RUHR-UNIVERSITÄT BOCHUM

RUB

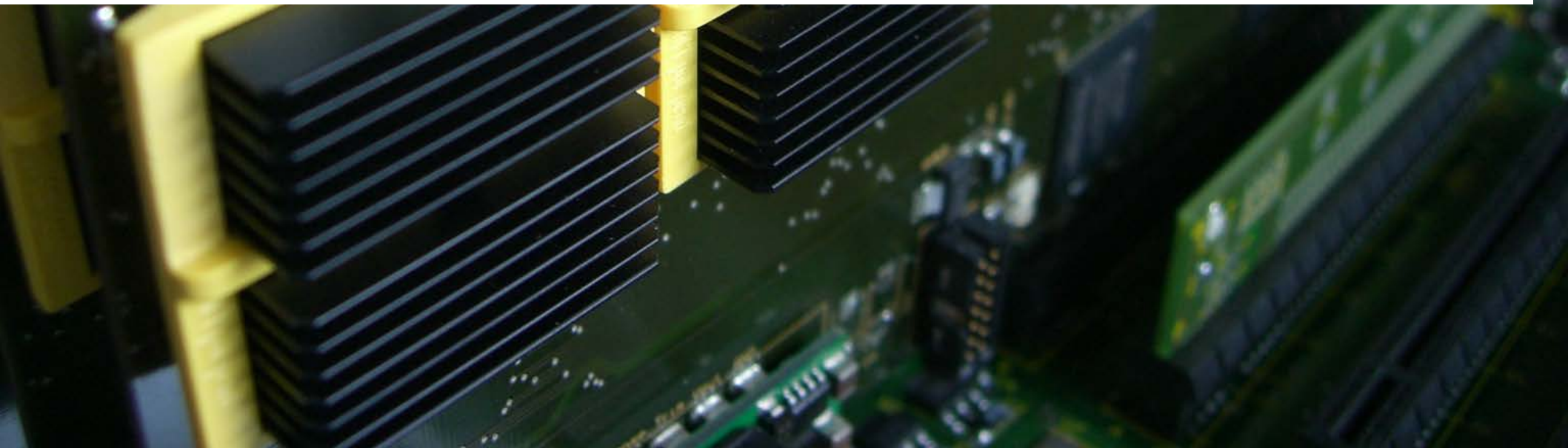
Smaller Keys for Code-based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices

CHES 2013, Santa Barbara, USA

Stefan Heyse, Ingo von Maurich and Tim Güneysu

Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany

August 22, 2013

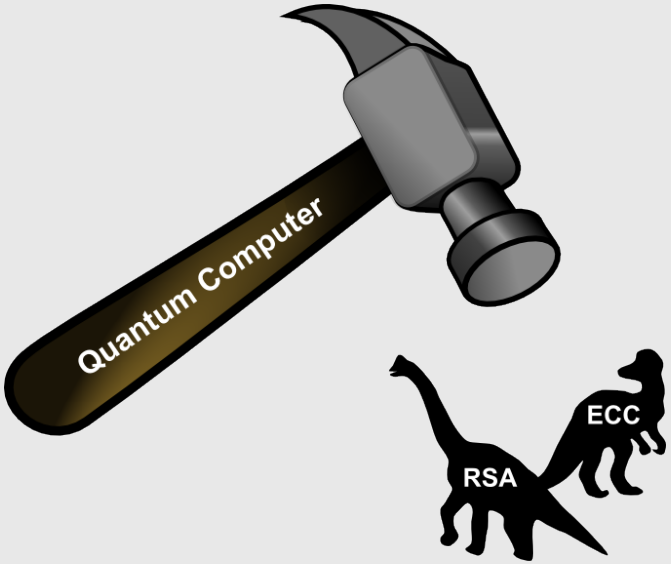


Motivation

- Factoring and discrete log problem vs. quantum computers
- Larger diversification of pk primitives needed
- McEliece and Niederreiter resist quantum attacks
- …and can outperform classical cryptosystems
- Drawback: large keys (often $\geq$ 50 kByte) vs. embedded devices
- Quasi-cyclic medium-density parity check codes (4800 bit pk, 80 bit security level) [MTSB12]

Open questions

- Performance on embedded devices?
- Which decoders should be used?
- Can existing decoders be improved?



Overview

Motivation

Background

Efficient Decoding of MDPC Codes

Implementing QC-MDPC McEliece

Results

Conclusions

Background on (QC-)MDPC Codes

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

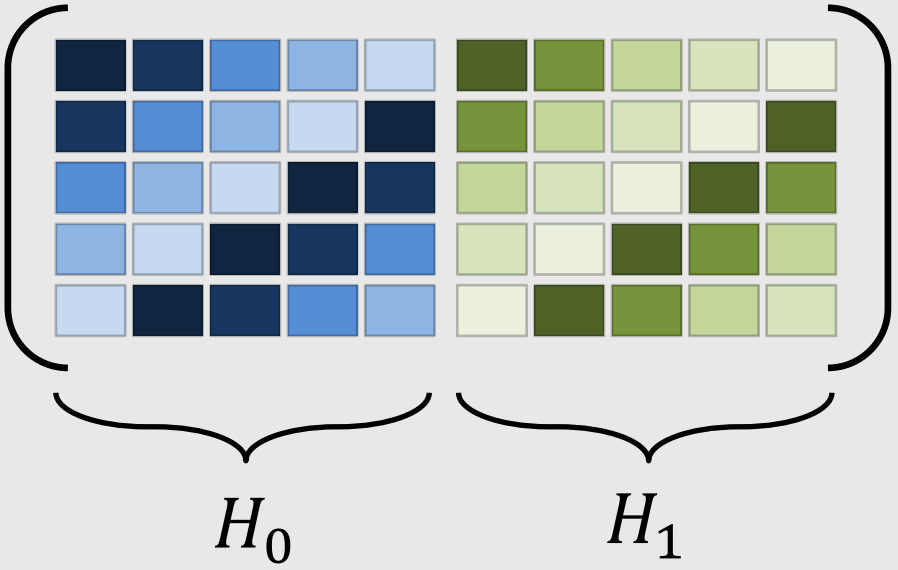
Code/Key Generation

1. Randomly pick n_0 first rows of parity check matrix blocks H_i
   $h_i \in F_2^n$ of weight w_i s.t. 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 $G = (I|Q)$,

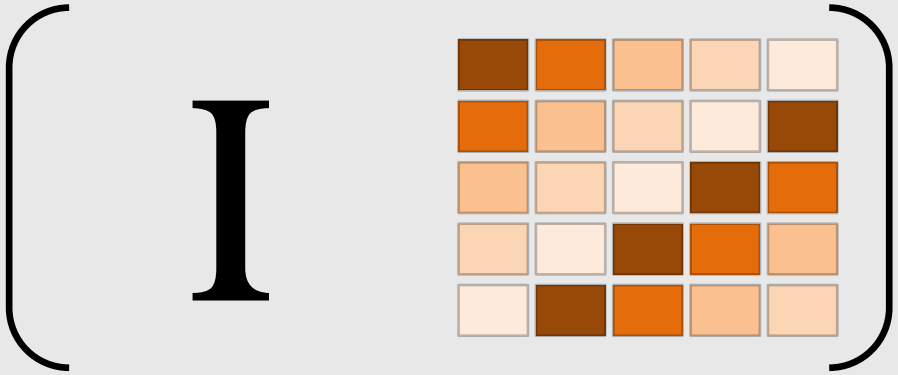
$$\mathrm{Q} = \begin{pmatrix} (H_{n_0-1}^{-1} * H_0)^T \\ (H_{n_0-1}^{-1} * H_1)^T \\ \ldots \\ (H_{n_0-1}^{-1} * H_{n_0-2})^T \end{pmatrix}$$

Background on (QC-)MDPC Codes

Parity check matrix H



Generator matrix G



(QC-)MDPC McEliece

Encryption

Message $m \in F_2^{(n-r)}$, 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 MDPC decoding algorithm.

$mG \leftarrow \Psi_H(mG + e)$

Extract m from the first $(n - r)$ positions.

Parameters for 80-bit equivalent symmetric security [MTSB12]

$n_0 = 2, n = 9600, r = 4800, w = 90, t = 84$

Overview

Motivation

Background

Efficient Decoding of MDPC Codes

Implementing QC-MDPC McEliece

Results

Conclusions

Efficient Decoding of MDPC Codes

General Decoding Principle

1. *Compute syndrome* s of the received codeword
2. *Count the number of unsatisfied parity-check-equations* $\#_{upc}$ for each codeword bit
3. *Flip codeword bits* that violate more than b equations
4. *Recompute syndrome*
5. *Repeat* until $s = 0$ or reaching predefined maximum of iterations (decoding failure)

- Main difference is how threshold b is computed
  - Precompute b_i for each iteration [Gal62]
  - $b = max_{upc}$ [HP03]
  - $b = max_{upc} - \delta$ [MTSB12]

Decoding Optimizations

Observations

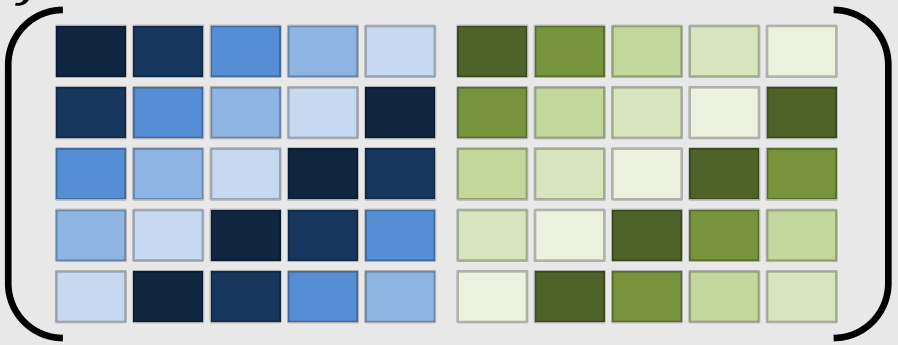
- Decoders recompute syndrome after each iteration
- Syndrome computation is expensive!

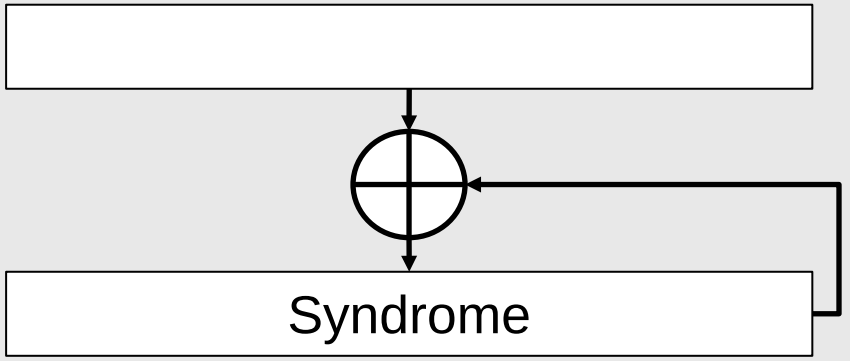
Optimizations

- If threshold exceeded, flip codeword bit j
  → the syndrome changes
- Syndrome does not change arbitrarily!
  $s_{new} = s_{old} + h_j$

⟶ No syndrome recomputation

⟶ Decoding with up-to-date syndrome





Benchmarking

- Derived several decoders
  - Direct vs. temporary syndrome update
  - Different threshold techniques
- Decoding failure if no success within 10 iterations
- C implementation on Intel Xeon E5345 CPU@2.33 GHz
- 1000 random QC-MDPC codes with

$$n_0 = 2, n = 9600, r = 4800, w = 90, t = 84$$

100,000 random decoding tries for each decoder

Most Efficient (QC-)MDPC Decoders

Proposed Decoder 1

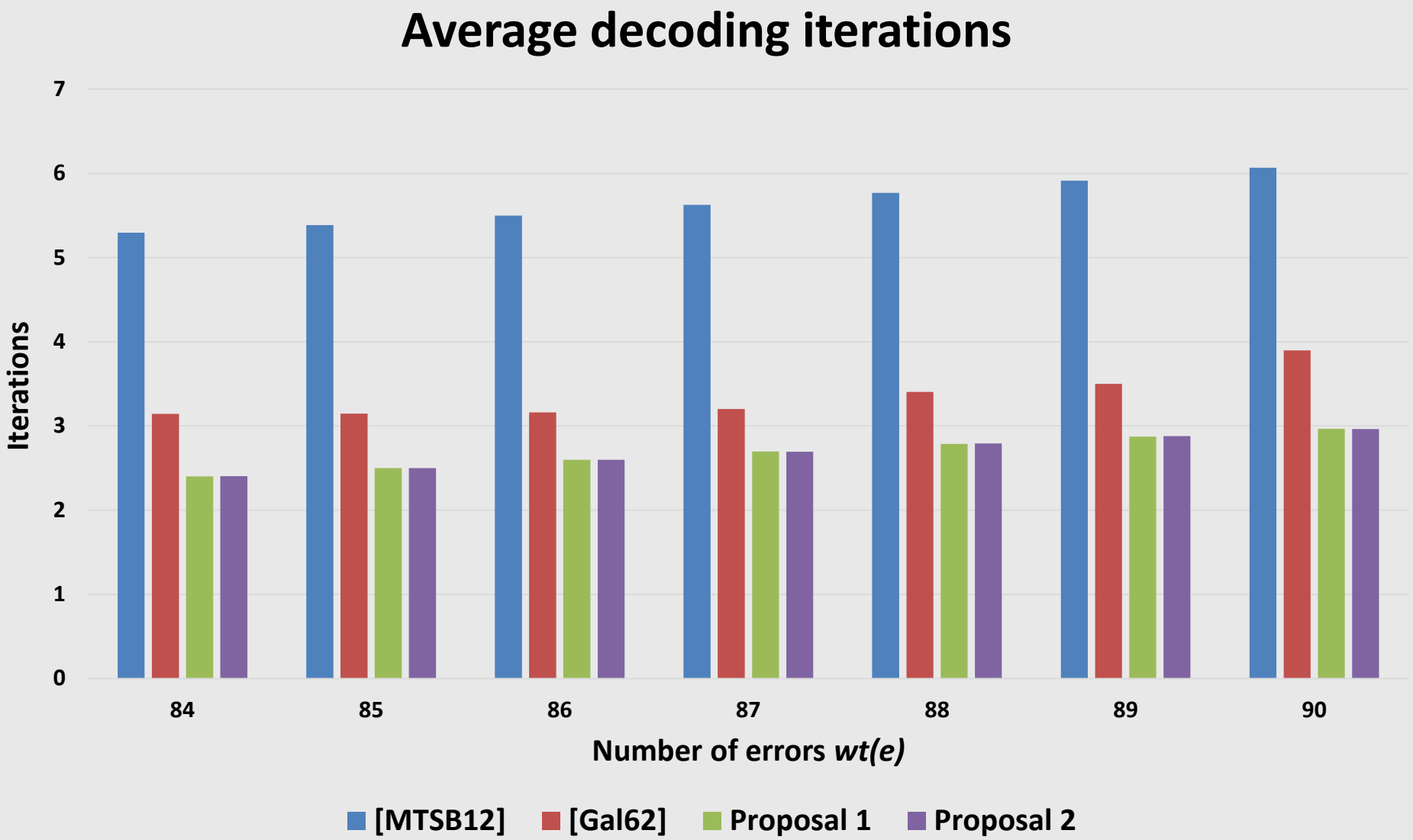
1. Compute the syndrome
2. Count $\#_{upc}$ for each bit, flip the current codeword bit j if $\#_{upc}$ exceeds threshold b_i and add h_j to the syndrome

Proposed Decoder 2

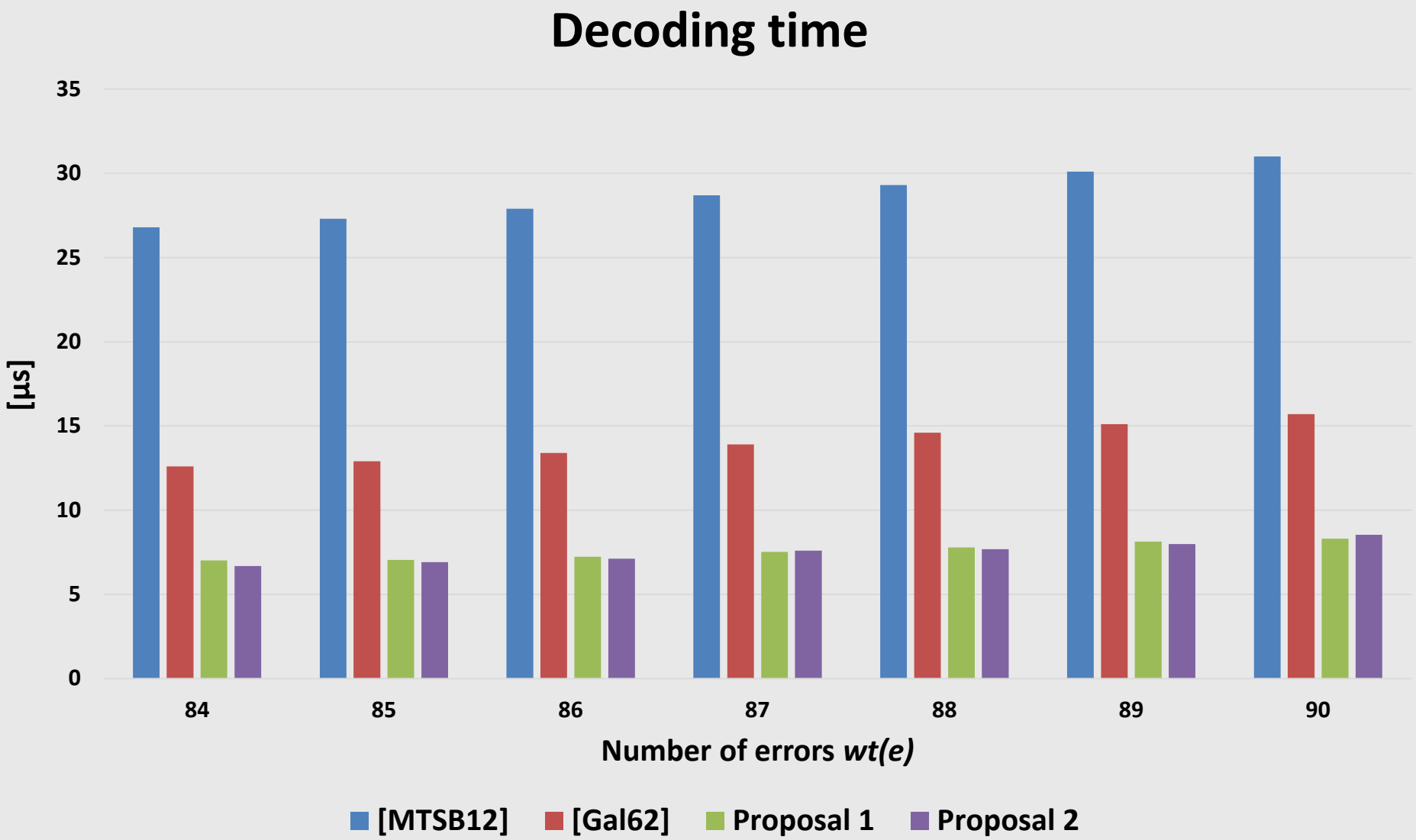
- Decoder 1 + additionally checks $s = 0$ after each update

*See paper for a full description of all evaluated decoders

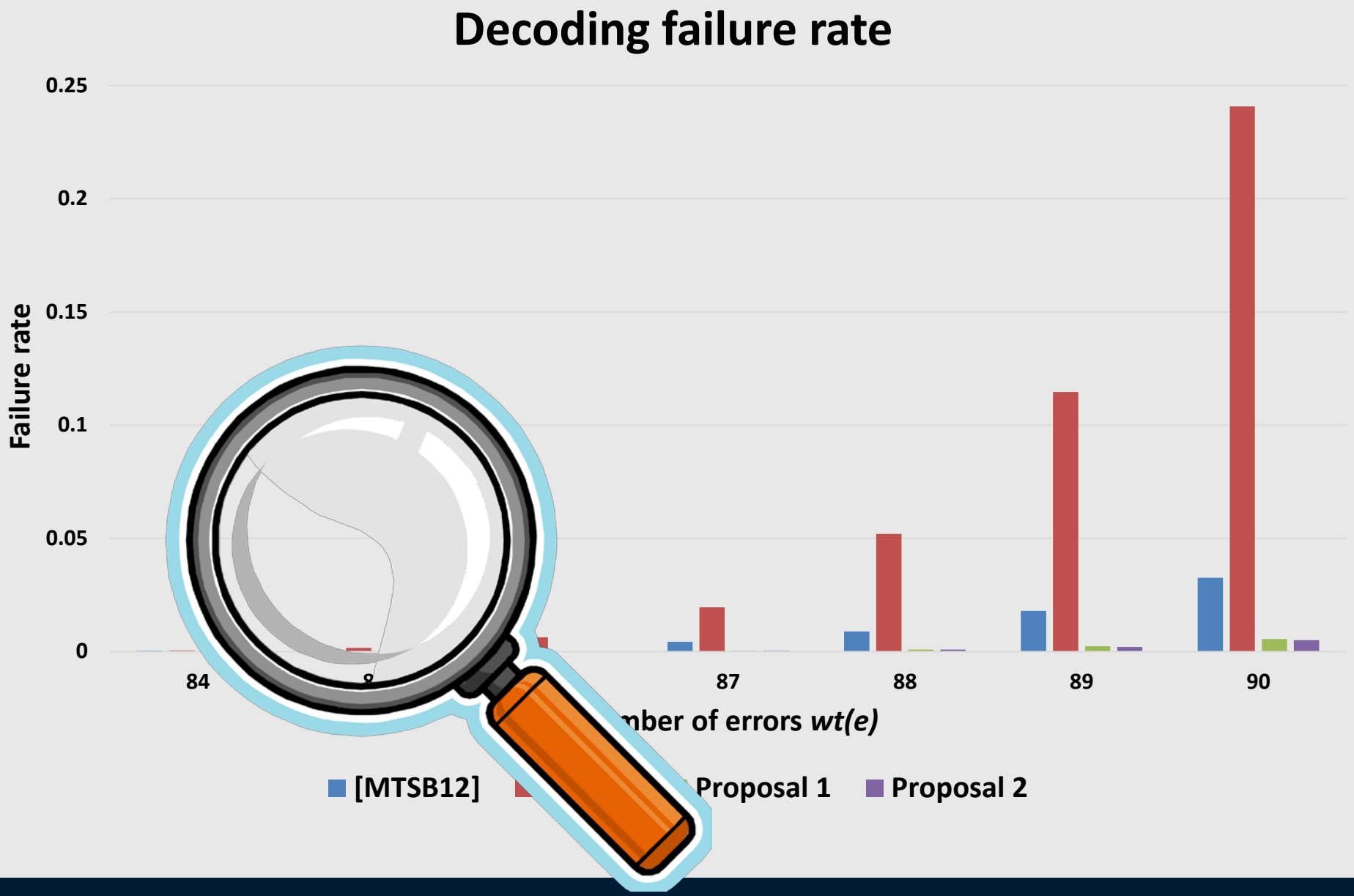
Decoder Evaluation



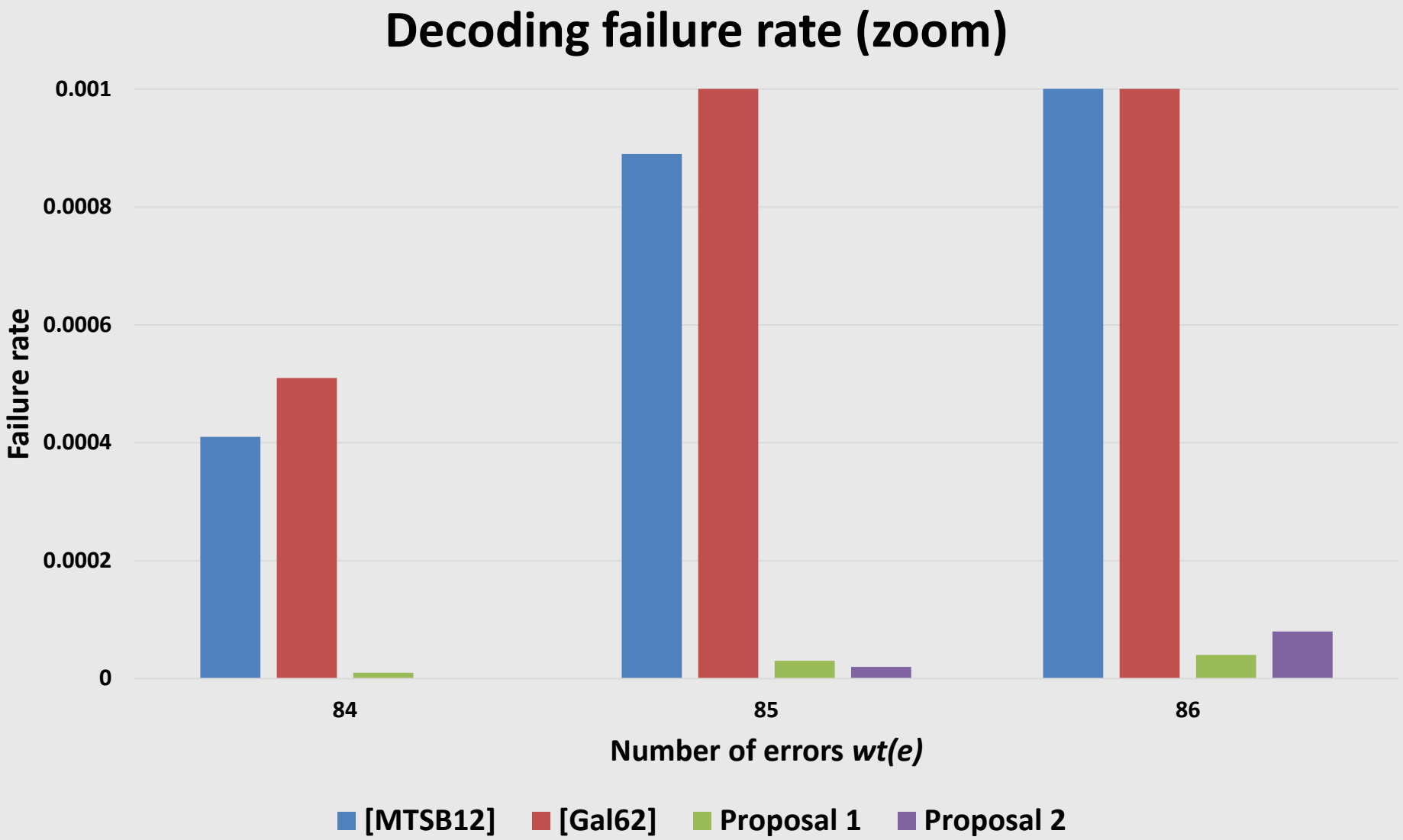
Decoder Evaluation



Decoder Evaluation



Decoder Evaluation



Overview

Motivation

Background

Efficient Decoding of MDPC Codes

Implementing QC-MDPC McEliece

Results

Conclusions

Implementation Platforms

- Reconfigurable Hardware: Xilinx Virtex-6 XC6VLX240T FPGA
  - Powerful FPGA
  - 37,680 slices, each slice has four 6-input LUTs and eight FFs
- Embedded microcontroller: Atmel AVR ATxmega256A3
  - Low-cost 8-bit microcontroller
  - 16 kByte SRAM, 256 kByte program memory
  - Clocked at 32 MHz

FPGA Design Considerations

- Overall FPGA design goal: **high speed**
- Relatively small keys ⟶ store operands directly in logic, no BRAMs
- Implemented proposal 1, early exit requires variable rotations
- Excluded TRNG for error generation and CCA2 conversion

Microcontroller Design Considerations

Overall microcontroller design goal: **small memory footprint**

Encoder

- Straightforward: rotate and accumulate
- Rolled vs. unrolled public key rotation

Decoder (based on Proposal 2)

- Generating the next h_i requires to shift 4800 bit (600 byte)
- But rows are sparse, storing positions of set bits needs 45 counters
- Shifting requires to increment 45 counters
- Adding sparse to full polynomial by flipping 45 bits

Overview

Motivation

Background

Efficient Decoding of MDPC Codes

Implementing QC-MDPC McEliece

Results

Conclusions

FPGA Results

- Post-PAR for Xilinx Virtex-6 XC6VLX240T
- Encryption takes 4,800 cycles
- Average decryption cycles
  - Small: $4{,}800 + 2 + 2.4002 * (9{,}620 + 2) = 27{,}896.7$ cycles
  - Fast: $4{,}800 + 2 + 2.4002 * (4{,}810 + 2) = 16{,}351.8$ cycles

Aspect	Encoder	Decoder (small)	Decoder (fast)
FFs	14,426 (4%)	32,974 (10%)	46,515 (15%)
LUTs	8,856 (5%)	36,554 (24%)	46,249 (30%)
Slices	2,920 (7%)	10,271 (27%)	17,120 (45%)
Frequency	351.3 MHz	222.5 MHz	190.6 MHz
Time/Op	13.66 µs	125.38 µs	85.79 µs
Throughput	351.3 Mbit/s	38.3 Mbit/s	55.9 Mbit/s
Encode	4,800 cycles	-	-
Compute Syndrome	-	4,800 cycles	4,800 cycles
Check Zero	-	2 cycles	2 cycles
Flip Bits	-	9,620 cycles	4,810 cycles
Overall average	4,800 cycles	27,896.7 cycles	16,351.8 cycles

FPGA Comparison

- PK size: 0.59 kByte vs. 100.5 kByte [SWM+10], 63.5 kByte [GDU+12]
- Performance evaluation: Time/operation vs. Mbit/s
- Faster than previous McEliece implementations

Scheme	Platform	f [MHz]	Bits	Time/Op	Cycles	Mbit/s	Slices	BRAM
This work (enc)	XC6VLX240T	351.3	4,800	13.66 µs	4,800	351.3	2,920	0
This work (dec)	XC6VLX240T	190.6	4,800	85.79 µs	16,352	55.9	17,120	0
This work (dec iter.)	XC6VLX240T	222.5	4,800	125.38 µs	27,897	38.3	10,271	0
McEliece (enc) [SWM+10]	XC5VLX110T	163	512	500 µs	n/a	1.0	14,537	75[1]
McEliece (dec) [SWM+10]	XC5VLX110T	163	512	1,290 µs	n/a	0.4	14,537	75[1]
McEliece (dec) [GDU+12]	XC5VLX110T	190	1,751	500 µs	94,249	3.5	1,385	5

Microcontroller Results

Encoder

- Very frequent memory access (>50% of the runtime)
- 0.8s@32Mhz

Decoder

- Shifting sparse polynomial in 720 cycles
- Adding sparse polynomial to syndrome in 2,200 cycles
- Again very frequent memory access
- 2.7sec@32Mhz

	Platform	SRAM	Flash
[enc]	ATxmega256	606 Byte	3,705 Byte
[enc unrolled]	ATxmega256	606 Byte	5,496 Byte
[dec]	ATxmega256	198 Byte	2,218 Byte

Microcontroller Comparison

- Much smaller than previous McEliece implementations
- Faster and smaller than RSA
- More cycles/op than most competitors

Scheme	Platform	SRAM	Flash	Cycles/Op	Cycles/byte
This work [enc]	ATxmega256	606 Byte	3,705 Byte	37,440,137	62,400
This work [enc unrolled]	ATxmega256	606 Byte	5,496 Byte	26,767,463	44,612
This work [dec]	ATxmega256	198 Byte	2,218 Byte	86,874,388	146,457
McEliece [enc] [13]	ATxmega256	512 Byte	438 kByte	14,406,080	65,781
McEliece [dec] [13]	ATxmega256	12 kByte	130.4 kByte	19,751,094	90,187
McEliece [enc] [20]	ATxmega256	3.5 kByte	11 kByte	6,358,400	39,493
McEliece [dec] [20]	ATxmega256	8.6 kByte	156 kByte	33,536,000	208,298
McEliece [enc] [10]	ATxmega256	-	-	4,171,734	260,733
McEliece [dec] [10]	ATxmega256	-	-	14,497,587	906,099
ECC-P160 [19]	ATmega128	282 Byte	3682 Byte	6,480,000	324,000
RSA-1024 random [19]	ATmega128	930 Byte	6292 Byte	87,920,000	686,875

Overview

Motivation

Background

Efficient Decoding of MDPC Codes

Implementing QC-MDPC McEliece

Results

Conclusions

Conclusions

- High throughput FPGA and low memory footprint microcontroller implementations with **practical** key sizes
- Two optimized QC-MDPC decoders
- Incentive for further cryptanalytical investigation to establish confidence
- Source code (C and VHDL) available at

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

RUHR-UNIVERSITÄT BOCHUM

RUB

Smaller Keys for Code-based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices

CHES 2013, Santa Barbara, USA

Stefan Heyse, Ingo von Maurich and Tim Güneysu

Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany

August 22, 2013

Thank you!
Questions?

References

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

[GDU+12] S. Ghosh, J. Delvaux, L. Uhsadel, and I. Verbauwhede. A Speed Area Optimized Embedded Co-processor for McEliece Cryptosystem. In Application-Specific Systems, Architectures and Processors (ASAP), 2012 IEEE 23rd International Conference on, pages 102 –108, 2012.

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

[MTSB12] R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto. MDPC-McEliece: New McEliece Variants from Moderate Density Parity-Check Codes. Cryptology ePrint Archive, Report 2012/409, 2012. http://eprint.iacr.org/

[SWM+10] A. Shoufan, T. Wink, H. G. Molter, S. A. Huss, and E. Kohnert. A Novel Cryptoprocessor Architecture for the McEliece Public-Key Cryptosystem. IEEE Trans. Computers, 59(11):1533–1546, 2010.

FPGA Comparison

- Performance evaluation: Time/operation vs. Mbit/s
- PK size: 0.59 kByte vs. 100.5 kByte [38], 63.5 kByte [16][21]

Scheme	Platform	f [MHz]	Bits	Time/Op	Cycles	Mbit/s	FFs	LUTs	Slices	BRAM
This work (enc)	XC6VLX240T	351.3	4,800	13.66 µs	4,800	351.3	14,426	8,856	2,920	0
This work (dec)	XC6VLX240T	190.6	4,800	85.79 µs	16,352	55.9	46,515	46,249	17,120	0
This work (dec iter.)	XC6VLX240T	222.5	4,800	125.38 µs	27,897	38.3	32,974	36,554	10,271	0
McEliece (enc) [38]	XC5VLX110T	163	512	500 µs	n/a	1.0	n/a	n/a	14,537	75[1]
McEliece (dec) [38]	XC5VLX110T	163	512	1,290 µs	n/a	0.4	n/a	n/a	14,537	75[1]
McEliece (dec) [16]	XC5VLX110T	190	1,751	500 µs	94,249	3.5	n/a	n/a	1,385	5
Niederreiter (enc) [21]	XC6VLX240T	300	192	0.66 µs	200	290.9	875	926	315	17
Niederreiter (dec) [21]	XC6VLX240T	250	192	58.78 µs	14,500	3.3	12,861	9,409	3,887	9
Ring-LWE (enc) [17]	XC6VLX240T	n/a	256	8.10 µs	n/a	15.8	143,396	298,016	n/a	0[2]
Ring-LWE (dec) [17]	XC6VLX240T	n/a	256	8.15 µs	n/a	15.7	65,174	124,158	n/a	0[2]
NTRU (enc/dec) [23]	XCV1600E	62.3	251	1.54/1.41 µs	96/88	163/178	5,160	27,292	14,352	0
ECC-P224 [18]	XC4VFX12	487	224	365.10 µs	177,755	0.61	1,892	1,825	1,580	11[3]
ECC-163 [34]	XC5VLX85T	167	163	8.60 µs	1436	18.9	n/a	10,176	3,446	0
ECC-163 [35]	Virtex-4	45.5	163	12.10 µs	552	13.4	n/a	n/a	12,430	0
ECC-163 [12]	Virtex-II	128	163	35.75 µs	4576	4.56	n/a	n/a	2251	6
RSA-1024 [42]	XC5VLX30T	450	1,024	1,520 µs	684,000	0.67	n/a	n/a	3,237	5[4]

QC-MDPC McEliece FPGA Implementation

RUB

QC-MDPC Encryption

- Given first 4800-bit row g of G and message m, compute $x = mG + e$
- G is of systematic form → 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

QC-MDPC McEliece FPGA Implementation

RUB

QC-MDPC Decryption

- Syndrome computation $s = Hx^T$, with $H = [H_0|H_1]$
  - Given 9600-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
- $s = 0?$
  - Logical OR tree, lowest level based on 6-input LUTs
  - Added registers to minimize critical path

QC-MDPC McEliece FPGA Implementation

RUB

QC-MDPC Decryption

- 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 accumulates overall HW
  - Parallel vs. iterative design
- Bit-flipping step
  - If HW exceeds threshold b_i the corresponding bit in codeword x_0 and/or x_1 is flipped
  - Syndrome is updated by XORing current secret poly h_0 and/or h_1
  - Generate next row h and repeat