RUHR-UNIVERSITÄT BOCHUM

RUB

Tweaking Code-Based Cryptography for Embedded Systems

DIMACS Workshop on The Mathematics of Post-Quantum Cryptography

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

1/12/2015

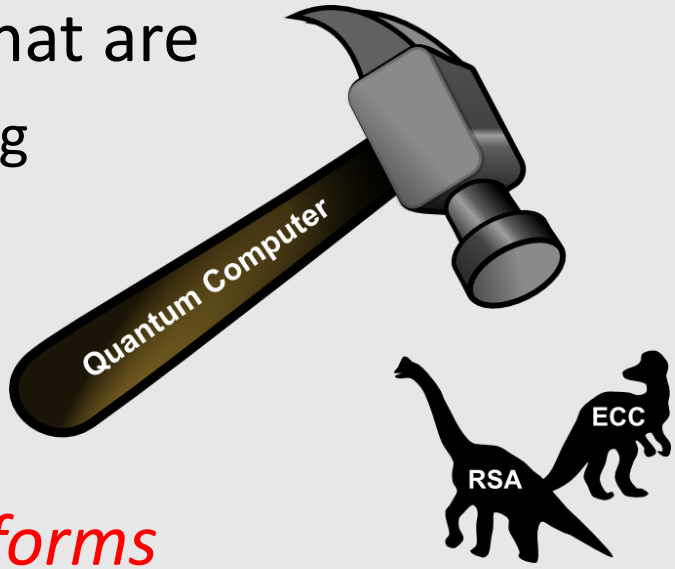
Motivation

- **High demand for security in the Internet of Things (IoT)**
- **Requirements**
  - Highly embedded/cost-sensitive
  - Long life-time/security
  - Diversity of target platforms
  - Simple physical accessibility
- **Consequences**
  - Quantum-computer resistant cryptography
  - Implementations for a wide range of cheap embedded devices

Motivation

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

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



➔ *Code-based Crypto on Embedded Platforms*

Overview

Motivation

Background

Efficient Decoding Techniques

Implementing QC-MDPC McEliece

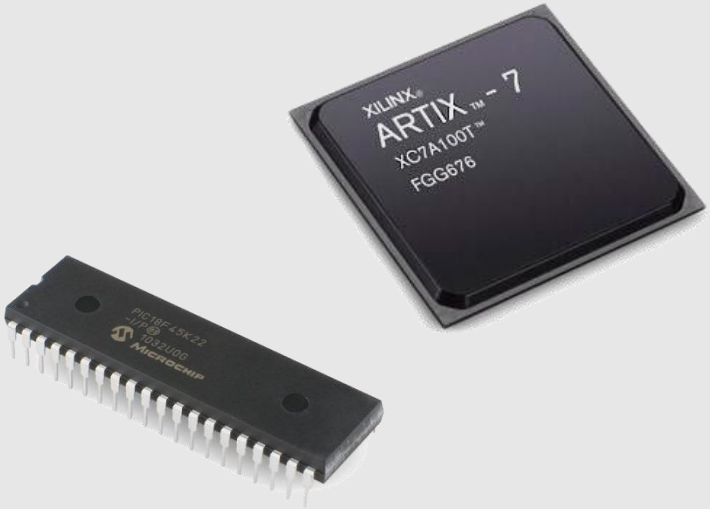
Side-Channel Attacks

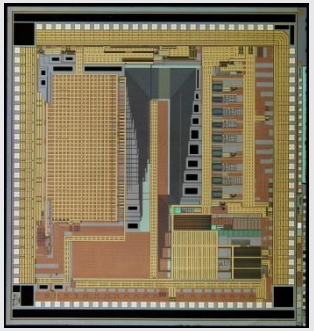
Countermeasures

Cryptography on Embedded Devices

Common computing platforms of embedded devices

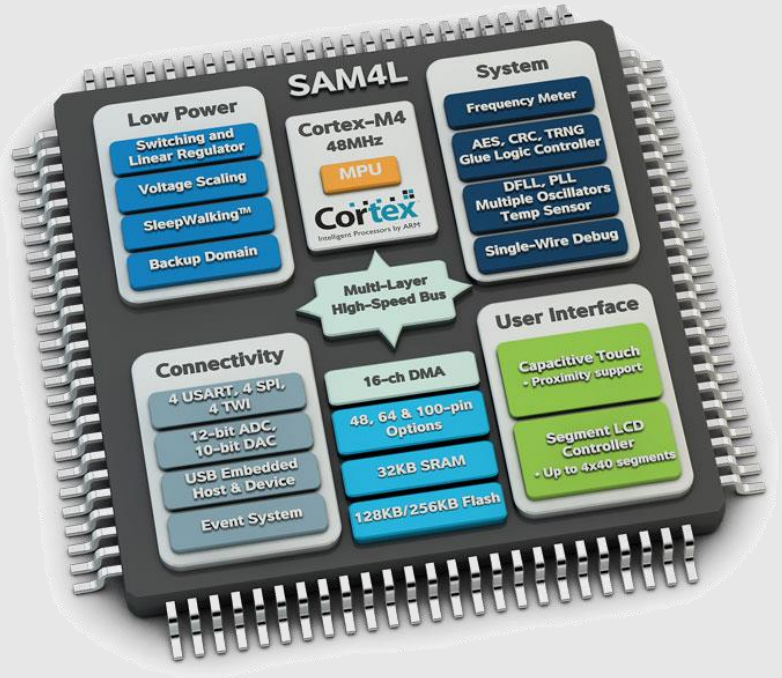
- *Microcontrollers (µC)*
  - Small 8/16/32-bit CPU, small RAM (≈ 512B-256KB), a bit more Flash (≈ 4KB-1MB)
- *Reconfigurable Hardware (FPGA)*
  - LUT-based logic functions, flip-flops, some 18/36 kBit block memories and DSP units
- Application-Specific Integrated Circuits (ASIC)
  - Dedicated hardware design of an individual application



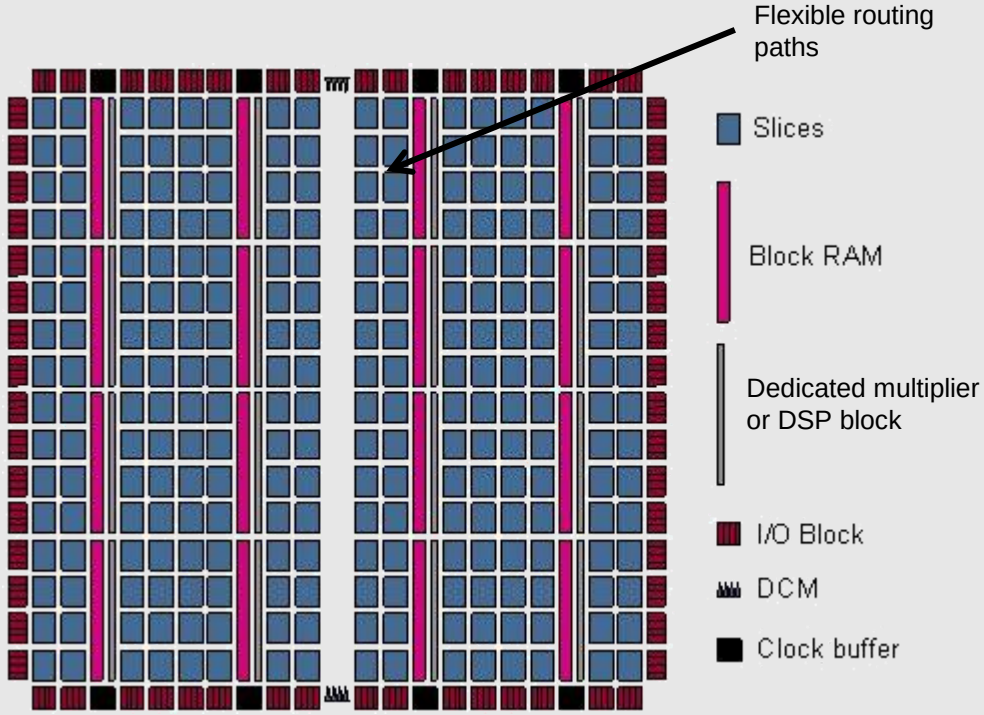


Cryptography on Embedded Devices

Microcontroller Architecture
AVR & *ARM M4 architectures*



FPGA Architecture
Altera/Xilinx FPGA

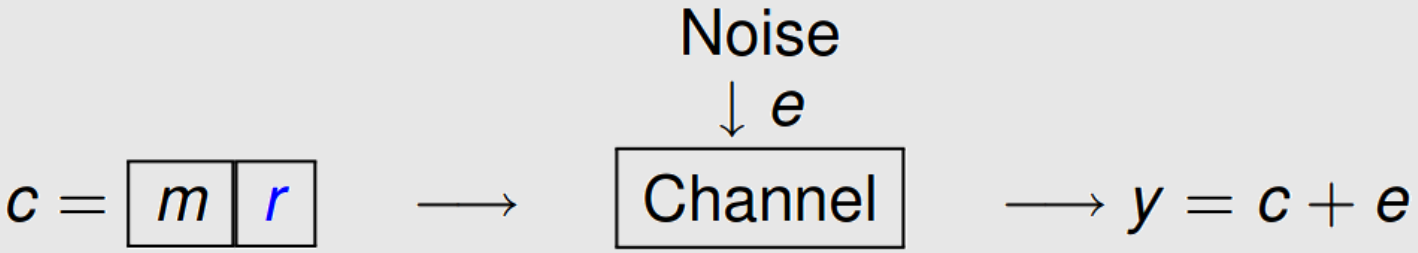


A slice contains

- 2-4 Look-Up Tables (LUT) as logic function generators
- 2-8 flip flops for data storage

Cryptography with Linear Codes?

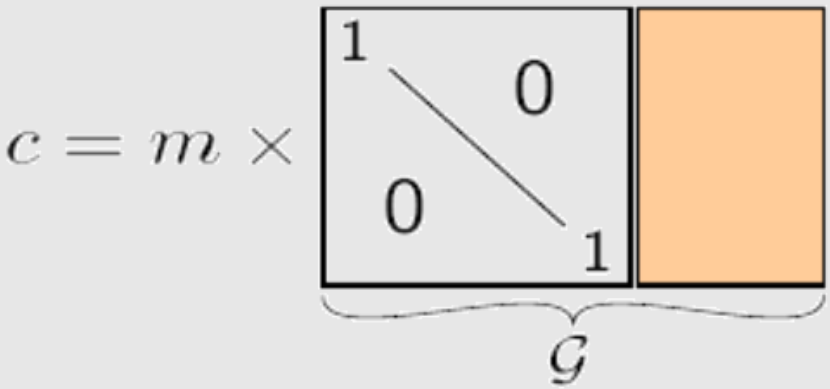
- Error-Correcting Codes are well-known in a large variety of applications
- Detection/Correction of errors in noisy channels by adding redundancy



- **Observation**:
  Some problems in code-based theory are NP-complete
  ➔ **Possible Foundation of Code-Based Cryptosystems (CBC)**

Linear Codes and Cryptography

- **Generator** and **parity check matrices** for encoding and decoding
- Matrices in **systematic form** minimize time and storage



Matrix size of G:
k x n

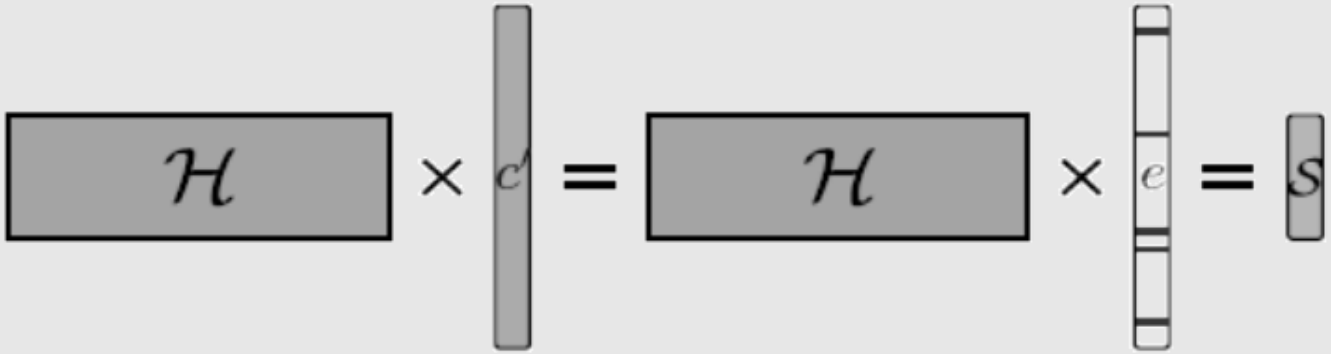
- Rows of G form a basis for the code **C[n, k, d]** of **length n** with **dimension k** and **minimum distance d**

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**



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

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

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

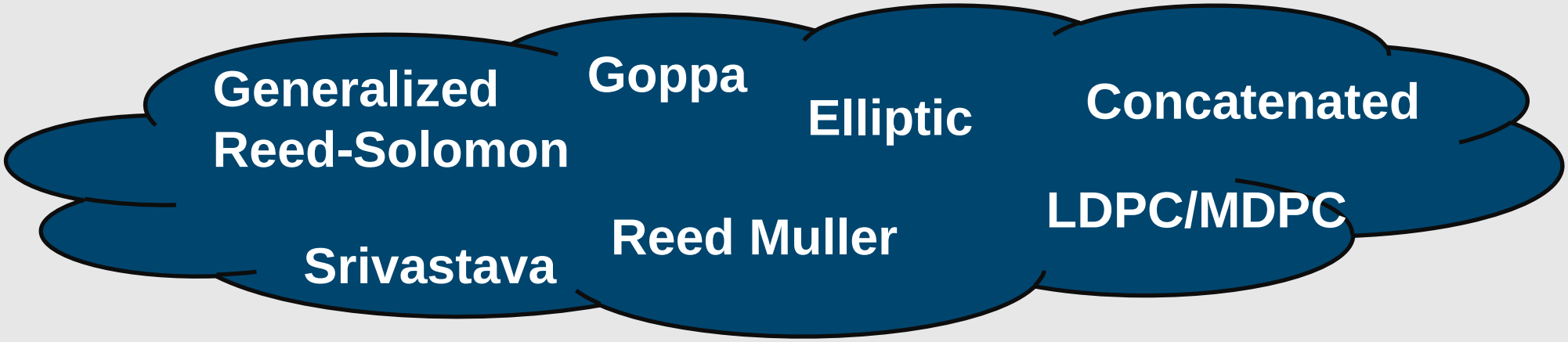
Security Level	**Parameters** (n, k, t), errors added	**Size** K_{pub} in KBits	**Size** K_{sec} $(g(z) \mid \mathcal{L} \mid M^{-1})$ KBits
Short-term (60 bit)	$(1024, 644, 38), 38$	239	$(0.37 \mid 10 \mid 141)$
Mid-term I (80 bit)	$(2048, 1751, 27), 27$	507	$(0.29 \mid 22 \mid 86)$
Mid-term II (128 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**

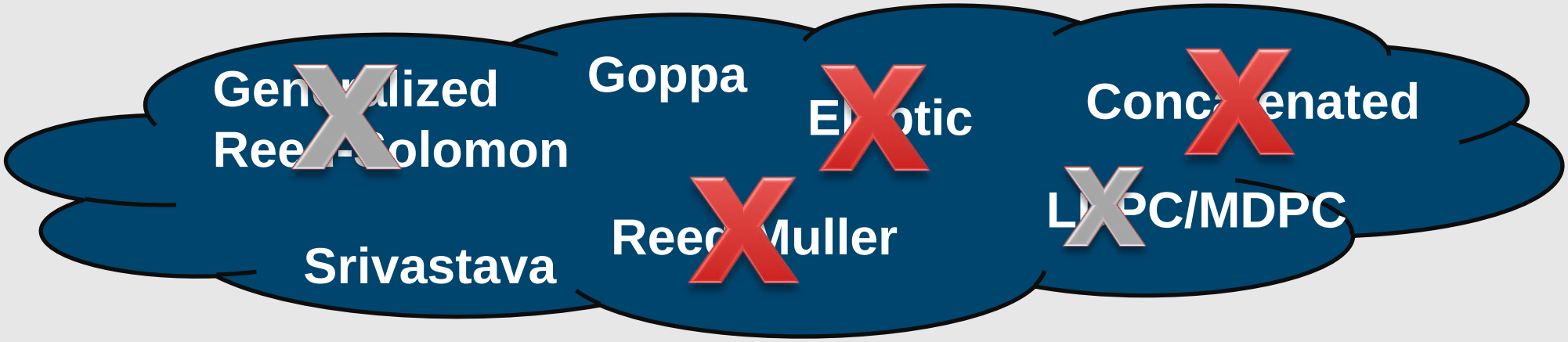
Code-based Cryptosystems

Suitable codes for code-based cryptography?



Code-based Cryptosystems

Suitable codes for code-based cryptography?

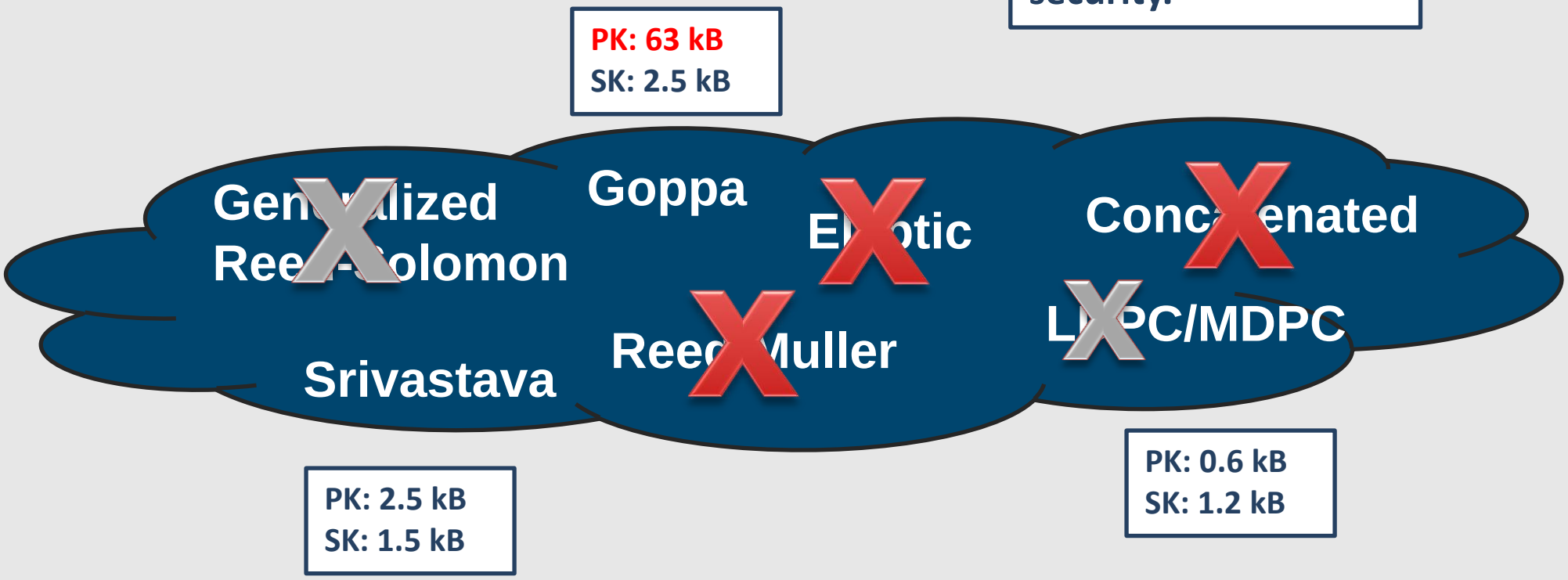


See Anja‘s and Nicolas‘ talks on Wednesday!

Code-based Cryptosystems

Suitable codes for code-based cryptography?

Key sizes for ≈ 80-bit equivalent symmetric security.



See Anja‘s and Nicolas‘ talks on Wednesday!

QC-MDPC Codes for Cryptography [MTSB13]

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

Code/Key Generation

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

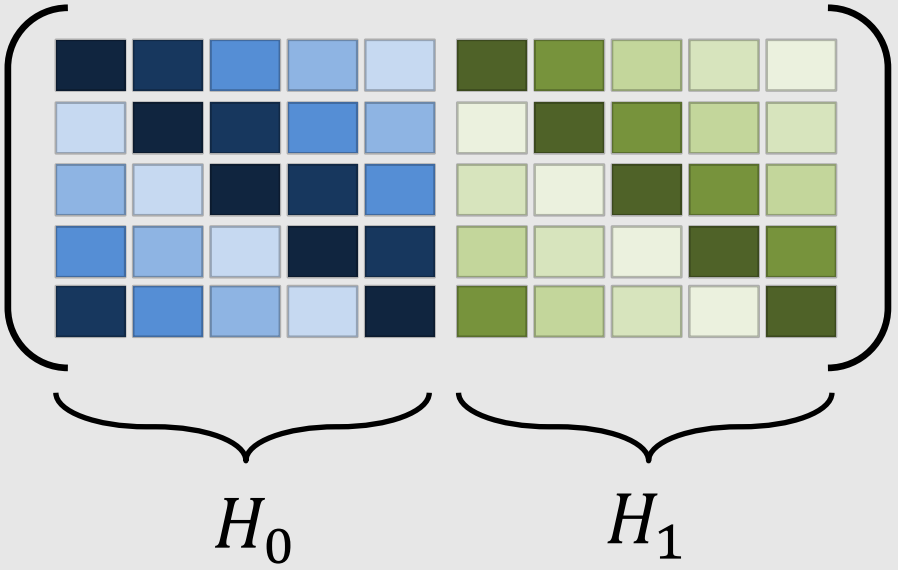
$$\mathrm{Q} = \begin{pmatrix} (H_{n_0-1}^{-1} * H_0)^T \\ (H_{n_0-1}^{-1} * H_1)^T \\ \ldots \\ (H_{n_0-1}^{-1} * H_{n_0-2})^T \end{pmatrix}$$

See Marco's talk!

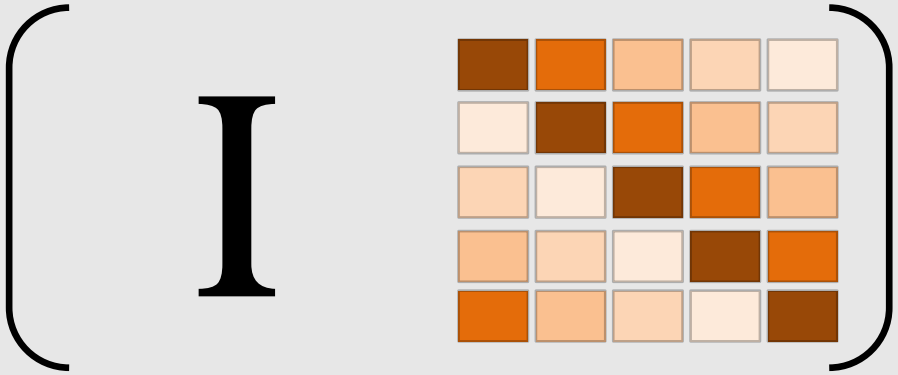
Background on QC-MDPC Codes

Parity check matrix H

$n_0 = 2$



Generator matrix G



(QC-)MDPC McEliece

Encryption

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

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

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]

$$n_0 = 2, n = 9602, r = 4801, w = 90, t = 84$$

Overview

Motivation

Background

Efficient Decoding Techniques

Implementing QC-MDPC McEliece

Side-Channel Attacks

Countermeasures

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]

Improving the Syndrome Recomputation

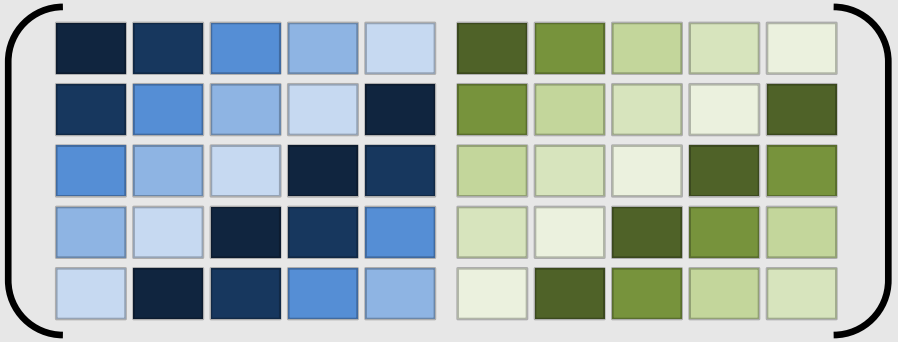
Observations

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

Proposed Optimization

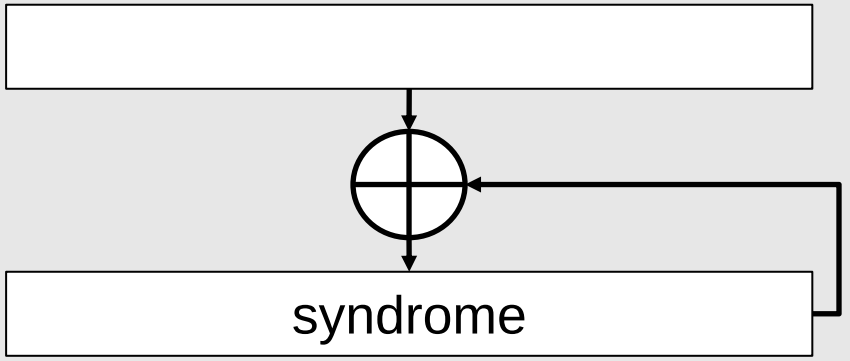
- Syndrome does not change arbitrarily!

$$s_{new} = s_{old} + h_j$$



→ Tracking changes allows to omit syndrome recomputation

→ Decoding based on up-to-date syndrome



Improving the Error-Correcting Capability

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

Proposed Optimization (adaptive thresholds)

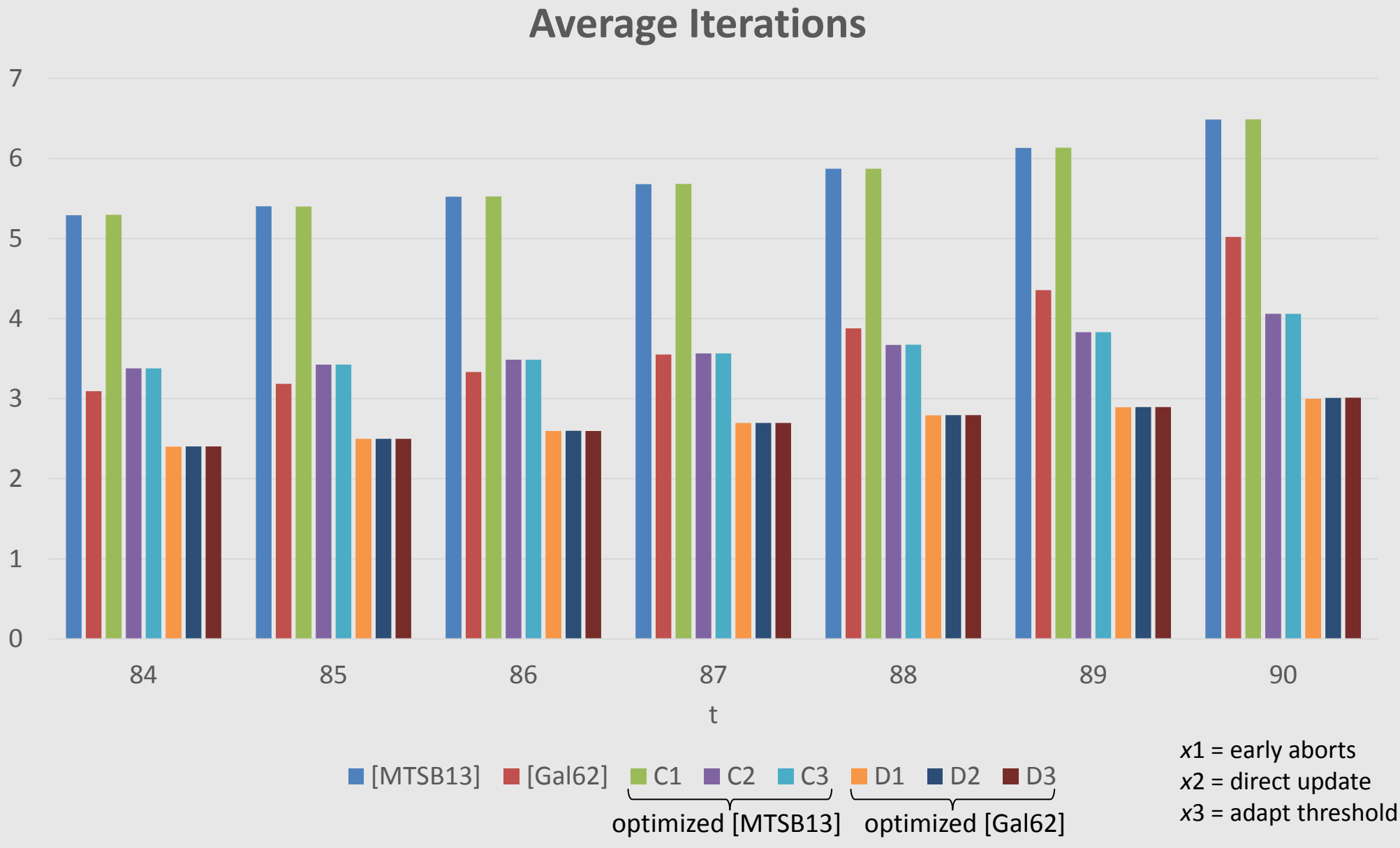
- Increment precomputed thresholds after decoding failure and restart
- If decoding fails again, increment Δ up to some fixed Δ_{max}
- Achieved best results for $\Delta = 1$ and incrementing $\Delta = \Delta + 1$
- Similar approach to $b = max_{upc} - \delta$ [MTSB13]

Benchmarking

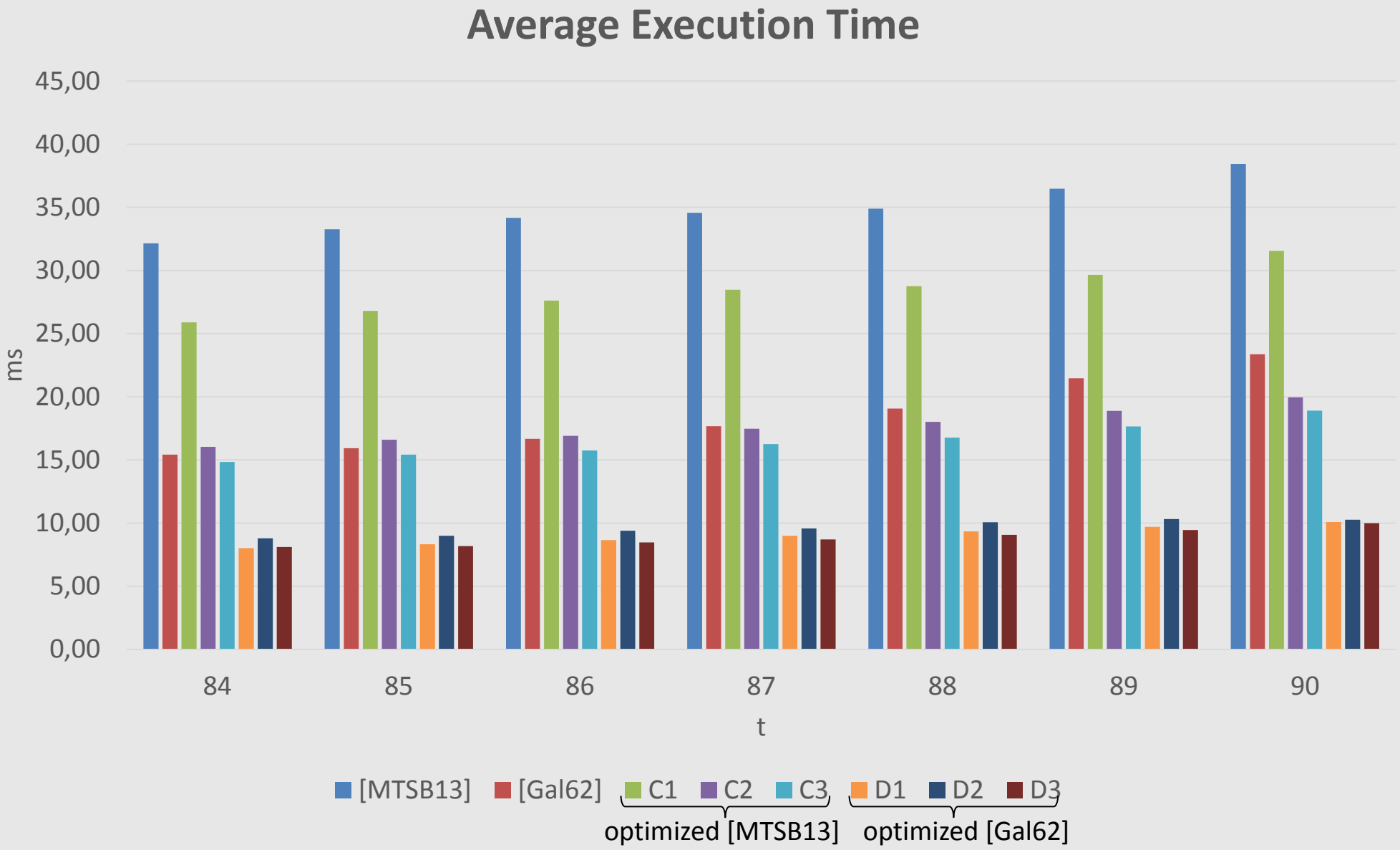
- **Empirical study on several decoder variants**
  - Direct vs. temporary syndrome update
  - Precomputed vs. adaptive thresholds
  - Modifying thresholds upon decoding failures

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

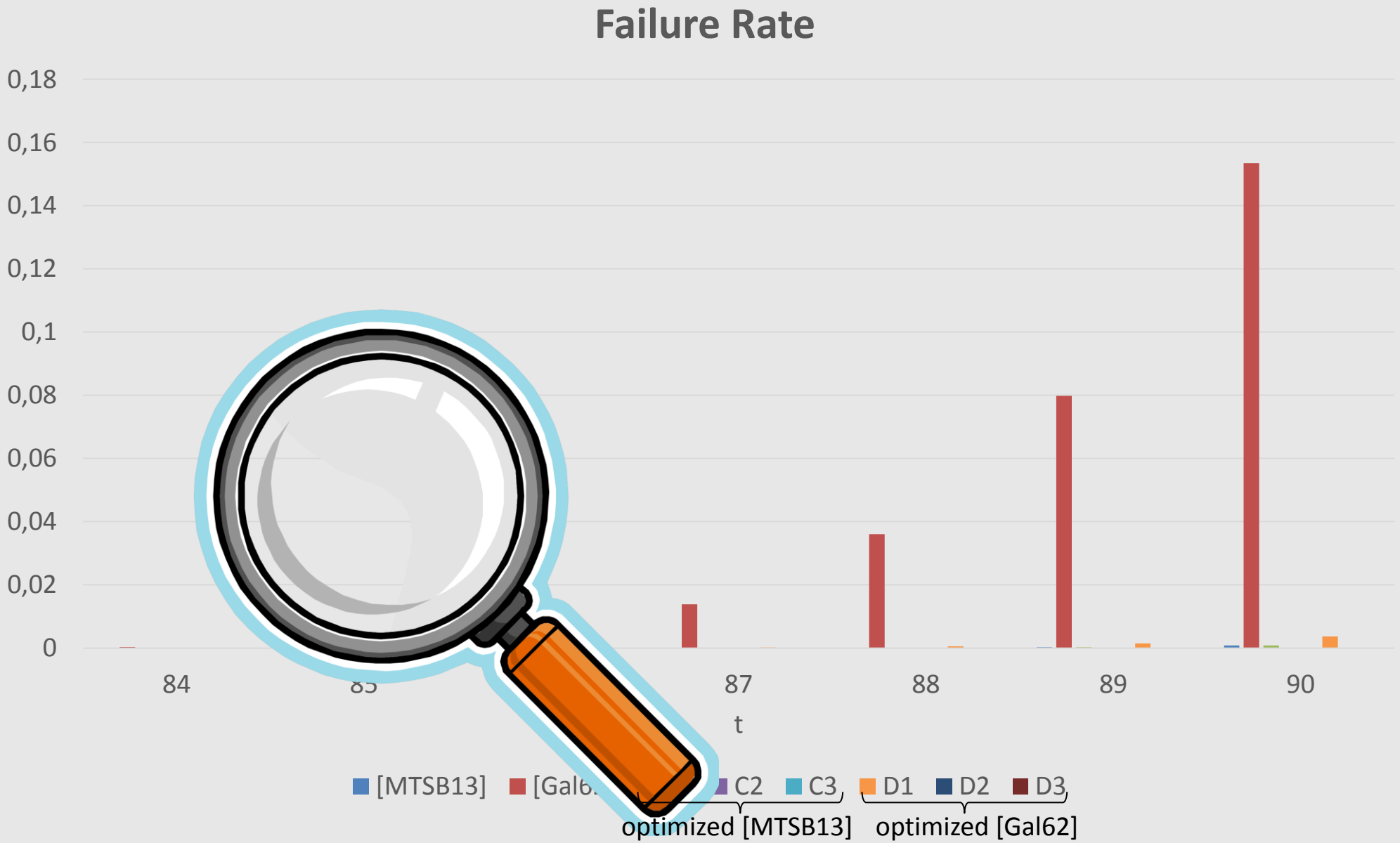
Decoder Evaluation Results



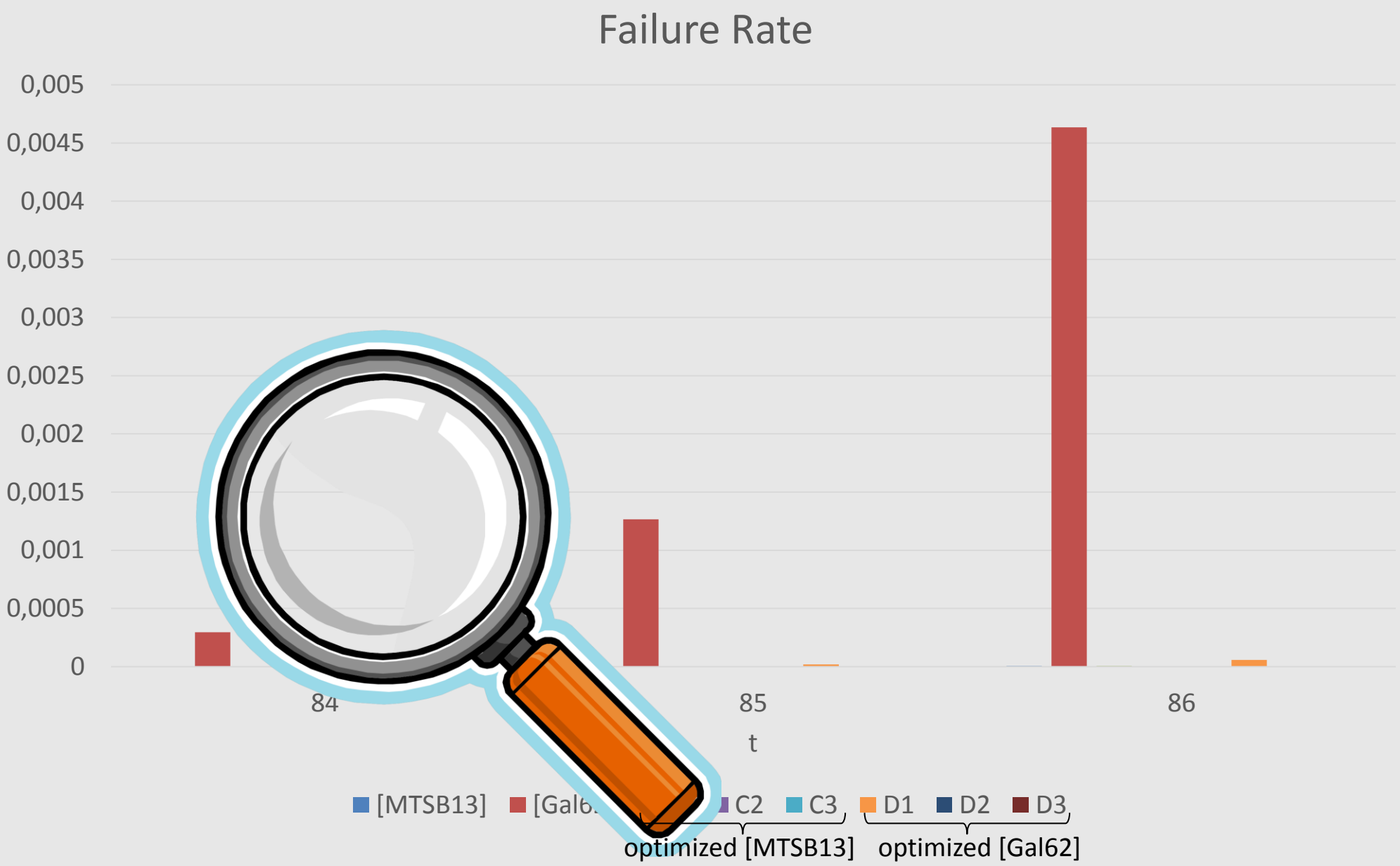
Decoder Evaluation Results



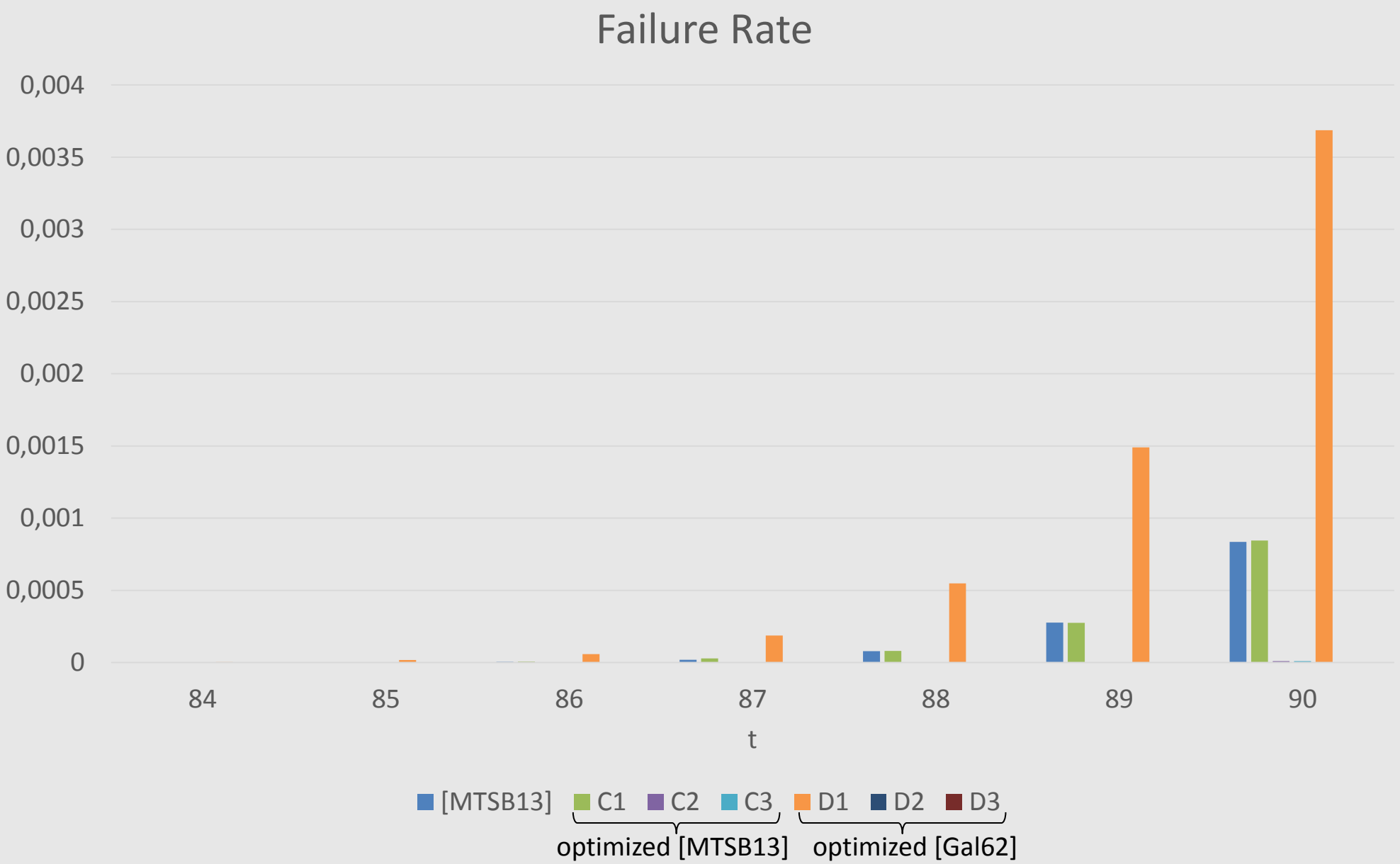
Decoder Evaluation Results



Decoder Evaluation Results



Decoder Evaluation Results



Decoder Evaluation Results

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

Overview

Motivation

Background

Efficient Decoding Techniques

Implementing QC-MDPC McEliece

Side-Channel Attacks

Countermeasures

Exploring Design Options

- First design goal: **high-performance using dedicated hardware**

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

- Relatively small keys → **store operands directly in logic,** no BRAMs
- Count $\#_{upc}$ for current row $h = [h_0|h_1]$

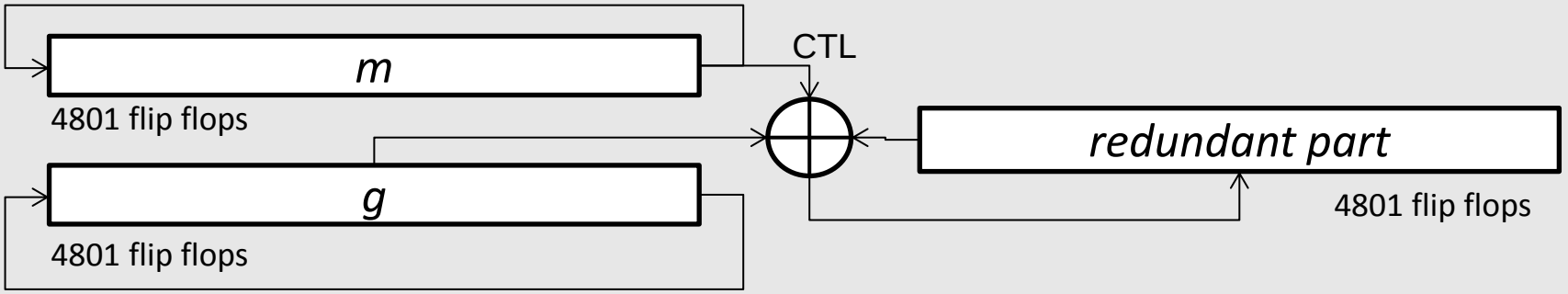
  → Compute Hamming weight HW(s AND h_0), HW(s AND h_1)
- Additional TRNG for error generation and CCA2 conversion required

FPGA High-Speed Encryption

QC-MDPC Encryption

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

$$\left(\mathrm{I} \quad g = \right)$$



FPGA High-Speed Decryption

QC-MDPC Decryption

- **Syndrome computation** $s = Hx^T$, with $H = [H_0|H_1]$
  - Given 9602-bit $h = [h_0|h_1]$ and $x = [x_0|x_1]$
  - Sequentially iterate over every bit of x_0 and x_1 in parallel, rotate h_0 and h_1 accordingly
  - If bit in x_0 and/or x_1 is set, XOR current h_0 and/or h_1 to intermediate syndrome
  - Technically similar to encryption (except for two parallel blocks)
- **Challenge**: Compare $s = 0$?
  - Logical OR tree, lowest level based on 6-input LUTs
  - Added registers to minimize critical path

FPGA High-Speed Decryption

RUB

QC-MDPC Decryption

- **Challenge**: Count $\#_{upc}$ for current row $h = [h_0|h_1]$

  $\rightarrow$ Compute HW(s AND h_0), HW(s AND h_1)
  - Split AND results into 6-bit blocks and lookup HW
  - Adder tree with registers on every level to accumulate total HW
  - Iterative vs. parallel design

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

High-Speed FPGA Results

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

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

High-Speed FPGA Comparison

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

Scheme	Platform	f [MHz]	Bits	Time/Op	Cycles	Mbit/s	Slices	BRAM
This work (enc)	XC6VLX240T	351.7	4,801	13.7 µs	4,801	351.7	2,924	0
This work (dec)	XC6VLX240T	199.3	4,801	82.1 µs	16,363	58.5	10,988	0
This work (dec iter.)	XC6VLX240T	222.5	4,801	125.4 µs	27,919	38.3	10,364	0
McEliece (enc) [Shoufan et al. 2010]	XC5VLX110T	163	512	500 µs	n/a	1.0	14,537	75[1]
McEliece (dec) [Shoufan et al. 2010]	XC5VLX110T	163	512	1,290 µs	n/a	0.4	14,537	75[1]
McEliece (dec) [Ghosh et al. 2012]	XC5VLX110T	190	1,751	500 µs	94,249	3.5	1,385	5
Niederreiter (enc) [Heyse and Güneysu 2012]	XC6VLX240T	300	192	0.66 µs	200	290.9	315	17
Niederreiter (dec) [Heyse and Güneysu 2012]	XC6VLX240T	250	192	58.78 µs	14,500	3.3	3,887	9
Ring-LWE (enc) [Roy et al. 2013]	XC6VLX75T	313	256	20.1 µs	6,300	12.7	n/a	2[1]
Ring-LWE (dec) [Roy et al. 2013]	XC6VLX75T	313	256	9.1 µs	2,800	28.1	n/a	2[1,2]
Ring-LWE (enc) [Pöppelmann and Güneysu 2013]	XC6VLX75T	262	256	26.2 µs	6,861	9.8	1,506	12[1,2]
Ring-LWE (enc) [Pöppelmann and Güneysu 2013]	XC6VLX75T	262	256	16.8 µs	4,404	15.2	1,506	12[1,2]
NTRU (enc/dec) [Kamal and Youssef 2009]	XCV1600E	62.3	251	1.54/1.41 µs	96/88	163/178	14,352	0
ECC-P224 [Güneysu and Paar 2008]	XC4VFX12	487	224	365.10 µs	177,755	0.6	1,580	11[3]
ECC-163 [Rebeiro et al. 2012]	XC5VLX85T	167	163	8.60 µs	1436	18.9	3,446	0
ECC-163 [Roy et al. 2012]	Virtex-4	45.5	163	12.10 µs	552	13.4	12,430	0
ECC-163 [Dimitrov et al. 2006]	Virtex-II	128	163	35.75 µs	4576	4.6	2251	6
RSA-1024 [Suzuki and Matsumoto 2011]	XC5VLX30T	450	1,024	1,520 µs	684,000	0.7	3,237	5[4]

Design Considerations

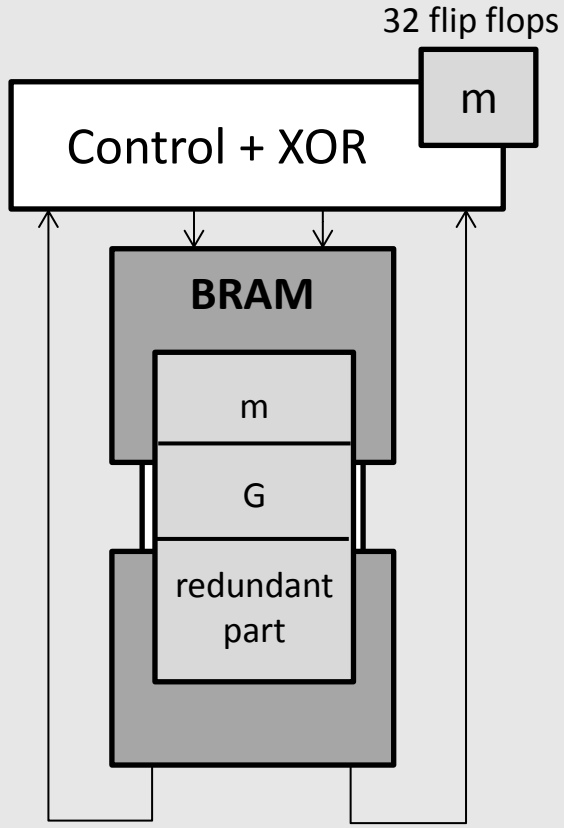
- Second design goal: **low resource/costs in hardware**

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

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

- Additional TRNG for error generation and CCA2 conversion required

FPGA Low-Resource Encryption



QC-MDPC Encryption

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

FPGA Low-Resource Encryption

RUB

QC-MDPC Encryption

- Rotating g is the most performance-critical operation
  - 4801-bit vector g is stored in 151 32-bit memory cells
  - Need to rotate g 4801 times
  - For each 4801-bit rotation: 152 32-bit load, rotate, store
- **Implementation**
  - *Read-First* mode can read cell content before overwriting it with new data
  - Read a cell, rotate it, and store it back to the next cell after reading its content
    - 1 clock cycle, one data port
    - Cyclically rotated addresses in memory

FPGA Low-Resource Decryption

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

- 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

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

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

- Compute Q from first row of $H_1^{-1}$and H_0

Implementing Encryption

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

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

Implementing Decryption

Recall syndrome computation; parity check matrix in sparse

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

Decoding iteration

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

Implementation Results

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
- What about side-channel resistance?

Overview

Motivation

Background

Efficient Decoding Techniques

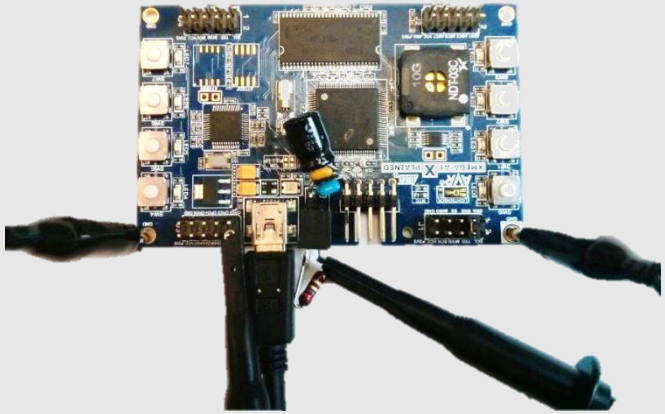
Implementing QC-MDPC McEliece

Side-Channel Attacks

Countermeasures

SCA Setup

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



Message Recovery Attack

Setting: device encrypts a symmetric key under some PK

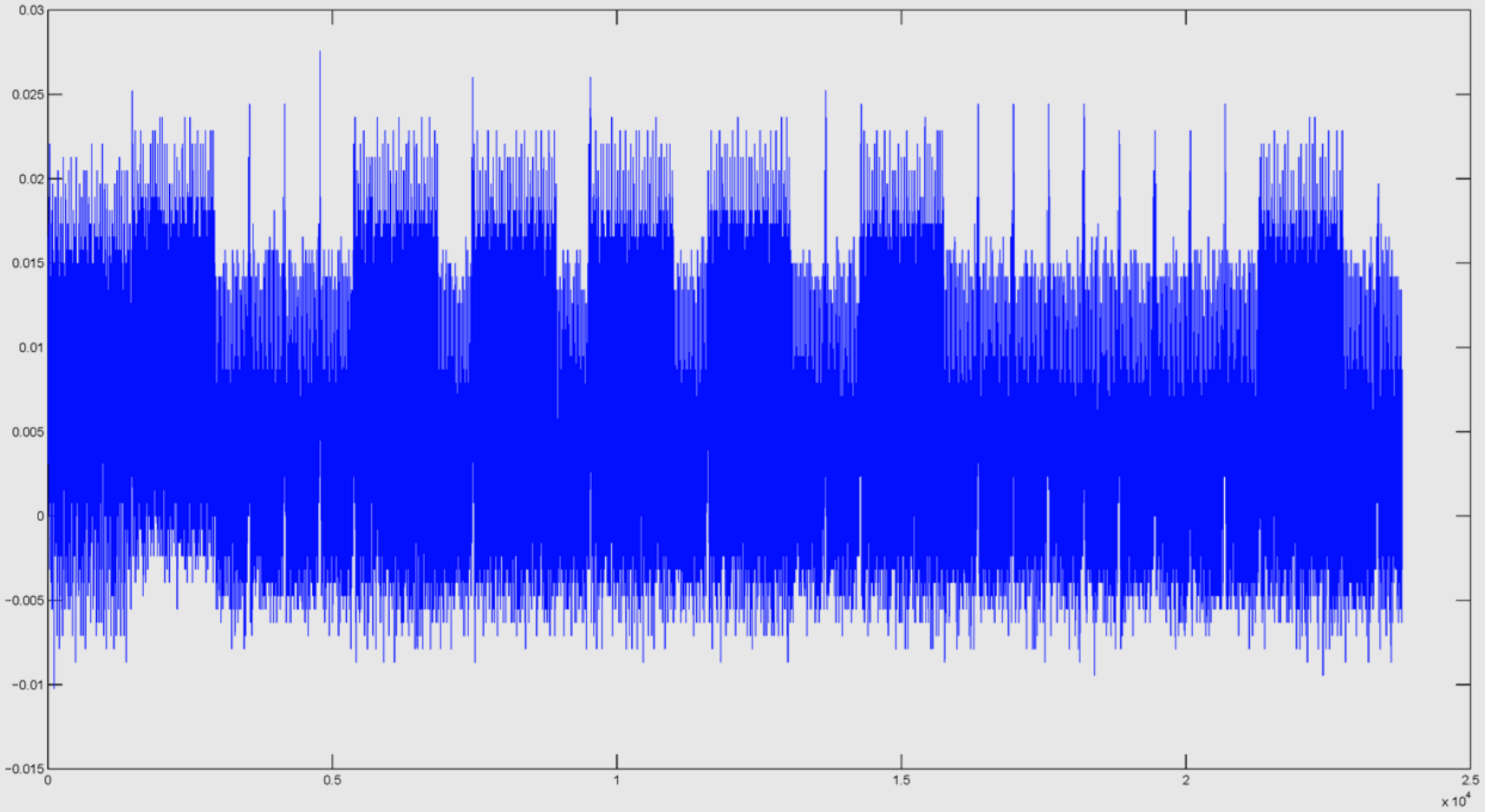
- Recall encryption

$$x = mG + e$$

- **Process:**
  - m selects rows of G
  - Each row has length 4801
    → addition is memory-intense operation

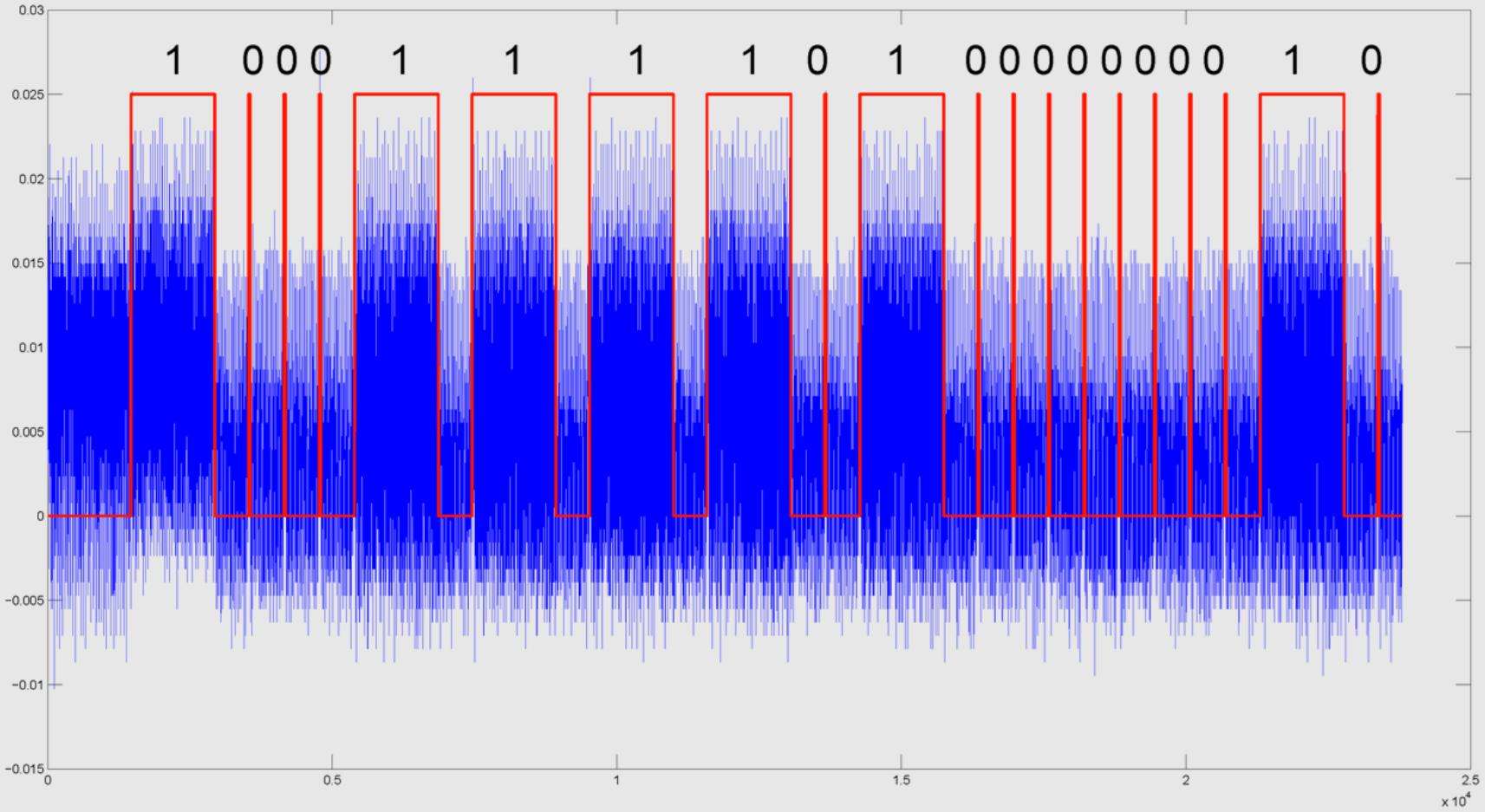
- **Can we detect if a row is accumulated or not?**

Message Recovery Attack – AVR



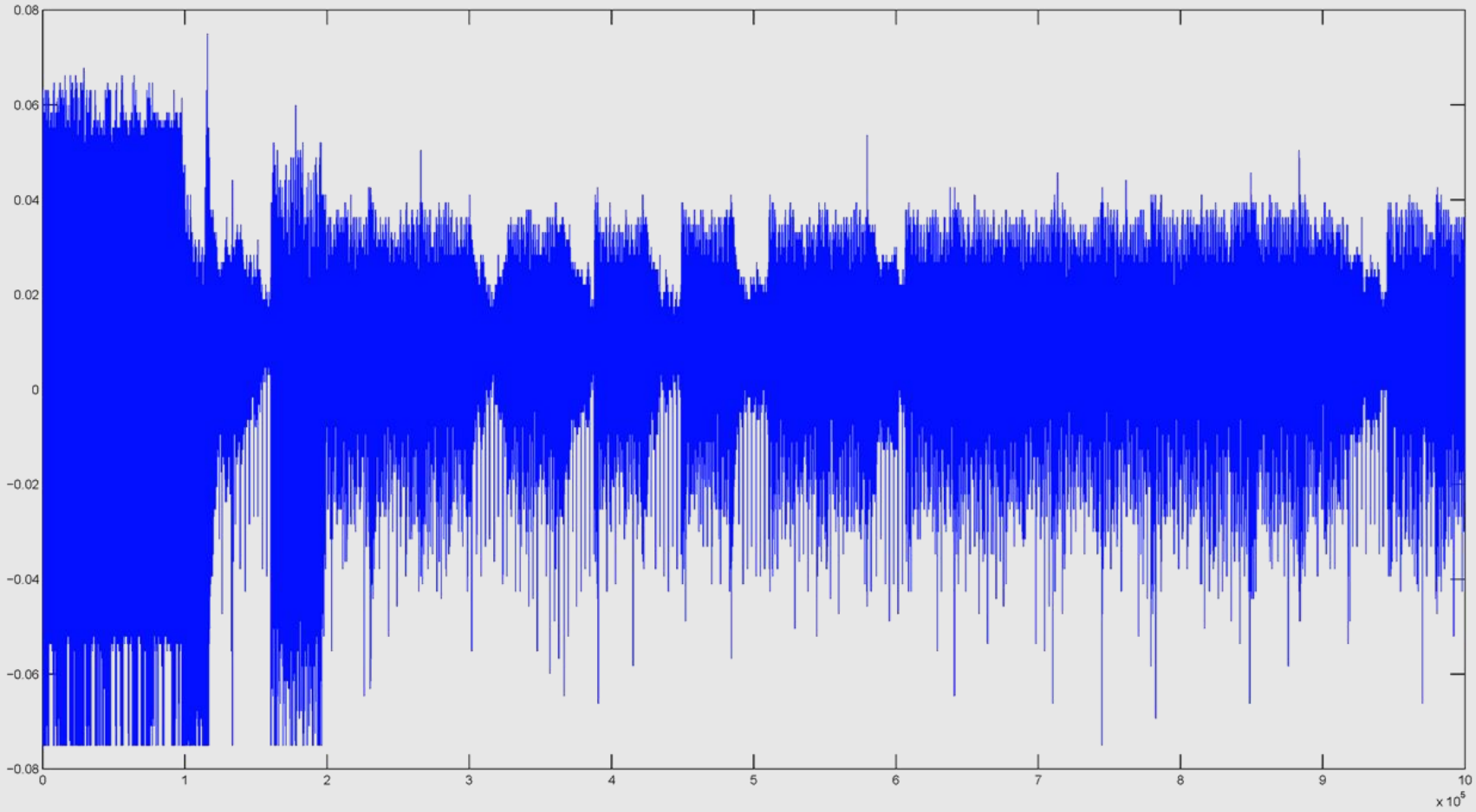
$$m = 0x8F402..$$

Message Recovery Attack – AVR



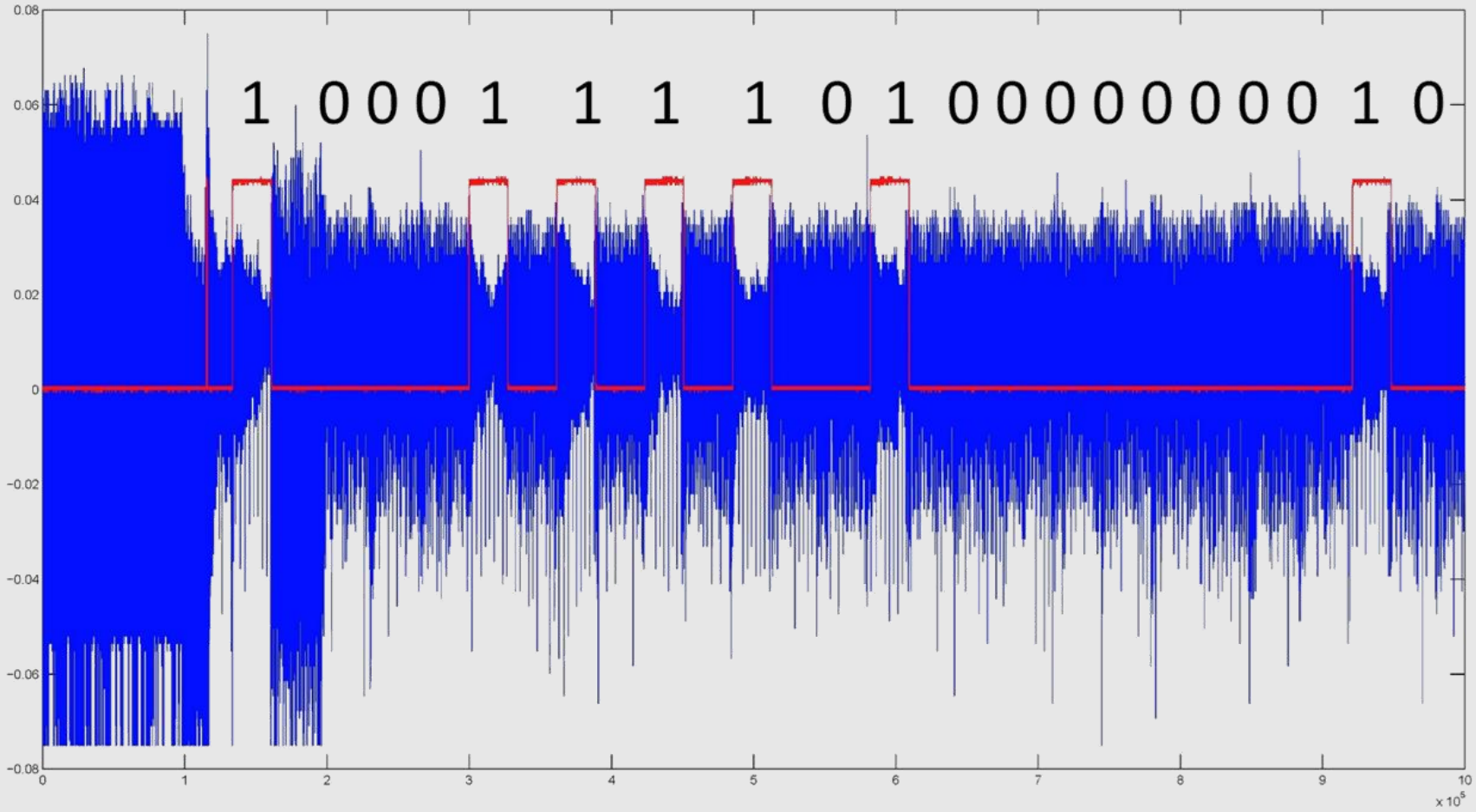
$$m = 0x8F402..$$

Message Recovery Attack – STM32



$$m = 0x8F402..$$

Message Recovery Attack – STM32



$$m = 0x8F402..$$

Secret Key Recovery Attack

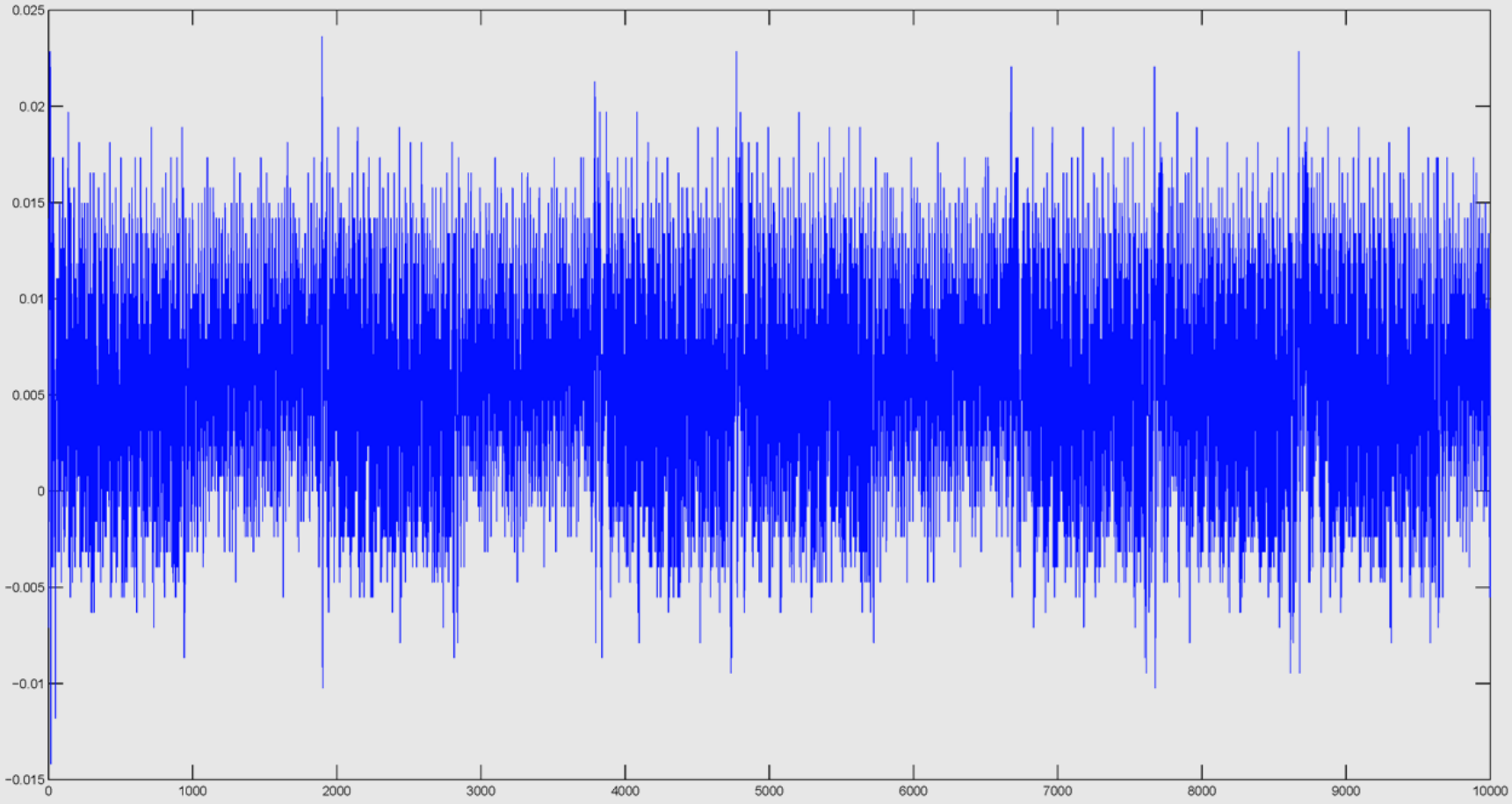
Setting: device decrypts some ciphertext, known or chosen

Possible leakage of information: sparse representation

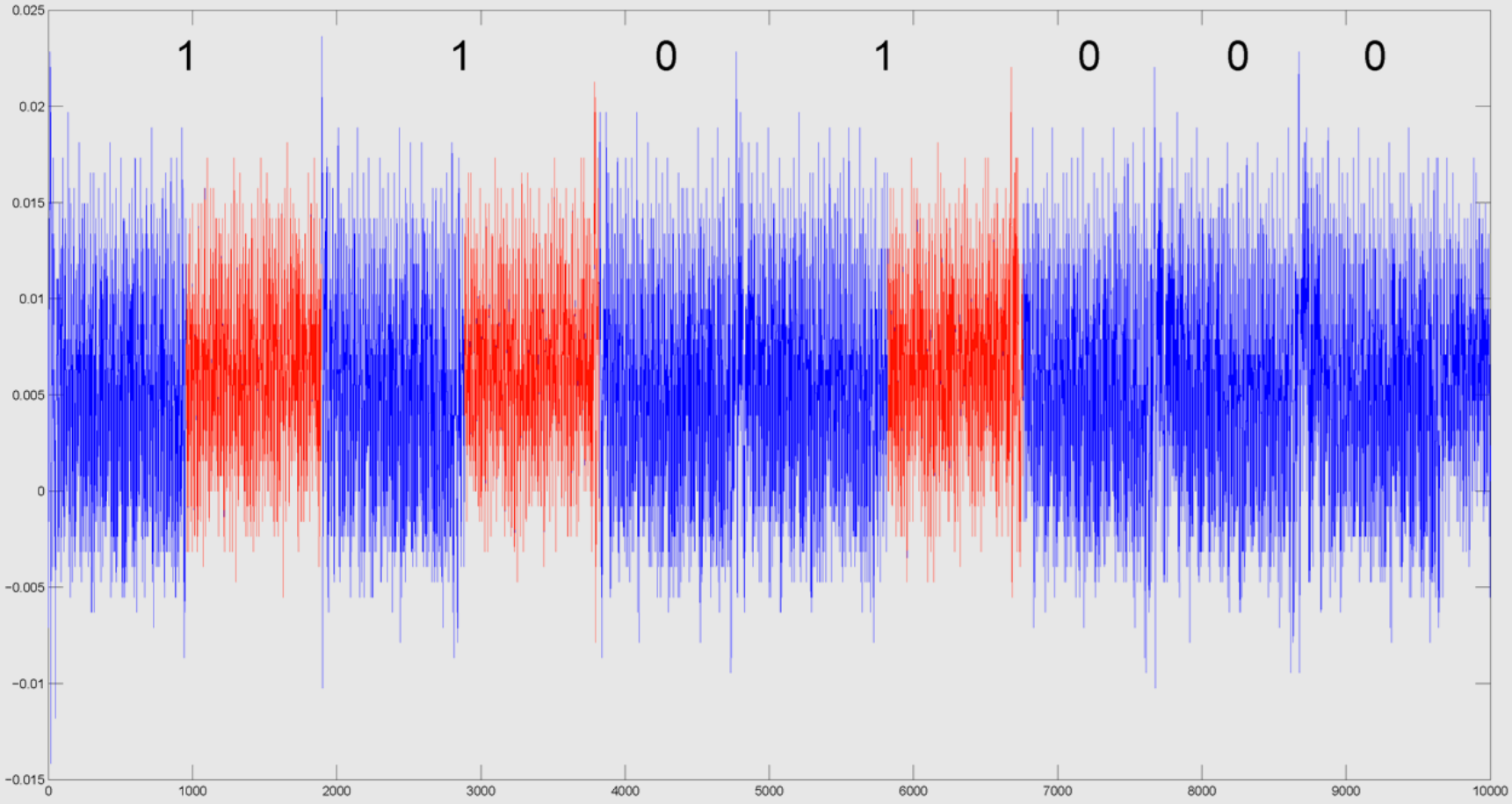
- Only one row of H is stored
- Cyclic shifts generate the following rows
- Sparse rows are stored using 2*45 counters
- Counters are incremented to generate next row
- If a counter exceeds r, it has to be reset to zero (carry)

Can we detect such overflows?

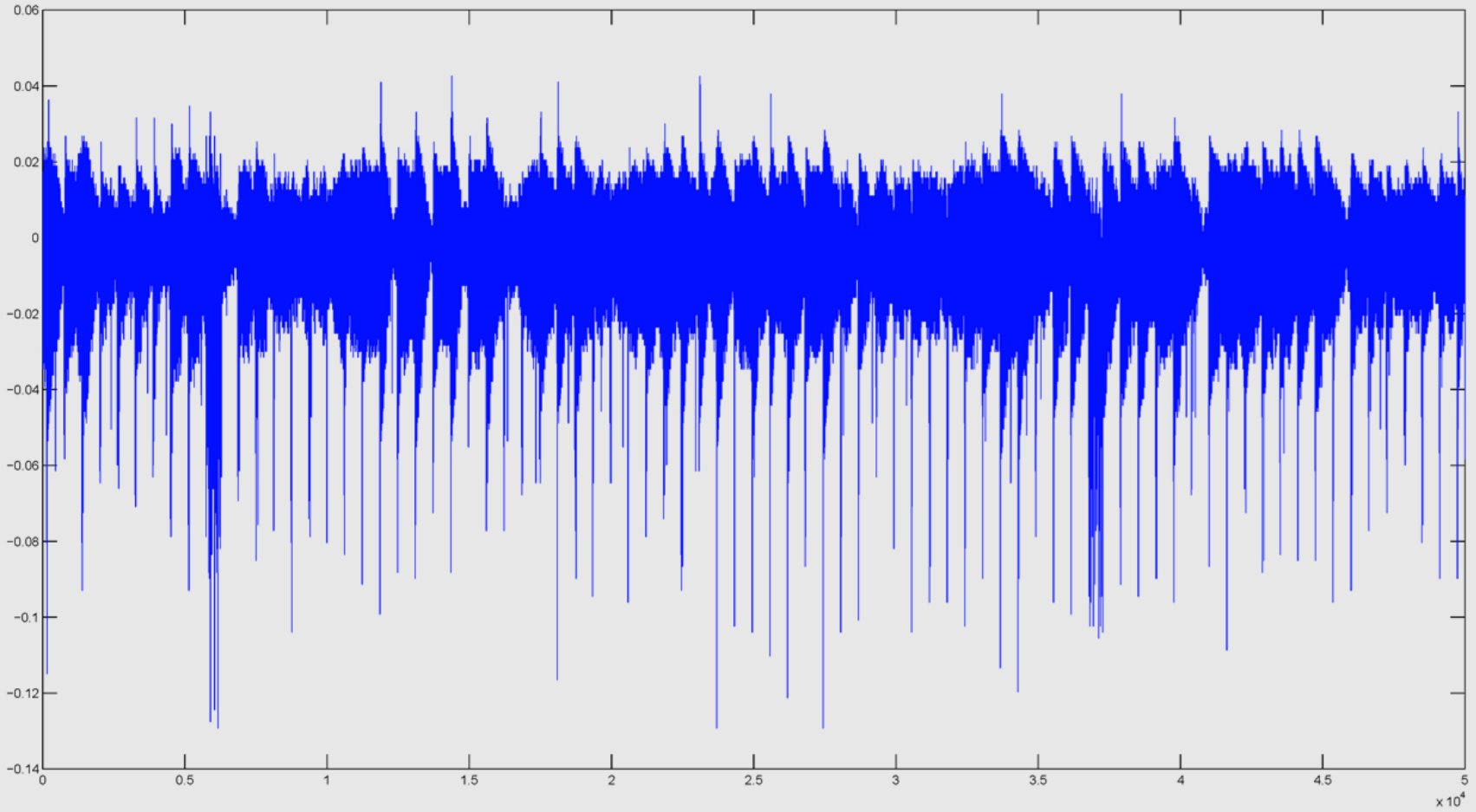
Secret Key Recovery Attack – AVR



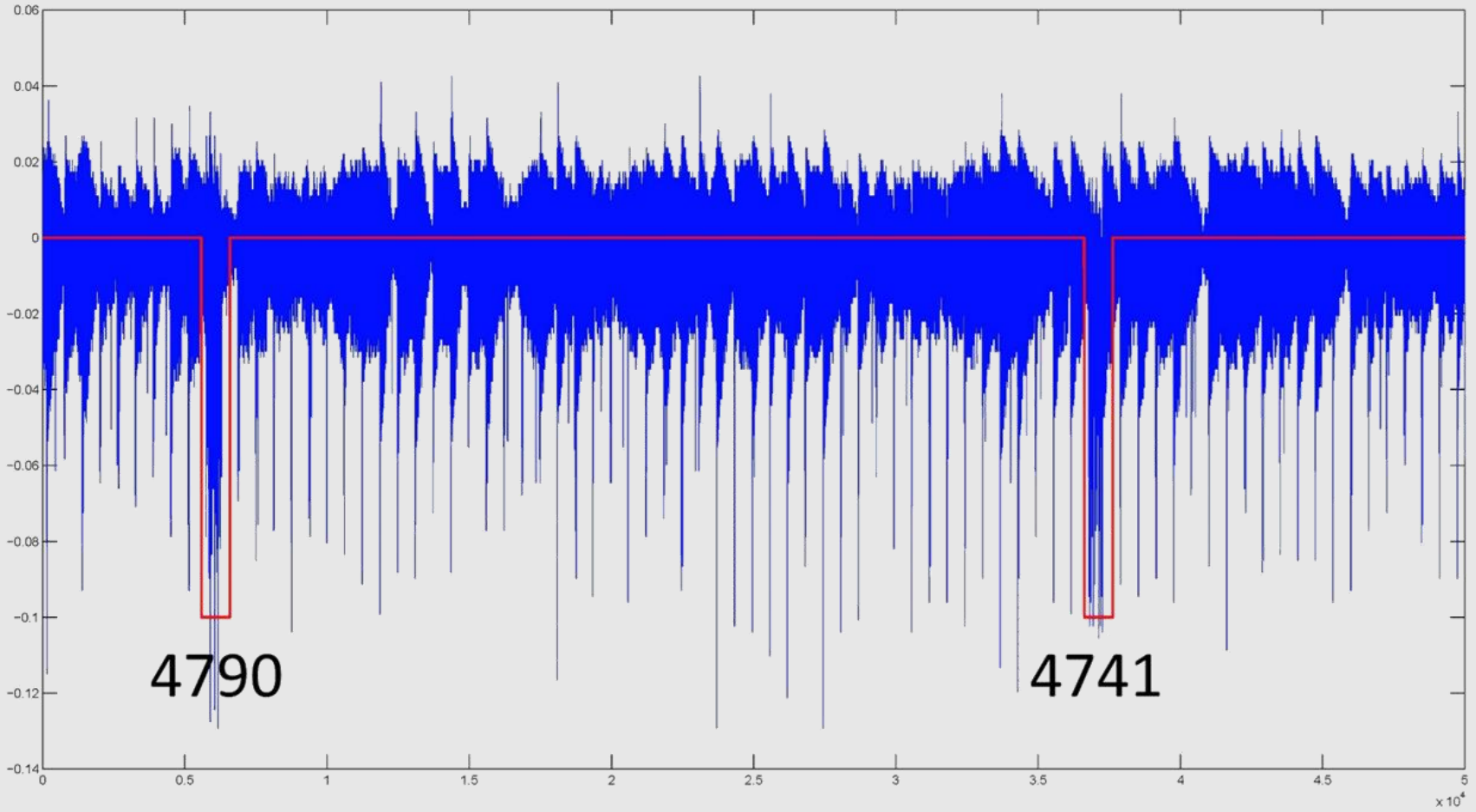
Secret Key Recovery Attack – AVR



Secret Key Recovery Attack – STM32



Secret Key Recovery Attack – STM32



Overview

Motivation

Background

Efficient Decoding Techniques

Implementing QC-MDPC McEliece

Side-Channel Attacks

Countermeasures

General Considerations

Goal: prevent timing and SPA attacks on AMR µC

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

Protecting the Encryption

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

$$r = r \oplus ((0 - m_i) \wedge g_i)$$

Protecting the Decryption

Problem: Counter recovery revealed in sparse representation

Idea: store the full matrix

→ infeasible since 2*(4801*4801) bit = 5.5 Mbyte

Alternative protection of the row rotation of H:

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

Protecting the Decryption

There are more dependencies on secret data!

- **Early aborts on comparison**
  - **Essential**: Test syndrome for zero after every decoding iteration
  - Comparison leaks information about syndrome when aborting after first word != 0 is found
  - **Remedy**: compute OR of all 32-bit blocks of the syndrome and test the result for zero
- **Early abort when decoding reaches $s = 0$**
  - Leaks number of decoding iterations
  - **Remedy**: test the syndrome after reaching max. #iterations
  - Decoding still works as before
- **Further dependencies → see paper @PQCrypto14**

Implementation Results

Scheme	Platform	Cycles/Op	Time
McE MDPC (enc)	STM32F407	16,771,239	100 ms
McE MDPC (dec)	STM32F407	37,171,833	221 ms
McE MDPC (enc, ct)	STM32F407	7,018,493	42 ms
McE MDPC (dec, ct_1)	STM32F407	42,129,589	251 ms
McE MDPC (dec, ct_2)	STM32F407	85,571,555	509 ms
McE MDPC (dec, ct_3)	STM32F407	93,745,754	558 ms
McE MDPC (enc)	ATxmega256	26,767,463	836 ms
McE MDPC (dec)	ATxmega256	86,874,388	2,71 s

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

ct1= early iteration abort; ct2= first syndrome comp. accelerated; ct3=constant time

Implementation Results

Scheme	Platform	Cycles/Op	Time
McE MDPC (enc)	STM32F407	16,771,239	100 ms
McE MDPC (dec)	STM32F407	37,171,833	221 ms
McE MDPC (enc, ct)	STM32F407	7,018,493	42 ms
McE MDPC (dec, ct_1)	STM32F407	42,129,589	251 ms
McE MDPC (dec, ct_2)	STM32F407	85,571,555	509 ms
McE MDPC (dec, ct_3)	STM32F407	93,745,754	558 ms
McE MDPC (enc)	ATxmega256	26,767,463	836 ms
McE MDPC (dec)	ATxmega256	86,874,388	2,71 s

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

ct1= early iteration abort; ct2= first syndrome comp. accelerated; ct3=constant time

Conclusions and Outlook

- **Efficient McEliece implementations with practical key sizes**
  - High-performance and low cost FPGA design exploration
  - Microcontroller implementation for 8-bit AVR and 32-bit ARM devices
- **Side-channel attacks on encryption and decryption**
  - SPA attacks and countermeasures; DPA and fault injection is under investigation

- **Papers and source code available at**

  http://www.sha.rub.de/research/projects/code/

- **Future and on-going work**:
  - Niederreiter encryption and key transport protocols
  - CS-MDPC codes
  - Countermeasures against DPA & fault-injection attacks

RUHR-UNIVERSITÄT BOCHUM

RUB

Tweaking Code-Based Cryptography for Embedded Systems

DIMACS Workshop on The Mathematics of Post-Quantum Cryptography

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

1/12/2015

Thank you!

References

[Gal62] R. Gallager. Low-density Parity-check Codes. Information Theory, IRE Transactions on, 8(1):21–28, 1962.

[HMG13] S. Heyse, I. von Maurich, and T. Güneysu. Smaller Keys for Code-Based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices. CHES 2013: 273-292.

[HP03] W. Huffman and V. Pless. Fundamentals of Error-Correcting Codes. Cambridge University Press, 2003.

[MG14a] I. von Maurich and T. Güneysu. Lightweight Code-based Cryptography: QC-MDPC McEliece Encryption on Reconfigurable Devices. DATE 2014: 38:1-38:6.

[MG14b] I, von Maurich and T. Güneysu. Towards Side-Channel Resistant Implementations of QC-MDPC McEliece Encryption on Constrained Devices, PQCrypto 2014.

[MTSB13] Rafael Misoczki, Jean-Pierre Tillich, Nicolas Sendrier, Paulo S. L. M. Barreto: MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes. ISIT 2013: 2069-2073.