Tweaking Code-Based Cryptography for Embedded Systems

DIMACS Workshop on The Mathematics of Post-Quantum Cryptography

Tim Güneysu, Ingo von Maurich
Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany 1/12/2015
Motivation

- High demand for security in the Internet of Things (IoT)

Requirements
- Highly embedded/cost-sensitive
- Long life-time/security
- Diversity of target platforms
- Simple physical accessibility

Consequences
- Quantum-computer resistant cryptography
- Implementations for a wide range of cheap embedded devices
Motivation

- **Cryptography in the era of quantum computing**
  - Symmetric: Security level for key lengths is halved (Grover) ... not good but we can fix it.
  - Asymmetric: Polytime attacks on RSA and Elliptic Curve exist (Shor) ... so it’s essential to have alternatives ready!

- **Task**: Deploy new asymmetric schemes that are
  - resistant to attacks from quantum computing
  - as efficient as RSA and ECC on our today’s and future computing platforms
  - available with many implementations

⇒ *Code-based Crypto on Embedded Platforms*
Overview

Motivation

Background

Efficient Decoding Techniques
Implementing QC-MDPC McEliece
Side-Channel Attacks
Countermeasures
Cryptography on Embedded Devices

Common computing platforms of embedded devices

- **Microcontrollers (µC)**
  - Small 8/16/32-bit CPU, small RAM (≈ 512B-256KB), a bit more Flash (≈ 4KB-1MB)

- **Reconfigurable Hardware (FPGA)**
  - LUT-based logic functions, flip-flops, some 18/36 kBit block memories and DSP units

- **Application-Specific Integrated Circuits (ASIC)**
  - Dedicated hardware design of an individual application
Microcontroller Architecture
AVR & ARM M4 architectures

FPGA Architecture
Altera/Xilinx FPGA

A slice contains
- 2-4 Look-Up Tables (LUT) as logic function generators
- 2-8 flip flops for data storage
Cryptography with Linear Codes?

- Error-Correcting Codes are well-known in a large variety of applications
- Detection/Correction of errors in noisy channels by adding redundancy

\[ c = \begin{array}{c} m \\ r \end{array} \quad \rightarrow \quad \begin{array}{c} \text{Channel} \\ y = c + e \end{array} \]

- Observation:
  Some problems in code-based theory are NP-complete
  \[ \Rightarrow \text{Possible Foundation of Code-Based Cryptosystems (CBC)} \]
• **Generator** and **parity check matrices** for encoding and decoding

• Matrices in **systematic form** minimize time and storage

- Rows of \( G \) form a basis for the code \( C[n, k, d] \) of length \( n \) with dimension \( k \) and minimum distance \( d \)

Matrix size of \( G \): \( k \times n \)
Linear Codes and Cryptography

- Parity check matrix $H$ is a $(n-k) \cdot k$ matrix orthogonal to $G$
- Defines the dual $C$ of the code $C$ via scalar product
  \[ C^\perp = \{ y \in \mathbb{F}_q^n | x \cdot y = 0, \forall x \in C \} \]
- A codeword $c \in C$ if and only if $Hc = 0$
- The term $s = Hc' = Hc + He$ is the syndrome of the error

\[ \mathcal{H} \times c = \mathcal{H} \times e = s \]
**McEliece Encryption Scheme [1978]**

**Key Generation**
Given a \([n, k]\)-code \(C\) with generator matrix \(G\) and error correcting capability \(t\)

*Private Key*: \((S, G, P)\), where \(S\) is a scrambling and \(P\) is a permutation matrix

*Public Key*: \(G' = S \cdot G \cdot P\)

**Encryption**
Message \(m \in \mathbb{F}_2^k\), error vector \(e \in_R \mathbb{F}_2^n\), \(\text{wt}(e) \leq t\)

\(x \leftarrow mG' + e\)

**Decryption**
Let \(\Psi_H\) be a \(t\)-error-correcting decoding algorithm.

\(m \cdot S \leftarrow \Psi_H(x \cdot P^{-1})\), removes the error \(e \cdot P^{-1}\)

Extract \(m\) by computing \(m \cdot S \cdot S^{-1}\)
Security Parameters (Goppa Codes)

- **Original proposal**: McEliece with binary Goppa codes
- Code properties determine key size, **matrices are often large**
- Code parameters revisited by Bernstein, Lange and Peters
- Public key is a $k \times (n - k)$ bit matrix (redundant part only)

| Security Level      | Parameters $(n, k, t)$, errors added | $K_{pub}$ size in KBits | $K_{sec}$ size $(g(z) | L | M^{-1})$ KBits |
|---------------------|-------------------------------------|-------------------------|-----------------------------------|
| Short-term (60 bit) | (1024, 644, 38), 38                | 239                     | (0.37 | 10 | 141)                        |
| Mid-term I (80 bit) | (2048, 1751, 27), 27               | 507                     | (0.29 | 22 | 86)                         |
| Mid-term II (128 bit)| (2690, 2280, 56), 57              | 913                     | (0.38 | 18 | 164)                       |
| Long-term (256 bit)  | (6624, 5129, 115), 117             | 7,488                   | (1.45 | 84 | 2,183)                     |
• **Selection of the employed code is a highly critical issue**
  – Properties of code determine key size, **short keys essential**
  – Structures in codes reduce key size, but can enable attacks
  – Encoding is a fast operation **on all platforms** (matrix multiplication)
  – Decoding requires **efficient techniques** in terms of time and memory

• **Basic McEliece is only CPA-secure; conversion required**

• **Protection against side-channel and fault-injection attacks**
Suitable codes for code-based cryptography?

- Generalized Reed-Solomon
- Goppa
- Elliptic
- Concatenated
- Reed Muller
- LDPC/MDPC
- Srivastava
Suitable codes for code-based cryptography?

- Generalized Reed-Solomon
- Goppa
- Srivastava
- Elliptic
- Concatenated
- LDPC/MDPC

See Anja’s and Nicolas’ talks on Wednesday!
Suitable codes for code-based cryptography?

- Generalized Reed-Solomon
  - PK: 63 kB
  - SK: 2.5 kB
- Goppa
  - PK: 2.5 kB
  - SK: 1.5 kB
- Srivastava
- Reed-Muller
  - PK: 63 kB
  - SK: 2.5 kB
- Elliptic
- Concatenated LDPC/MDPC
  - PK: 0.6 kB
  - SK: 1.2 kB

Key sizes for ≈ 80-bit equivalent symmetric security.

See Anja’s and Nicolas’ talks on Wednesday!
QC-MDPC Codes for Cryptography [MTSB13]

- $t$-error correcting $(n, r, w)$-QC-MDPC code of length $n = n_0 r$
- Parity-check matrix $H$ consists of $n_0$ blocks with fixed row weight $w$

**Code/Key Generation**

1. Generate $n_0$ first rows of parity-check matrix blocks $H_i$
   
   $h_i \in R F_2^r$ of weight $w_i$, $w = \sum_{i=0}^{n_0-1} w_i$

2. Obtain remaining rows by $r - 1$ quasi-cyclic shifts of $h_i$

3. $H = [H_0 | H_1 | ... | H_{n_0-1}]$

4. Generator matrix of systematic form $G = (I_k | Q)$

   $$Q = \begin{pmatrix}
   (H_{n_0-1}^{-1} * H_0)^T \\
   (H_{n_0-1}^{-1} * H_1)^T \\
   \vdots \\
   (H_{n_0-1}^{-1} * H_{n_0-2})^T
   \end{pmatrix}$$

See Marco's talk!
Background on QC-MDPC Codes

Parity check matrix $H$

$n_0 = 2$

Generator matrix $G$
### (QC-)MDPC McEliece

**Encryption**
Message \( m \in F_2^k \), error vector \( e \in_R F_2^n \), \( wt(e) \leq t \)
\[
x \leftarrow mG + e
\]

**Decryption**
Let \( \Psi_H \) be a \( t \)-error-correcting (QC-)MDPC decoding algorithm.
\[
mG \leftarrow \Psi_H(mG + e)
\]
Extract \( m \) from the first \( k \) positions.

Parameters for 80-bit equivalent symmetric security [MTSB13]
\[
n_0 = 2, n = 9602, r = 4801, w = 90, t = 84
\]
Overview

Motivation
Background

**Efficient Decoding Techniques**
Implementing QC-MDPC McEliece
Side-Channel Attacks
Countermeasures
“Bit-Flipping” Decoder

1. Compute syndrome $s$ of the ciphertext
2. Count unsatisfied parity-check-equations $\#_{\text{upc}}$ for each ciphertext bit
3. Flip ciphertext bits that violate $\geq b$ equations
4. Recompute syndrome
5. Repeat until $s = 0$ or reaching max. iterations (decoding failure)

How to determine threshold $b$?

- Precompute $b_i$ for each iteration [Gal62]
- $b = \max_{\text{upc}}$ [HP03]
- $b = \max_{\text{upc}} - \delta$ [MTSB13]
Observations

- Decoders recompute the syndrome after each iteration
- Syndrome computation $s = Hx^T$ is expensive!
- If threshold exceeded, flip codeword bit $j \rightarrow$ syndrome changes

Proposed Optimization

- Syndrome does not change arbitrarily!
  \[ s_{\text{new}} = s_{\text{old}} + h_j \]

$\rightarrow$ Tracking changes allows to omit syndrome recomputation

$\rightarrow$ Decoding based on up-to-date syndrome
Improving the Error-Correcting Capability

Error-correcting capability can be improved when using precomputed thresholds $b_i$ [Gal62]

Proposed Optimization (adaptive thresholds)

- Increment precomputed thresholds after decoding failure and restart
- If decoding fails again, increment $\Delta$ up to some fixed $\Delta_{max}$
- Achieved best results for $\Delta = 1$ and incrementing $\Delta = \Delta + 1$
- Similar approach to $b = max_{upc} - \delta$ [MTSB13]
Benchmarking

- **Empirical study on several decoder variants**
  - Direct vs. temporary syndrome update
  - Precomputed vs. adaptive thresholds
  - Modifying thresholds upon decoding failures

- **Simulation/evaluation on AMD Opteron 6276 CPUs @2.3 GHz**
  - 1,000 random codes with $n_0 = 2, n = 9602, r = 4801, w = 90$
  - 10,000 random decoding trials for each code
  - Evaluate different error weights $t = \{84, \ldots, 90\}$
Decoder Evaluation Results

Average Iterations

- t

- x1 = early aborts
- x2 = direct update
- x3 = adapt threshold

[MTSB13] [Gal62] C1 C2 C3 D1 D2 D3
optimized [MTSB13] optimized [Gal62]
Decoder Evaluation Results

Average Execution Time

<table>
<thead>
<tr>
<th>t</th>
<th>84</th>
<th>85</th>
<th>86</th>
<th>87</th>
<th>88</th>
<th>89</th>
<th>90</th>
</tr>
</thead>
<tbody>
<tr>
<td>ms</td>
<td>30,00</td>
<td>35,00</td>
<td>40,00</td>
<td>45,00</td>
<td>50,00</td>
<td>55,00</td>
<td>60,00</td>
</tr>
</tbody>
</table>

- [MTSB13]
- [Gal62]
- C1
- C2
- C3
- D1
- D2
- D3

optimized [MTSB13]
optimized [Gal62]
Decoder Evaluation Results

Failure Rate

- [MTSB13]
- [Gal62]
- C1
- C2
- C3
- D1
- D2
- D3

optimized [MTSB13]
optimized [Gal62]
Decoder Evaluation Results

![](image)

**Failure Rate**

<table>
<thead>
<tr>
<th>Failure Rate</th>
<th>0.0005</th>
<th>0.001</th>
<th>0.0015</th>
<th>0.002</th>
<th>0.0025</th>
<th>0.003</th>
<th>0.0035</th>
<th>0.004</th>
<th>0.0045</th>
<th>0.005</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- [MTSB13]
- [Gal62]
- C1
- C2
- C3
- D1
- D2
- D3
- Optimized [MTSB13]
- Optimized [Gal62]
Decoder Evaluation Results

Failure Rate

- [MTSB13]
- C1
- C2
- C3
- D1
- D2
- D3

Optimized [MTSB13]
Optimized [Gal62]
Decoder Evaluation Results

- Direct syndrome update halves the execution time
- Decoding iterations are reduced from 5.3/3.1 to 2.4 on average
- Adapting the precomputed thresholds upon a decoding failure yields very low failure rates
- Within 140,000,000 decoding tries only a single one failed at t=90
Overview

Motivation
Background
Efficient Decoding Techniques
**Implementing QC-MDPC McEliece**
Side-Channel Attacks
Countermeasures
Exploring Design Options

- First design goal: high-performance using dedicated hardware

- **Powerful FPGA**: Xilinx Virtex-6 XC6VLX240T FPGA
  - Powerful, expensive (US$ 2000)
  - 37,680 slices, each with 4x6-input LUTs and 8 FFs
  - 416 Block RAMs (36 kBit)

- Relatively small keys → *store operands directly in logic*, no BRAMs
- Count \( \#_{upc} \) for current row \( h = [h_0| h_1] \)
  → Compute Hamming weight \( \text{HW}(s \text{ AND } h_0), \text{HW}(s \text{ AND } h_1) \)
- Additional TRNG for error generation and CCA2 conversion required
QC-MDPC Encryption

- Given first 4801-bit row $g$ of $G$ and message $m$, compute $x = mG + e$
- $G$ is of systematic form $\rightarrow$ first half of $x$ is equal to $m$
- Computation of redundant part
  - Iterate over message bit by bit and rotate $g$ accordingly
  - If message bit is set, XOR current $g$ to the redundant part

\[
\begin{align*}
\text{FPGA High-Speed Encryption} \\
\text{QC-MDPC Encryption} \\
\text{- Given first 4801-bit row } g \text{ of } G \text{ and message } m, \text{ compute } x = mG + e \\
\text{- } G \text{ is of systematic form } \rightarrow \text{ first half of } x \text{ is equal to } m \\
\text{- Computation of redundant part} \\
\text{  - Iterate over message bit by bit and rotate } g \text{ accordingly} \\
\text{  - If message bit is set, XOR current } g \text{ to the redundant part}
\end{align*}
\]
FPGA High-Speed Decryption

QC-MDPC Decryption

- **Syndrome computation** \( s = Hx^T \), with \( H = [H_0|H_1] \)
  - Given 9602-bit \( h = [h_0|h_1] \) and \( x = [x_0|x_1] \)
  - Sequentially iterate over every bit of \( x_0 \) and \( x_1 \) in parallel, rotate \( h_0 \) and \( h_1 \) accordingly
  - If bit in \( x_0 \) and/or \( x_1 \) is set, XOR current \( h_0 \) and/or \( h_1 \) to intermediate syndrome
  - Technically similar to encryption (except for two parallel blocks)

- **Challenge**: Compare \( s = 0 \)?
  - Logical OR tree, lowest level based on 6-input LUTs
  - Added registers to minimize critical path
FPGA High-Speed Decryption

QC-MDPC Decryption

- **Challenge**: Count $\#_{\text{upc}}$ for current row $h = [h_0 | h_1]$
  
  - Compute HW($s \text{ AND } h_0$), HW($s \text{ AND } h_1$)
  - Split AND results into 6-bit blocks and lookup HW
  - Adder tree with registers on every level to accumulate total HW
  - Iterative vs. parallel design

- **Implementing bit-flipping**
  - If HW exceeds threshold $b_i$ the corresponding bit in $x_0/x_1$ is flipped
  - Syndrome is updated by XORing current secret poly $h_0$ and/or $h_1$
  - Generate next row $h$ by rotation and repeat
### High-Speed FPGA Results

- Post-PAR for Xilinx Virtex-6 XC6VLX240T
- Encryption takes 4,801 cycles
- Average decryption cycles
  - Iterative: \(4,801 + 2 + 2.4 \times (9,622 + 2) = 27,919\) cycles
  - Parallel: \(4,801 + 2 + 2.4 \times (4,811 + 2) = 16,363\) cycles

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Encoder</th>
<th>Decoder (iterative)</th>
<th>Decoder (parallel)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFs</td>
<td>14,429 (4%)</td>
<td>32,962 (10%)</td>
<td>41,714 (13%)</td>
</tr>
<tr>
<td>LUTs</td>
<td>9,201 (6%)</td>
<td>36,502 (24%)</td>
<td>42,274 (28%)</td>
</tr>
<tr>
<td>Slices</td>
<td>2,924 (7%)</td>
<td>10,364 (27%)</td>
<td>10,988 (29%)</td>
</tr>
<tr>
<td>Frequency</td>
<td>351.7 MHz</td>
<td>222.5 MHz</td>
<td>199.3 MHz</td>
</tr>
<tr>
<td>Time/Op</td>
<td>13.7 µs</td>
<td>125.4 µs</td>
<td>82.1 µs</td>
</tr>
<tr>
<td>Throughput</td>
<td>351.7 Mbit/s</td>
<td>38.3 Mbit/s</td>
<td>58.5 Mbit/s</td>
</tr>
<tr>
<td>Encode</td>
<td>4,801 cycles</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Compute Syndrome</td>
<td>-</td>
<td>4,801 cycles</td>
<td>4,801 cycles</td>
</tr>
<tr>
<td>Check Zero</td>
<td>-</td>
<td>2 cycles</td>
<td>2 cycles</td>
</tr>
<tr>
<td>Flip Bits</td>
<td>-</td>
<td>9,622 cycles</td>
<td>4,811 cycles</td>
</tr>
<tr>
<td>Overall average</td>
<td>4,801 cycles</td>
<td>27,918.9 cycles</td>
<td>16,363.3 cycles</td>
</tr>
</tbody>
</table>
## High-Speed FPGA Comparison

- **PK size:** 0.6 kByte vs. 100.5 kByte [SWM+10], 63.5 kByte [GDU+12]
- **Performance metric:** Time/operation vs. Mbit/s
- **Faster than previous McEliece implementations (no CCA2 yet)**

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Platform</th>
<th>$f$ [MHz]</th>
<th>Bits</th>
<th>Time/Op</th>
<th>Cycles</th>
<th>Mbit/s</th>
<th>Slices</th>
<th>BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>This work (enc)</td>
<td>XC6VLX240T</td>
<td>351.7</td>
<td>4,801</td>
<td>13.7 μs</td>
<td>4,801</td>
<td>351.7</td>
<td>2,924</td>
<td>0</td>
</tr>
<tr>
<td>This work (dec)</td>
<td>XC6VLX240T</td>
<td>199.3</td>
<td>4,801</td>
<td>82.1 μs</td>
<td>16,363</td>
<td>58.5</td>
<td>10,988</td>
<td>0</td>
</tr>
<tr>
<td>This work (dec iter.)</td>
<td>XC6VLX240T</td>
<td>222.5</td>
<td>4,801</td>
<td>125.4 μs</td>
<td>27,919</td>
<td>38.3</td>
<td>10,364</td>
<td>0</td>
</tr>
<tr>
<td>McEliece (enc) [Shoufan et al. 2010]</td>
<td>XC5VLX110T</td>
<td>163</td>
<td>512</td>
<td>500 μs</td>
<td>n/a</td>
<td>1.0</td>
<td>14,537</td>
<td>75</td>
</tr>
<tr>
<td>McEliece (dec) [Shoufan et al. 2010]</td>
<td>XC5VLX110T</td>
<td>163</td>
<td>512</td>
<td>1,290 μs</td>
<td>n/a</td>
<td>0.4</td>
<td>14,537</td>
<td>75</td>
</tr>
<tr>
<td>McEliece (dec) [Ghosh et al. 2012]</td>
<td>XC5VLX110T</td>
<td>190</td>
<td>1,751</td>
<td>500 μs</td>
<td>94,249</td>
<td>3.5</td>
<td>1,385</td>
<td>5</td>
</tr>
<tr>
<td>Niederreiter (enc) [Heyse and Güneysu 2012]</td>
<td>XC6VLX240T</td>
<td>300</td>
<td>192</td>
<td>0.66 μs</td>
<td>200</td>
<td>290.9</td>
<td>315</td>
<td>17</td>
</tr>
<tr>
<td>Niederreiter (dec) [Heyse and Güneysu 2012]</td>
<td>XC6VLX240T</td>
<td>250</td>
<td>192</td>
<td>58.78 μs</td>
<td>14,500</td>
<td>3.3</td>
<td>3,887</td>
<td>9</td>
</tr>
<tr>
<td>Ring-LWE (enc) [Roy et al. 2013]</td>
<td>XC6VLX75T</td>
<td>313</td>
<td>256</td>
<td>20.1 μs</td>
<td>6,300</td>
<td>12.7</td>
<td>n/a</td>
<td>2</td>
</tr>
<tr>
<td>Ring-LWE (dec) [Roy et al. 2013]</td>
<td>XC6VLX75T</td>
<td>313</td>
<td>256</td>
<td>9.1 μs</td>
<td>2,800</td>
<td>28.1</td>
<td>n/a</td>
<td>2</td>
</tr>
<tr>
<td>Ring-LWE (enc) [Pöppelmann and Güneysu 2013]</td>
<td>XC6VLX75T</td>
<td>262</td>
<td>256</td>
<td>26.2 μs</td>
<td>6,861</td>
<td>9.8</td>
<td>1,506</td>
<td>12</td>
</tr>
<tr>
<td>Ring-LWE (dec) [Pöppelmann and Güneysu 2013]</td>
<td>XC6VLX75T</td>
<td>262</td>
<td>256</td>
<td>16.8 μs</td>
<td>4,404</td>
<td>15.2</td>
<td>1,506</td>
<td>12</td>
</tr>
<tr>
<td>NTRU (enc/dec) [Kamal and Youssef 2009]</td>
<td>XCV1600E</td>
<td>62.3</td>
<td>251</td>
<td>1.54/1.41 μs</td>
<td>96/88</td>
<td>163/178</td>
<td>14,352</td>
<td>0</td>
</tr>
<tr>
<td>ECC-P224 [Güneysu and Paar 2008]</td>
<td>XC4VFX12</td>
<td>487</td>
<td>224</td>
<td>365.10 μs</td>
<td>177,755</td>
<td>0.6</td>
<td>1,580</td>
<td>11</td>
</tr>
<tr>
<td>ECC-163 [Rebeiro et al. 2012]</td>
<td>XC5VLX83T</td>
<td>167</td>
<td>163</td>
<td>8.60 μs</td>
<td>1436</td>
<td>18.9</td>
<td>3,446</td>
<td>0</td>
</tr>
<tr>
<td>ECC-163 [Roy et al. 2012]</td>
<td>Virtex-4</td>
<td>45.5</td>
<td>163</td>
<td>12.10 μs</td>
<td>552</td>
<td>13.4</td>
<td>12,430</td>
<td>0</td>
</tr>
<tr>
<td>ECC-163 [Dimitrov et al. 2006]</td>
<td>Virtex-II</td>
<td>128</td>
<td>163</td>
<td>35.75 μs</td>
<td>4576</td>
<td>4.6</td>
<td>2251</td>
<td>6</td>
</tr>
<tr>
<td>RSA-1024 [Suzuki and Matsumoto 2011]</td>
<td>XC5VLX30T</td>
<td>450</td>
<td>1,024</td>
<td>1,520 μs</td>
<td>684,000</td>
<td>0.7</td>
<td>3,237</td>
<td>5</td>
</tr>
</tbody>
</table>
Design Considerations

- Second design goal: **low resource/costs in hardware**

- **Low-Cost Device**: Xilinx Spartan-6 XC6SLX4 FPGA
  - Low-cost (US$ 15)
  - 600 slices, 4800 Flip-Flops, 2400 LUTs
  - 12 Block RAMs (18 kBit)

- Process keys and operands **within BRAMs**
  - 18 kBit dual-ported block memories; 32-bit data path
  - Two 32-bit values can be read/written in one clock cycle
  - Rotating $g/h$ is the most performance-critical operation

- Additional TRNG for error generation and CCA2 conversion required
FPGA Low-Resource Encryption

QC-MDPC Encryption

- Given first 4801-bit row $g$ of $G$ and message $m$, compute $x = mG + e$
- Storage requirements
  - One 18 kBit BRAM is sufficient to store message $m$, row $g$ and the redundant part (3x4801-bit vectors)
  - But only two data ports are available
  - Read out 32-bit of the message and store them in a separate register
- Error addition
  - Instead of starting with an all-zero redundant part we preload it with the second half of the error vector
FPGA Low-Resource Encryption

QC-MDPC Encryption

- Rotating \( g \) is the most performance-critical operation
  - 4801-bit vector \( g \) is stored in 151 32-bit memory cells
  - Need to rotate \( g \) 4801 times
  - For each 4801-bit rotation: 152 32-bit load, rotate, store

- **Implementation**
  - *Read-First* mode can read cell content before overwriting it with new data
  - Read a cell, rotate it, and store it back to the next cell after reading its content
    - 1 clock cycle, one data port
    - Cyclically rotated addresses in memory
FPGA Low-Resource Decryption

QC-MDPC Decryption

- Secret key and ciphertext consist of two blocks
  - Iterative vs. parallel design
  - Decoding is complex task → parallel processing

- BRAM-based implementation: storage requirements
  - Secret key (2x4801 bit)
  - Ciphertext (2x4801 bit)
  - Syndrome (4801 bit)
  - In total 3 BRAMs due to memory and port access requirements
FPGA Low-Resource Decryption

QC-MDPC Decryption

- Syndrome computation $s = Hx^T$
  - Similar technique as for encoding
- Compare $s = 0$?
  - Compute binary OR of all 32-bit blocks of the syndrome
- Count $\#_{upc}$
  - Hamming weight of syndrome AND $h_0/h_1$ (32-bit at a time)
  - Accumulate Hamming weight
- Bit-flipping
  - If $\#_{upc} \geq b_i$ invert ciphertext bit(s) and XOR $h_0/h_1$ to the syndrome while rotating both
Lightweight FPGA Results

- Post-PAR for Xilinx Spartan-6 XC6SLX4 & Virtex-6 XC6VLX240T
- Encryption takes 735,000 cycles
- Decryption takes 4,274,000 cycles on average

<table>
<thead>
<tr>
<th>Aspect</th>
<th>Virtex-6 XC6VLX240T</th>
<th></th>
<th>Spartan-6 XC6SLX4</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encryption</td>
<td>Decryption</td>
<td>Encryption</td>
<td>Decryption</td>
</tr>
<tr>
<td>FFs</td>
<td>120</td>
<td>412</td>
<td>119</td>
<td>413</td>
</tr>
<tr>
<td>LUTs</td>
<td>224</td>
<td>568</td>
<td>226</td>
<td>605</td>
</tr>
<tr>
<td>Slices</td>
<td>68</td>
<td>148</td>
<td>64</td>
<td>159</td>
</tr>
<tr>
<td>BRAM</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>Frequency</td>
<td>334 MHz</td>
<td>318 MHz</td>
<td>213 MHz</td>
<td>186 MHz</td>
</tr>
<tr>
<td>Time/Op</td>
<td>2.2 ms</td>
<td>13.4 ms</td>
<td>3.4 ms</td>
<td>23.0 ms</td>
</tr>
</tbody>
</table>
### Lightweight FPGA Comparison

- Realistic public key size (0.6 kByte vs. 50-100 kByte)
- Smallest McEliece FPGA implementation
- Sufficient performance for many applications

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Platform</th>
<th>Time/Op</th>
<th>FFs</th>
<th>LUTs</th>
<th>Slices</th>
<th>BRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lightweight McE (enc)</td>
<td>XC6SLX4</td>
<td>3.4 ms</td>
<td>119</td>
<td>226</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>Lightweight McE (dec)</td>
<td>XC6SLX4</td>
<td>23.0 ms</td>
<td>413</td>
<td>605</td>
<td>159</td>
<td>3</td>
</tr>
<tr>
<td>Lightweight McE (enc)</td>
<td>XC6VLX240T</td>
<td>2.2 ms</td>
<td>120</td>
<td>224</td>
<td>68</td>
<td>1</td>
</tr>
<tr>
<td>Lightweight McE (dec)</td>
<td>XC6VLX240T</td>
<td>13.4 ms</td>
<td>412</td>
<td>568</td>
<td>148</td>
<td>3</td>
</tr>
<tr>
<td>High-performance McE (enc)</td>
<td>XC6VLX240T</td>
<td>13.7 μs</td>
<td>14,429</td>
<td>9,201</td>
<td>2,924</td>
<td>0</td>
</tr>
<tr>
<td>High-performance McE (dec)</td>
<td>XC6VLX240T</td>
<td>125.4 μs</td>
<td>32,974</td>
<td>36,554</td>
<td>10,271</td>
<td>0</td>
</tr>
<tr>
<td>[Eisenbarth et al. 2009] (enc)</td>
<td>XC3S1400AN</td>
<td>2.2 ms</td>
<td>804</td>
<td>1,044</td>
<td>668</td>
<td>3</td>
</tr>
<tr>
<td>[Eisenbarth et al. 2009] (dec)</td>
<td>XC3S1400AN</td>
<td>21.6 ms</td>
<td>8,977</td>
<td>22,034</td>
<td>11,218</td>
<td>20</td>
</tr>
<tr>
<td>[Ghosh et al. 2012] (dec)</td>
<td>XC5VLX110T</td>
<td>0.5 ms</td>
<td>n/a</td>
<td>n/a</td>
<td>1,385</td>
<td>5</td>
</tr>
<tr>
<td>[Ghosh et al. 2012] (dec)</td>
<td>XC3S1400AN</td>
<td>1.02 ms</td>
<td>2,505</td>
<td>4,878</td>
<td>2,979</td>
<td>5</td>
</tr>
<tr>
<td>[Pöppelmann and Güneysu 2014] (enc)</td>
<td>XC6SLX9</td>
<td>0.9 ms</td>
<td>238</td>
<td>317</td>
<td>95</td>
<td>2^1</td>
</tr>
<tr>
<td>[Pöppelmann and Güneysu 2014] (dec)</td>
<td>XC6SLX9</td>
<td>0.4 ms</td>
<td>87</td>
<td>112</td>
<td>32</td>
<td>1^1</td>
</tr>
<tr>
<td>RSA [Helion 2010]</td>
<td>Spartan6-3</td>
<td>345 ms</td>
<td>n/a</td>
<td>n/a</td>
<td>135</td>
<td>1</td>
</tr>
</tbody>
</table>
32-bit ARM Microcontroller

**ARM-based 32-bit Microcontroller**
- STM32F407@168MHz
- 32-bit ARM Cortex-M4
- 1 Mbyte flash, 192 kbyte SRAM
- Crypto functions: TRNG, 3DES, AES, SHA-1/-256, HMAC co-processor
- Costs: roughly US$ 10

**AVR-based 8-bit Microcontroller**
- ATXmega128A1@32MHz
- 8-bit AVR Xmega Family
- 256 Kbyte flash, 8 Kbyte SRAM
- Crypto functions: DES, AES
- Costs: roughly US$ 10
Implementing Key Generation

- Memory is a scarce resource on microcontrollers
- Generate and store random sparse vectors of length 4801 with 45 bits set → store set bit locations only

Generating secret key $H = [H_0|H_1]$
- Generate first row of $H_1$, repeat if not invertible
- Generate first row of $H_0$
- Convert to sparse representation → 90 counters

Computing public key $G = [I|Q]$
- Compute $Q$ from first row of $H_1^{-1}$ and $H_0$
Implementing Encryption

- **Recall operation principle as for low-cost hardware**
  - All processes are based on 32-bit based operations
  - Set bits in message $m$ select rows of the public key $G$
  - Parse $m$ bit-by-bit, XOR current row of $G$ if bit is set

- **Error addition for encryption**
  - Use TRNG to provide random bits to add $t$ errors
  - Obtain individual error indices by rejection sampling from $\lceil \log_2 n \rceil = 14$ bit
Implementing Decryption

Recall syndrome computation; parity check matrix in sparse

- Parse ciphertext bit-by-bit
- XOR row of the secret key if corresponding ciphertext bit is set

Decoding iteration

- Count #bits that are set in the syndrome and current row of the parity-check matrix blocks → use 90 counters
- Compare #bits to decoding threshold
- Invert current ciphertext bit if #bits above threshold
- Add current row to syndrome
- Generate next row → increment counters (check overflows)
### Implementation Results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Platform</th>
<th>Cycles/Op</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>McE MDPC (keygen)</td>
<td>STM32F407</td>
<td>148,576,008</td>
<td>884 ms</td>
</tr>
<tr>
<td>McE MDPC (enc)</td>
<td>STM32F407</td>
<td>16,771,239</td>
<td>100 ms</td>
</tr>
<tr>
<td>McE MDPC (dec)</td>
<td>STM32F407</td>
<td>37,171,833</td>
<td>221 ms</td>
</tr>
<tr>
<td>McE MDPC (enc)</td>
<td>ATxmega256</td>
<td>26,767,463</td>
<td>836 ms</td>
</tr>
<tr>
<td>McE MDPC (dec)</td>
<td>ATxmega256</td>
<td>86,874,388</td>
<td>2,71 s</td>
</tr>
</tbody>
</table>

- 8-Bit AVR platform too slow for real-world deployment
  - Key generation excessive, decryption roughly 3 seconds
- 32-bit ARM is a suitable platform and provides built-in TRNG
- What about side-channel resistance?
Overview

Motivation
Background
Efficient Decoding Techniques
Implementing QC-MDPC McEliece
Side-Channel Attacks
Countermeasures
SCA Setup

- **Timing and Simple Power Analysis (μC only)**
- **Modify evaluation boards for both implementations**
  - 8-bit AVR ATxmega256 (Xplained-A1)
  - 32-bit ARM STM32F407
- Removed all capacitors and coils between VDD and GND
- Measurement resistor in the VDD path
- PicoScope 5203, 500MS/s, 250 MHz bandwidth
Message Recovery Attack

**Setting**: device encrypts a symmetric key under some PK

- Recall encryption

\[ x = mG + e \]

- Process:
  
  - \( m \) selects rows of \( G \)
  
  - Each row has length 4801
    \[ \rightarrow \] addition is memory-intensive operation

- Can we detect if a row is accumulated or not?
Message Recovery Attack – AVR

\[ m = 0x8F402.. \]
Message Recovery Attack – AVR

\[ m = 0x8F402.. \]
Message Recovery Attack – STM32

\[ m = 0x8F402.. \]
Message Recovery Attack – STM32

\[ m = 0x8F402.. \]
Secret Key Recovery Attack

**Setting**: device decrypts some ciphertext, known or chosen

Possible leakage of information: sparse representation

- Only one row of H is stored
- Cyclic shifts generate the following rows
- Sparse rows are stored using 2*45 counters
- Counters are incremented to generate next row
- If a counter exceeds $r$, it has to be reset to zero (carry)

Can we detect such overflows?
Secret Key Recovery Attack – AVR
Secret Key Recovery Attack – STM32
Secret Key Recovery Attack – STM32
Overview

Motivation
Background
Efficient Decoding Techniques
Implementing QC-MDPC McEliece
Side-Channel Attacks

Countermeasures
General Considerations

**Goal**: prevent timing and SPA attacks on AMR µC

- Runtime independent of secret data
- Program flow independent of secret data
- Critical code in ARM assembly
Protecting the Encryption

- **Dummy operations**: Always perform row addition, independent of message bits
- Can be detected in a fault-injection setting
- **Preferred choice**: Apply masking for constant runtime
- Generate with \((0 - m_i)\) either all-zero or all-one vector
- Compute redundant part \(r\) as

\[
r = r \oplus ((0 - m_i) \land g_i)
\]
Protecting the Decryption

Problem: Counter recovery revealed in sparse representation

Idea: store the full matrix
→ infeasible since $2 \times (4801 \times 4801)$ bit = 5.5 Mbyte

Alternative protection of the row rotation of $H$:

- Avoid using ordered counters
- Increment counter, compare to maximum value $r$
- If counter is smaller than $r$, the negative flag is set
- Load negative flag $N$ from status register
- $c_i = c_i \land (0 - N)$
Protecting the Decryption

There are more dependencies on secret data!

- Early aborts on comparison
  - Essential: Test syndrome for zero after every decoding iteration
  - Comparison leaks information about syndrome when aborting after first word \( \neq 0 \) is found
  - Remedy: compute OR of all 32-bit blocks of the syndrome and test the result for zero

- Early abort when decoding reaches \( s = 0 \)
  - Leaks number of decoding iterations
  - Remedy: test the syndrome after reaching max. #iterations
  - Decoding still works as before

- Further dependencies → see paper @PQCrypto14
## Implementation Results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Platform</th>
<th>Cycles/Op</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>McE MDPC (enc)</td>
<td>STM32F407</td>
<td>16,771,239</td>
<td>100 ms</td>
</tr>
<tr>
<td>McE MDPC (dec)</td>
<td>STM32F407</td>
<td>37,171,833</td>
<td>221 ms</td>
</tr>
<tr>
<td>McE MDPC (enc, ct)</td>
<td>STM32F407</td>
<td>7,018,493</td>
<td>42 ms</td>
</tr>
<tr>
<td>McE MDPC (dec, ct₁)</td>
<td>STM32F407</td>
<td>42,129,589</td>
<td>251 ms</td>
</tr>
<tr>
<td>McE MDPC (dec, ct₂)</td>
<td>STM32F407</td>
<td>85,571,555</td>
<td>509 ms</td>
</tr>
<tr>
<td>McE MDPC (dec, ct₃)</td>
<td>STM32F407</td>
<td>93,745,754</td>
<td>558 ms</td>
</tr>
<tr>
<td>McE MDPC (enc)</td>
<td>ATxmega256</td>
<td>26,767,463</td>
<td>836 ms</td>
</tr>
<tr>
<td>McE MDPC (dec)</td>
<td>ATxmega256</td>
<td>86,874,388</td>
<td>2.71 s</td>
</tr>
</tbody>
</table>

5.7 kByte (0.6%) Flash, 2.7 kByte (1.4%) SRAM, including keys

*ct1= early iteration abort; ct2= first syndrome comp. accelerated; ct3=constant time*
# Implementation Results

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Platform</th>
<th>Cycles/Op</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>McE MDPC (enc)</td>
<td>STM32F407</td>
<td>16,771,239</td>
<td>100 ms</td>
</tr>
<tr>
<td>McE MDPC (dec)</td>
<td>STM32F407</td>
<td>37,171,833</td>
<td>221 ms</td>
</tr>
<tr>
<td>McE MDPC (enc, ct)</td>
<td>STM32F407</td>
<td>7,018,493</td>
<td>42 ms</td>
</tr>
<tr>
<td>McE MDPC (dec, ct₁)</td>
<td>STM32F407</td>
<td>42,129,589</td>
<td>251 ms</td>
</tr>
<tr>
<td>McE MDPC (dec, ct₂)</td>
<td>STM32F407</td>
<td>85,571,555</td>
<td>509 ms</td>
</tr>
<tr>
<td>McE MDPC (dec, ct₃)</td>
<td>STM32F407</td>
<td>93,745,754</td>
<td>558 ms</td>
</tr>
<tr>
<td>McE MDPC (enc)</td>
<td>ATxmega256</td>
<td>26,767,463</td>
<td>836 ms</td>
</tr>
<tr>
<td>McE MDPC (dec)</td>
<td>ATxmega256</td>
<td>86,874,388</td>
<td>2,71 s</td>
</tr>
</tbody>
</table>

5.7 kByte (0.6%) Flash, 2.7 kByte (1.4%) SRAM, including keys

*ct₁ = early iteration abort; ct₂ = first syndrome comp. accelerated; ct₃ = constant time*
Conclusions and Outlook

- Efficient McEliece implementations with practical key sizes
  - High-performance and low cost FPGA design exploration
  - Microcontroller implementation for 8-bit AVR and 32-bit ARM devices

- Side-channel attacks on encryption and decryption
  - SPA attacks and countermeasures; DPA and fault injection is under investigation

- Papers and source code available at

- Future and on-going work:
  - Niederreiter encryption and key transport protocols
  - CS-MDPC codes
  - Countermeasures against DPA & fault-injection attacks
Tweaking Code-Based Cryptography for Embedded Systems

DIMACS Workshop on The Mathematics of Post-Quantum Cryptography

Tim Güneysu, Ingo von Maurich
Horst Görtz Institute for IT-Security, Ruhr-Universität Bochum, Germany

1/12/2015

Thank you!
References


