</tr>
<tr>
<td>4–5</td>
<td>2</td>
<td>651.58 (± 5.81)</td>
<td>24.18 (± 0.61)</td>
<td>109.85 (± 6.75)</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>354.45 (± 5.72)</td>
<td>25.95 (± 0.91)</td>
<td>112.36 (± 3.48)</td>
</tr>
<tr>
<td></td>
<td>4</td>
<td>257.84 (± 2.97)</td>
<td>27.44 (± 0.38)</td>
<td>149.93 (± 4.65)</td>
</tr>
</tbody>
</table>
Thanks for the attention

Questions?
https://www.ledacrypt.org