Part I: Introduction to Post Quantum Cryptography
Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017
Overview

• Goals
  – Provide a high-level introduction to Post-Quantum Cryptography (PQC)
  – Introduce selected implementation details (HW/SW) for some PQC classes (Focus: Encryption)
  – Highlight open challenges for PQC schemes

• Topics/Parts
  1. Introduction to PQC
  2. Hardware Implementation of PQC
  3. (Embedded) Software Implementation of PQC
Tutorial Outline – Part I

• Introduction
• Classes of Post-Quantum Cryptography (PQC)
  – Code-Based Cryptography
  – Lattice-Based Cryptography
  – Hash-Based Cryptography
• Lessons Learned
Long-Term Security in Embedded Devices

- For many today’s applications and systems long-term security is an essential requirement

- Many processing platforms have tight constraints with their computational resources

- 5-25 years

- 10-30 years

- 10 years

- > 15 years
Security of Practical Cryptographic Primitives

- Cryptosystems must combine security and efficiency

- Embedded devices mostly deploy standardized cryptography
  - Symmetric encryption: Advanced Encryption Standard
  - Asymmetric encryption: RSA (Factorization Problem), ElGamal or Elliptic Curve Cryptography (DLOG Problem)

- No „hard“ security guarantees are available for these real-world cryptosystems

- Common practise: Parameters chosen to resist best known (cryptanalytic) attack
Best Attacks on Cryptosystems

• **Attacks on symmetric cryptosystems**
  – Modern symmetric ciphers follow well-understood principles
  – For „solid“ ciphers best attack is exhaustive key search
  – Scaling key sizes to achieve long-term security

• **Attacks on asymmetric cryptosystems**
  – Virtually all asymmetric cryptosystems are based on factorization or DLOG problem
  – Best attacks with subexponential complexity
    • General Number Field Sieve (on RSA)
    • Index Calculus (on DLOG)
Key Size Recommendations

- Security parameters assuming *today’s* algorithmic knowledge and computing capabilities of a powerful attacker (e.g. NSA)

<table>
<thead>
<tr>
<th>Security (bits) (symmetric)</th>
<th>RSA</th>
<th>DLOG</th>
<th>EC</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>480</td>
<td>480</td>
<td>96</td>
</tr>
<tr>
<td>56</td>
<td>640</td>
<td>640</td>
<td>112</td>
</tr>
<tr>
<td>64</td>
<td>816</td>
<td>816</td>
<td>128</td>
</tr>
<tr>
<td>80</td>
<td>1248</td>
<td>1248</td>
<td>160</td>
</tr>
<tr>
<td>112</td>
<td>2432</td>
<td>2432</td>
<td>224</td>
</tr>
<tr>
<td>128</td>
<td>3248</td>
<td>3248</td>
<td>256</td>
</tr>
<tr>
<td>160</td>
<td>5312</td>
<td>5312</td>
<td>320</td>
</tr>
<tr>
<td>192</td>
<td>7936</td>
<td>7936</td>
<td>384</td>
</tr>
<tr>
<td>256</td>
<td>15424</td>
<td>15424</td>
<td>512</td>
</tr>
</tbody>
</table>

Source: ECRYPT II Yearly Key Size Report

- Short-term security (days to months)
- Mid-term security (years to decades)
- Long-term security (many years)
Public-Key Cryptography and Long-Term Security

Quantum Factorization of 143 on a Dipolar-Coupling Nuclear Magnetic Resonance System

Nanyang Xu,1 Jing Zhu,1,2 Dawei Lu,1 Xianyi Zhou,1 Xinhua Peng,1,† and Jiangfeng Du1,†
1Hefei National Laboratory for Physical Sciences at Microscale and Department of Modern Physics, University of Science and Technology of China, Hefei, Anhui, 230026, China
2Department of Physics, Shanghai Key Laboratory for Magnetic Resonance, East China Normal University, Shanghai 200062

Quantum Adiabatic Factoring

Quantum Adiabatic Factoring (QAF) is a quantum algorithm that can factor integers. It is a method for solving the problem of integer factorization using quantum computation. The algorithm works by starting with a Hamiltonian that is easy to factor and then evolving it into the Hamiltonian that corresponds to the integer we want to factor. The evolution is controlled by adjusting the parameters of the Hamiltonian in such a way that the ground state of the system will be the integer we want to factor. This is done by using a process called adiabatic quantum computation, which is a type of quantum computation that operates in a way that is analogous to the operation of a classical computer.

Quantum factorization of 56153 with only 4 qubits

Nikesh S. Dattani,1,2,* Nathaniel Bryans3,†
1Quantum Chemistry Laboratory, Kyoto University, 606-8502, Kyoto, Japan, 2Physical & Theoretical Chemistry Laboratory, Oxford University, OX1 3QZ, Oxford, UK, 3University of Calgary, T2N 4N1, Calgary, Canada. *dattani.nike@gmail.com, †nbryans1@gmail.com

Quantum factorization of 143

The NMR factorization of 143 in 2012 [1] began with the multiplication table:

| Table 1: Multiplication table for 11 × 13 = 143 in binary. |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
|                | \( p \)          | \( p^2 \)        | \( p^3 \)        | \( p^4 \)        | \( q \)          | \( q^2 \)        | \( q^3 \)        |
| \( p \)        | 1                | \( p_2 \)        | \( p_1 \)        | 1                | \( q_1 \)        | \( q_2 \)        | \( q_1 \)        |
| \( q \)        | 1                | \( q_2 \)        | \( q_1 \)        | 1                | \( p_2 \)        | \( p_1 \)        | 1               |
Alternatives for Public-Key Cryptography

- **Research** on alternative public-key cryptosystems is required
  ➔ NIST Call for PQC (Nov 30)

- Foundation on NP-hard problems?

- No polynomial-time attacks (such as Grover‘s/Shor‘s alg.) with quantum computers

- Efficiency in implementations comparable to currently employed cryptosystems
Post-Quantum Cryptography

• Definition
  – Class of cryptographic schemes based on the **classical computing paradigm**
  – Designed to provide security in the era of powerful quantum computers

• Important:
  – PQC ≠ quantum cryptography!
Post-Quantum Cryptography - Categories

- Five main branches of post-quantum crypto:
  - Code-based
  - Lattice-based
  - Hash-based
    - Multivariate-quadratic
    - Supersingular isogenies
- Should support public-key encryption and/or digital signatures
CHES History in PQC

- CHES has a long tradition on the implementation of PQC cryptosystems:
  - CHES 2004: Yang et al.: TTS on SmartCards
  - CHES 2008: Bogdanov et al.: MQ-Cryptosystems in HW
  - CHES 2009: Eisenbarth et al.: MicroEliece
  - CHES 2011: Session on Lattice-based attacks (3 papers)
  - CHES 2012: High-Performance McEliece+MQ+Lattices; GLS-Cryptosystem
  - CHES 2013: McBits + QC-MDPC McEliece Implementations
  - CHES 2014: RingLWE + Lattice-based Signature Implementations
  - CHES 2015: Session on Lattice crypto (2 papers), Homomorphic Encryption
  - CHES 2016: QcBits, Fault-Attack on BLISS signature scheme
  - CHES 2017: Tomorrow‘s session on PQC (3 papers)
Research Directions in PQC

- Propose novel robust and failure-proof cryptographic constructions
- Efficient constant-time implementation techniques and algorithmic tweaks
- Physical resistance against side-channel analysis and fault-injection attacks
- Improve cryptanalysis to foster confidence considering potential attacks
- Identify secure parameters against attacks from quantum-computers
- Compatible implementations for IoT devices, Internet infrastructures and Cloud services
Outline

• Introduction

• Classes of Post-Quantum Cryptography (PQC)
  – Code-Based Cryptography
  – Lattice-Based Cryptography
  – Hash-Based Cryptography

• Lessons Learned
Introduction to Code-based Cryptography

• Error-Correcting Codes are well-known in a large variety of applications

• Detection/Correction of errors in noisy channels by adding redundancy

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

• Observation:
  Some problems in code-based theory are NP-complete
  ➔ Possible foundation for Code-Based Cryptosystems (CBC)
Linear Codes and Cryptography

- **Linear codes**: Error correcting codes for which redundancy depends linearly on the information.
- **Generator and parity check matrices** for encoding and decoding.
- Rows of G form a basis for the code $C[n, k, d]$ of length $n$ with **dimension $k$** and **minimum distance $d$**.
- Matrices can be in systematic form minimizing time/storage.

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**
Syndrome Decoding Problem

- **Input given**
  - $H$: parity check matrix of size $(n - k) \cdot n$
  - $s$: vector of $\text{GF}(2^{n-k})$
  - $t$: positive integer (defined by error correction capability)

- **Problem:** Is there a vector $e$ in $\text{GF}(2^n)$ of weight $w(e) \leq t$ s.t. $H \cdot e^T = s$

- Syndrome decoding problem is **NP-complete**
  - E.R. BERLEKAMP, R.J. MCELIECE and H.C. VAN TILBORG
Niederreiter Encryption Scheme [1986]

**Key Generation**
Given a code $C[n, k, d]$ with parity check matrix $H$ and error correcting capability $t$

Private Key: $(S, H, P)$, where $S$ is a scrambling and $P$ a permutation matrix

Public Key: $\widehat{H} = S \cdot H \cdot P$

**Encryption**
Encode the message $m$ into an error vector $e \in R F_2^n$, $wt(e) \leq t$

$x \leftarrow \widehat{H} \cdot e^T$

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

$P m^T \leftarrow \Psi_H (S^{-1} \cdot x)$

Extract $m$ by transposing the computation $P^{-1} \cdot P m^T$. 
McEliece Encryption Scheme
[1978]

Key Generation
Given a code $C[n, k, d]$ with generator matrix $G$ and error correcting capability $t$

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

Public Key: $\hat{G} = S \cdot G \cdot P$

Encryption
Message $m \in F_2^{(n-r)}$, error vector $e \in_R F_2^n$, $wt(e) \leq t$

$x \leftarrow m\hat{G} + e$

Decryption
Let $\Psi_H$ be a $t$-error-correcting decoding algorithm.

$Sm \leftarrow \Psi_H(x \cdot P^{-1})$ removing the error $e$

Extract $m$ by computing $S^{-1} \cdot Sm$
Taxonomy of Code-Based Encryption Schemes

Code-based Encryption Schemes*

McEliece [M78]  Niederreiter [N86]

Generalized Reed-Solomon  Goppa Rank-Metric  Elliptic
Srivastava  Reed Muller  Concatenated LRPC/LDPC/MDPC

* This is a selection based on presenter’s choice.
Taxonomy of Code-Based Encryption Schemes

Code-based Encryption Schemes*

McEliece [M78]  Niederreiter [N86]

Generalized Reed-Solomon  Goppa  Reed-Muller
Rank-Metric  Elliptic  LRPC/LDPC/MDPC
Srivastava

* This is a selection based on presenter’s choice.
Key Aspects of Code-Based Cryptography

- Focus on encryption, signature schemes are inefficient
- Selection of the employed code is a highly critical issue
  - Properties of code determine key size, matrices are often large
  - Structures in codes reduce key size, but might enable attacks
  - Encoding is fast on most 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
Outline

• Introduction

• **Classes of Post-Quantum Cryptography (PQC)**
  – Code-Based Cryptography
  – Lattice-Based Cryptography
  – Hash-Based Cryptography

• Conclusions
Lattice-based Cryptography – Basics

• **Hard problem**: Shortest/Closest Vector Problem (SVP/CVP) in the worst case

• **Typically thought to be**
  – Unpractical but provably secure
  – Practical but without proof (GGH/NTRU)
  – **Lately**: Ideal lattices can potentially combine both

• **More constructions feasible beyond classical PKC**: hash functions, PRFs, identity-based encryption, homomorphic encryption
Learning with Errors

Solving of a system of linear equations

Blue is given; Find (learn) red $\rightarrow$ Solve linear system

Use Gaussian elimination

\[
\begin{array}{cccc}
4 & 1 & 11 & 10 \\
5 & 5 & 9 & 53 \\
3 & 9 & 0 & 10 \\
1 & 3 & 3 & 2 \\
12 & 7 & 3 & 4 \\
6 & 5 & 11 & 4 \\
3 & 3 & 5 & 0 \\
\end{array}
\times
\begin{array}{c}
6 \\
9 \\
11 \\
11 \\
\end{array}
= 
\begin{array}{c}
4 \\
8 \\
1 \\
10 \\
4 \\
12 \\
9 \\
\end{array}
\]

secret

\[
\begin{array}{c}
\mathbb{Z}_{13}^{7\times4} \\
\mathbb{Z}_{13}^{4\times1} \\
\mathbb{Z}_{13}^{7\times1} \\
\end{array}
\]
Learning with Errors

Solving of a system of linear equations

<table>
<thead>
<tr>
<th>random $\mathbb{Z}_{13}^{7 \times 4}$</th>
<th>secret $\mathbb{Z}_{13}^{4 \times 1}$</th>
<th>small noise $\mathbb{Z}_{13}^{7 \times 1}$</th>
<th>looks random $\mathbb{Z}_{13}^{7 \times 1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>11</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>53</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Blue is given; Find red $\Rightarrow$ Learning with Errors (LWE) Problem
Key Aspects of Lattice-based Systems

• Encryption and signature systems are both feasible (and secure)
  – Significant ciphertext expansion for (R-)LWE encryption
  – Decryption error probability with (R-)LWE encryption

• Random Sampling not only from uniform but also from Discrete Gaussian distributions (not a trivial task!)

• Most operations are efficient and parallizable
  – (Ideal lattices) Make use of FFT for polynomial multiplication
  – (Standard lattices) Matrix-vector arithmetic

• Reasonably large public and private keys
  – Given for encryption/signatures constructions
  – Unclear for advanced services such as functional encryption (e.g., FHE)
Outline

• Introduction

• **Classes of Post-Quantum Cryptography (PQC)**
  – Code-Based Cryptography
  – Lattice-Based Cryptography
  – Hash-Based Cryptography

• Lessons Learned
Hash-based Cryptography: Lamport-Diffie One-Time Signatures (LD-OTS, 1979)

- **Definition**: Given a security parameter $n$, the set of $n$-bit vectors $U_n = \{0,1\}^n$ and a one-way function $h : U_n \rightarrow U_n$

- **Secret key**: Generate $2n \times n$-bit vector $X = (x_{(0,0)}, x_{(0,1)}, x_{(1,0)}, x_{(1,1)}, \ldots, x_{(n-1,1)})$

- **Public Key**: Compute $Y = (y_{(0,0)}, \ldots, y_{(n-1,1)}) \ \forall y_{i,j} = f(x_{i,j})$

- **Publish** public key $Y$
Hash-based Cryptography: Lamport-Diffie One-Time Signatures (LD-OTS, 1979)

- **Definition**: Given a published public key $Y$ and an $n$-bit message $M = (m_0, \ldots, m_{n-1})$ to sign.

- **Sign**: Generate signature $\sigma = (x_{(0,m_0)}, \ldots, x_{(n-1,m_{n-1})})$ by revealing corresponding $x_{(i,m_i)}$ secret bits.

- **Verify**: Check that for $f(\sigma_i) = y_{(i,m_i)} \forall i = [0, n - 1]$.
Extension for Multiple Use: Merkle’s Signature Scheme

- **Idea by R. Merkle [1979]:** reduces the validity of many OTS verification keys to a single verification key using a binary tree.

- **Properties and Requirements**
  - Max. signature count determined by height $H$ of tree (fixed at setup).
  - Needs to keep track of already used signatures in the tree → stateful signature scheme.
  - Can be used with any one-time signature scheme and (collision-resistant) cryptographic hash function.

\[
\[
V_0[1] = g(Y_1)\]
\[
V_0[2] = g(Y_2)\]
\[
V_0[3] = g(Y_0)\]
\[
V_0[4] = g(Y_4)\]
\[
V_0[5] = g(Y_5)\]
\[
V_0[6] = g(Y_6)\]
\[
V_0[7] = g(Y_7)\]
Merkle Signature Scheme

Principle

- Let \( g: \{0,1\}^* \rightarrow \{0,1\}^n \) be a hash function with security parameter \( n \)
- Fix height \( H \) and generate \( 2^H \) LD-OTS key pairs \((X_i, Y_i)\) with \( 0 \leq i < 2^H \)
- **Notation**: \( V_i[j] \) with \( 0 \leq i \leq H \) and \( 0 \leq j < 2^{H-i} \)

**Example**: \( H = 3 \)

**Computation rule** for inner nodes: \( V_i[j] = g(V_{i-1}[2j] || V_{i-1}[2j+1]) \) with \( 0 < i \leq H \) and \( 0 \leq j < 2^i \)
Key Aspects of Hash-based Cryptographic Systems

• Only signature schemes available, no encryption
• Moderate requirements for implementations
  – Second preimage (older schemes: collision) resistant hash function
  – Pseudorandom functions for OTS (XMSS)
• Hard limitation on the number of signatures per tree
  – Height of the tree determines max. # of signatures (issue with DoS attacks for real-world systems)
  – Requires track record of signatures already used (critical in untrusted environments!)
  – Increasing tree height increases memory requirements and computational complexity
Outline

• Introduction

• Classes of Post-Quantum Cryptography (PQC)
  – Code-Based Cryptography
  – Lattice-Based Cryptography
  – Hash-Based Cryptography

• Lessons Learned
Lessons Learned

• **Post-Quantum Cryptography essential for long-term security**
  – Code-based encryption schemes are the most mature candidates
  – Digital signatures from hash-based cryptography with high confidence respect to security and under standardization
  – Lattice-based cryptography has high potential and extremely high versatility

• **Next topics in this tutorial** (selection due to time constraints)
  – Efficient implementation strategies for Code-Based Cryptosystems
  – Efficient implementation of Lattice-Based Cryptosystems
Part I: Introduction to Post Quantum Cryptography

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017

Thank you! Questions?
Part II: Hardware Architectures for Post Quantum Cryptography

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017

including slides by Ingo von Maurich and Thomas Pöppelmann
Tutorial Outline – Part II

Code-based Cryptography
Efficient Code-based Implementations
Lattice-based Cryptography
Efficient Lattice-based Implementations
Lessons Learned
Recall: 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 $e \in \mathbb{F}_2^n$, $\text{wt}(e) \leq t$

$x \leftarrow mG' + e$

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 (Binary 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 | Size $K_{pub}$ in KBits | Size $K_{sec}$ $(g(z) | L | M^{-1})$ 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                   | $(1.45 | 84 | 2,183)$                    |
Code-based Cryptography for Embedded Devices

- 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
Quasi-Cyclic Moderate Density Check Codes (QC-MDPC)

- \( t \)-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 \( G = (I_k|Q) \)
   \[
   Q = \begin{pmatrix}
   (H_{n_0-1}^{-1} \times H_0)^T \\
   (H_{n_0-1}^{-1} \times H_1)^T \\
   \vdots \\
   (H_{n_0-1}^{-1} \times H_{n_0-2})^T
   \end{pmatrix}
   \]
Background on QC-MDPC Codes

Parity check matrix $H$

$n_0 = 2$

Generator matrix $G$
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]

$n_0 = 2, n = 9602, r = 4801, w = 90, t = 84$
Tutorial Outline – Part II

Code-based Cryptography

**Efficient Code-based Implementations**

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned
Hardware Implementation of Building Blocks for McEliece/Niederreiter

- **Two Operations**
  - **Encryption/Encoding:**
    - Matrix-vector multiplication (with large matrices, either to be stored or to be generated on-the-fly);
    - TRNG for error generation
  - **Decryption/Decoding:**
    - Code-specific syndrome decoding; hard-decision decoding with simple (bitwise) operations preferred
    - Inverse-matrix-vector multiplication
Efficient Decoding of MDPC Codes

Decoders for LDPC/MDPC codes: bit flipping and belief propagation

“Bit-Flipping” Decoder

1. Compute syndrome $s$ of the ciphertext
2. Count unsatisfied parity-check-equations $\#_{upc}$ 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]
- $b = \max_{upc}$ [HP03]
- $b = \max_{upc} - \delta$ [MTSB13]
FPGA Low-Resource Encryption

Target: Xilinx Spartan-6 FPGA

Scheme: 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
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>21</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>11</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>
Tutorial Outline – Part II

Code-based Cryptography
Efficient Code-based Implementations

Lattice-based Cryptography
Efficient Lattice-based Implementations
Lessons Learned
Lattice-Based Cryptography

- Recall: Benefits of Lattice-Based Cryptography
  - We can get *signatures* and *public key encryption* from lattices and also *more advanced services* (IBE, FHE)
  - A lot of development on theory side; schemes are improving
  - Implementation of lattice-based cryptography is a young field; only done for a few years (except maybe for NTRU)
To be Ideal or not Ideal?

Two important lines of research: random lattices and ideal lattices

- Major impact on implementation (theory not that much)
- Security for random lattices is better understood (ideal lattices are more structured)

- **Random Lattices**
  - Operations on large matrices (e.g., 532x840)
  - Mostly matrix-vector multiplication modulo $q < 2^{32}$
  - Large public keys (e.g., 532x840 matrix)

- **Ideal Lattices**
  - Operations on polynomials with 256 or 512 coefficients
  - Mostly polynomial multiplication modulo $q < 2^{32}$
  - Public keys are one (or two) polynomials with 256 or 512 coefficients
Learning with Errors

Solving of a system of linear equations

_secret_

\[
\mathbb{Z}_{13}^{7 \times 4} \times \mathbb{Z}_{13}^{4 \times 1} = \mathbb{Z}_{13}^{7 \times 1}
\]

<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
<th>11</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5</td>
<td>9</td>
<td>53</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>11</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>9</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Use Gaussian elimination

Blue is given; Find (learn) red ➔ Solve linear system
Learning with Errors

Solving of a system of linear equations

<table>
<thead>
<tr>
<th>random</th>
<th>secret</th>
<th>small noise</th>
<th>looks random</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathbb{Z}_{13}^{7\times4}$</td>
<td>$\mathbb{Z}_{13}^{4\times1}$</td>
<td>$\mathbb{Z}_{13}^{7\times1}$</td>
<td>$\mathbb{Z}_{13}^{7\times1}$</td>
</tr>
</tbody>
</table>

\[
\begin{array}{cccc}
4 & 1 & 11 & 10 \\
5 & 5 & 9 & 53 \\
6 & 5 & 11 & 4 \\
3 & 3 & 5 & 0 \\
\end{array}
\times
\begin{array}{c}
\text{red} \\
\text{red} \\
\text{red} \\
\text{red} \\
\text{red} \\
\text{red} \\
\text{red} \\
\text{red} \\
\text{red} \\
\end{array}
\]

\[
\begin{array}{c}
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\text{small noise} \\
\end{array}
\]

\[
\begin{array}{c}
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\text{looks random} \\
\end{array}
\]

Blue is given; Find red $\Rightarrow$ Learning with errors
(Ring) Learning with Errors

From learning with errors to ring-learning with errors

\[ \mathbb{Z}_{13}^{7 \times 4} \]

- Shift first line on every line
- Use rule that we negate \( x \) in case of wrap around (e.g., \( 10 \Rightarrow -10 \equiv 3 \mod 13 \))

Only one line has to be stored
Ring Learning with Errors: Principle

- **Ideal lattices** correspond to ideals in the ring $R = \frac{\mathbb{Z}_q[x]}{(x^n+1)}$

- **Ring Learning With Errors (RLWE)** sample is: $t = as + e \in R$ for uniform $a \in R$ and small discrete Gaussian distributed $s, e \leftarrow D_\sigma$
  - Search-RLWE: Find $s$ when given $t$ and $a$
  - Decision-RLWE: Distinguish $t$ from uniform when given $t$ and $a$
Example:

Polynomial Addition in $R = \frac{\mathbb{Z}_q[x]}{(x^n+1)}$

- Assume ring $R = \frac{\mathbb{Z}_q[x]}{(x^n+1)}$
- Assume parameters $q = 5$ and $n = 4$
- $v = 4x^3 + 2x^2 + 0x^1 + 1 \quad = (4,2,0,1)$
- $k = 2x^3 + 1x^2 + 4x^1 + 0 \quad = (2,1,4,0)$
- $s = v + k = (4 + 2 \mod 5,2 + 1,4,1) = (1,3,4,1)$
Example:

Polynomial Multiplication in $R = \frac{Z_q[x]}{\langle x^{n+1} \rangle}$

- $k = (2, 1, 4, 0)$
- $s = (1, 3, 4, 1)$

Task: $z = s \ast k = (3, 0, 2, 0)$
Discrete Gaussian Distribution

- $D_\sigma$ is defined by assigning weight proportional to
  \[ \rho_\sigma(x) = \exp\left(-\frac{x^2}{2\sigma^2}\right) \]

<table>
<thead>
<tr>
<th>Uniform</th>
<th>-1501</th>
<th>1020</th>
<th>502</th>
<th>...</th>
<th>-1900</th>
<th>572</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gaussian</td>
<td>-1</td>
<td>4</td>
<td>-8</td>
<td>...</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Remark on Arithmetic of $x$-distributed values:
- Uniform $\ast$ Gaussian = Uniform
- Gaussian $\ast$ Gaussian = larger Gaussian

\[ R = \frac{Z_4093[x]}{\langle x^{256} + 1 \rangle} \]
Gaussian Sampling: Options

- Rejection Sampling
- Bernoulli Sampling
- Cumulative Distribution Table (CDT) Sampling
- Knuth-Yao Sampling

[DDLL14] Lattice Signatures and Bimodal Gaussians, Léo Ducas and Alain Durmus and Tancrède Lepoint and Vadim Lyubashevsky, CRYPTO ’13
Ring-LWE Encryption Scheme
[LP11/LPR10]

**Gen**: Choose \( a \leftarrow R \) and \( r_1, r_2 \leftarrow D_\sigma \); 
**pk**: \( p = r_1 - a \cdot r_2 \in R \); 
**sk**: \( r_2 \)

**Enc**(\( a, p, m \in \{0,1\}^n \)): \( e_1, e_2, e_3 \leftarrow D_\sigma \). \( \overline{m} = encode(m) \). Ciphertext:
\[
\begin{align*}
    c_1 &= a \cdot e_1 + e_2, \\
    c_2 &= p \cdot e_1 + e_3 + \overline{m}
\end{align*}
\]

**Dec**(\( c = [c_1, c_2], r_2 \)): Output
\[
\begin{align*}
    decode(c_1 \cdot r_2 + c_2)
\end{align*}
\]

**Correctness**: \( c_1 r_2 + c_2 = (ae_1 + e_2)r_2 + pe_1 + e_3 + \overline{m} \)
\[
\begin{align*}
    &= r_2 ae_1 + r_2 e_2 + r_1 e_1 - r_2 ae_1 + e_3 + \overline{m} = \overline{m} + r_2 e_2 + r_1 e_1 + e_3
\end{align*}
\]
Ring-LWE Encryption: Parameters

Error correction

- **Encode**($m$)
  - Return $m \cdot q/2$

- **Decode**($x$)
  - If $(1/4q < x < 3/4q)$
    - Return 1
  - Else return 0
Ring-LWE Encryption: Parameters

- **Message and ciphertext:**
  - Message space: \( n \) bits
  - Expansion \( 2 \cdot \log_2 (q) \)
  - Two large polynomials \( (c_1, c_2) \)
- **Public key:** one or two large polynomials \( (a, p) \)
- **Secret key:** small polynomial \( r_2 \)

| Parameter sets            | \( n \) | \( p \)   | \( \sigma \) | \( |c_1, c_2| \) | \( |sk| \) | \( |pk| \) | Security   |
|---------------------------|--------|----------|--------------|----------------|--------|--------|-----------|
| (256, 4093, 8.35 [LP11])  | 256    | 4093     | \~4.5        | 6,144          | 1,792  | 6,144  | \~106 bits |
| (256, 7681, 11.32) [GFSBH12] | 256    | 7681     | \~4.8        | 6,656          | 1,792  | 6,656  | \~106 bits |
| (512, 12289, 12.18) [GFSBH12] | 512    | 12289    | \~4.9        | 14,336         | 3,584  | 14,336 | \~256 bits |
Tutorial Outline – Part II

Code-based Cryptography
Efficient Code-based Implementations
Lattice-based Cryptography
**Efficient Lattice-based Implementations**
Lessons Learned
Two main components

- Polynomial multiplier for \( n = \{256, 512, 1024\} \) over specific rings with coefficients with less than \( \log_2(q) < 24 \) bits
- Discrete Gaussian sampler with precisely defined precision \( \sigma \)
Hardware Implementation:
Low-Cost Design for Xilinx Spartan-6

• Row-wise polynomial multiplication ($ae_1/p e_1$)
  – Simple address generation
  – Sample coefficient of $e_1$, add row of $c_1$ then add row of $c_2$, add coefficient of $e_2$ and $e_3$

• Key and ciphertext are stored in block memory

• DSP block for arithmetic ($q \times q$-bit multiplier)
Hardware Implementation: Low Area

Post-place-and-route performance on a Spartan-6 LX9 FPGA.

<table>
<thead>
<tr>
<th>Operation</th>
<th>LUT/LUTM/FF/Slice</th>
<th>BRAM9/DSP</th>
<th>MHz</th>
<th>Cycles</th>
<th>OP/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLWEenc</td>
<td>(256, 4093, 8.35)</td>
<td>360/36/290/114</td>
<td>2/1</td>
<td>128</td>
<td>avg. 136986</td>
</tr>
<tr>
<td>RLWEenc</td>
<td>(256, 4096, 8.25)</td>
<td>282/35/238/95</td>
<td>2/1</td>
<td>144</td>
<td>avg. 136212</td>
</tr>
<tr>
<td>RLWEenc</td>
<td>(256, 4093, 8.35)</td>
<td>162/18/136/51</td>
<td>1/1</td>
<td>179</td>
<td>avg. 66304</td>
</tr>
<tr>
<td>RLWEenc</td>
<td>(256, 4096, 8.35)</td>
<td>94/18/87/32</td>
<td>1/1</td>
<td>189</td>
<td>avg. 66338</td>
</tr>
</tbody>
</table>

- Usage of $q = 4096$ leads to area improvement and higher clock frequency
- Performance is still very good
- Area consumption is low, especially for decryption

Area savings by power of two modulus
Ring-LWE: Can we do better?

- Schoolbook polynomial multiplication is simple and independent of parameters
- Performance is reasonable but can still be improved
- Remember: according to schoolbook multiplication, we need $n^2$ multiplications modulo $q$ for one polynomial multiplication
  - $128^2 = 16384$
  - $256^2 = 65536$
  - $512^2 = 262144$
  - $1024^2 = 1048576$

Can we do better?
Optimization: Polynomial Multiplication based on NTT

• Include algorithmic tweaks for fast polynomial multiplication
• The Number Theoretic Transform (NTT) is a discrete Fourier transform (DFT) defined over a finite field or ring. For a given primitive $n$-th root of unity $\omega$ the NTT is defined as:

  – Forward transformation: NTT
    \[ A[i] = \sum_{j=0}^{n-1} a[j] \omega^{ij}, i = 0, 1, \ldots, n \]
  – Inverse transformation: INTT
    \[ a[i] = n^{-1} \sum_{j=0}^{n-1} A[j] \omega^{-ij}, i = 0, 1, \ldots, n \]

• NTT exists if $q$ is a prime, $n$ a power of two and if $q \equiv 1 \mod 2n$
• **Example**: Ring-LWE encryption: $7681 \mod 2 \cdot 256 = 1$
NTT for Lattice Cryptography: Convolution Theorem

- With the convolution theorem we can basically multiply two vectors/polynomials with the help of the NTT
  - \( c = \text{INTT} \left( \text{NTT}(a) \circ \text{NTT}(b) \right) \)
  - Efficient algorithms are known for bi-direction conversion

- **Negative Wrapped Convolution:**
  - Polynomial multiplication in \( Z_q[x]/\langle x^n + 1 \rangle \)
  - Runtime \( O(n \log n) \)
  - No appending of zeros required (as for regular convolution)
  - Implicit polynomial reduction by \( x^n + 1 \)
Efficient Computation of the NTT (Cooley-Tukey)

1: func NTT(a ∈ ℜ)
2: a ← BitRev(a)
3: N ← n
4: m ← 2
5: while m ≤ N do
6: s ← 0
7: while s < N do
8: for i to m/2 − 1 do
9: N ← i · n/m
10: k ← s + i
11: l ← s + i + m/2
12: c ← a[k]
13: d ← a[l]
14: a[k] ← c + ωN d mod q
15: a[l] ← c − ωN d mod q
16: end for
17: s ← s + m
18: end while
19: m ← m · 2
20: end while
21: Return a
22: end func

- Bitreversal required (NTT_{no→bo})
- Precomputation of powers of ω possible
- Arithmetic is basically multiplication and reduction modulo q (\(\frac{n}{2} \log_2(n)\) times)
- Further optimizations still possible
## Ring-LWE Encryption on FPGA

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Device</th>
<th>Resources (LUT/FF/BRAM/DSP)</th>
<th>OP</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLWEenc (area, our work) ((n = 256, q = 7681))</td>
<td>S6LX9</td>
<td>396/290/2/1</td>
<td>Enc</td>
<td>1.07 ms</td>
</tr>
<tr>
<td></td>
<td>S6LX9</td>
<td>180/136/1/1</td>
<td>Dec</td>
<td>370.00 µs</td>
</tr>
<tr>
<td>RLWEenc [RVM+14] ((n = 256, q = 7681))</td>
<td>V6LX75T</td>
<td>1349/860/2/1</td>
<td>Enc</td>
<td>20.10 µs</td>
</tr>
<tr>
<td></td>
<td>V6LX75T</td>
<td>-</td>
<td>Dec</td>
<td>9.10 µs</td>
</tr>
<tr>
<td>RLWEenc [GFS+12] ((n = 256, q = 7681))</td>
<td>V6LX240T</td>
<td>146 718/82 463/-/-</td>
<td>Gen</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>V6LX240T</td>
<td>298 016/143 396/-/-</td>
<td>Enc</td>
<td>8.05 µs</td>
</tr>
<tr>
<td></td>
<td>V6LX240T</td>
<td>124 158/65 174/-/-</td>
<td>Dec</td>
<td>8.10 µs</td>
</tr>
<tr>
<td>Niederreiter [HG12] ((80\text{-bit security}))</td>
<td>V6LX240T</td>
<td>888/875/17/-</td>
<td>Enc</td>
<td>0.66 µs</td>
</tr>
<tr>
<td></td>
<td>V6LX240T</td>
<td>9 409/12 861/9/-</td>
<td>Dec</td>
<td>57.78 µs</td>
</tr>
<tr>
<td>NTRU [KY09]</td>
<td>XCV1600E</td>
<td>27 292/5 160</td>
<td>Enc</td>
<td>1.54 µs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-</td>
<td>Dec</td>
<td>1.41 µs</td>
</tr>
<tr>
<td>1024-bit mod. Exp. [Suz07]</td>
<td>XC4VFX12</td>
<td>3 937 slice/17 DSP</td>
<td>-</td>
<td>1.71 ms</td>
</tr>
<tr>
<td>ECC-P224 [GP08]</td>
<td>XC4VFX12</td>
<td>1 825/1 892/11/26</td>
<td>-</td>
<td>365.10 µs</td>
</tr>
<tr>
<td>ECC-B233 [RRM12]</td>
<td>XC5VLX85T</td>
<td>18 097 LUT/5 644 slice</td>
<td>-</td>
<td>12.30 µs</td>
</tr>
</tbody>
</table>

NTT is very fast but still quite small.

Lots of improvement since [GFS+12].
Tutorial Outline – Part II

Code-based Cryptography
Efficient Code-based Implementations
Lattice-based Cryptography
Efficient Lattice-based Implementations

Lessons Learned
Lessons Learned

- **Efficient McEliece implementations with practical key sizes**
  - QC-MDPC codes are an efficient alternative to binary Goppa codes
  - Note: consider attacks on decryption failure rate (ASIACRYPT 2016)
  - Low-cost FPGA implementation practical for key agreement scheme (in prep)

- **Efficient R-LWE encryption are extremely efficient**
  - R-LWE (and variants) also allow signature + advanced schemes
  - FPGA implementations more efficient than RSA, en par with ECC

- **Papers and source code available at**
  
  http://www.seceng.rub.de/research/projects/pqc/

- **For more papers and codes, see project websites of**

  - Horizon 2020
  - PQCRYPTO ICT-645622
  - SAFEcrypto ICT-644729
Part II: Hardware Architectures for Post Quantum Cryptography

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017

Thank you! Questions?
Part III: Post Quantum Cryptography in Embedded Software

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017

including slides by Ingo von Maurich and Thomas Pöppelmann
Tutorial Outline – Part III

Code-based Cryptography
Efficient Code-based Implementations
Lattice-based Cryptography
Efficient Lattice-based Implementations
Lessons Learned
Recall: 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 \(e \in \mathbb{R} \mathbb{F}_2^n\), \(\text{wt}(e) \leq t\)
\(x \leftarrow mG' + e\)

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 \(m \cdot S \cdot S^{-1}\)
(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]
$n_0 = 2, n = 9602, r = 4801, w = 90, t = 84$
Tutorial Outline – Part III

Code-based Cryptography

**Efficient Code-based Implementations**

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned
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 (Plain) 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 (Plain) 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
- Improved QcBits software for Cortex-M4 by Chou (CHES 2016)
Further Implementation Remarks and Requirements

• **CCA2-Security for McEliece Encryption:**
  – Additional conversion (e.g., via Fujisaki-Okamoto, includes the necessity for hash-function and re-encryption)

• **Side-Channel Attacks:**
  – Masking schemes (SCA) for McEliece by Eisenbarth et al. [SAC15], does not include CCA2 security

• **Decryption Failure Rate Attacks:**
  – Guo et al [ASIACRYPT16] identifies correlation between decoding failures in iterative decoders (bit flipping decoding)
Tutorial Outline – Part III

Code-based Cryptography
Efficient Code-based Implementations

Lattice-based Cryptography
Efficient Lattice-based Implementations
Lessons Learned
Ring-LWE Encryption Scheme [LP11/LPR10]

**Gen**: Choose $a \leftarrow R$ and $r_1, r_2 \leftarrow D_\sigma$; $pk$: $p = r_1 - a \cdot r_2 \in R$; $sk$: $r_2$

$\text{Enc}(a, p, m \in \{0,1\}^n)$: $e_1, e_2, e_3 \leftarrow D_\sigma$. $\bar{m} = \text{encode}(m)$. Ciphertext:

$[c_1 = a \cdot e_1 + e_2, c_2 = p \cdot e_1 + e_3 + \bar{m}]$

$\text{Dec}(c = [c_1, c_2], r_2)$: Output $\text{decode}(c_1 \cdot r_2 + c_2)$

**Correctness**: $c_1 r_2 + c_2 = (a e_1 + e_2) r_2 + p e_1 + e_3 + \bar{m}$

$= r_2 a e_1 + r_2 e_2 + r_1 e_1 - r_2 a e_1 + e_3 + \bar{m} = \bar{m} + r_2 e_2 + r_1 e_1 + e_3$
## Ring-LWE Encryption: Parameters

| Parameter sets                    | $n$  | $p$   | $\sigma$ | $|c_1, c_2|$ | $|sk|$  | $|pk|$  | security   |
|-----------------------------------|------|-------|----------|--------------|--------|--------|------------|
| (256, 4093, 8.35 [LP11])          | 256  | 4093  | ~4.5     | 6,144        | 1,792  | 6,144  | ~106 bits  |
| (256, 7681, 11.32) [GFSBH12]      | 256  | 7681  | ~4.8     | 6,656        | 1,792  | 6,656  | ~106 bits  |
| (512, 12289, 12.18) [GFSBH12]     | 512  | 12289 | ~4.9     | 14,336       | 3,584  | 14,336 | ~256 bits  |

- **Message and ciphertext:**
  - Message space: $n$ bits
  - Expansion $2 \cdot \log_2(q)$
  - Two large polynomials $(c_1, c_2)$

- **Public key:** one or two large polynomials $(a, p)$

- **Secret key:** small polynomial $(r_2)$
Tutorial Outline – Part III

Code-based Cryptography
Efficient Code-based Implementations

Lattice-based Cryptography
Efficient Lattice-based Implementations

Lessons Learned
void encrypt(poly a, poly p, unsigned char * plaintext, poly c1, poly c2) {
    int i, j; poly e1, e2, e3;
    gauss_poly(e1); gauss_poly(e2); gauss_poly(e3);
    poly_init(c1, 0, n); // init with 0
    poly_init(c2, 0, n); // init with 0

    for(i = 0; i < n; i++) { // multiplication loops
        for(j = 0; j < n; j++) {
            c1[(i + j) % n] = modq(c1[(i + j) % n] + (a[i] * e1[j] * (i + j >= n ? -1 : 1)));
            c2[(i + j) % n] = modq(c2[(i + j) % n] + (p[i] * e1[j] * (i + j >= n ? -1 : 1)));
        }
        c1[i] = modq(c1[i] + e2[i]);
        c2[i] = (plaintext[i >> 3] & (1 << (i % 8))) ? modq(c2[i] + e3[i] + q/2) : modq(c2[i] + e3[i]);
    }
}
Software Implementation
Main Functions for R-LWE

• Two main components
  – Polynomial multiplier for $n = \{256, 512, 1024\}$ over specific rings with coefficients with less than $\log_2(q) < 24$ bits
  – Discrete Gaussian sampler with precisely defined precision $\sigma$ and tail cut $\tau$
Intermediate Results

- Implementation of RLWE-Encryption on the AVR 8-bit ATxmega processor running at 32 MHz

<table>
<thead>
<tr>
<th>Operation</th>
<th>(n=192, q=4093)</th>
<th>(n=256, q=4093)</th>
<th>(n=320, q=4093)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle counts and stack usage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>encrypt</td>
<td>7,294,976 (446 bytes)</td>
<td>12,289,025 (573 bytes)</td>
<td>21,684,224 (705 bytes)</td>
</tr>
<tr>
<td>decrypt</td>
<td>3,205,121 (444 bytes)</td>
<td>5,491,712 (571 bytes)</td>
<td>9,798,655 (703 bytes)</td>
</tr>
<tr>
<td>SchoolMul</td>
<td>3,264,511</td>
<td>5,571,584</td>
<td>9,967,615</td>
</tr>
<tr>
<td>SampleGauss$_\sigma$</td>
<td>55,296</td>
<td>73,727</td>
<td>89,088</td>
</tr>
<tr>
<td>AddEncode</td>
<td>12,773</td>
<td>16,915</td>
<td>20,707</td>
</tr>
<tr>
<td>Decode</td>
<td>3,331</td>
<td>4,419</td>
<td>5,507</td>
</tr>
<tr>
<td>Static memory consumption in bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash</td>
<td>4,648</td>
<td>5,032</td>
<td>5,418</td>
</tr>
<tr>
<td>RAM</td>
<td>824</td>
<td>1,088</td>
<td>1,352</td>
</tr>
</tbody>
</table>

- Schoolbook multiplication (SchoolMul)
- Encryption is two multiplications and decryption one
Recall Improvement: Polynomial Multiplication with NTT

• Number Theoretic Transform (NTT) is a discrete Fourier transform (DFT) defined over a finite field or ring. For a given primitive $n$-th root of unity $\omega$ the NTT is defined as:

  – Forward transformation: NTT
    - $A[i] = \sum_{j=0}^{n-1} a[j] \omega^{ij}, i = 0, 1, \ldots, n$

  – Inverse transformation: INTT
    - $a[i] = n^{-1} \sum_{j=0}^{n-1} A[j] \omega^{-ij}, i = 0, 1, \ldots, n$

• NTT exists if $q$ is a prime, $n$ a power of two and if $q \equiv 1 \mod 2n$
Efficient Computation of the NTT (Textbook)

```python
1: func NTT(a ∈ ℜ)
2:     a ← BitRev(a)
3:     N ← n
4:     m ← 2
5:     while m ≤ N do
6:         s ← 0
7:         while s < N do
8:             for i to m/2 − 1 do
9:                 N ← i · n/m
10:                k ← s + i
11:                l ← s + i + m/2
12:                c ← a[k]
13:                d ← a[l]
14:                a[k] ← c + ω^N d mod q
15:                a[l] ← c − ω^N d mod q
16:             end for
17:         s ← s + m
18:     end while
19:     m ← m · 2
20: end while
21: Return a
```

- Bitreversal required ($\text{NTT}_{n_0 \rightarrow b_0}$)
- Precomputation of powers of $\omega$ possible
- Arithmetic is basically multiplication and reduction modulo $q \left( \frac{n}{2} \log_2(n) \right)$ times
Optimization of NTT Computation

Removal of expensive “helper” functions

• **Problem**: Permutation (Bitrev) of polynomial is expensive
  – “Standard” $\text{NTT}_{bo\rightarrow no}$ requires bitreversed input and produces naturally ordered output
  – Bitreversal before each forward or inverse NTT

• **Solution**: NTT algorithm can be written as
  – Natural to bitreversed for forward: $\text{NTT}_{no\rightarrow bo}$
  – Bitreversed to natural for inverse: $\text{INTT}_{bo\rightarrow no}$
  – No bitreversal necessary anymore:
    • $\text{INTT}_{bo\rightarrow no}(\text{NTT}_{no\rightarrow bo}(a) \circ \text{NTT}_{no\rightarrow bo}(b))$
Optimization of NTT Computation

Removal of expensive “helper” functions

• **Problem**: Multiplication by scalar $n^{-1}$ in inverse transformation is expensive

• **Solution**: In lattice-based crypto we usually multiply by pretransformed constants (e.g., $\tilde{a}$, $\tilde{p}$, or $r_2$)
  – Put $n^{-1}$ into these constants
  – Multiplication by scalar does not change much as
    • $x \cdot \text{NTT}(a) \Leftrightarrow \text{NTT}(x \cdot a)$
  – Store $\tilde{a}' = n^{-1}\tilde{a}$
Optimization of NTT Computation

Removal of expensive “helper” functions

- **Problem**: Multiplication by powers of $\psi$ and $\psi^{-1}$ (PowMul) is expensive

- **Solution**: Merge powers of $\psi$ into twiddle factors
  - Only possible with forward transformation and current butterfly (see next picture)
Optimization of NTT Computation

- Combines all tricks for forward transformation
- We cannot merge powers of $\psi^{-1}$; We have to multiply after transformation is finished
Optimization of NTT Computation

- Usage of Gentlemen-Sande (GS) butterfly instead of Cooley-Tukey (CT) allows merging of inverse multiplication by powers of $\psi^{-1}$
  - CT: $a + \omega b$ and $a - \omega b$
  - GS: $a + b$ and $(a - b)\omega$
Optimization of NTT Computation

- We save several steps compared to straightforward approach
- Almost no additional costs (if we store twiddle factors)
  - No multiplication by one in first stage anymore
  - Can be mitigated by using lookup tables if coefficients for $e$ are small

(*) FFT people probably know most of these tricks
Optimization of NTT Computation

How to accelerate the multiplication core operation

- Address generation for NTT is cheap and well researched (see FFT)
- The only expensive computation is the butterfly, which boils down to
  - a $\log_2(q) \times \log_2(q)$ multiplication
  - a mod $q$ modulo reduction
  - two additions or subtractions modulo $q$
- Implementation of the butterfly depends on target architecture
  - General methods like Montgomery or Barret reduction
  - Reductions that depend on special primes like Solinas primes
Ring-LWE Encryption on ATXmega (ATXMEGA128A1)

- Moderate performance impact of larger parameter set
- Very fast decryption
- Some pitfalls in practice (only CPA and decryption errors)

<table>
<thead>
<tr>
<th>Operation</th>
<th>(n=256, q=7681)</th>
<th>(n=512, q=12289)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cycle counts and stack usage</td>
<td>Cycle counts and stack usage</td>
</tr>
<tr>
<td>RLWEnc_{ENC}</td>
<td>874,347 (109 bytes)</td>
<td>2,196,945 (102 bytes)</td>
</tr>
<tr>
<td>RLWEnc_{DEC}</td>
<td>215,863 (73 bytes)</td>
<td>600,351 (68 bytes)</td>
</tr>
<tr>
<td>NTT^{CT}_{no\to bo}</td>
<td>185,360</td>
<td>502,896</td>
</tr>
<tr>
<td>INTT^{GS,\psi^{-1}}_{bo\to no}</td>
<td>168,853</td>
<td>427,827</td>
</tr>
<tr>
<td>SampleGauss_{\sigma}</td>
<td>84,001</td>
<td>170,861</td>
</tr>
<tr>
<td>PwMulFlash</td>
<td>22,012</td>
<td>53,891</td>
</tr>
<tr>
<td>AddEncode</td>
<td>16,884</td>
<td>37,475</td>
</tr>
<tr>
<td>Decode</td>
<td>4,407</td>
<td>8,759</td>
</tr>
<tr>
<td></td>
<td>Cycle counts of obsolete functions</td>
<td>Cycle counts of obsolete functions</td>
</tr>
<tr>
<td>NTT^{CT}_{bo\to no}</td>
<td>198,491</td>
<td>521,872</td>
</tr>
<tr>
<td>BitRev</td>
<td>29,696</td>
<td>75,776</td>
</tr>
<tr>
<td>BitrevDual</td>
<td>32,768</td>
<td>79,872</td>
</tr>
<tr>
<td>PowMul_{\psi}</td>
<td>35,068</td>
<td>96,603</td>
</tr>
<tr>
<td></td>
<td>Static memory consumption in bytes</td>
<td>Static memory consumption in bytes</td>
</tr>
<tr>
<td>Complete binary</td>
<td>6,668</td>
<td>9,258</td>
</tr>
<tr>
<td>RAM</td>
<td>1,088</td>
<td>2,144</td>
</tr>
</tbody>
</table>
## Ring-LWE Encryption on ATXmega Family

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Device</th>
<th>Operation</th>
<th>Cycles</th>
<th>OP / s</th>
</tr>
</thead>
<tbody>
<tr>
<td>RLWEenc-la ( n = 256 )</td>
<td>AX128</td>
<td>Enc/Dec</td>
<td>874,347</td>
<td>36.60</td>
</tr>
<tr>
<td>RLWEenc-Ila ( n = 512 )</td>
<td>AX128</td>
<td>Enc/Dec</td>
<td>2,196,945</td>
<td>14.57</td>
</tr>
<tr>
<td>RLWEenc-la ( n = 256 )  [LSR+15]</td>
<td>AX128</td>
<td>Enc/Dec</td>
<td>666,671</td>
<td>48</td>
</tr>
<tr>
<td>RLWEenc-Ila ( n = 512 )  [LSR+15]</td>
<td>AX128</td>
<td>Enc/Dec</td>
<td>2,721,372</td>
<td>11.76</td>
</tr>
<tr>
<td>RLWEenc-la ( n = 256 )  [BSJ14]</td>
<td>AT64</td>
<td>Enc/Dec</td>
<td>3,042,675</td>
<td>2.63</td>
</tr>
<tr>
<td>RLWEenc-la ( n = 256 )  [BJ14]</td>
<td>AX64</td>
<td>Enc/Dec</td>
<td>5,024,000</td>
<td>6.37</td>
</tr>
<tr>
<td>QC-MDPC [HvMG13]</td>
<td>AX128</td>
<td>Enc/Dec</td>
<td>26,767,463</td>
<td>1.20</td>
</tr>
<tr>
<td>RSA-1024 [GPW+04]</td>
<td>AT128</td>
<td>Enc/Dec</td>
<td>3,440,000</td>
<td>2.33</td>
</tr>
<tr>
<td>RSA-1024[LGK10]</td>
<td>AT128</td>
<td>priv. key</td>
<td>75,680,000</td>
<td>0.11</td>
</tr>
<tr>
<td>Curve25519[DHH+ar]</td>
<td>AT2560</td>
<td>Point mul.</td>
<td>13,900,397</td>
<td>1.15</td>
</tr>
<tr>
<td>ECC-ecp160r1 [GPW+04]</td>
<td>AT128</td>
<td>Point mul.</td>
<td>6,480,000</td>
<td>1.23</td>
</tr>
</tbody>
</table>

- Schoolbook was 12 million
- Code size is not significantly increased
- Sampler is the bottleneck

Ring-LWE Encryption on Other Platforms [CRV+15]

<table>
<thead>
<tr>
<th>Platform</th>
<th>Key Gen.</th>
<th>Encrypt</th>
<th>Decrypt</th>
<th>$n, q, \sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARM7TDMI [12]</td>
<td>575 047</td>
<td>878 454</td>
<td>226 235</td>
<td>$P_1$</td>
</tr>
<tr>
<td>ATMega64 [12]</td>
<td>2 770 592</td>
<td>3 042 675</td>
<td>1 368 969</td>
<td>$P_1$</td>
</tr>
<tr>
<td>ATxmega64A3 [11]</td>
<td></td>
<td>- 5 024 000</td>
<td>2 464 000</td>
<td>$P_1$</td>
</tr>
<tr>
<td>Core 2 Duo [3]</td>
<td></td>
<td>9 300 000</td>
<td>4 560 000</td>
<td>1 710 000</td>
</tr>
<tr>
<td>Cortex-M4F</td>
<td></td>
<td>117 009</td>
<td>121 166</td>
<td>43 324</td>
</tr>
<tr>
<td>Core 2 Duo [3]</td>
<td>13 590 000</td>
<td>9 180 000</td>
<td>3 540 000</td>
<td>$P_2$</td>
</tr>
<tr>
<td>Cortex-M4F</td>
<td></td>
<td>252 002</td>
<td>261 939</td>
<td>96 520</td>
</tr>
</tbody>
</table>

$P_1 = (256, 7 681, 11.31/\sqrt{2\pi}), \ P_2 = (512, 12 289, 12.18/\sqrt{2\pi})$

1 Cycles estimated from reported execution time.

2 Cycles estimated from reported execution time with a clock speed of 32 MHz.

Table from [CRV+15]: Ruan de Clercq, Sujoy Sinha Roy, Frederik Vercauteren, Ingrid Verbauwhede: *Efficient software implementation of ring-LWE encryption*. DATE 2015: 339-344
Further Implementation Remarks and Requirements

• **CCA2-Security:**
  – Additional conversion (e.g., via Fujisaki-Okamoto, includes the necessity for hash-function and re-encryption)

• **Side-Channel Attacks:**
  – Masking schemes (SCA) by Reparaz et al [CHES15, PQCryptO16], does not include CCA2 security

• **Fault-Injection Attacks:**
  – Loop-Abort attacks by Espitau et al. [ePrint 16]
  – Fault Sensitivity by Bindel et al. [FDTC16]
Tutorial Outline – Part III

Code-based Cryptography
Efficient Code-based Implementations
Lattice-based Cryptography
Efficient Lattice-based Implementations

Lessons Learned
Lessons Learned

- **Efficient McEliece implementations with practical key sizes**
  - QC-MDPC codes are an efficient alternative also in software
  - Note: consider reported issues with decryption error (ASIACRYPT 2016)
  - Physical attacks are more challenging to counter with probabilistic decoding

- **Efficient R-LWE encryption are extremely efficient**
  - R-LWE (and variants) also allow signature + advanced schemes
  - Software implementations very efficient compared to ECC and RSA

- **Papers and source code available at**
  

- **For more papers and codes, see project websites of**
  
  - [Horizon 2020](http://www.seceng.rub.de/research/projects/pqc/)
  - [SAFEcrypto](http://www.seceng.rub.de/research/projects/pqc/)
  - [PQCryptO ICT-645622](http://www.seceng.rub.de/research/projects/pqc/)
  - [ICT-644729](http://www.seceng.rub.de/research/projects/pqc/)
Part III: Post Quantum Cryptography
in Embedded Software
Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

Thank you! Questions?