Constructive aspects of code-based cryptography

Marco Baldi
Università Politecnica delle Marche
Ancona, Italy

m.baldi@univpm.it
Code-based cryptography

• Cryptographic primitives based on the decoding problem

• Main challenge: put the adversary in the condition of decoding a random-like code

• Everything started with the McEliece (1978) and Niederreiter (1986) public-key cryptosystems

• A large number of variants originated from them

• Some private-key cryptosystems were also derived

• The extension to digital signatures is still challenging (most concrete proposals: Courtois-Finiasz-Sendrier (CFS) and Kabatianskii-Krouk-Smeets (KKS) schemes)
Main ingredients (McEliece)

• Private key:

\[ \{G, S, P\} \]

- **G**: generator matrix of a \( t \)-error correcting \((n, k)\) Goppa code
- **S**: \( k \times k \) non-singular dense matrix
- **P**: \( n \times n \) permutation matrix

• Public key:

\[ G' = S \cdot G \cdot P \]

The private and public codes are permutation equivalent!
Main ingredients (McEliece)

- Encryption map:
  \[ x = u \cdot G' + e \]

- Decryption map:
  \[ x' = x \cdot P^{-1} = u \cdot S \cdot G + e \cdot P^{-1} \]

  all errors are corrected, so we have:

  \[ u' = u \cdot S \] at the decoder output

  \[ u = u' \cdot S^{-1} \]
Main ingredients (McEliece)

• Goppa codes are classically used as secret codes

• Any degree-\( t \) (irreducible) polynomial generates a different Goppa code (very large families of codes with the same parameters and correction capability)

• Their matrices are non-structured, thus their storage requires \( kn \) bits, which are reduced to \( rk \) bits with a CCA2 secure conversion

• The public key size grows quadratically with the code length
Niederreiter cryptosystem

• Exploits the same principle, but uses the code parity-check matrix \((\mathbf{H})\) in the place of the generator matrix \((\mathbf{G})\)

• Secret key: \(\{\mathbf{H}, \mathbf{S}\} \rightarrow\) Public key: \(\mathbf{H}' = \mathbf{S}\mathbf{H}\)

• Message mapped into a weight-\(t\) error vector \((\mathbf{e})\)
• Encryption: \(\mathbf{x} = \mathbf{H}'\mathbf{e}^T\)
• Decryption: \(\mathbf{s} = \mathbf{S}^{-1}\mathbf{x} = \mathbf{H}\mathbf{e}^T \rightarrow\) syndrome decoding \((\mathbf{e})\)

• In this case there is no permutation (identity), since passing from \(\mathbf{G}\) to \(\mathbf{H}\) suffices to hide the Goppa code (indeed the permutation could be avoided also in McEliece)
Permutation equivalence

• Using permutation equivalent private and public codes works for the original system based on Goppa codes

• Many attempts of using other families of codes (RS, GRS, convolutional, RM, QC, QD, LDPC) have been made, aimed at reducing the public key size

• In most cases, they failed due to permutation equivalence between the private and the public code

• In fact, permutation equivalence was exploited to recover the secret key from the public key
Permutation equivalence (2)

• Can we remove permutation equivalence?

• We need to replace $P$ with a more general matrix $Q$

• This way, $G' = S \cdot G \cdot Q$ and the two codes are no longer permutation equivalent

• Encryption is unaffected

• Decryption: $x' = x \cdot Q^{-1} = u \cdot S \cdot G + e \cdot Q^{-1}$
Permutation equivalence (3)

• How can we guarantee that $e' = e \cdot Q^{-1}$ is still correctable by the private code?

• We shall guarantee that $e'$ has a low weight

• This is generally impossible with a randomly designed matrix $Q$

• But it becomes possible through some special choices of $Q$
Design of $Q$: first approach

• Design $Q^{-1}$ as an $n \times n$ sparse matrix, with average row and column weight equal to $m$:
  \[
  1 < m \ll n
  \]

• This way, $w(e') \leq m \cdot w(e)$ and $w(e') \approx m \cdot w(e)$ due to the matrix sparse nature

• $w(e')$ is always $\leq m \cdot w(e)$ with regular matrices ($m$ integer)

• The same can be achieved with irregular matrices ($m$ fractional), with some trick in the design of $Q$
Design of $Q$: second approach

• Design $Q^{-1}$ as an $n \times n$ sparse matrix $T$, with average row and column weight equal to $m$, summed to a low rank matrix $R$, such that:

$$e \cdot Q^{-1} = e \cdot T + e \cdot R$$

• Then:
  – Use only intentional error vectors $e$ such that $e \cdot R = 0$
  ...or...
  – Make Bob informed of the value of $e \cdot R$
LDPC-code based cryptosystems
(example of use of the first approach)

SpringerBriefs in Electrical and Computer Engineering
(preprint available on ResearchGate)
LDPC codes

- Low-Density Parity-Check (LDPC) codes are capacity-achieving codes under Belief Propagation (BP) decoding

- They allow a random-based design, which results in large families of codes with similar characteristics

- The low density of their matrices could be used to reduce the key size, but this exposes the system to key recovery attacks

- Hence, the public code cannot be an LDPC code, and permutation equivalence to the private code must be avoided

---


LDPC codes (2)

• LDPC codes are linear block codes
  – $n$: code length
  – $k$: code dimension
  – $r = n - k$: code redundancy
  – $G$: $k \times n$ generator matrix
  – $H$: $r \times n$ parity-check matrix
  – $d_v$: average $H$ column weight
  – $d_c$: average $H$ row weight

• LDPC codes have parity-check matrices with:
  – Low density of ones ($d_v \ll r$, $d_c \ll n$)
  – No more than one overlapping symbol 1 between any two rows/columns
  – No short cycles in the associated Tanner graph

\[
H = \begin{bmatrix}
0 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 0 \\
1 & 1 & 0 & 1 & 0 & 1 & 1 \\
\end{bmatrix}
\]
LDPC decoding

• LDPC decoding can be accomplished through the Sum-Product Algorithm (SPA) with Log-Likelihood Ratios (LLR)

• For a random variable $U$:

$$LLR(U) = \ln \left( \frac{\Pr(U = 0)}{\Pr(U = 1)} \right)$$

• The initial LLRs are derived from the channel

• They are then updated by exchanging messages on the Tanner graph
LDPC decoding for the McEliece PKC

- The McEliece encryption map is equivalent to transmission over a special Binary Symmetric Channel with error probability $p = t/n$

- LLR of \textit{a priori} probabilities associated with the codeword bit at position $i$:

  $$LLR(x_i) = \ln\left[\frac{P(x_i = 0 \mid y_i = y)}{P(x_i = 1 \mid y_i = y)}\right]$$

- Applying the Bayes theorem:

  $$LLR(x_i \mid y_i = 0) = \ln\left(\frac{1-p}{p}\right) = \ln\left(\frac{n-t}{t}\right)$$

  $$LLR(x_i \mid y_i = 1) = \ln\left(\frac{p}{1-p}\right) = \ln\left(\frac{t}{n-t}\right)$$
Bit flipping decoding

- LDPC decoding can also be accomplished through hard-decision iterative algorithms known as bit-flipping (BF).

- During an iteration, every check node sends each neighboring variable node the binary sum of all its neighboring variable nodes, excluding that node.

- In order to send a message back to each neighboring check node, a variable node counts the number of unsatisfied parity-check sums from the other check nodes.

- If this number overcomes some threshold, the variable node flips its value and sends it back, otherwise, it sends its initial value unchanged.

- BF is well suited when soft information from the channel is not available (as in the McEliece cryptosystem).
Decoding threshold

- Differently from algebraic codes, the **decoding radius** of LDPC codes is not easy to estimate.

- Their error correction capability is statistical (with a high mean).

- For iterative decoders, the **decoding threshold** of large ensembles of codes can be estimated through density evolution techniques.

- The decoding threshold of BF decoders can be found by iterating simple closed-form expressions.

<table>
<thead>
<tr>
<th>$n$ [bits]</th>
<th>12288</th>
<th>15360</th>
<th>18432</th>
<th>21504</th>
<th>24576</th>
<th>27648</th>
<th>30720</th>
<th>33792</th>
<th>36864</th>
<th>39936</th>
<th>43008</th>
<th>46080</th>
<th>49152</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R = 2/3$</td>
<td>$d_v = 13$</td>
<td>190</td>
<td>237</td>
<td>285</td>
<td>333</td>
<td>380</td>
<td>428</td>
<td>476</td>
<td>523</td>
<td>571</td>
<td>619</td>
<td>666</td>
<td>714</td>
</tr>
<tr>
<td>$d_v = 15$</td>
<td>192</td>
<td>240</td>
<td>288</td>
<td>336</td>
<td>384</td>
<td>432</td>
<td>479</td>
<td>527</td>
<td>575</td>
<td>622</td>
<td>670</td>
<td>718</td>
<td>766</td>
</tr>
<tr>
<td>$R = 3/4$</td>
<td>$d_v = 13$</td>
<td>181</td>
<td>225</td>
<td>270</td>
<td>315</td>
<td>360</td>
<td>405</td>
<td>450</td>
<td>495</td>
<td>540</td>
<td>585</td>
<td>630</td>
<td>675</td>
</tr>
<tr>
<td>$d_v = 15$</td>
<td>187</td>
<td>233</td>
<td>280</td>
<td>327</td>
<td>374</td>
<td>421</td>
<td>468</td>
<td>515</td>
<td>561</td>
<td>608</td>
<td>655</td>
<td>702</td>
<td>749</td>
</tr>
</tbody>
</table>
Quasi-Cyclic codes

• A linear block code is a Quasi-Cyclic (QC) code if:
  1. Its dimension and length are both multiple of an integer $p$ ($k = k_0p$ and $n = n_0p$)
  2. Every cyclic shift of a codeword by $n_0$ positions yields another codeword

• The generator and parity-check matrices of a QC code can assume two alternative forms:
  – Circulant of blocks
  – Block of circulants
QC-LDPC codes with rate \((n_0 - 1)/n_0\)

- For \(r_0 = 1\), we obtain a particular family of codes with length \(n = n_0p\), dimension \(k = k_0p\) and rate \((n_0 - 1)/n_0\)

- \(H\) has the form of a single row of circulants:
  \[
  H = \begin{bmatrix} H^c_0 & H^c_1 & \cdots & H^c_{n_0-1} \end{bmatrix}
  \]
  completely described by its first row

- In order to be non-singular, \(H\) must have at least one non-singular block (suppose the last)

- In this case, \(G\) (in systematic form) is easily derived:
  \[
  G = \begin{bmatrix} \left( H^c_{n_0-1} \right)^{-1} \cdot H^c_0^T \\ \left( H^c_{n_0-1} \right)^{-1} \cdot H^c_1^T \\ \vdots \\ \left( H^c_{n_0-1} \right)^{-1} \cdot H^c_{n_0-2}^T \end{bmatrix}
  \]
  completely described by its \((k + 1)\)-th column
Random-based design

- A **Random Difference Family** (RDF) is a set of subsets of a finite group $G$ such that every non-zero element of $G$ appears no more than once as a difference of two elements in a subset.

- An RDF can be used to obtain a QC-LDPC matrix free of length-4 cycles in the form:

  $$H = \left[ H_0^c \quad H_1^c \quad \ldots \quad H_{n_0-1}^c \right]$$

- The random-based approach allows to design large families of codes with fixed parameters.

- The codes in a family share the characteristics that mostly influence LDPC decoding, thus they have equivalent error correction performance.
An example

• RDF over $\mathbb{Z}_{13}$:
  – $\{1, 3, 8\}$ (differences: 2, 11, 7, 6, 5, 8)
  – $\{5, 6, 9\}$ (differences: 1, 12, 4, 9, 3, 10)

• Parity-check matrix ($n_0 = 2, p = 13$):

\[
H = \begin{bmatrix}
0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0
\end{bmatrix}
\]
Attacks

• In addition to classical attacks against McEliece, some specific attacks exist against QC-LDPC codes

• **Dual-code attacks**: search for low weight codewords in the dual of the public code in order to recover the secret (and sparse) $H$

• **QC code weakness**: exploit the QC nature to facilitate information set decoding (decode one out of many) and low weight codeword searches

• Their work factor depends on the complexity of information set decoding (ISD)
Dual code attacks

• Avoiding permutation equivalence is fundamental to counter these attacks

• We use $Q^{-1}$ with row and column weight $m \ll n$

• $Q$ and $Q^{-1}$ are formed by $n_0 \times n_0$ circulant blocks with size $\rho$ to preserve the QC nature in the public code

• The public code has parity-check matrix $H' = H(Q^{-1})^T$

• The row weight of $H'$ is about $m$ times that of $H$
Security level and Key Size

• Minimum attack WF for $m = 7$:

<table>
<thead>
<tr>
<th>$p$ [bits]</th>
<th>4096</th>
<th>5120</th>
<th>6144</th>
<th>7168</th>
<th>8192</th>
<th>9216</th>
<th>10240</th>
<th>11264</th>
<th>12288</th>
<th>13312</th>
<th>14336</th>
<th>15360</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_0 = 3$</td>
<td>254</td>
<td>254</td>
<td>254</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
<td>284</td>
</tr>
<tr>
<td>$d_v = 13$</td>
<td>263</td>
<td>264</td>
<td>273</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
</tr>
<tr>
<td>$d_v = 15$</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
</tr>
<tr>
<td>$d_v = 13$</td>
<td>273</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
</tr>
<tr>
<td>$d_v = 15$</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
<td>275</td>
</tr>
</tbody>
</table>

• Key size (bytes):

<table>
<thead>
<tr>
<th>$p$ [bits]</th>
<th>4096</th>
<th>5120</th>
<th>6144</th>
<th>7168</th>
<th>8192</th>
<th>9216</th>
<th>10240</th>
<th>11264</th>
<th>12288</th>
<th>13312</th>
<th>14336</th>
<th>15360</th>
<th>16384</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n_0 = 3$</td>
<td>1024</td>
<td>1280</td>
<td>1536</td>
<td>1792</td>
<td>2048</td>
<td>2304</td>
<td>2560</td>
<td>2816</td>
<td>3072</td>
<td>3328</td>
<td>3584</td>
<td>3840</td>
<td>4096</td>
</tr>
<tr>
<td>$n_0 = 4$</td>
<td>1536</td>
<td>1920</td>
<td>2304</td>
<td>2688</td>
<td>3072</td>
<td>3456</td>
<td>3840</td>
<td>4224</td>
<td>4608</td>
<td>4992</td>
<td>5376</td>
<td>5760</td>
<td>6144</td>
</tr>
</tbody>
</table>

Comparison with Goppa codes

- Comparison considering the Niederreiter version with 80-bit security (CCA2 secure conversion)

<table>
<thead>
<tr>
<th>Solution</th>
<th>n</th>
<th>k</th>
<th>t</th>
<th>Key size [bytes]</th>
<th>Enc. compl.</th>
<th>Dec. compl.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goppa based</td>
<td>1632</td>
<td>1269</td>
<td>33</td>
<td>57581</td>
<td>48</td>
<td>7890</td>
</tr>
<tr>
<td>QC-LDPC based</td>
<td>24576</td>
<td>18432</td>
<td>38</td>
<td>2304</td>
<td>1206</td>
<td>1790 (BF)</td>
</tr>
</tbody>
</table>

- For the QC-LDPC code-based system, the key size grows linearly with the code length, due to the quasi-cyclic nature of the codes, while with Goppa codes it grows quadratically.
MDPC code-based variants

- An alternative is to use Moderate-Density Parity-Check (MDPC) codes in the place of LDPC codes

- This means to incorporate the density of $Q^{-1}$ into the private code, which is no longer an LDPC code

- Then the public code can still be permutation equivalent to the private code

- QC-MDPC code based variants can be designed too

---

MDPC code-based variants (2)

• It appears that the short cycles in the Tanner graph are no longer a problem with MDPC codes

• Therefore, their matrices can be designed completely at random

• This has permitted to obtain the first security reduction (to the random linear code decoding problem) for these schemes

• On the other hand, decoding MDPC codes is more complex than for LDPC codes (due to denser graphs)
Irregular codes

• Irregular LDPC codes achieve higher error correction capability than regular ones

• This can be exploited to increase the system efficiency by reducing the code length...

• ...although the QC structure and the need to avoid enumeration impose some constraints

160-bit security

<table>
<thead>
<tr>
<th>QC-LDPC code type</th>
<th>$n_0$</th>
<th>$d'_v$</th>
<th>$t$</th>
<th>$d_v$</th>
<th>$n$</th>
<th>Key size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>regular</td>
<td>4</td>
<td>97</td>
<td>79</td>
<td>13</td>
<td>54616</td>
<td>5121</td>
</tr>
<tr>
<td>irregular</td>
<td>4</td>
<td>97</td>
<td>79</td>
<td>13</td>
<td>46448</td>
<td>4355</td>
</tr>
</tbody>
</table>

Symmetric variants

• The same principles can also be exploited to build a symmetric cryptosystem inspired to the Barbero-Ytrehus system

• Also in this case, QC-LDPC codes allow to achieve considerable reductions in the key size

• A QC-LDPC matrix is used as a part of the private key

• The sparse nature of the circulant matrices is also exploited by using run-length coding and Huffman coding to achieve a very compact representation of the private key

GRS-code based cryptosystems
(example of use of the second approach)
Replacing Goppa with GRS codes

• GRS codes are maximum distance separable codes, thus have optimum error correction capability

• This would allow to reduce the public key size

• GRS codes are widespread, and already implemented in many practical systems

• On the other hand, they are more structured than Goppa codes (and wild Goppa codes)
Weakness of GRS codes

• When the public code is permutation equivalent to the private code, the latter can be recovered

• This was first shown by the Sidelnikov-Shestakov attack against the GRS code-based Niederreiter cryptosystem
Avoiding permutation equivalence

• Public parity-check matrix (Niederreiter):

\[ H' = S^{-1} \cdot H \cdot Q^{-1} \]

• \( Q^{-1} = R + T \)

• \( R \): dense \( n \times n \) matrix with rank \( z \ll n \)

• \( T \): sparse \( n \times n \) matrix with average row and column weight \( m \ll n \)

• All matrices are over \( GF(q) \)

Avoiding permutation equivalence (2)

• Example of construction of $R$:
  – take two matrices $a$ and $b$ defined over $\text{GF}(q)$, having size $z \times n$ and rank $z$
  – Compute $R = b^T \cdot a$

• Encryption:
  – Alice maps the message into an error vector $e$ with weight $[t/m]$
  – Alice computes the ciphertext as $x = H' \cdot e^T$
Avoiding permutation equivalence (3)

• Decryption:
  • Bob computes \( x' = S \cdot x = H \cdot Q^{-1} \cdot e^T = H \cdot (b^T \cdot a + T) \cdot e^T = H \cdot b^T \cdot y + H \cdot T \cdot e^T \), where \( y = a \cdot e^T \)
  • We suppose that Bob knows \( y \), then he computes \( x'' = x' - H \cdot b^T \cdot y = H \cdot T \cdot e^T \)
  • \( e' = T \cdot e^T \) has weight \( \leq t \), thus \( x'' \) is a correctable syndrome
  • Bob recovers \( e' \) by syndrome decoding through the private code
  • He multiplies the result by \( T^{-1} \) and demaps \( e \) into the secret message
Main issue

• How can Bob be informed of the value of \( y = a \cdot e^T \)?

• Two possibilities:
  – Alice knows \( a \) (which is made public), computes \( y \) and sends it along with the ciphertext (or select only error vectors such that \( y \) is known (all-zero)).
  – Alice does not know \( a \) and Bob has to guess the value of \( y \)

• Both them have pros and cons
A History of proposals and attacks

Subcode vulnerability

• When $a$ is public, an attacker can look at $H_s = \begin{bmatrix} H' \\ a \end{bmatrix}$

• For any codeword $c$ in this subcode: $S^{-1}HTc^T = 0$

• Hence, the effect of the dense matrix $R$ is removed

• When $T$ is a permutation matrix, the subcode defined by $H_s$ is permutation-equivalent to a subcode of the secret code

• The dimension of the subcode is $n - \text{rank}\{H_s\}$
Distinguishing attacks

- When $a$ is private, Bob has to guess the value of $y$
- The number of attempts he needs increases as $q^z$
- Therefore only very small values of $z (z = 1)$ are feasible
- When $z = 1$ and $m$ is small, the system can be attacked by exploiting distinguishers
- These attacks, recently improved, force us to use very large values of $m (m ≈ 2)$ when $z = 1$
Avoiding attacks

• Publish a such that z can be increased, but avoid subcode attacks

• This could be achieved by reducing the dimension of the subcode to zero, which occurs for $z \geq k$

• Let us consider $z = k$ (can be extended to $z \geq k$): in this case $H_S$ is a square invertible matrix

• The attacker could consider the system

\[
\begin{bmatrix}
  x \\
  y
\end{bmatrix} = H_S \cdot e^T
\]
Avoiding attacks (2)

• This further attacks is avoided if:
  – we design $b$ such that it has rank $z' < z$ and make a basis of the kernel of $b^T$ public (through a $z' \times z$ matrix $B$)
  – rather than sending $y$ along with the ciphertext, Alice computes and sends $y' = y + v$, where $v$ is a $z \times 1$ vector in the kernel of $b^T$ (that is, $b^T v = 0$)
  – $v$ is obtained as a non-trivial random linear combination of the basis vectors

• This way, when Bob computes $b^T y'$ he still obtains $b^T y$, but the attack is avoided since $y$ is hidden
ISD WF and Key Size

- **Goppa code-based (PK: $H'$ over GF(2))**
  
<table>
<thead>
<tr>
<th>$n$</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>3004</td>
</tr>
<tr>
<td>$t$</td>
<td>91</td>
</tr>
<tr>
<td>WF</td>
<td>180.1</td>
</tr>
<tr>
<td>KS</td>
<td>400.4</td>
</tr>
</tbody>
</table>

- **GRS code-based (PK: $\{H', a, B\}$ over GF(512))**
  
<table>
<thead>
<tr>
<th>$n$</th>
<th>511</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>311</td>
</tr>
<tr>
<td>$t$</td>
<td>100</td>
</tr>
<tr>
<td>WF</td>
<td>180.1</td>
</tr>
<tr>
<td>KS</td>
<td>295.9</td>
</tr>
</tbody>
</table>
Comparison

• Consider the instances of both systems with highest code rate able to reach $WF \geq 2^{180}$

• By using the GRS code-based system, we achieve a public key size reduction in the order of 26% over the classical one

• The gap is even larger by considering lower code rates
Digital signature schemes based on sparse syndromes

(another example of use of the second approach)
From PKC to Digital Signatures

- RSA
- McEliece

encryption map
Code-based signature schemes

• Simply inverting decryption with encryption does not work with code-based PKCs

• Some specific solution must be designed

• Two main code-based digital signature schemes:
  – Kabatianskii-Krouk-Smeets (KKS)
  – Courtois-Finiasz-Sendrier (CFS)

• CFS appears to be more robust than KKS
CFS

• Close to the original McEliece Cryptosystem
• Based on Goppa codes

• Public:
  – A hash function \( \mathcal{H}(\cdot) \)
  – A function \( \mathcal{F}(h) \) able to transform any hash digest \( h \) into a correctable syndrome through the code \( C \)

• Key generation:
  – The signer chooses a Goppa code able to correct \( t \) errors, having parity-check matrix \( H \)
  – He chooses a scrambling matrix \( S \) and publishes \( H' = SH \)
CFS (2)

• Signing the document $D$:
  – The signer computes $s = F(\mathcal{H}(D))$ and $s' = S^{-1} s$
  – He decodes the syndrome $s'$ through the secret code
  – The error vector $\mathbf{e}$ is the signature

• Verification:
  – The verifier computes $s = F(\mathcal{H}(D))$
  – He checks that $\mathbf{H}' \mathbf{e}^T = \mathbf{S} \mathbf{H} \mathbf{e}^T = \mathbf{S} \mathbf{S}^{-1} s = s$
The main issue is to find an efficient function $F(h)$.

In the original CFS there are two solutions:
- Appending a counter to $h = \mathcal{H}(D)$ until a valid signature is generated
- Performing complete decoding

Both these methods require codes with very special parameters:
- very high rate
- very small error correction capability
Weaknesses

• Codes with small $t$ and high rate could be decoded, with good probability, through the Generalized Birthday Paradox Algorithm (GBA)

• High rate Goppa codes have been discovered to produce public codes which are distinguishable from random codes

• The public key size and decoding complexity can be very large
A CFS variant

• Main differences:
  – Only a subset of sparse syndromes is considered
  – Goppa codes are replaced with low-density generator-
    matrix (LDGM) codes

• Main advantages:
  – Significant reductions in the public key size are achieved
  – Classical attacks against the CFS scheme are inapplicable
  – Decoding is replaced by a straightforward vector
    manipulation

Rationale

• If we use a secret code in systematic form and sparse syndromes, we can obtain **sparse signatures**

• An attacker instead can only forge dense signatures

• Example:
  – secret code: $H = [X|I]$, with $I$ an $r \times r$ identity matrix
  – $s$ is an $r \times 1$ sparse syndrome vector
  – the error vector $e = [0|s^T]$ is sparse and verifies $H e^T = s$
Issues

• The map $s \leftrightarrow e$ is trivial (and also linear!)

• The public syndrome should undergo (at least) a secret permutation before obtaining $e$

• Also $e$ should be disguised before being made public

• Sparsity is used to distinguish $e$ from other (forged) vectors in the same coset, but it should not endanger the system security
Key generation

• Private key: \{Q, H, S\}, with
  – H: \(r \times n\) parity-check matrix of the secret code \(C(n, k)\)
  – Q = R + T
  – R = a^T b, having rank \(z \ll n\)
  – T: sparse random matrix with row and column weight \(m_T\), such that Q is full rank
  – S: sparse non-singular \(n \times n\) matrix with average row and column weight \(m_S \ll n\)

• Public key: \(H' = Q^{-1} H S^{-1}\)
Signature generation

• Given the document $M$
• The signer computes $h = \mathcal{H}(M)$
• The signer finds $s = \mathcal{F}(h)$, with weight $w$, such that $b \cdot s = 0$ (this requires $2^z$ attempts, on average)
• The signer computes the private syndrome $s' = Q \cdot s$, with weight $\leq m_T w$
• The signer computes the private error vector $e = [0 | s'^T]$
• The signer selects a random codeword $c \in C$ with small weight $w_c$
• The signer computes the public signature of $M$ as $e' = (e + c) S^T$
Signature generation issues

- Without any random codeword \( c \), the signing map becomes linear, and signatures can be easily forged.

- With \( c \) having weight \( w_c \ll n \), the map becomes affine, and summing two signatures does not result in a valid signature.

- The signature should not change each time a document is signed, to avoid attacks exploiting many signatures of the same document.

- It suffices to choose \( c \) as a deterministic function of \( M \).
Signature verification

• The verifier receives the message $M$, its signature $e'$ and the parameters to use in $F$

• He checks that the weight of $e'$ is $\leq (m_Tw + w_c)m_S$, otherwise the signature is discarded

• He computes $s^* = F(H(M))$ and checks that it has weight $w$, otherwise the signature is discarded

• He computes $H' e'^T = Q^{-1} H S^{-1} S (e^T + c^T) = Q^{-1} H (e^T + c^T) = Q^{-1} H e^T = Q^{-1} s' = s$

• If $s = s^*$, the signature is accepted, otherwise it is discarded
LDGM codes

• LDGM codes are codes with a low density generator matrix $G$

• The row weight of $G$ is $w_g \ll n$

• They are useful in this cryptosystem because:
  – Large random-based families of codes can be designed
  – Finding low weight codewords is very easy
  – Structured codes (e.g. QC) can be designed
Attacks

• The signature $e'$ is an error vector corresponding to the public syndrome $s$ through the public code parity-check matrix $H'$

• If $e'$ has a low weight it is difficult to find, otherwise signatures could be forged

• If $e'$ has a too low weight the supports of $e$ and $c$ could be almost disjoint, and the link between the support of $s$ and that of $e'$ could be discovered

• Hence, the density of $e'$ must be:
  – sufficiently low to avoid forgeries
  – sufficiently high to avoid support decompositions
Attacks (2)

• If the matrix $S$ is (sparse and) regular, statistical arguments could be used to analyze large number of intercepted signatures (thanks to J. P. Tillich for pointing this out)

• This way, an attacker could discover which columns of $S$ have a symbol 1 in the same row

• By iterating the procedure, the structure of the matrix $S$ could be recovered (except for a permutation)

• This can be avoided by using an irregular matrix $S$ with the same average weight

Examples

<table>
<thead>
<tr>
<th>SL (bits)</th>
<th>n</th>
<th>k</th>
<th>p</th>
<th>w</th>
<th>wg</th>
<th>wc</th>
<th>z</th>
<th>mT</th>
<th>mS</th>
<th>A_{wc}</th>
<th>N_s</th>
<th>S_k (KiB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>9800</td>
<td>4900</td>
<td>50</td>
<td>18</td>
<td>20</td>
<td>160</td>
<td>2</td>
<td>1</td>
<td>9</td>
<td>2^{82.76}</td>
<td>2^{166.10}</td>
<td>117</td>
</tr>
<tr>
<td>120</td>
<td>24960</td>
<td>10000</td>
<td>80</td>
<td>23</td>
<td>25</td>
<td>325</td>
<td>2</td>
<td>1</td>
<td>14</td>
<td>2^{140.19}</td>
<td>2^{242.51}</td>
<td>570</td>
</tr>
<tr>
<td>160</td>
<td>46000</td>
<td>16000</td>
<td>100</td>
<td>29</td>
<td>31</td>
<td>465</td>
<td>2</td>
<td>1</td>
<td>20</td>
<td>2^{169.23}</td>
<td>2^{326.49}</td>
<td>1685</td>
</tr>
</tbody>
</table>

- For **80-bit security**, the original CFS system needs a Goppa code with $n = 2^{21}$ and $r = 2^{10}$, which gives a key size of 52.5 MiB

- By using the parallel CFS, the same security level is obtained with key sizes between 1.25 MiB and 20 MiB

- The proposed system requires a public key of only **117 KiB** to achieve 80-bit security (by using QC-LDGM codes)
Comments

• Permutation equivalence between private and public codes can be avoided

• This opens the way to the use of families of codes other than Goppa codes

• Both public-key encryption and digital signature schemes can take advantage of this

• This results in strong reductions in the size of the public keys