Part I: Introduction to Post Quantum Cryptography

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017

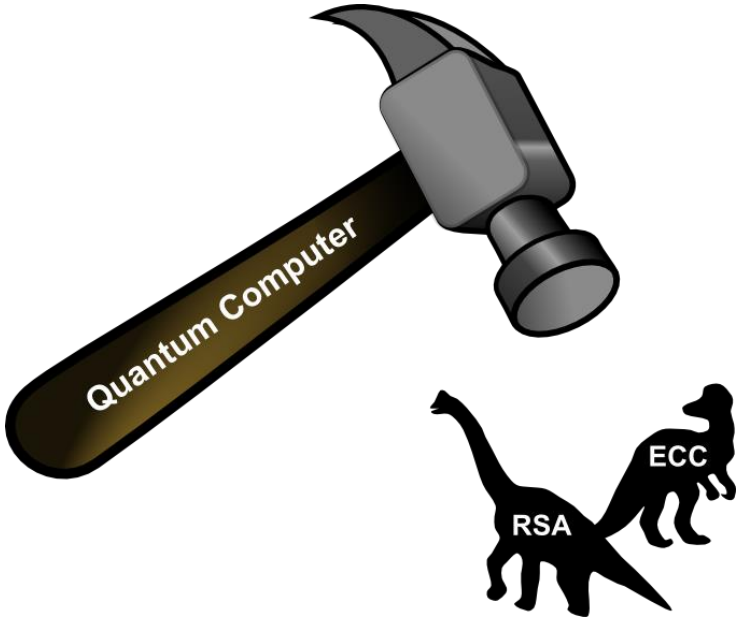
Overview

RUB

- **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

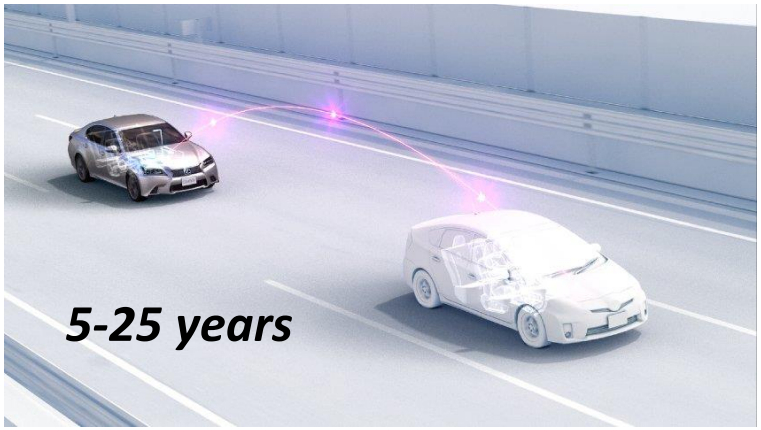


Long-Term Security in Embedded Devices

RUB

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

- Many processing platforms have tight constraints with their computational ressources



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)

Security (bits) (symmetric)	RSA	DLOG	EC
48	480	480	96
56	640	640	112
64	816	816	128
80	1248	1248	160
112	2432	2432	224
128	3248	3248	256
160	5312	5312	320
192	7936	7936	384
256	15424	15424	512

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

RUB

PRL 108, 130501 (2012) PHYSICAL REVIEW LETTERS week ending 30 MARCH 2012

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 Du[1,†]
[1]*Hefei National Laboratory for Physical Sciences at Microscale and Department of Modern Physics, University of Science and Technology of China, Hefei, Anhui, 230026, China*
[2]*Department of Physics, Shanghai Key Laboratory for Magnetic Resonance, East China Normal University, Shanghai 200062*

Quantum
Adiabatic qu
we report an
143 on a liqu
largest numb
adiabatic qua

ntal realization of Shor's
factoring algorithm
resonance

n*†, Gregory Breyta*,
& Isaac L. Chuang*†
nia 95120, USA
University, Stanford,

ter requires in order to
er N increases exponen-
wn at present[1]. Factoring
be intractable classically,
of widely used crypto-
however, could factor
hor's quantum factoring
the study of quantum
n of this algorithm has
implementation of the
actorization of $N = 15$
seven spin-1/2 nuclei in
n be manipulated with
agnetic resonance tech-
tore quantum informa-
taining many quantum
y the present work. The
stration of experimental
ntrol and modelling of
ar, we present a simple,
lecoherence effects[14] in

SIAM J. COMPUT.
Vol. 26, No. 5, pp. 1484–1509, Octob

POLYNOMIAL-TIME
AND DISCRETE L

Abstract. A digital comp
that is, it is believed able to si
time by at most a polynomia
consideration. This paper co
which are generally thought t
of several proposed cryptosys
on a hypothetical quantum
input size, e.g., the number

Key words. algorith
thesis, quantum computers

Quantum factorization of 56153 with only 4 qubits

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

December 1, 2014

The largest number factored on a quantum device reported until now was 143 [1]. That quantum computation, which used only 4 qubits at 300K, actually also factored much larger numbers such as 3599, 11663, and 56153, without the awareness of the authors of that work. Furthermore, unlike the implementations of Shor's algorithm performed thus far [2–8], these 4-qubit factorizations do not need to use prior knowledge of the answer. However, because they only use 4 qubits, these factorizations can also be performed trivially on classical computers. We discover a class of numbers for which the power of quantum information actually comes into play. We then demonstrate a 3-qubit factorization of 175, which would be the first quantum factorization

and no prior knowledge of the answer to the problem. To provide context, factoring the number 15 with Shor's algorithm and no prior knowledge of the answer to the problem, requires a minimum of 8 qubits (and more if error correction is attempted); and this has still never been done.

Quantum factorization of 143

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

Table 1: *Multiplication table for* $11 \times 13 = 143$ *in binary.*

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
p					1	p_2	p_1	1
q					1	q_2	q_1	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

RUB

- CHES has a long tradition on the implementation of PQC cryptosystems:
  - CHES 2001: Bailey et al.: NTRU in Small Devices
  - 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

RUB

- 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**

European Commission

PQCRYPTO
ICT-645622
(2015-2019)

DFG Deutsche Forschungsgemeinschaft

Implementierungsaspekte alternativer asymmetrischer Kryptosysteme
(2012-2017)

European Commission

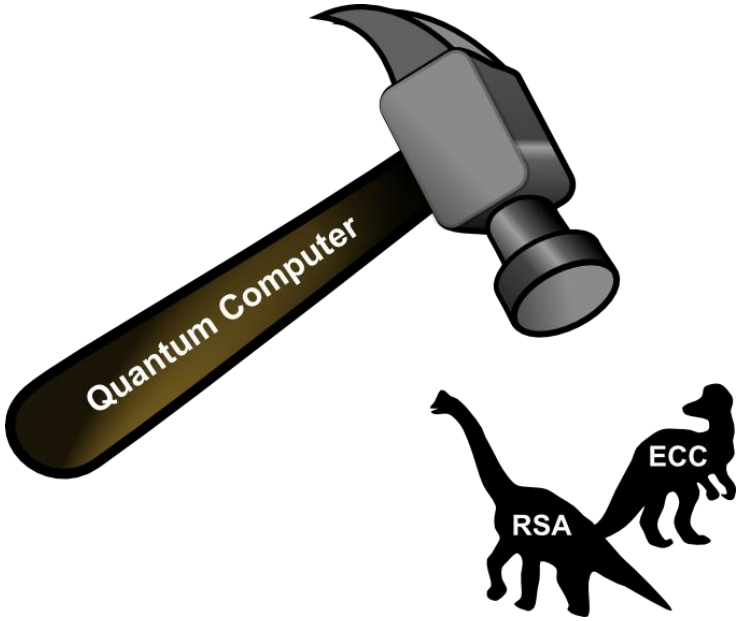
SAFE crypto

ICT-644729
(2015-2019)

+ more

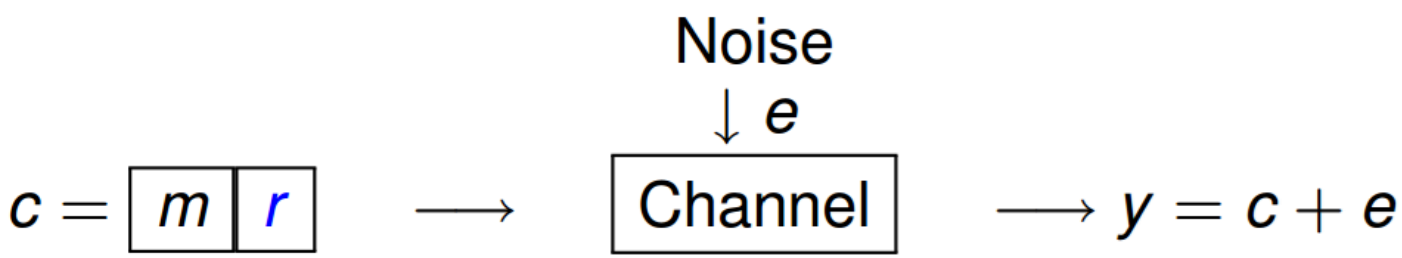
Outline

- Introduction
- **Classes of Post-Quantum Cryptography (PQC)**
  - **Code-Based Cryptography**
  - Lattice-Based Cryptography
  - Hash-Based Cryptography
- Lessons Learned



Introduction to Code-based Cryptography

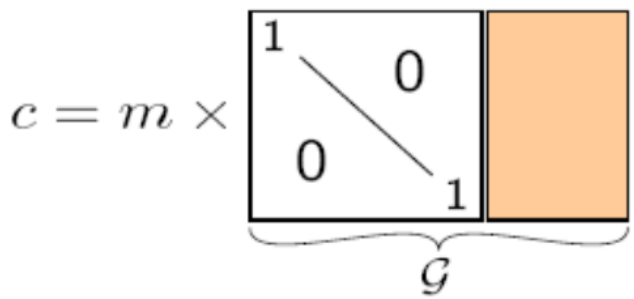
- Error-Correcting Codes are well-known in a large variety of applications
- Detection/Correction of errors in noisy channels by adding redundancy



- **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 $\mathbf{C[n, k, d]}$ of **length** $\mathbf{n}$ with **dimension** $\mathbf{k}$ and **minimum distance** $\mathbf{d}$
- Matrices can be in systematic form minimizing time/storage



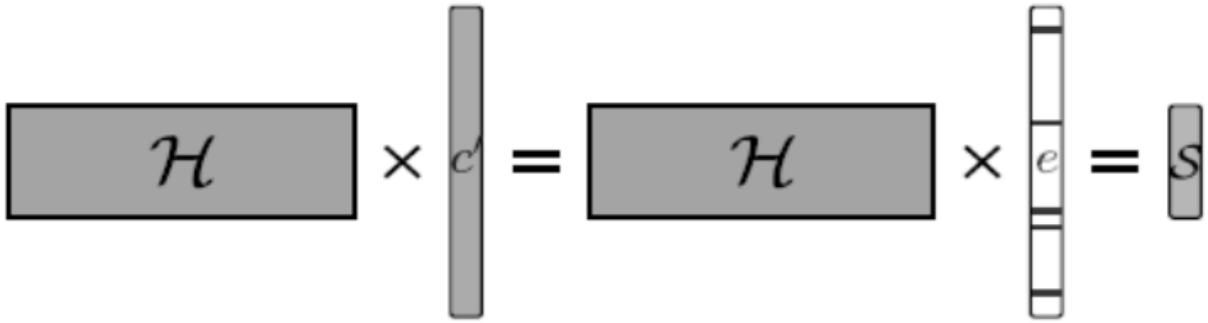
Matrix size of G:
k x n

Linear Codes and Cryptography

- **Parity check matrix H** is a $(n\text{-}k) \cdot k$ matrix orthogonal to G
- Defines the dual C of the code C via scalar product

$$\mathcal{C}^{\perp} = \{y \in \mathbb{F}_q^n | x \cdot y = 0, \forall x \in \mathcal{C}\}$$

- 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 $GF(2^{n-k})$
  - t : positive integer (defined by error correction capability)

- **Problem: Is there a vector e in $GF(2^n)$ 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
    On the inherent intractability of certain coding problems. IEEE Transactions on Information Theory, 24(3), May 1978.

Niederreiter Encryption Scheme [1986]

Key Generation
Given a code $\mathrm{C}[\mathrm{n}, \mathrm{k}, \mathrm{d}]$ 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 $e \in_R F_2^n, wt(e) \leq t$
$\mathrm{x} \leftarrow \widehat{H} \cdot e^T$

Decryption
Let Ψ_H be a t-error-correcting decoding algorithm.
$\mathrm{P}m^T \leftarrow \Psi_H(S^{-1} \cdot x)$
Extract m by transposing the computation $\mathrm{P}^{-1} \cdot \mathrm{P}m^T$.

McEliece Encryption Scheme [1978]

Key Generation

Given a code $\mathrm{C}[\mathrm{n}, \mathrm{k}, \mathrm{d}]$ 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 $e \in_R F_2^n$, $wt(e) \leq t$

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

Decryption

Let Ψ_H be a t-error-correcting decoding algorithm.

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

Extract m by computing $\mathrm{S}^{-1} \cdot Sm$

Taxonomy of Code-Based Encryption Schemes

RUB

Code-based Encryption Schemes*

McEliece [M78]

Niederreiter [N86]

Generalized Reed-Solomon

Goppa

Rank-Metric

Elliptic

Concatenated

Reed Muller

LRPC/LDCP/MDPC

Srivastava

* This is a selection based on presenter's choice.

Taxonomy of Code-Based Encryption Schemes

RUB

Code-based Encryption Schemes*

McEliece [M78]

Niederreiter [N86]

Generalized Reed-Solomon

Goppa

Rank-Metric

Srivastava

Reed-Muller

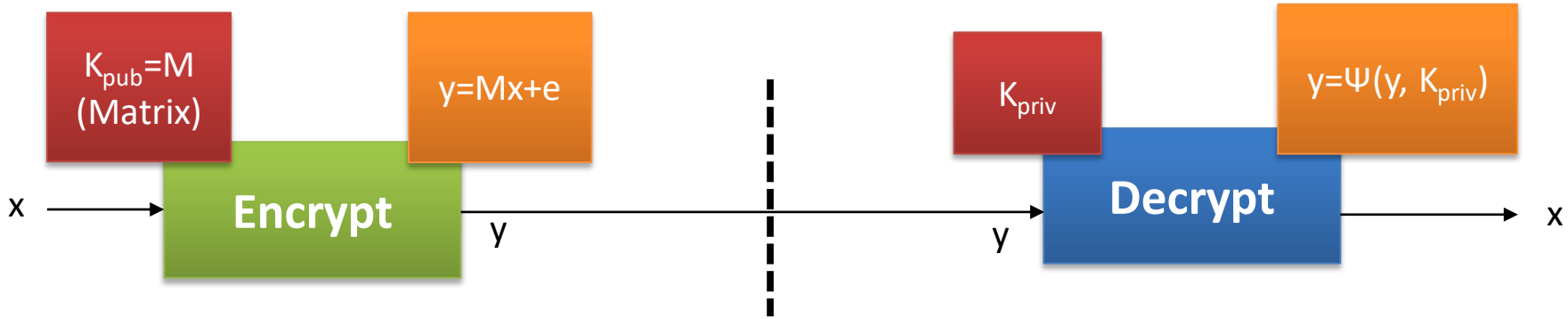
Elliptic

Concatenated

LRPC/LDPC/MDPC

* This is a selection based on presenter's choice.

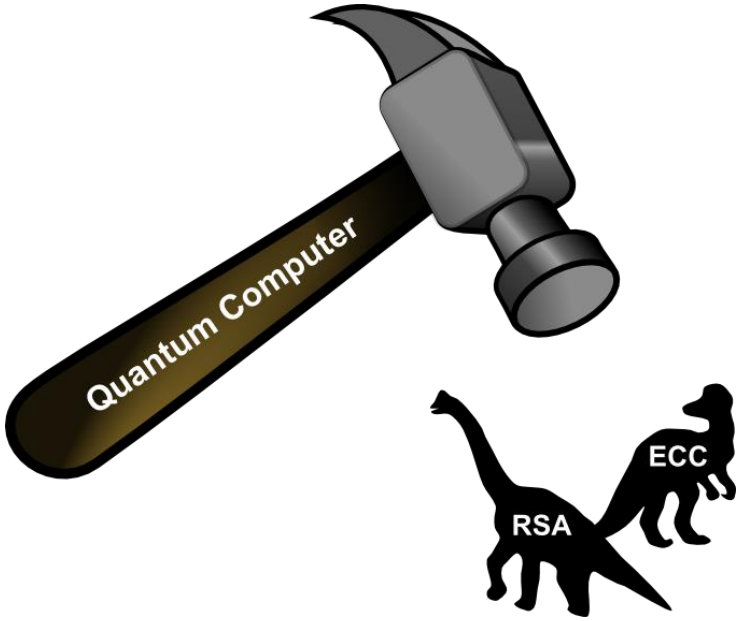
Key Aspects of Code-Based Cryptography



- **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**

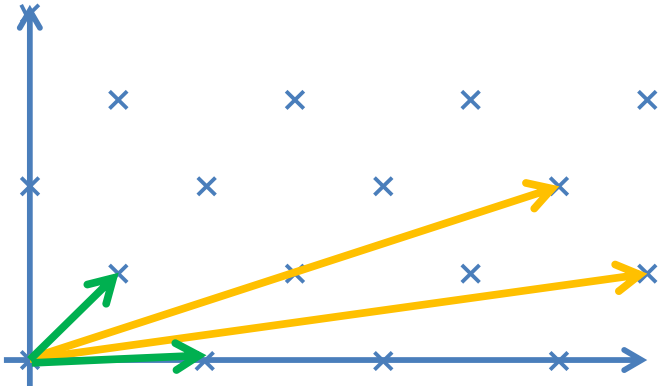
Outline

- 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

secret

$\mathbb{Z}_{13}^{7\times 4}$ $\mathbb{Z}_{13}^{4\times 1}$ $\mathbb{Z}_{13}^{7\times 1}$

$$\begin{bmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 53 \\ 3 & 9 & 0 & 10 \\ 1 & 3 & 3 & 2 \\ 12 & 7 & 3 & 4 \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{bmatrix} \times \begin{bmatrix} 6 \\ 9 \\ 11 \\ 11 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 1 \\ 10 \\ 4 \\ 12 \\ 9 \end{bmatrix}$$

Use Gaussian elimination

Blue is given; Find (learn) **red** ➔ Solve linear system

Learning with Errors

Solving of a system of linear equations

random	secret	small noise	looks random
$\mathbb{Z}_{13}^{7\times 4}$	$\mathbb{Z}_{13}^{4\times 1}$	$\mathbb{Z}_{13}^{7\times 1}$	$\mathbb{Z}_{13}^{7\times 1}$

$$\begin{pmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 53 \\ \vdots & \vdots & \vdots & \vdots \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{pmatrix} \times \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix} + \begin{pmatrix} \\ \\ \\ \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \\ 1 \\ 10 \\ 4 \\ 12 \\ 9 \end{pmatrix}$$

Gets large in practice $\mathbb{Z}_{4093}^{640\times 256}$

Blue is given; Find **red** ➔ Learning with Errors (LWE) Problem

Key Aspects of Lattice-based Systems

RUB

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

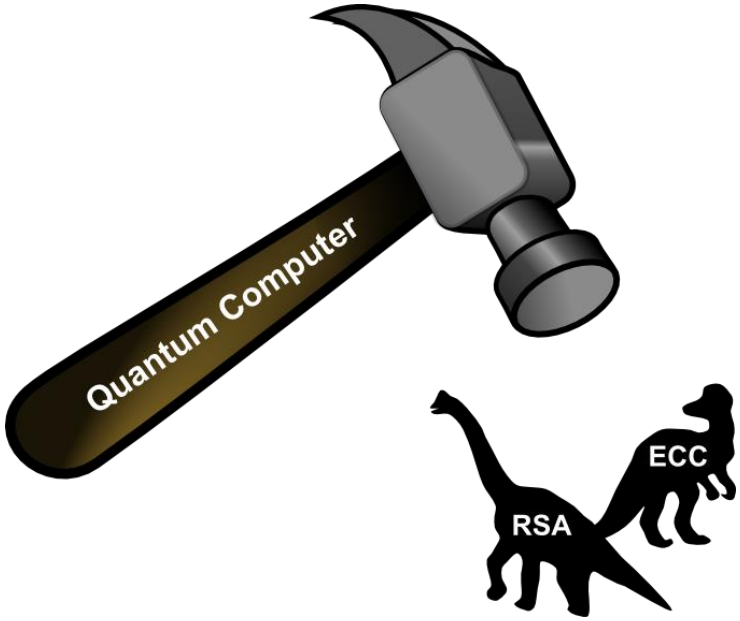
- **Random Sampling not only from uniform but also from Discrete Gaussian distributions** (not a trivial task!)

- **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)

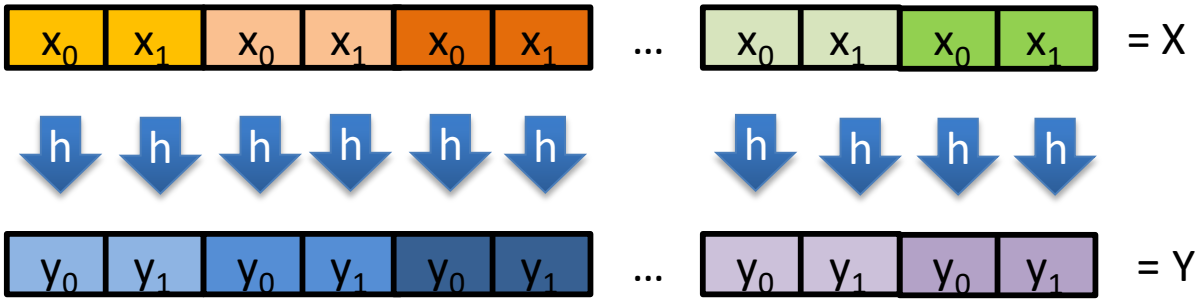
Outline

- 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)

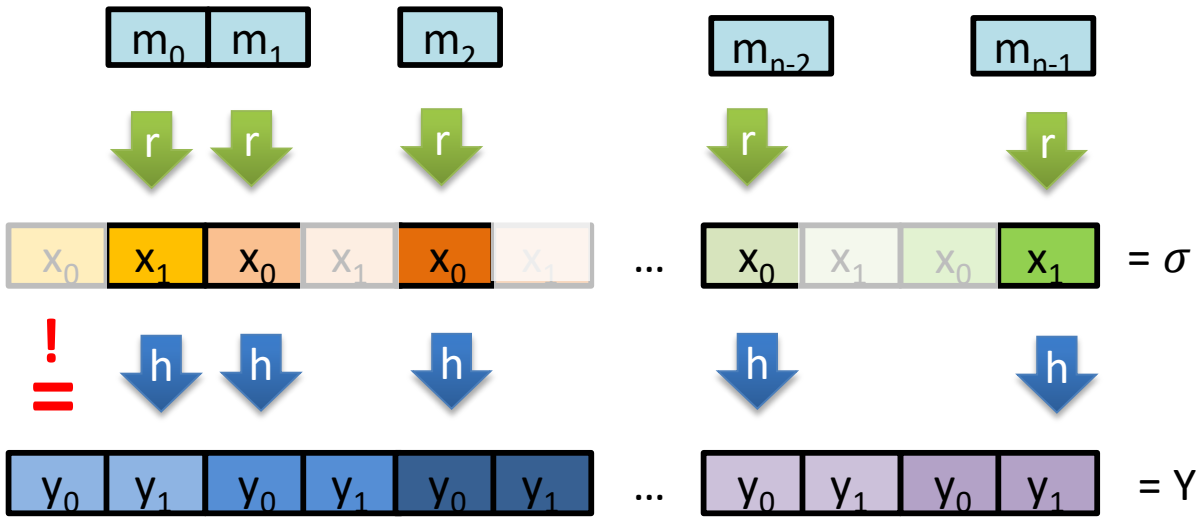
- **Definition**: Given a security parameter n, the set of n-bit vectors $U_n = \{0,1\}^n$ and a one-way function h: $U_n \rightarrow U_n$
- **Secret key**: Generate $2n \times n$-bit vector $X = (x_{(0,0)}, x_{(0,1)}, x_{(1,0)}, x_{(1,1)}, .., x_{(n-1,1)})$
- **Public Key** : Compute $Y = \left(y_{(0,0)}, .., y_{(n-1,1)}\right) \forall y_{i,j} = f(x_{i,j})$



- **Publish** public key Y

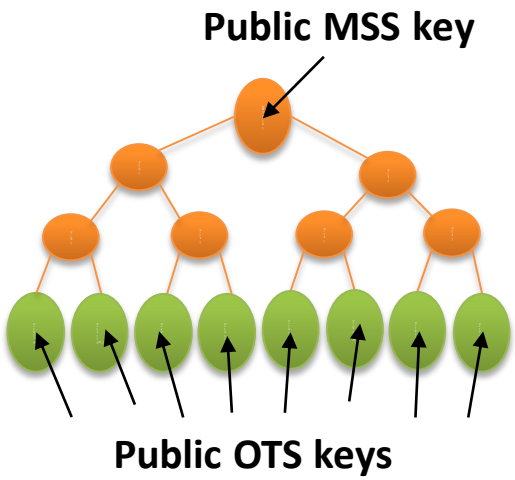
Hash-based Cryptography: Lamport-Diffie One-Time Signatures (LD-OTS, 1979)

- **Definition**: Given a published public key Y and an n-bit message $M = (m_0, \dots, m_{n-1})$ to sign
- **Sign**: Generate signature $\sigma = (x_{(0,m_0)}, \dots, x_{(n-1,m_{n-1})})$ 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

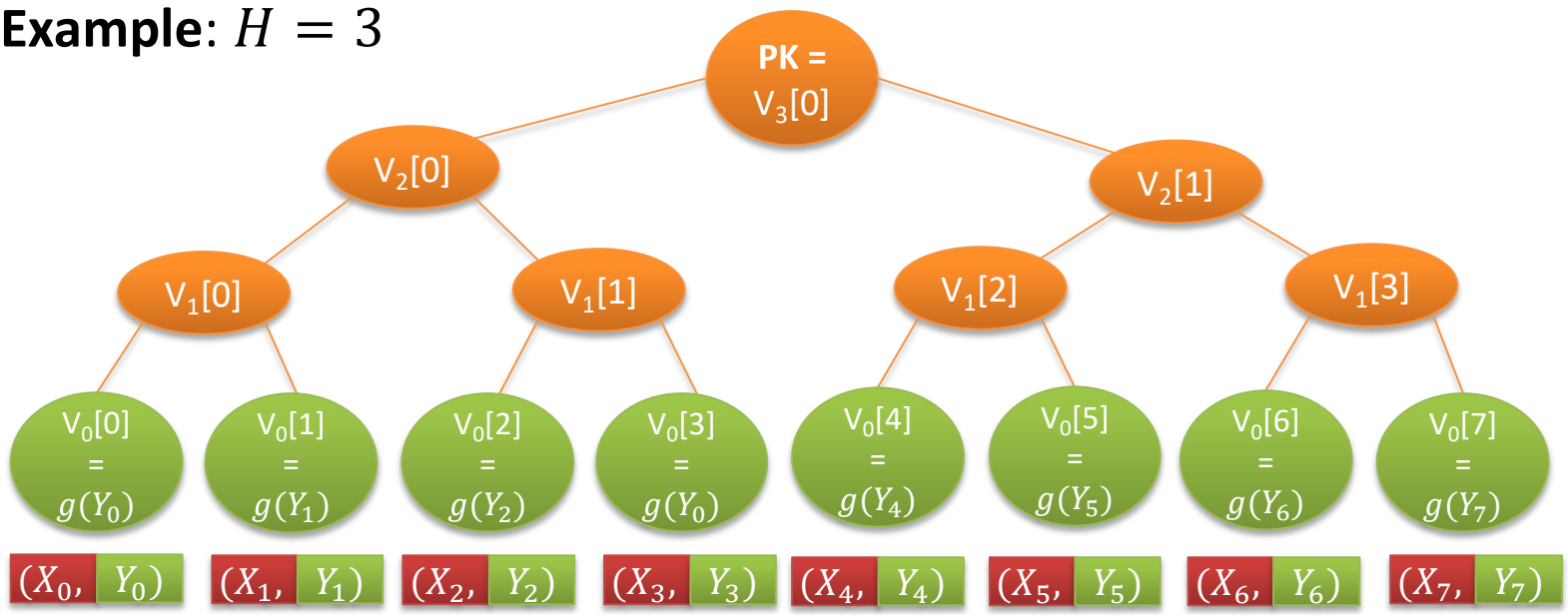


- **Properties and Requirements**
  - Max. signature count determined by height H of tree (fixed at setup)
  - Needs to keep track of already used signatures in the tree → stateful signature scheme
  - Can be used with any one-time signature scheme and (collision-resistant) cryptographic hash function

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: $V_i[j] = \mathrm{g}(V_{i-1}[2\mathrm{j}] \,||\, V_{i-1}[2\mathrm{j}+1])$ with $0 < i \leq \mathrm{H}$ and $0 \leq j < 2^i$

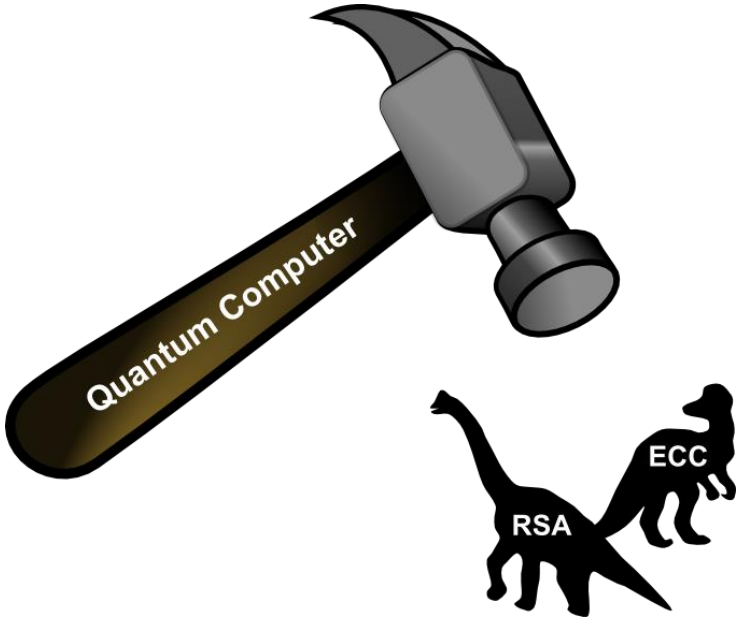
Key Aspects of Hash-based Cryptographic Systems

RUB

- **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

RUB

- **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

European Commission
PQCRYPTO
ICT-645622
SAFE crypto
ICT-644729
DFG Deutsche Forschungsgemeinschaft

Part I: Introduction to Post Quantum Cryptography

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017

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

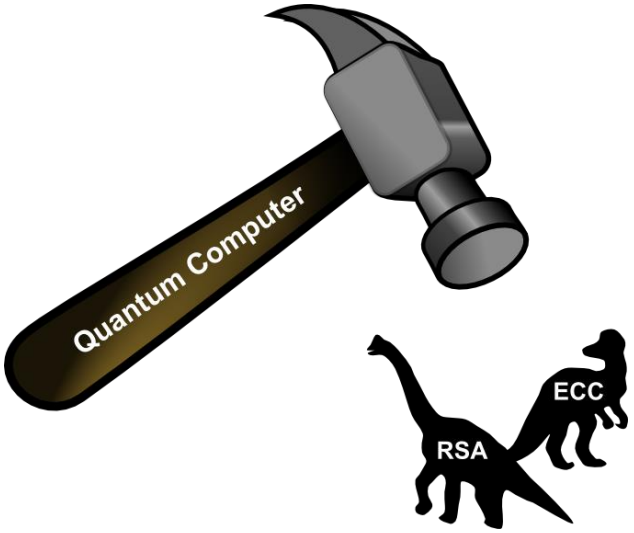
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned



Recall: McEliece Encryption Scheme [1978]

RUB

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 $\mathrm{e} \in_R \mathbb{F}_2^n$, $\mathrm{wt}(\mathrm{e}) \leq t$

$\mathrm{x} \leftarrow mG' + \mathrm{e}$

Decryption

Let Ψ_H be a t-error-correcting decoding algorithm.

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

Extract m by computing $m \cdot S \cdot S^{-1}$

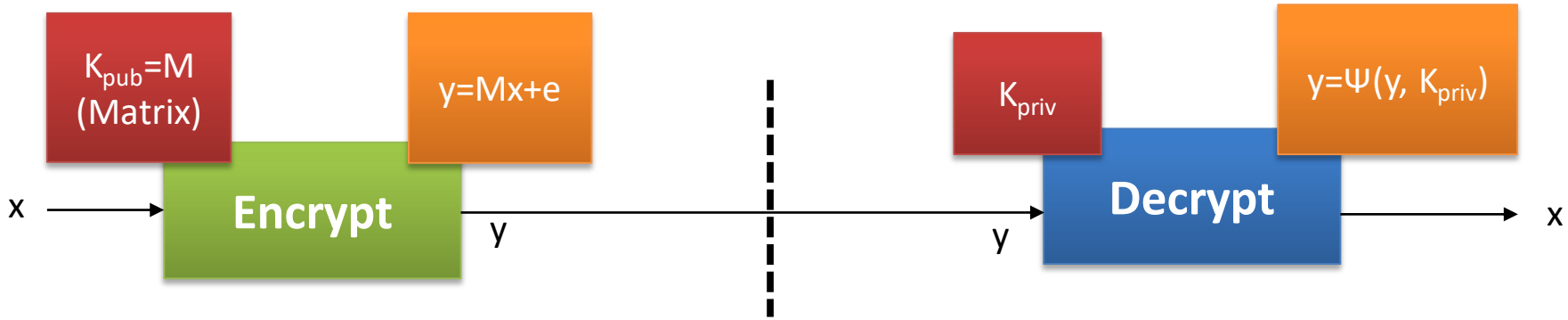
Security Parameters (Binary Goppa Codes)

RUB

- **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 $k * (n - k)$ bit matrix (redundant part only)

Security Level	Parameters (n, k, t), errors added	Size K_{pub} in KBits	Size K_{sec} $(g(z) \mid \mathcal{L} \mid M^{-1})$ KBits
Short-term (60 bit)	$(1024, 644, 38), 38$	239	$(0.37 \mid 10 \mid 141)$
Mid-term I (80 bit)	$(2048, 1751, 27), 27$	507	$(0.29 \mid 22 \mid 86)$
Mid-term II (128 bit)	$(2690, 2280, 56), 57$	913	$(0.38 \mid 18 \mid 164)$
Long-term (256 bit)	$(6624, 5129, 115), 117$	$7,488$	$(1.45 \mid 84 \mid 2,183)$

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)

RUB

- 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

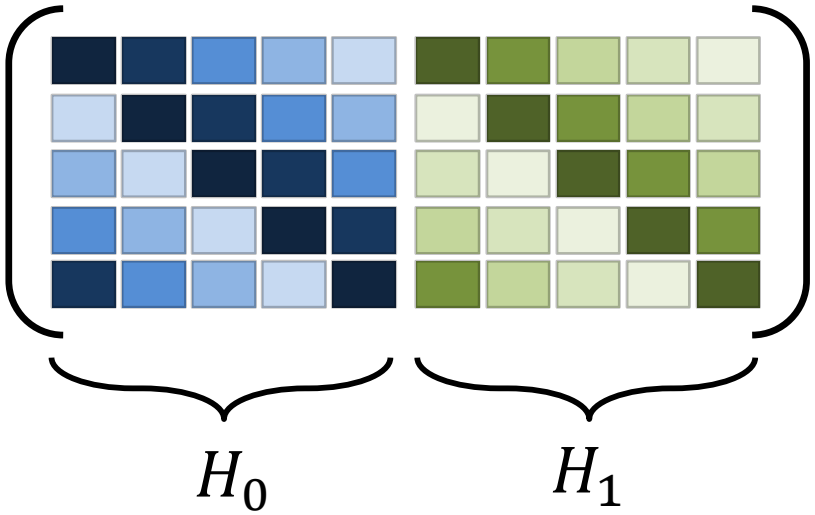
1. Generate n_0 first rows of parity-check matrix blocks H_i
   $h_i \in_R F_2^r$ of weight w_i, $\mathrm{w} = \sum_{i=0}^{n_0-1} w_i$
2. Obtain remaining rows by $r - 1$ quasi-cyclic shifts of h_i
3. $H = [H_0 | H_1 | \dots | H_{n_0-1}]$
4. Generator matrix of systematic form $G = (I_k | Q)$

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

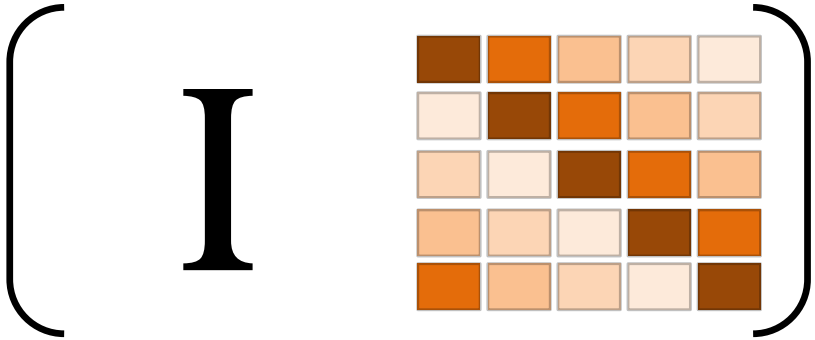
Background on QC-MDPC Codes

Parity check matrix H

$n_0 = 2$



Generator matrix G



(QC-)MDPC McEliece

RUB

Encryption

Message $m \in F_2^k$, error vector $e \in_R F_2^n$, $wt(e) \leq t$

$\mathrm{x} \leftarrow mG + e$

Decryption

Let Ψ_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

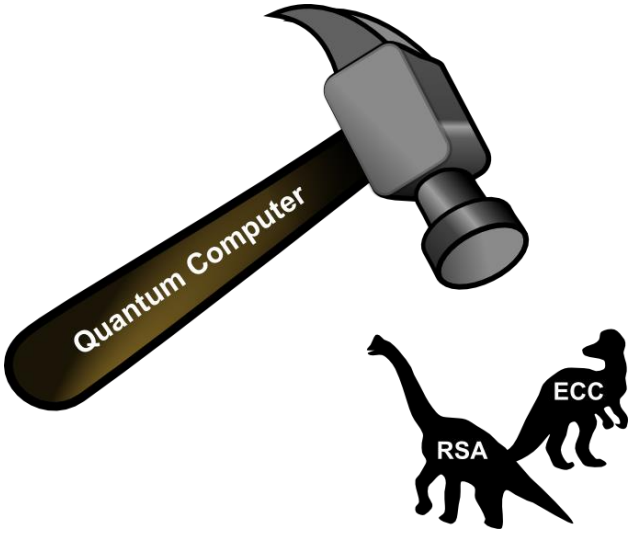
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

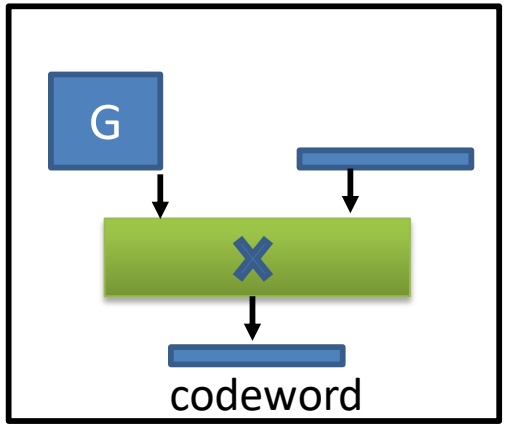
Lessons Learned

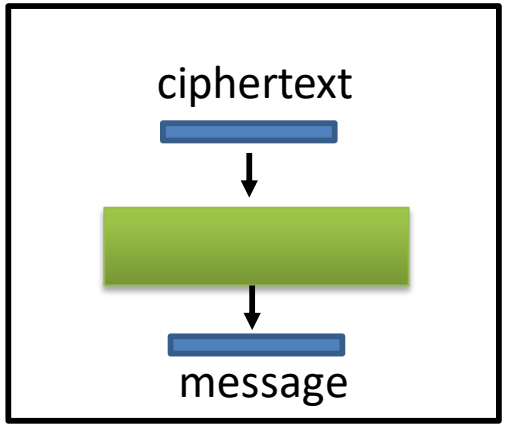


Hardware Implementation of Building Blocks for McEliece/Niederreiter

RUB

- **Two Operations**
  - **Encryption/Encoding:**
    - Matrix-vector multiplication (with large matricies, 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

RUB

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)

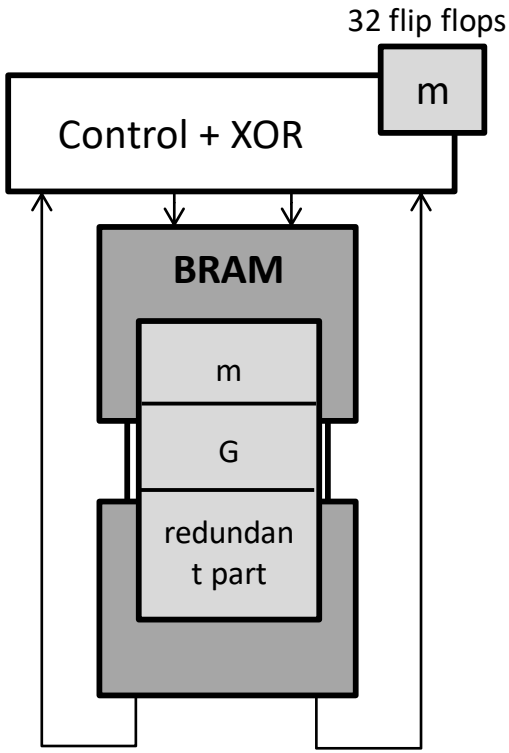
- How to determine threshold b ?
  - Precompute b_i for each iteration [Gal62]
  - $b = max_{upc}$ [HP03]
  - $b = max_{upc} - \delta$ [MTSB13]

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



FPGA Low-Resource Decryption

RUB

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

RUB

QC-MDPC Decryption

- Syndrome computation $s = Hx^T$
  - Similar technique as for encoding
- Compare $s = \mathbf{0}$?
  - 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

RUB

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

	Virtex-6 XC6VLX240T		Spartan-6 XC6SLX4	
Aspect	**Encryption**	**Decryption**	**Encryption**	**Decryption**
FFs	120	412	119	413
LUTs	224	568	226	605
Slices	68	148	64	159
BRAM	1	3	1	3
Frequency	334 MHz	318 MHz	213 MHz	186 MHz
Time/Op	2.2 ms	13.4 ms	3.4 ms	23.0 ms

Lightweight FPGA Comparison

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

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

Tutorial Outline – Part II

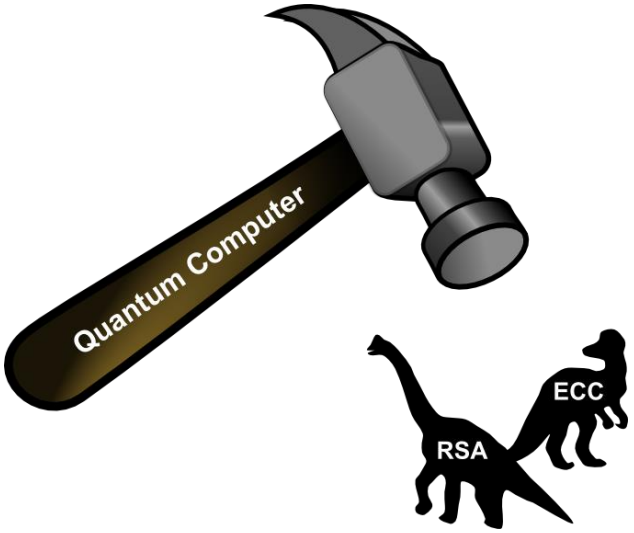
Code-based Cryptography

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)

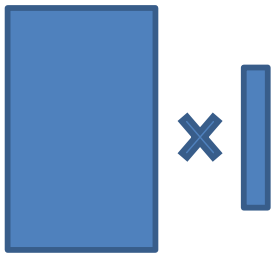
To be Ideal or not Ideal?

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

$\mathbb{Z}_{13}^{7\times 4}$ $\mathbb{Z}_{13}^{4\times 1}$ $\mathbb{Z}_{13}^{7\times 1}$

$$\begin{bmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 53 \\ 3 & 9 & 0 & 10 \\ 1 & 3 & 3 & 2 \\ 12 & 7 & 3 & 4 \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{bmatrix} \times \begin{bmatrix} 6 \\ 9 \\ 11 \\ 11 \end{bmatrix} = \begin{bmatrix} 4 \\ 8 \\ 1 \\ 10 \\ 4 \\ 12 \\ 9 \end{bmatrix}$$

Use Gaussian elimination

Blue is given; Find (learn) **red** ➔ Solve linear system

Learning with Errors

Solving of a system of linear equations

random	secret	small noise	looks random
$\mathbb{Z}_{13}^{7\times 4}$	$\mathbb{Z}_{13}^{4\times 1}$	$\mathbb{Z}_{13}^{7\times 1}$	$\mathbb{Z}_{13}^{7\times 1}$

$$\begin{pmatrix} 4 & 1 & 11 & 10 \\ 5 & 5 & 9 & 53 \\ \vdots & \vdots & \vdots & \vdots \\ 6 & 5 & 11 & 4 \\ 3 & 3 & 5 & 0 \end{pmatrix} \times \begin{pmatrix} \\ \\ \\ \\ \end{pmatrix} + \begin{pmatrix} \\ \\ \\ \\ \\ \\ \\ \end{pmatrix} = \begin{pmatrix} 4 \\ 8 \\ 1 \\ 10 \\ 4 \\ 12 \\ 9 \end{pmatrix}$$

Gets large in practice $\mathbb{Z}_{4093}^{640\times 256}$

Blue is given; Find **red** ➔ Learning with errors

(Ring) Learning with Errors

From learning with errors to ring-learning with errors

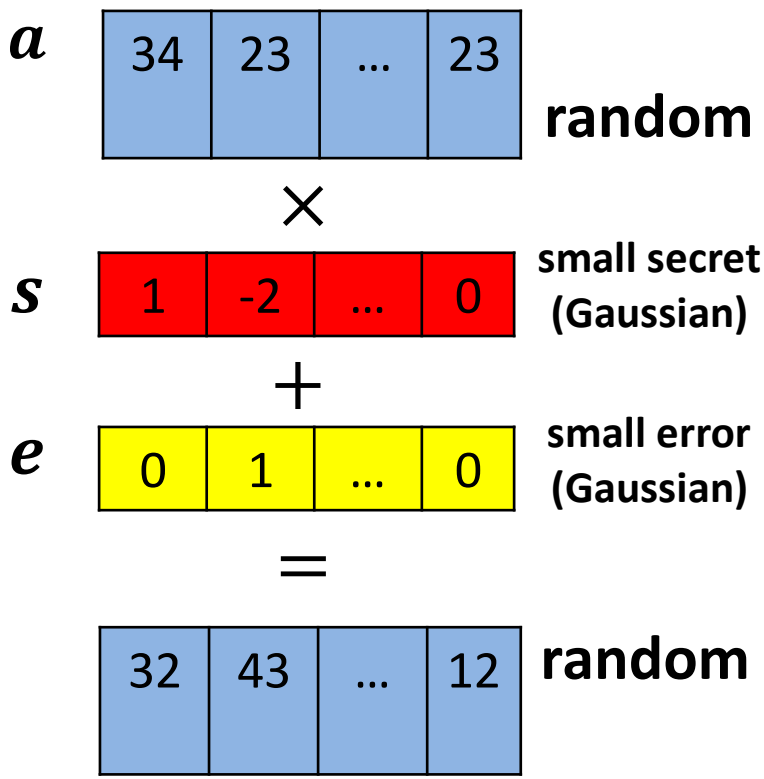
- Shift first line on every line
- Use rule that we negate x in case of wrap around (e.g., $10 \Rightarrow -10 \equiv 3 \bmod 13$)

$$\mathbb{Z}_{13}^{7\times 4}$$

4	1	11	10
3	4	1	11
2	3	4	1
12	2	3	4
9	12	2	3
10	9	12	2
11	10	9	12

Only one line has to be stored

Ring Learning with Errors: Principle



- **Ideal lattices** correspond to ideals in the ring $\mathrm{R} = \frac{Z_q[x]}{\langle x^n+1 \rangle}$
- **Ring Learning With Errors** (RLWE) sample is: $\mathbf{t} = \boldsymbol{as} + \boldsymbol{e} \in R$ for uniform $\boldsymbol{a} \in \mathrm{R}$ and small discrete Gaussian distributed $\boldsymbol{s}, \boldsymbol{e} \leftarrow D_\sigma$
  - Search-RLWE: Find $\mathbf{s}$ when given $\mathbf{t}$ and $\mathbf{a}$
  - Decision-RLWE: Distinguish $\mathbf{t}$ from uniform when given $\mathbf{t}$ and $\mathbf{a}$

Example:

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

- Assume ring $\mathrm{R} = \frac{Z_q[x]}{\langle x^n+1 \rangle}$
- Assume parameters $q = 5$ and $n = 4$
- $\boldsymbol{v} = 4x^3 + 2x^2 + 0x^1 + 1 \quad = (4,2,0,1)$
- $\mathbf{k} = 2x^3 + 1x^2 + 4x^1 + 0 \quad = (2,1,4,0)$
- $\boldsymbol{s} = \boldsymbol{v} + \boldsymbol{k} = (4 + 2 \bmod 5, 2 + 1, 4, 1) = (1,3,4,1)$

$\boldsymbol{v}$	4	2	0	1
	+			
$\boldsymbol{k}$	2	1	4	0
	=			
	6	3	4	1
	mod 5			
$\boldsymbol{s}$	1	3	4	1

Example:

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

- $\boldsymbol{k} = (2, 1, 4, 0)$
- $\boldsymbol{s} = (1, 3, 4, 1)$
- **Task:** $\boldsymbol{z} = \boldsymbol{s} * \boldsymbol{k} = (3, 0, 2, 0)$

multiply-then-reduce

+				0	0	0	0	0s
+			4	12	16	4		4s
+		1	3	4	1			1s
+	2	6	8	2				2s
	2	7	15	18	17	4	0	

mod 5/mod $x^{(n+1)}$

z	3	0	2	0

in-place multiplication

+	0	0	0	0	0s
+	12	16	4	-4	4s
+	4	1	-1	-3	1s
+	2	-2	-6	-8	2s
	18	15	-3	-15	

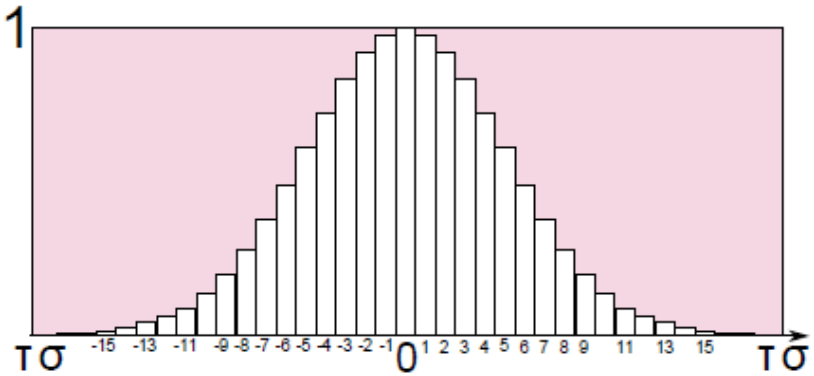
mod 5

z	3	0	2	0

Discrete Gaussian Distribution

- D_σ is defined by assigning weight proportional to

$$\rho_\sigma(x) = \exp\left(\frac{-x^2}{2\sigma^2}\right)$$



Uniform

-1501	1020	502	…	-1900	572

$\boldsymbol{a}$

$$\mathrm{R} = \frac{Z_{4093}[x]}{\langle x^{256} + 1\rangle}$$

Gaussian

-1	4	-8	…	0	1

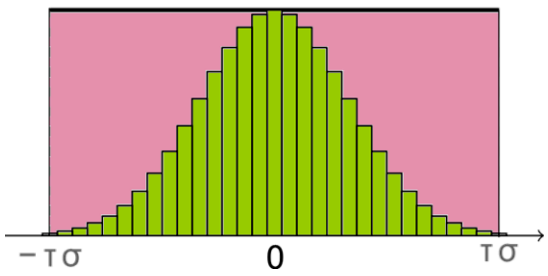
e

Remark on Arithmetic of x-distributed values:

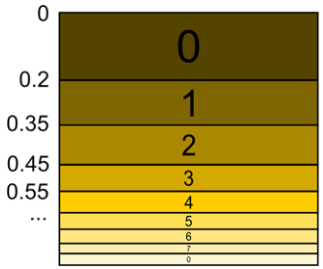
Uniform * Gaussian = Uniform

Gaussian * Gaussian = larger Gaussian

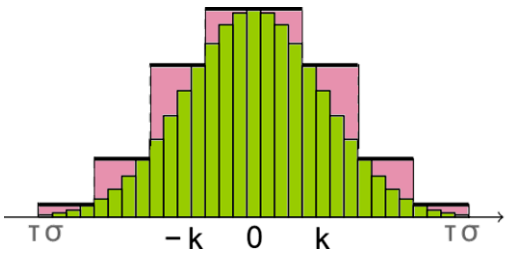
Gaussian Sampling: Options



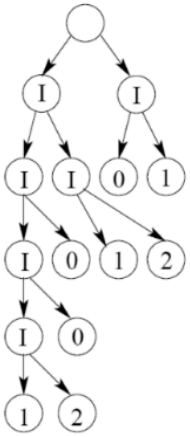
Rejection Sampling



Cumulative Distribution Table (CDT) Sampling



Bernoulli Sampling



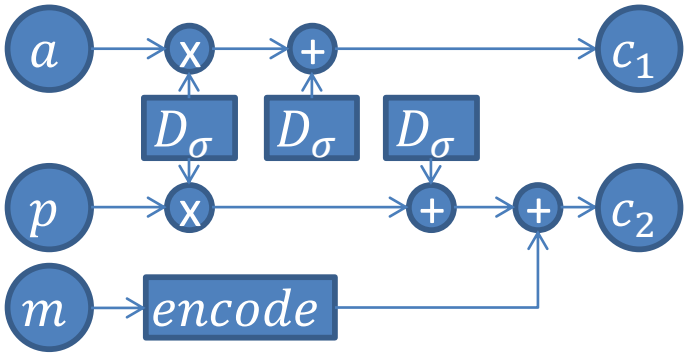
Knuth-Yao Sampling

[DG14] *Efficient sampling from discrete Gaussians for lattice-based cryptography on a constrained device*, Dwarakanath and Galbraith, Applicable Algebra in Engineering, Communication and Computing, 2014
[DDLL14] *Lattice Signatures and Bimodal Gaussians*, Léo Ducas and Alain Durmus and Tancrède Lepoint and Vadim Lyubashevsky, CRYPTO '13

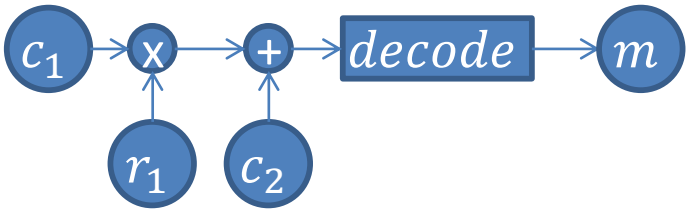
Ring-LWE Encryption Scheme [LP11/LPR10]

Gen: Choose $\boldsymbol{a} \leftarrow R$ and $\boldsymbol{r}_1, \boldsymbol{r}_2 \leftarrow D_\sigma$; *pk*: $\boldsymbol{p} = \boldsymbol{r}_1 - \boldsymbol{a} \cdot \boldsymbol{r}_2 \in \mathrm{R}$; *sk*: $\boldsymbol{r}_2$

Enc$(\boldsymbol{a}, \boldsymbol{p}, m \in \{0,1\}^n)$: $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3 \leftarrow D_\sigma$.
$\bar{\boldsymbol{m}} = encode(m)$. Ciphertext:
$[\boldsymbol{c}_1 = \boldsymbol{a} \cdot \boldsymbol{e}_1 + \boldsymbol{e}_2, \boldsymbol{c}_2 = \boldsymbol{p} \cdot \boldsymbol{e}_1 + \boldsymbol{e}_3 + \bar{\boldsymbol{m}}]$



Dec$(c = [\boldsymbol{c}_1, \boldsymbol{c}_2], \boldsymbol{r}_2)$: Output
$decode(\boldsymbol{c}_1 \cdot \boldsymbol{r}_2 + \boldsymbol{c}_2)$



Correctness: $\boldsymbol{c}_1\boldsymbol{r}_2 + \boldsymbol{c}_2 = (\boldsymbol{a}\boldsymbol{e}_1 + \boldsymbol{e}_2)\boldsymbol{r}_2 + \boldsymbol{p}\boldsymbol{e}_1 + \boldsymbol{e}_3 + \bar{\boldsymbol{m}}$
$= \boldsymbol{r}_2\boldsymbol{a}\boldsymbol{e}_1 + \boldsymbol{r}_2\boldsymbol{e}_2 + \boldsymbol{r}_1\boldsymbol{e}_1 - \boldsymbol{r}_2\boldsymbol{a}\boldsymbol{e}_1 + \boldsymbol{e}_3 + \bar{\boldsymbol{m}} = \bar{\boldsymbol{m}} + \boldsymbol{r}_2\boldsymbol{e}_2 + \boldsymbol{r}_1\boldsymbol{e}_1 + \boldsymbol{e}_3$

large

small

Ring-LWE Encryption: Parameters

$$R = \frac{Z_{4093}[x]}{\langle x^{256} + 1 \rangle}$$

Error correction

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

n −bit message/coefficients

m	0	1	…	1	0

$encode(m)$

$\bar{\boldsymbol{m}}$	0	2046	…	2046	0

- *Decode*(x)
  - *If* $(1/4q < x < 3/4q)$ **Return** 1
  - Else **return** 0

$\bar{\boldsymbol{m}} + \boldsymbol{r}_2\boldsymbol{e}_2 + \boldsymbol{r}_1\boldsymbol{e}_1 + \mathbf{e}_3$	402	1907	…	2631	4024

$decode(m)$

$\bar{\boldsymbol{m}}$	0	1	…	1	0

Ring-LWE Encryption: Parameters

RUB

Parameter sets	n	p	σ	$\lvert \boldsymbol{c}_1, \boldsymbol{c}_2 \rvert$	|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 $(\boldsymbol{c}_1, \boldsymbol{c}_2)$
- **Public key:** one or two large polynomials $(\boldsymbol{a}, \boldsymbol{p})$
- **Secret key:** small polynomial $(\boldsymbol{r}_2)$

Tutorial Outline – Part II

RUB

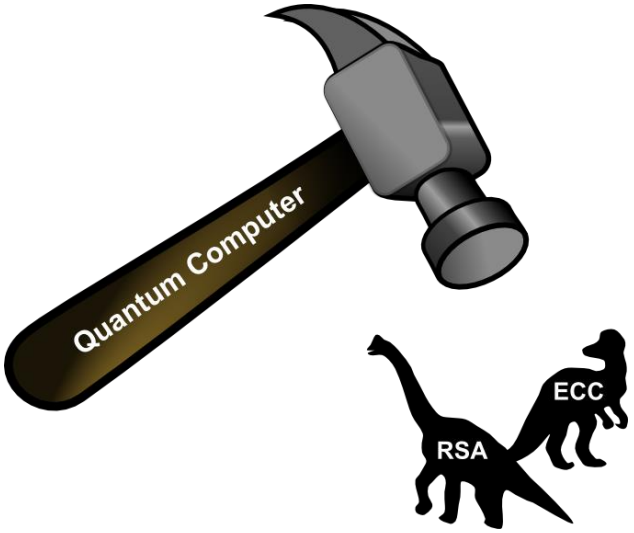
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

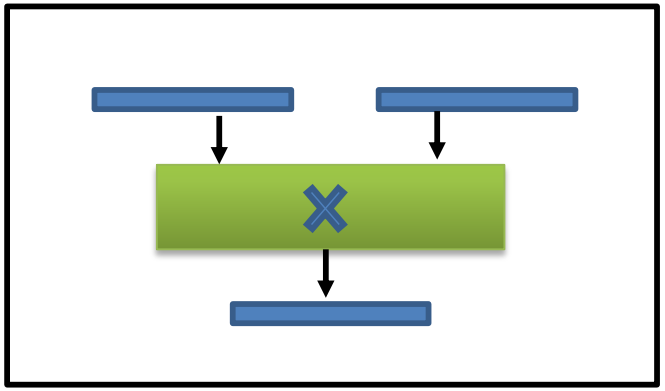
Efficient Lattice-based Implementations

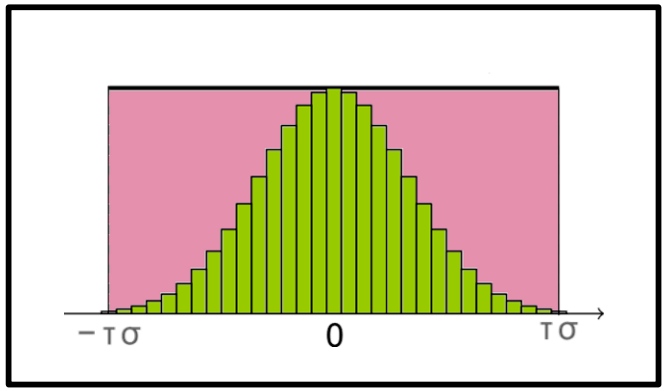
Lessons Learned



Hardware Implementation
Building Blocks for R-LWE

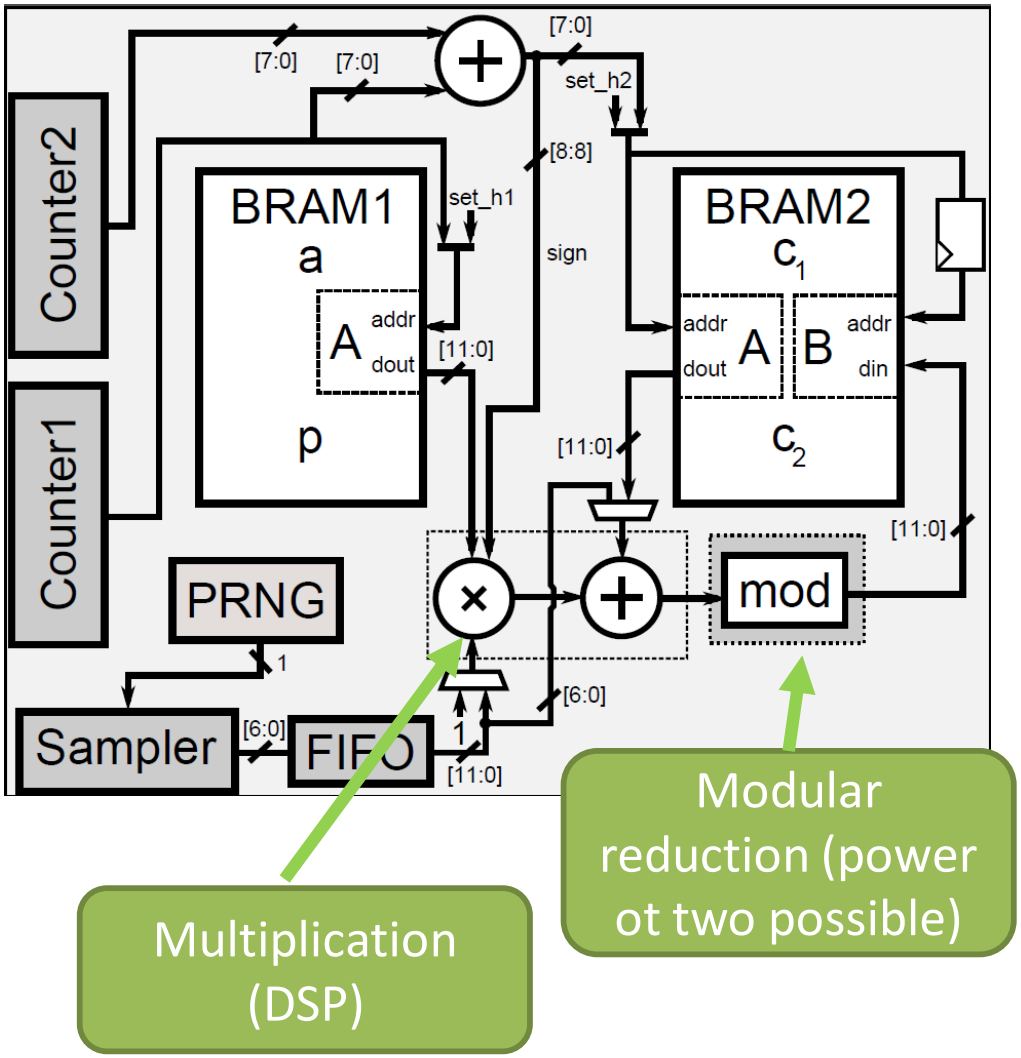
- **Two main components**
  - Polynomial multiplier for $n = \{256,512,1024\}$ over specific rings with coefficients with less than $\log2(q) < 24$ bits
  - Discrete Gaussian sampler with precisely defined precision σ





Hardware Implementation: Low-Cost Design for Xilinx Spartan-6

- Row-wise polynomial multiplication ($\boldsymbol{ae}_1/\boldsymbol{pe}_1$)
  - Simple address generation
  - Sample coefficient of $\boldsymbol{e}_1$, add row of $\boldsymbol{c}_1$ then add row of $\boldsymbol{c}_2$, add coefficient of $\boldsymbol{e}_2$ and $\boldsymbol{e}_3$
- Key and ciphertext are stored in block memory
- DSP block for arithmetic ($q \times q$-bit multipler)



Hardware Implementation: Low Area

RUB

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

Area savings by power of two modulus

Operation		LUT/LUTM/ FF/Slice	BRAM9/ DSP	MHz	Cycles	OP/s
$\mathsf{RLWEnc}^{\text{School}}_{\text{Enc}}$	(256, 4093, 8.35)	360/36/290/114	2/1	128	avg. 136 986	934
$\mathsf{RLWEnc}^{\text{School}}_{\text{Enc}}$	(256, 4096, 8.35)	282/35/238/95	2/1	144	avg. 136 212	1 057
$\mathsf{RLWEnc}^{\text{School}}_{\text{Dec}}$	(256, 4093, 8.35)	162/18/136/51	1/1	179	avg. 66 304	2 700
$\mathsf{RLWEnc}^{\text{School}}_{\text{Dec}}$	(256, 4096, 8.35)	94/18/87/32	1/1	189	avg. 66 338	2 849

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

Ring-LWE: Can we do better?

RUB

- 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?

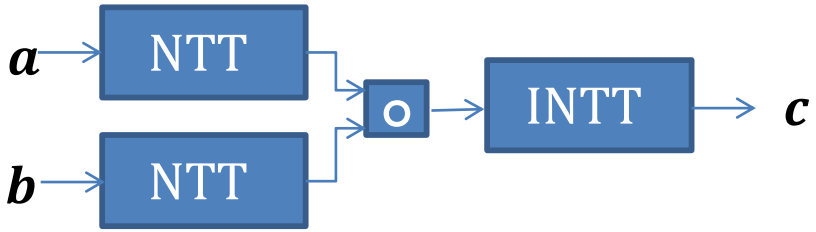
Optimization: Polynomial Multiplication based on NTT

RUB

- **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 ω the NTT is defined as:
  - Forward transformation: NTT
    - $\boldsymbol{A}[i] = \sum_{j=0}^{n-1} \boldsymbol{a}[j]\omega^{ij}, i = 0,1,\dots,n$
  - Inverse transformation: INTT
    - $\boldsymbol{a}[i] = n^{-1} \sum_{j=0}^{n-1} \boldsymbol{A}[j]\omega^{-ij}, i = 0,1,\dots,n$
- NTT exists if q is a prime, n a power of two and if $\mathrm{q} \equiv 1 \bmod 2n$
- **Example**: Ring-LWE encryption: $7681 \bmod 2 \cdot 256 = 1$

NTT for Lattice Cryptography: Convolution Theorem

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

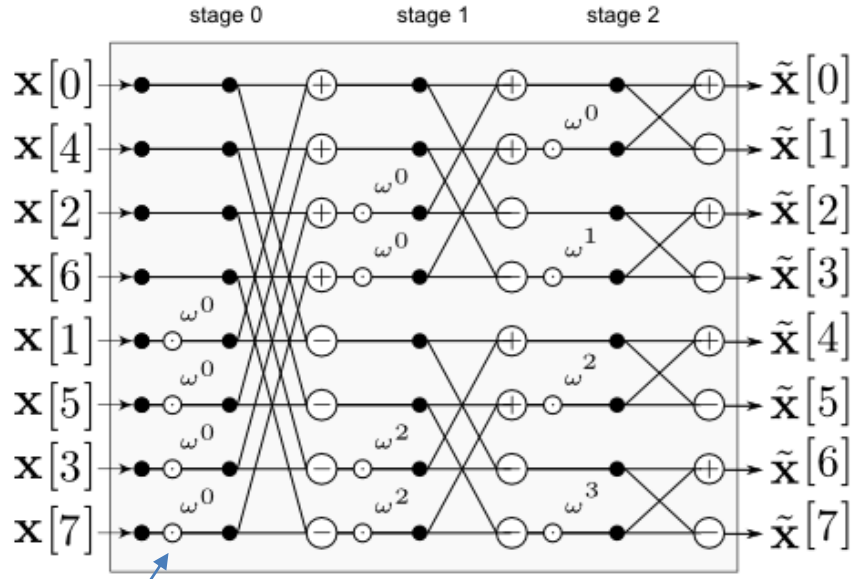


- **Negative Wrapped Convolution:**
  - Polynomial multiplication in $Z_q[x]/\langle x^n + 1\rangle$
  - Runtime $O(n \log n)$
  - No appending of zeros required (as for regular convolution)
  - Implicit polynomial reduction by $x^n + 1$

Efficient Computation of the NTT (Cooley-Tukey)

```
func NTT(a ∈ R)
    a ← BitRev(a)
    N ← n
    m ← 2
    while m ≤ N do
        s ← 0
        while s < N do
            for i to m/2 − 1 do
                N ← i · n/m
                k ← s + i
                l ← s + i + m/2
                c ← a[k]
                d ← a[l]
                a[k] ← c + ω^N d  mod q
                a[l] ← c − ω^N d  mod q
            end for
            s ← s + m
        end while
        m ← m · 2
    end while
    Return a
end func
```

twiddle factors



Multiplication by $\omega^0 = 1$

$\mathrm{NTT}^{CT}_{bo \to no}$

- Bitreversal required ($\mathrm{NTT}_{no \to bo}$)
- 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

RUB

Scheme	Device	Resources (LUT/FF/BRAM/DSP)	OP	Speed
RLWEenc(area, **our work**)	S6LX9	396/290/2/1	Enc	1.07 ms
($n = 256, q = 7681$)	S6LX9	180/136/1/1	Dec	370.00 µs
RLWEenc [RVM+14]	V6LX75T	1 349/860/2/1	Enc	20.10 µs
($n = 256, q = 7681$)	V6LX75T	-	Dec	9.10 µs
RLWEenc [GFS+12]	V6LX240T	146 718/82 463/-/-	Gen	-
($n = 256, q = 7681$)	V6LX240T	298 016/143 396/-/-	Enc	8.05 µs
	V6LX240T	124 158/65 174/-/-	Dec	8.10 µs
Niederreiter [HG12]	V6LX240T	888/875/17/-	Enc	0.66 µs
(80-bit security)	V6LX240T	9 409/12 861/9/-	Dec	57.78 µs
NTRU [KY09]	XCV1600E	27 292/5 160	Enc	1.54 µs
		-	Dec	1.41 µs
1024-bit mod. Exp. [Suz07]	XC4VFX12	3 937 slice/17 DSP	-	1.71 ms
ECC-P224 [GP08]	XC4VFX12	1 825/1 892/11/26	-	365.10 µs
ECC-B233 [RRM12]	XC5VLX85T	18 097 LUT/5 644 slice	-	12.30 µs

NTT is very fast but still quite small

Lots of improvement since [GFS+12]

Tutorial Outline – Part II

RUB

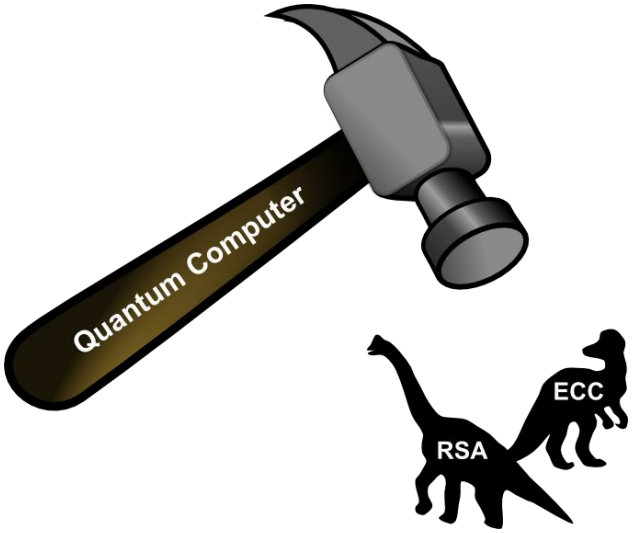
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned



Lessons Learned

RUB

- **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

SAFE crypto
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

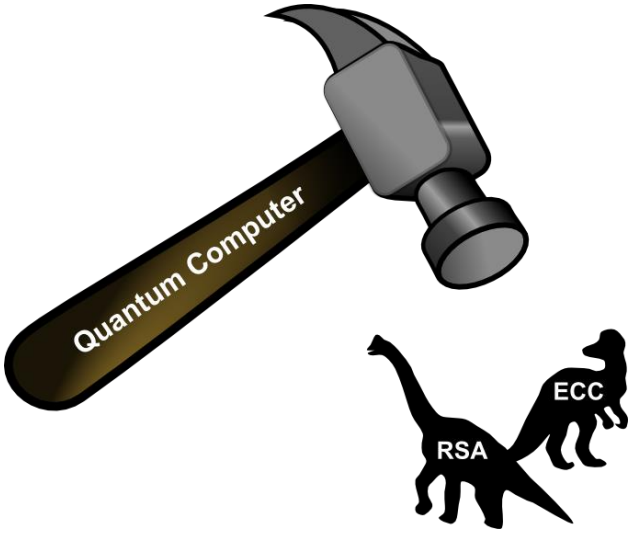
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned



Recall: McEliece Encryption Scheme [1978]

RUB

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 $\mathrm{e} \in_R \mathbb{F}_2^n$, $\mathrm{wt}(\mathrm{e}) \leq t$

$\mathrm{x} \leftarrow mG' + \mathrm{e}$

Decryption

Let Ψ_H be a t-error-correcting decoding algorithm.

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

Extract m by computing $m \cdot S \cdot S^{-1}$

(QC-)MDPC McEliece

RUB

Encryption

Message $m \in F_2^k$, error vector $e \in_R F_2^n$, $wt(e) \leq t$

$\mathrm{x} \leftarrow mG + e$

Decryption

Let Ψ_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

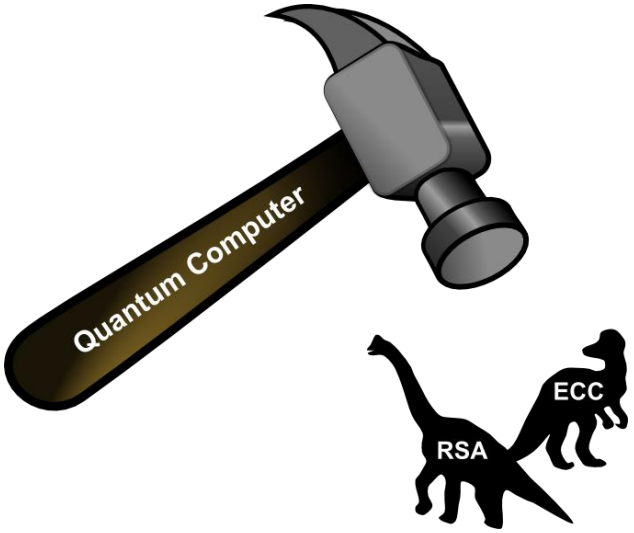
Code-based Cryptography

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

RUB

- **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 $\boldsymbol{H} = [\boldsymbol{H_0}|\boldsymbol{H_1}]$

- 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 $\boldsymbol{G} = [\boldsymbol{I}|\boldsymbol{Q}]$

- 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

RUB

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

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

- 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

RUB

- **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

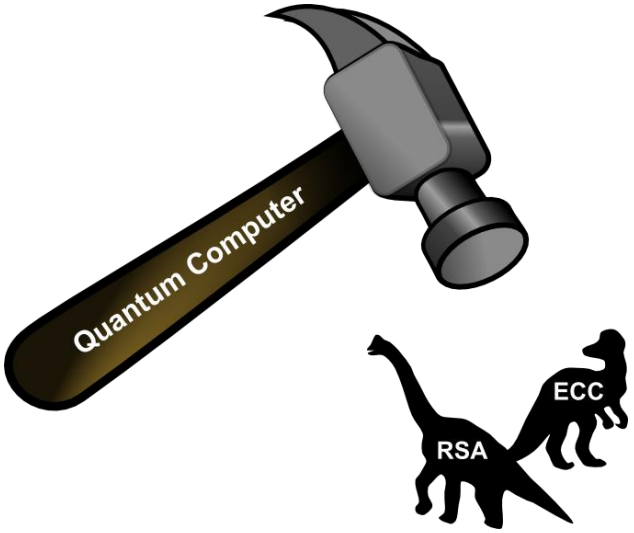
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

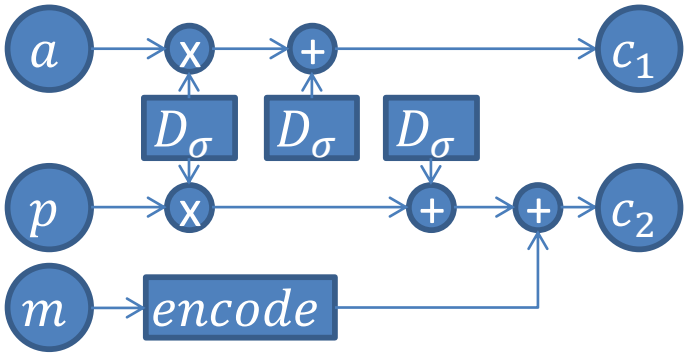
Lessons Learned



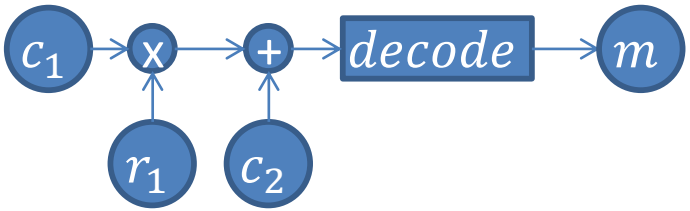
Ring-LWE Encryption Scheme [LP11/LPR10]

Gen: Choose $\boldsymbol{a} \leftarrow R$ and $\boldsymbol{r}_1, \boldsymbol{r}_2 \leftarrow D_\sigma$; *pk*: $\boldsymbol{p} = \boldsymbol{r}_1 - \boldsymbol{a} \cdot \boldsymbol{r}_2 \in \mathrm{R}$; *sk*: $\boldsymbol{r}_2$

Enc$(\boldsymbol{a}, \boldsymbol{p}, m \in \{0,1\}^n)$: $\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3 \leftarrow D_\sigma$.
$\bar{\boldsymbol{m}} = encode(m)$. Ciphertext:
$[\boldsymbol{c}_1 = \boldsymbol{a} \cdot \boldsymbol{e}_1 + \boldsymbol{e}_2, \boldsymbol{c}_2 = \boldsymbol{p} \cdot \boldsymbol{e}_1 + \boldsymbol{e}_3 + \bar{\boldsymbol{m}}]$



Dec$(c = [\boldsymbol{c}_1, \boldsymbol{c}_2], \boldsymbol{r}_2)$: Output
$decode(\boldsymbol{c}_1 \cdot \boldsymbol{r}_2 + \boldsymbol{c}_2)$



Correctness: $\boldsymbol{c}_1\boldsymbol{r}_2 + \boldsymbol{c}_2 = (\boldsymbol{a}\boldsymbol{e}_1 + \boldsymbol{e}_2)\boldsymbol{r}_2 + \boldsymbol{p}\boldsymbol{e}_1 + \boldsymbol{e}_3 + \bar{\boldsymbol{m}}$
$= \boldsymbol{r}_2\boldsymbol{a}\boldsymbol{e}_1 + \boldsymbol{r}_2\boldsymbol{e}_2 + \boldsymbol{r}_1\boldsymbol{e}_1 - \boldsymbol{r}_2\boldsymbol{a}\boldsymbol{e}_1 + \boldsymbol{e}_3 + \bar{\boldsymbol{m}} = \bar{\boldsymbol{m}} + \boldsymbol{r}_2\boldsymbol{e}_2 + \boldsymbol{r}_1\boldsymbol{e}_1 + \boldsymbol{e}_3$

large

small

Ring-LWE Encryption: Parameters

RUB

Parameter sets	n	p	σ	$\lvert \boldsymbol{c}_1, \boldsymbol{c}_2 \rvert$	|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 $(\boldsymbol{c}_1, \boldsymbol{c}_2)$
- **Public key:** one or two large polynomials $(\boldsymbol{a}, \boldsymbol{p})$
- **Secret key:** small polynomial $(\boldsymbol{r}_2)$

Tutorial Outline – Part III

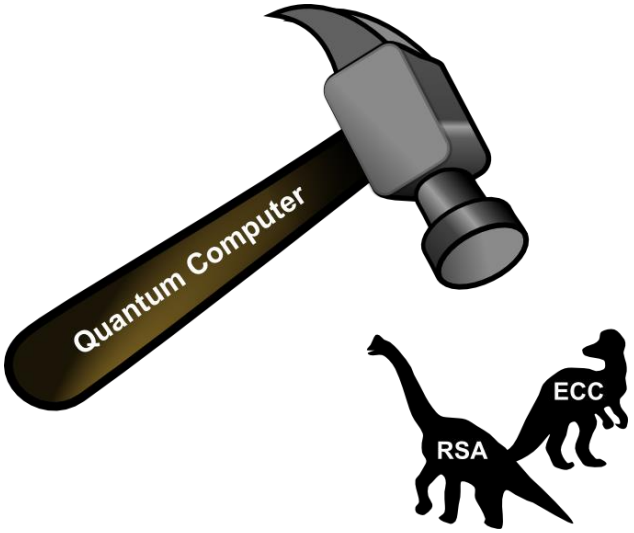
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned



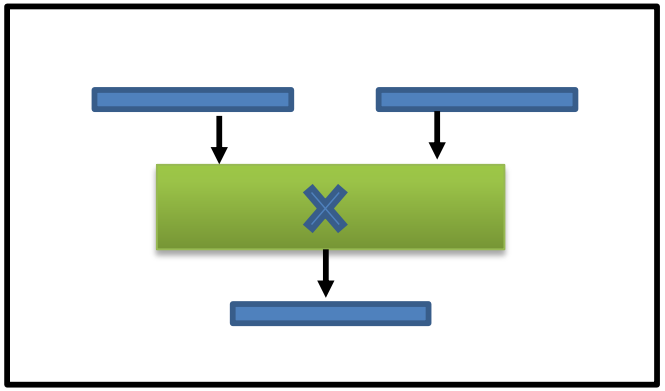
Simple Implementation of RLWE-Encryption

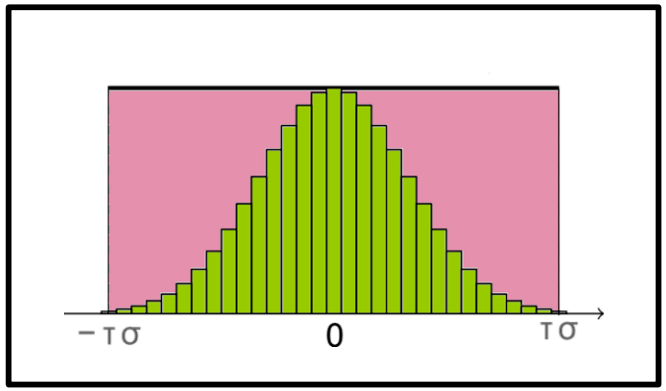
```
void encrypt(poly a, poly p, unsigned char * plaintext, poly c1, poly c2)
{
    int i,j; poly e1,e2,e3;
    gauss_poly(e1); gauss_poly(e2); gauss_poly(e3);

    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]);
    }
}
```

This has to be fast

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 $\log2(q) < 24$ bits
  - Discrete Gaussian sampler with precisely defined precision σ and tail cut τ





Intermediate Results

RUB

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

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

- Schoolbook multiplication (SchoolMul)
- Encryption is two multiplications and decryption one

Recall Improvement: Polynomial Multiplication with NTT

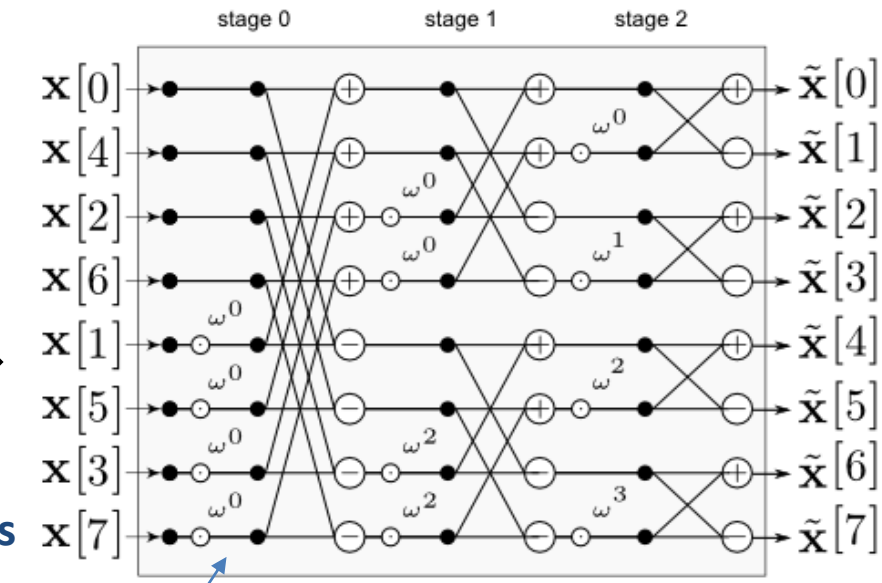
- Number Theoretic Transform (NTT) is a discrete Fourier transform (DFT) defined over a finite field or ring. For a given primitive n-th root of unity ω the NTT is defined as:
  - Forward transformation: NTT
    - $\boldsymbol{A}[i] = \sum_{j=0}^{n-1} \boldsymbol{a}[j]\omega^{ij}, i = 0,1,\dots,n$
  - Inverse transformation: INTT
    - $\boldsymbol{a}[i] = n^{-1} \sum_{j=0}^{n-1} \boldsymbol{A}[j]\omega^{-ij}, i = 0,1,\dots,n$
- NTT exists if q is a prime, n a power of two and if

$$\mathrm{q} \equiv 1 \bmod 2n$$

Efficient Computation of the NTT (Textbook)

```
func NTT(a ∈ R)
    a ← BitRev(a)
    N ← n
    m ← 2
    while m ≤ N do
        s ← 0
        while s < N do
            for i to m/2 − 1 do
                N ← i · n/m
                k ← s + i
                l ← s + i + m/2
                c ← a[k]
                d ← a[l]
                a[k] ← c + ω^N d  mod q
                a[l] ← c − ω^N d  mod q
            end for
            s ← s + m
        end while
        m ← m · 2
    end while
    Return a
end func
```

twiddle factors



Multiplication by $\omega^0 = 1$

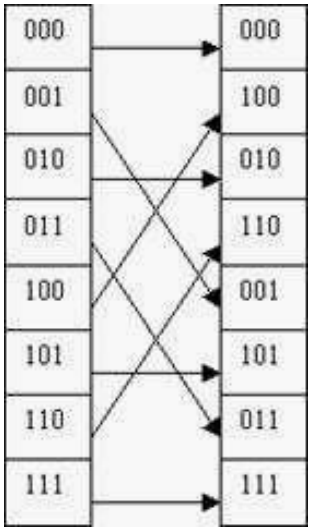
$\mathrm{NTT}^{CT}_{bo \to no}$

- Bitreversal required ($\mathrm{NTT}_{no \to bo}$)
- Precomputation of powers of ω possible
- Arithmetic is basically multiplication and reduction modulo q ($\frac{n}{2}\log_2(n)$ times)

Optimization of NTT Computation

Removal of expensive "helper" functions

- Problem: Permutation (Bitrev) of polynomial is expensive
  - "Standard" $\mathrm{NTT}_{bo \to no}$ 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: $\mathrm{NTT}_{no \to bo}$
  - Bitreversed to natural for inverse: $\mathrm{INTT}_{bo \to no}$
  - No bitreversal necessary anymore:
    - $\mathrm{INTT}_{bo \to no}(\mathrm{NTT}_{no \to bo}(\boldsymbol{a}) \circ \mathrm{NTT}_{no \to bo}(\boldsymbol{b}))$

Optimization of NTT Computation

RUB

Removal of expensive "helper" functions

- <u>Problem</u>: Multiplication by scalar n^{-1} in inverse transformation is expensive

- <u>Solution:</u> In lattice-based crypto we usually multiply by pretransformed constants (e.g., $\widetilde{\boldsymbol{a}}$, $\widetilde{\boldsymbol{p}}$, or $\boldsymbol{r}_2$)
  - Put n^{-1} into these constants
  - Multiplication by scalar does not change much as
    - $\mathrm{x} \cdot \mathrm{NTT}(\boldsymbol{a}) \Leftrightarrow \mathrm{NTT}(x \cdot \boldsymbol{a})$
  - Store $\widetilde{\boldsymbol{a}}' = n^{-1}\widetilde{\boldsymbol{a}}$

Optimization of NTT Computation

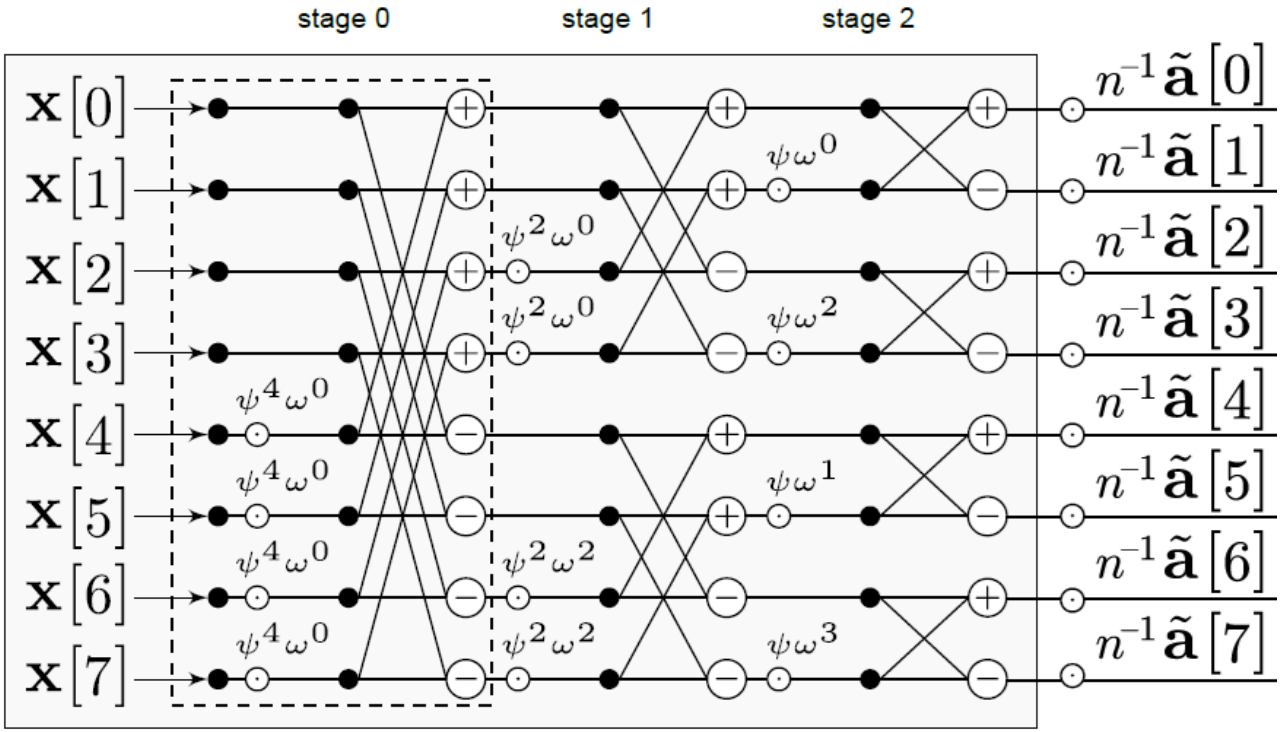
RUB

Removal of expensive "helper" functions

- Problem: Multiplication by powers of ψ and ψ^{-1} (PowMul) is expensive

- Solution: Merge powers of ψ into twiddle factors
  - Only possible with forward transformation and current butterfly (see next picture)

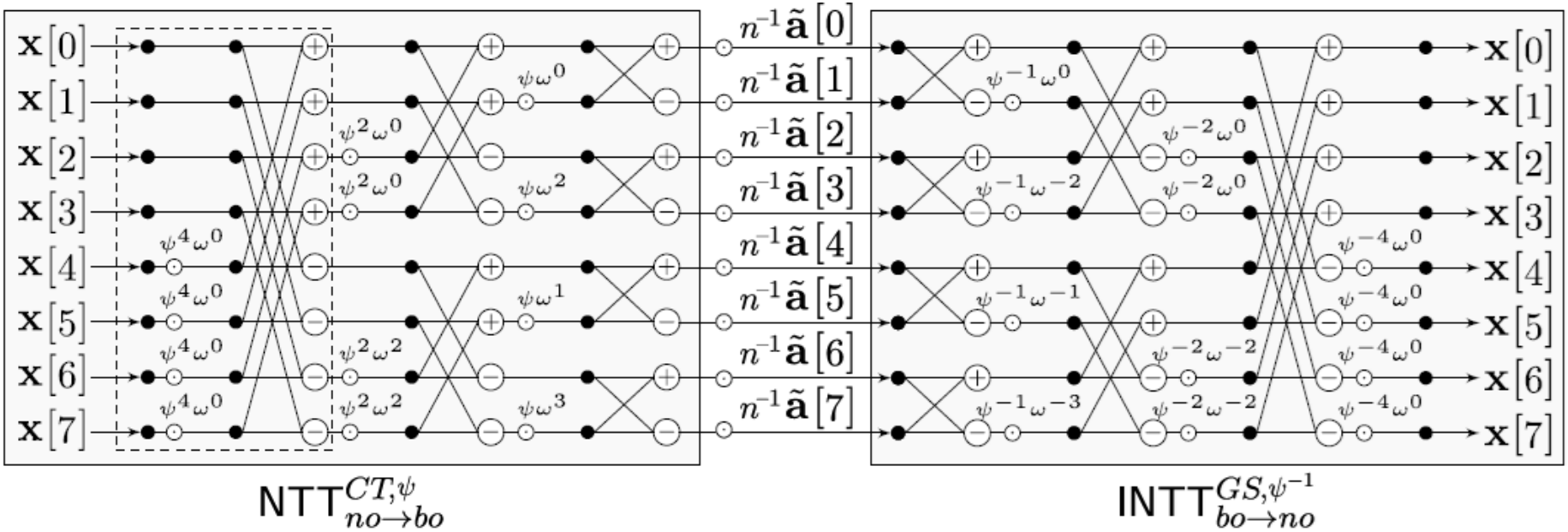
Optimization of NTT Computation



$\mathrm{NTT}^{CT,\psi}_{no\rightarrow bo}$

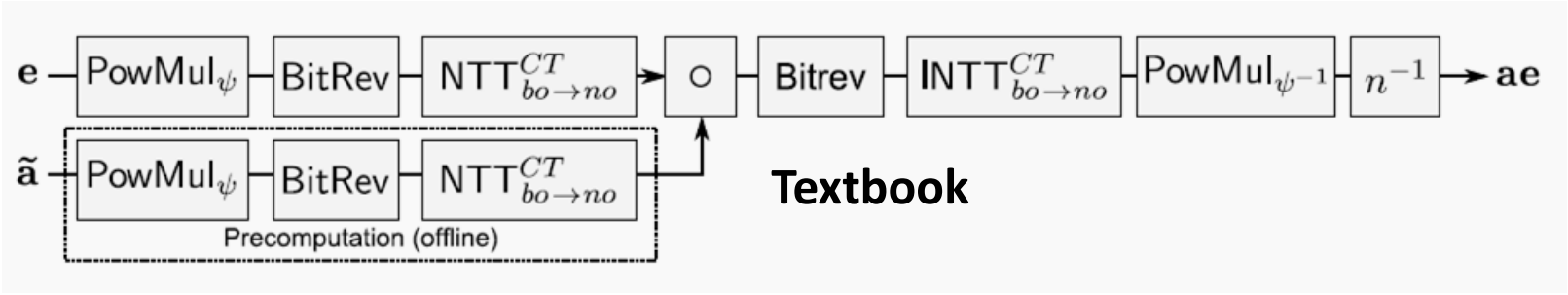
- Combines all tricks for forward transformation
- We cannot merge powers of ψ^{-1}; We have to multiply after transformation is finished

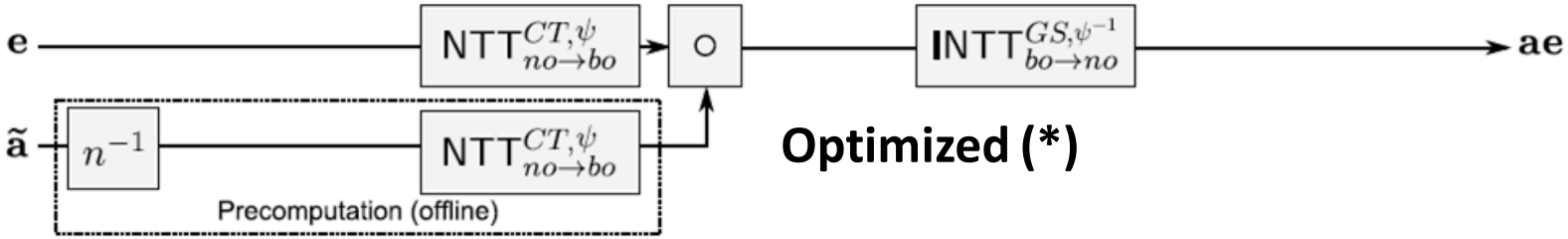
Optimization of NTT Computation



- Usage of Gentlemen-Sande (GS) butterfly instead of Cooley-Tukey (CT) allows merging of inverse multiplication by powers of ψ^{-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

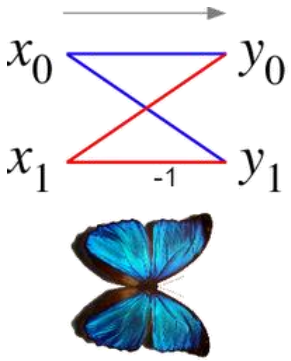
(*) FFT people probably know most of these tricks

Optimization of NTT Computation

RUB

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 $\bmod q$ 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)

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

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

Ring-LWE Encryption on ATXmega Family

RUB

Scheme	Device	Operation	Cycles		OP/s	
RLWEenc-Ia ($n = 256$)	AX128	Enc/Dec	874,347	215,863	36.60	148.24
RLWEenc-IIa ($n = 512$)	AX128	Enc/Dec	2,196,945	600,351	14.57	53.30
RLWEenc-Ia ($n = 256$) [LSR+15]	AX128	Enc/Dec	666,671	299,538	48	106.83
RLWEenc-IIa ($n = 512$) [LSR+15]	AX128	Enc/Dec	2,721,372	700,999	11.76	45.65
RLWEenc-Ia ($n = 256$) [BSJ14]	AT64	Enc/Dec	3,042,675	1,368,969	2.63	5.84
RLWEenc-Ia ($n = 256$) [BJ14]	AX64	Enc/Dec	5,024,000	2,464,000	6.37	12.98
QC-MDPC [HvMG13]	AX128	Enc/Dec	26,767,463	86,874,388	1.20	0.36
RSA-1024 [GPW+04]	AT128	Enc/Dec	3,440,000	87,920,000	2.33	0.09
RSA-1024† [LGK10]	AT128	priv. key	75,680,000		0.11	
Curve25519† [DHH+ar]	AT2560	Point mul.	13,900,397		1.15	
ECC-ecp160r1 [GPW+04]	AT128	Point mul.	6,480,000		1.23	

Schoolbook was 12 million

Code size is not significantly increased

Sampler is the bottleneck

[POG15] High-Performance Ideal Lattice-Based Cryptography on 8-bit ATxmega Microcontrollers, Thomas Pöppelmann, Tobias Oder, and Tim Güneysu, Latincrypt'15

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

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

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

[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

RUB

- **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

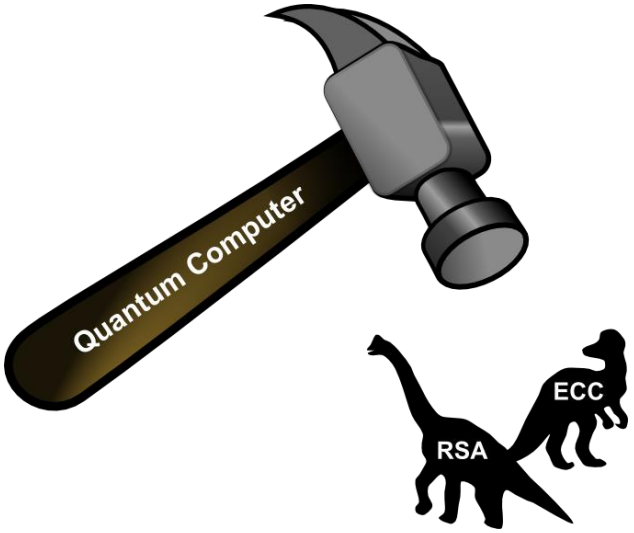
Code-based Cryptography

Efficient Code-based Implementations

Lattice-based Cryptography

Efficient Lattice-based Implementations

Lessons Learned



Lessons Learned

RUB

- **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**

  http://www.seceng.rub.de/research/projects/pqc/

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

HORIZON 2020

PQCRYPTO
ICT-645622

SAFE crypto
ICT-644729

Part III: Post Quantum Cryptography in Embedded Software

Tutorial@CHES 2017 - Taipei

Tim Güneysu
Ruhr-Universität Bochum & DFKI

04.10.2017