RUHR-UNIVERSITÄT BOCHUM

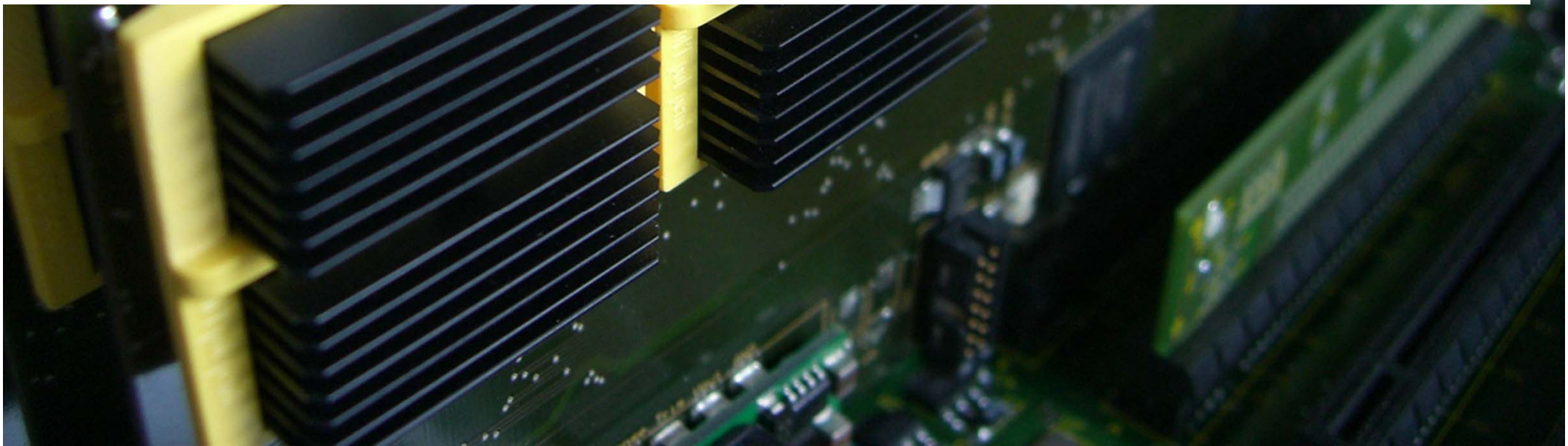
RUB

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

4th Code-based Cryptography Workshop 2013, Rocquencourt, France

Stefan Heyse, Ingo von Maurich and Tim Güneysu
Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany

June 10, 2013



Overview

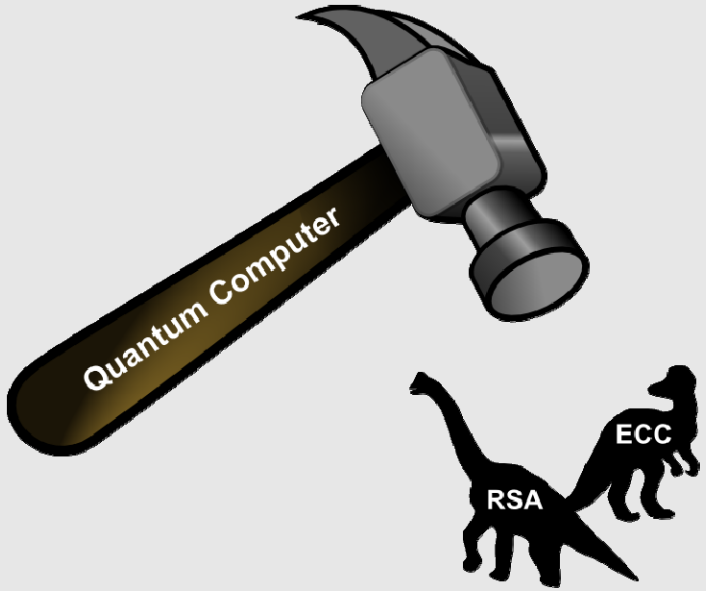
1. Motivation
2. Background
3. Efficient Decoding of MDPC Codes
4. Implementing QC-MDPC McEliece
5. Results
6. Conclusions

Overview

1. **Motivation**
2. Background
3. Efficient Decoding of MDPC Codes
4. Implementing QC-MDPC McEliece
5. Results
6. Conclusions

1. Motivation

- Quantum computers solve factoring and discrete log problem
- Code-based cryptosystems McEliece and Niederreiter resist quantum attacks and can outperform classical cryptosystems
- Main drawback: large keys (often $\geq$ 50 kByte) vs. embedded devices
- Misoczki et al. proposed quasi-cyclic medium-density parity check codes (QC-MDPC) (4800 bit pk, 80 bit security level) [MTSB12]
- Open questions
  - How does QC-MDPC McEliece perform on embedded devices?
  - Which decoders should be used?
  - Can known decoders be improved?



Overview

1. Motivation
2. **Background**
3. Efficient Decoding of MDPC Codes
4. Implementing QC-MDPC McEliece
5. Results
6. Conclusions

2. Background on MDPC Codes

- Original McEliece uses binary Goppa codes
- Main problem: large keys
- Many proposals to use codes with more compact representations, several were broken
- [MRS00,BCG06,BCG07,BC07,BBC08] say: use low-density parity check (LDPC) codes or even quasi-cyclic LDPC codes!
- [OTD10] cryptanalyzed some (QC-)LDPC proposals
- [MTSB12] say: use (QC-)MDPC codes, they resist known LDPC attacks and give small keys!
- Not broken (yet?)

2. Background on MDPC Codes

Definition 1 (Linear codes).
A binary (n, r)-linear code C of length n, dimension $(n - r)$ and co-dimension r, is a $(n - r)$-dimensional vector subspace of F_2^n.

It is spanned by the rows of a matrix $G \in F_2^{(n-r)n}$, called a generator matrix of C.

The generator matrix is the kernel of a matrix $H \in F_2^{r*n}$ and called the parity-check matrix of C.

The codeword $c \in C$ of a vector $m \in F_2^{(n-r)}$ is given by $c = mG$. Given a vector $e \in F_2^n$, we obtain the syndrome $s = He^T \in F_2^r$.

2. Background on MDPC Codes

Definition 2 (Quasi-cyclic codes).
A (n, r)-linear code is quasi-cyclic (QC) if there is some integer n_0 such that every cyclic shift of a codeword by n_0 positions is again a codeword.
When $n = n_0 p$, for some integer p, it is possible and convenient to have both generator and parity check matrices composed by $p * p$ circulant blocks.
A circulant block is completely described by its first row (or column) and the algebra of $p * p$ binary circulant matrices is isomorphic to the algebra of polynomials modulo $x^p - 1$ in F_2.

2. Background on MDPC Codes

Definition 3 (MDPC codes).
A (n, r, w)-MDPC code is a linear code of length n and co-dimension r admitting a parity check matrix with constant row weight w.

- If MDPC codes are quasi-cyclic, they are called (n, r, w)-QC-MDPC codes
- LDPC codes typically have small constant row weights (usually, less than 10)
- For MDPC codes, row weights scaling in $\mathcal{O}(\sqrt{n * \log(n)})$ are assumed

2. (QC-)MDPC McEliece

- t-error correcting (n, r, w)-QC-MDPC code with $n = n_0 p,\ r = p$

Key Generation:

1. Pick random words $h_i \in F_2^n$ of weight w_i such that w $= \sum_{i=0}^{n_0-1} w_i$
2. Define h_i as first row of parity check matrix block H_i
3. Obtain remaining $r - 1$ rows by $r - 1$ quasi-cyclic shifts of h_i
4. $H = [H_0|H_1| \ldots |H_{n_0-1}]$ is composed of n_0 circulant blocks
5. Generator matrix G is of systematic form $G = (I|Q)$,

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

2. (QC-)MDPC McEliece

Encryption:

To encrypt $m \in F_2^{(n-r)}$ into $x \in F_2^n$ select error vector $e \in F_2^n$ with $wt(e) \leq t$ at random. Then compute $\mathrm{x} \leftarrow mG + e$.

Decryption:

Let Ψ_H be a t-error-correcting MDPC decoding algorithm. Compute $mG \leftarrow \Psi_H(mG + e)$ and extract m from the first $(n - r)$ positions of mG.

Parameters for 80-bit equivalent symmetric security [MTSB12]:

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

Overview

1. Motivation
2. Background
3. **Efficient Decoding of MDPC Codes**
4. Implementing QC-MDPC McEliece
5. Results
6. Conclusions

3. Efficient Decoding of MDPC Codes

- Decoding is usually the most complex task in CBC
- Many LDPC/MDPC decoding algorithms, we focus on bit-flipping
- General decoding principle
  1. Compute syndrome s of the received codeword x
  2. Check the number of unsatisfied parity-check-equations $\#_{upc}$ associated with each codeword bit
  3. Flip each codeword bit that violates more than b equations
- Iterate until syndrome becomes zero or a predefined maximum of iterations is reached (decoding failure)
- Main difference between decoders is how threshold b is computed
  - (Pre-)compute new b for each iteration
  - $b = max_{upc}$
  - $b = max_{upc} - \delta$, for some small δ

3. Efficient Decoding of MDPC Codes

Decoder A [MTSB12]

1. Compute the syndrome
2. Compute $\#_{upc}$ for each codeword bit to determine max_{upc}
3. Compute $\#_{upc}$ again and flip all codeword bits that violate $\geq max_{upc} - \delta$ equations
4. Recompute syndrome and compare to zero

Decoder B [Gal62]

1. Compute the syndrome
2. Compute $\#_{upc}$ for each bit and directly flip the current codeword bit if $\#_{upc}$ is larger than a precomputed threshold b_i
3. Recompute syndrome and compare to zero

3. Efficient Decoding of MDPC Codes

Observations

- Decoder A and B recompute the syndrome after each iteration
- Syndrome computation is expensive!

Optimizations

- If $\#_{upc}$ exceeds the current threshold, the corresponding codeword bit j is flipped and the syndrome changes
- But the syndrome does not change arbitrarily!
  $s_{new} = s_{old} + h_j$, where h_j is the row of H corresponding to bit j
- By keeping track of which codeword bits are flipped we can update the syndrome at runtime

$\longrightarrow$ Recomputation is not required anymore

$\longrightarrow$ We always decode with a up-to-date syndrome

3. Efficient Decoding of MDPC Codes

- Derived several decoders
  - Direct vs. temporary syndrome update method
  - Combined with different threshold techniques
  - Precomputed b_i as proposed by [MTSB12]
  - For $b = max_{upc} - \delta$, chosing $\delta = 5$ requires the least iterations
  - Constantly check if syndrome becomes zero
- Measured 1000 random QC-MDPC codes with
  $$n_0 = 2, n = 9600, r = 4800, w = 90, t = 84$$
  and 100,000 random decoding tries for each decoder
- Decoding failure if no success within 10 iterations
- Measured on a Intel Xeon E5345 CPU@2.33 GHz

3. Efficient Decoding of MDPC Codes

- The following decoders require the least amount of iterations and provide the best decoding failure rates

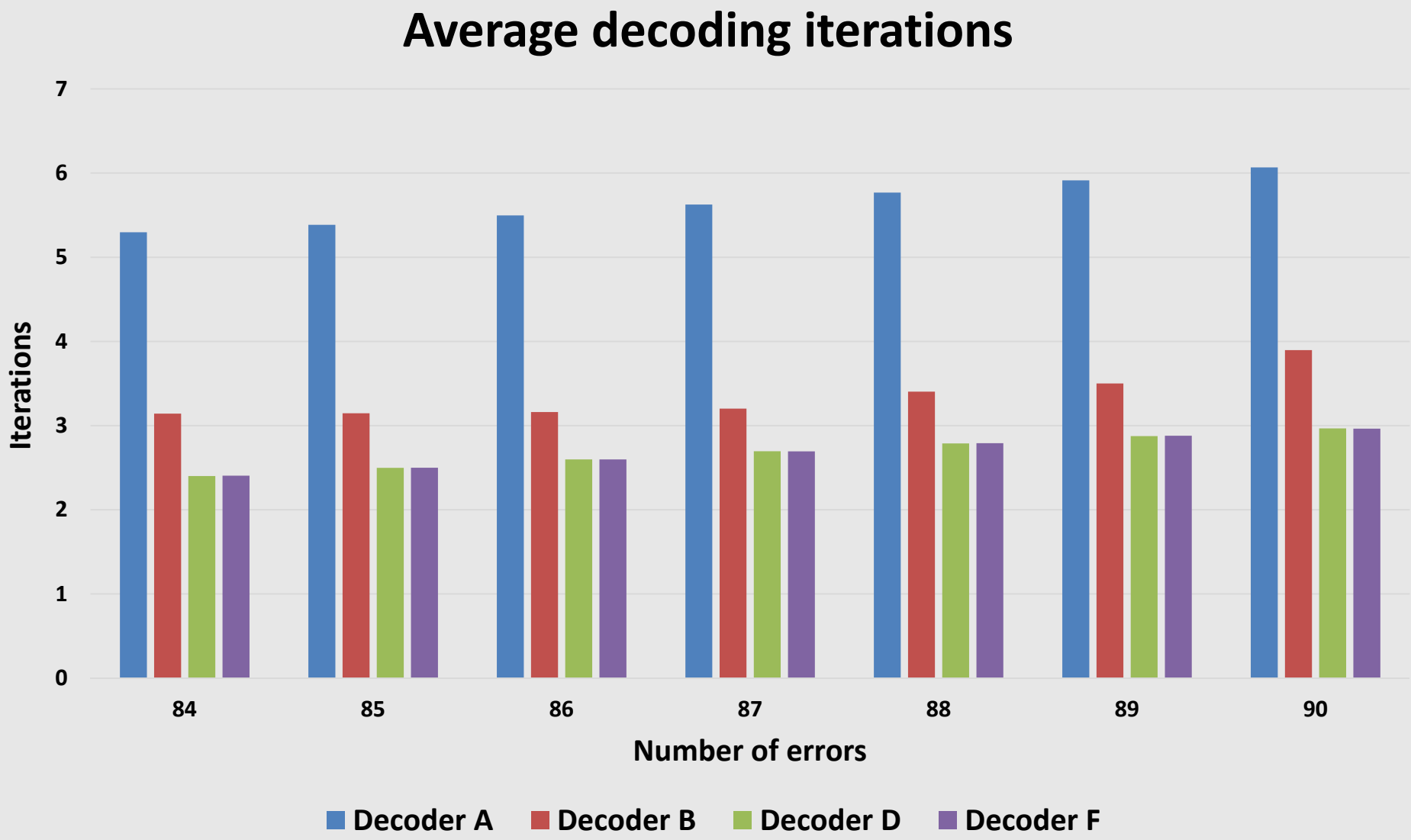
Decoder D

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

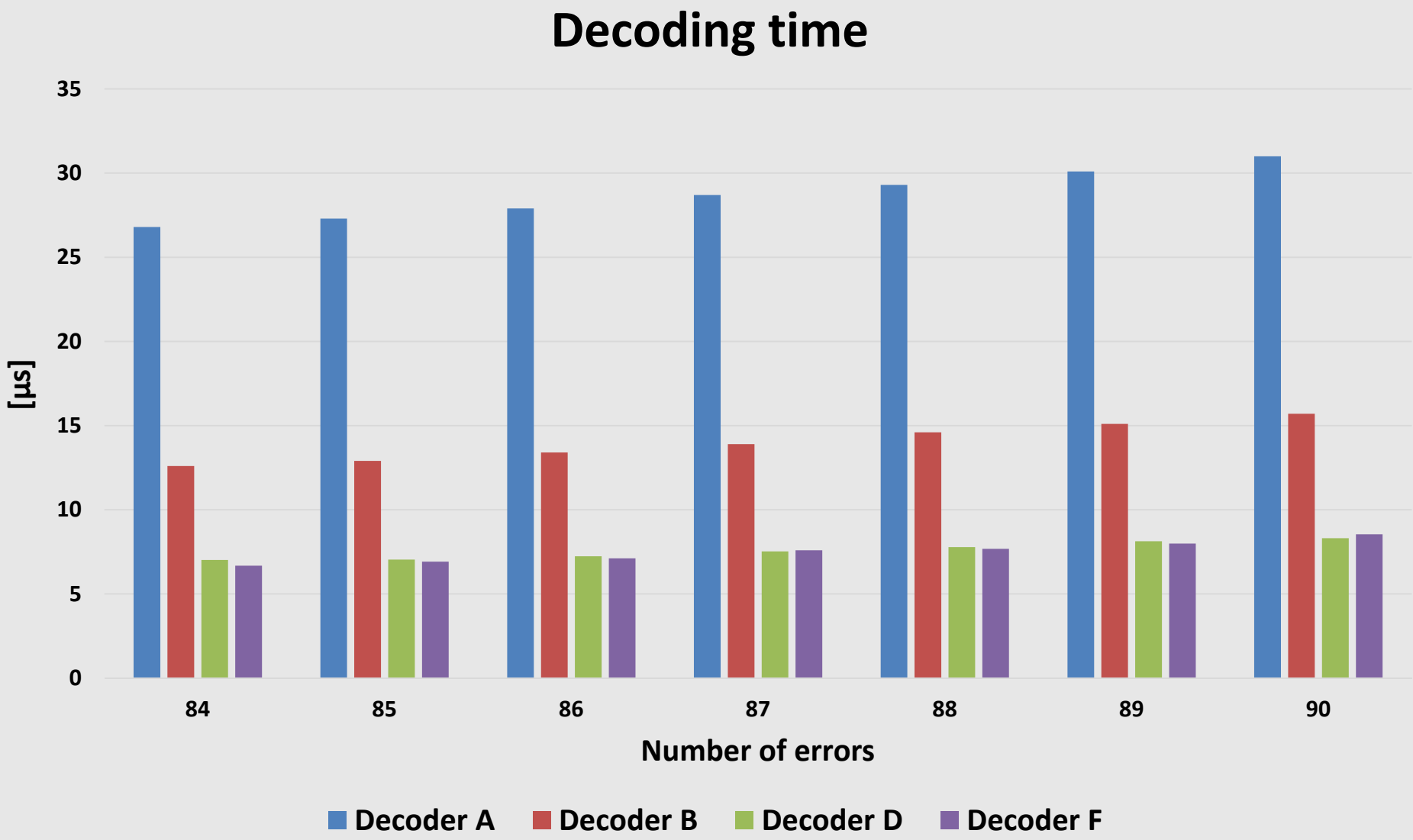
Decoder F

- Same as D, but additionally compares the syndrome to zero after each update and aborts immediately if it becomes zero

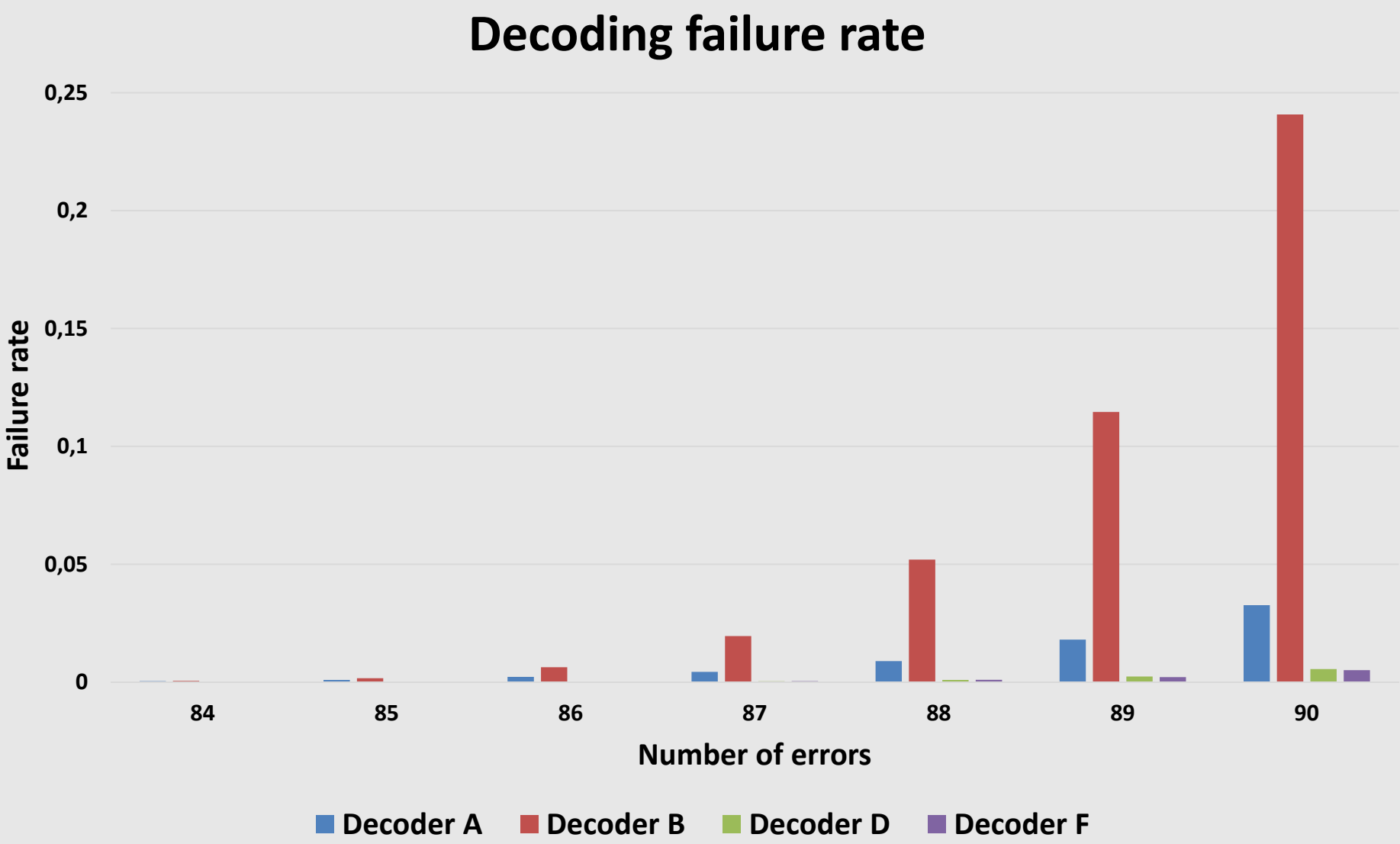
3. Efficient Decoding of MDPC Codes



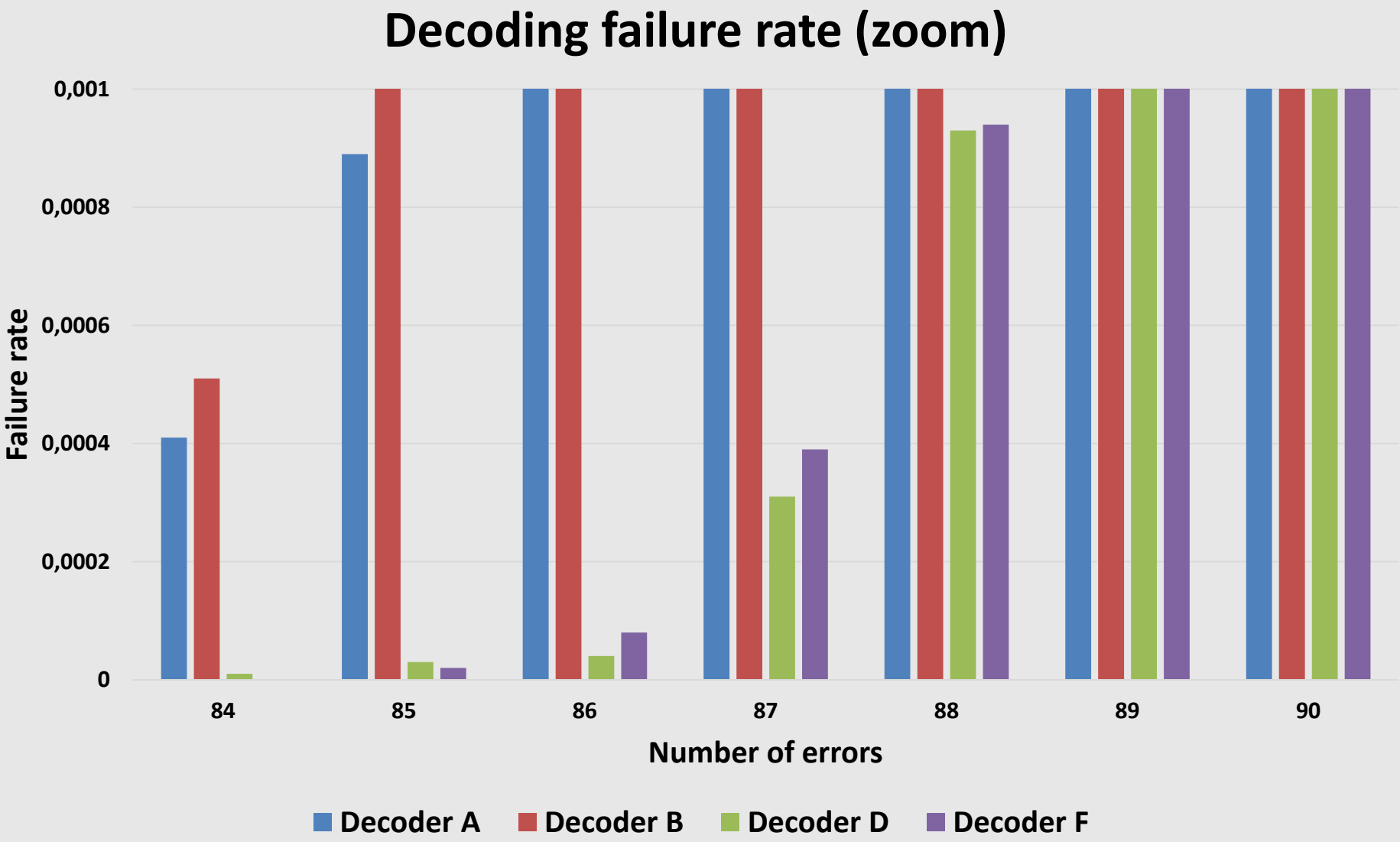
3. Efficient Decoding of MDPC Codes



3. Efficient Decoding of MDPC Codes



3. Efficient Decoding of MDPC Codes



Overview

1. Motivation
2. Background
3. Efficient Decoding of MDPC Codes
4. **Implementing QC-MDPC McEliece**
5. Results
6. Conclusions

4. Implementation Platforms

- Reconfigurable Hardware: Xilinx Virtex-6 FPGA
  - Powerful, up-to-date FPGA
  - (Ten-)thousands of slices, each slice contains four 6-input lookup tables (LUTs), eight FFs, and surrounding logic
  - Embedded resources such as block memories (BRAM) and digital signal processors (DSP)
- Embedded microcontroller: Atmel AVR ATxmega
  - Popular low-cost 8-bit microcontroller
  - Wide range of cryptographic and non-cryptographic applications

4. Implementing QC-MDPC McEliece

- Recall, parameters for 80-bit security are

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

Data sizes

- 4800-bit public key
- 9600-bit sparse secret key, 90 bits set
- 4800-bit plaintext
- 9600-bit ciphertext

- Secret key and ciphertext consist of two separate 4800-bit blocks

4. FPGA Design Considerations

- Overall FPGA design goal: high speed
- Relatively small keys → store operands directly in logic, no BRAMs
- Sparsity of secret polynomials is not exploited
  - Requires to implement 90 13-bit counters 
  - Increment all counters to generate the next row 
  - E.g., when computing the syndrome we would have to build a 4800-bit vector from the counters and XOR this vector to the current syndrome 
  - Alternatively read content of each counter and flip corresponding bits in the current syndrome 
- Implemented decoder D, early exit would require variable shifts
- Simple I/O interface keeps overhead small to get close to the actual resource consumptions
- TRNG for random error generation is out-of-scope

4. QC-MDPC McEliece FPGA Implementation

RUB

QC-MDPC Encryption

- Given first 4800-bit row g of G and message m, compute $c = mG$ and afterwards $x = c + e$
- G is of systematic form → first half of c is equal to m
- Computation of redundant part Q
  - Iterate over message bit by bit and rotate g accordingly
  - If message bit is set, XOR current g to the redundant part
  - 3x 4800-bit registers for g, m, and Q

4. 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 after the second level to minimize critical path

4. 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 until all rows of H have been checked

4. Microcontroller Design Considerations

Overall microcontroller design goal: small memory footprint

Encoder

- Straightforward: copy, rotate and accumulate
- Rolled vs. unrolled public key rotation
- Whole message is not required to start encryption
- Encrypt-while-transfer allows to hide part of the encryption time

Decoder

- Generating the next h_i requires to shift 600 bytes
- But h_0 and h_1 are sparse, storing positions of set bits just needs 45*2=90 byte
- Shifting requires to increment 45 counters
- Adding sparse to full polynomial by flipping 45 bits
- Decoder F is used

Overview

1. Motivation
2. Background
3. Efficient Decoding of MDPC Codes
4. Implementing QC-MDPC McEliece
5. **Results**
6. Conclusions

5. FPGA Results

- Post-PAR for Xilinx Virtex-6 XC6VLX240T
- Xilinx ISE 14.5
- Average decoding cycles
  - Iterative: $4{,}800 + 2 + 2.4002 * (9{,}620 + 2) = 27{,}896.7$ cycles
  - Non-iterative: $4{,}800 + 2 + 2.4002 * (4{,}810 + 2) = 16{,}351.8$ cycles

Aspect	Encoder	Decoder (iterative)	Decoder (non-iterative)
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

5. FPGA Comparison

- Performance evaluation: Time/operation vs. Mbit/s
- PK size: 0.59 kByte vs. 100.5 kByte [37], 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) [37]	XC5VLX110T	163	512	500 µs	n/a	1.0	n/a	n/a	14,537	75[1]
McEliece (dec) [37]	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 [33]	XC5VLX85T	167	163	8.60 µs	1436	18.9	n/a	10,176	3,446	0
ECC-163 [34]	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 [41]	XC5VLX30T	450	1,024	1,520 µs	684,000	0.67	n/a	n/a	3,237	5[4]

5. Microcontroller Results

Encoder

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

Decoder

- Shifting sparse poly in 720 cycles
- Adding sparse poly to syndrome in 2,200 cycles
- Very frequent memory access and looping over and over again (10 iterations over 2*4800 rows → 100k polynomial shifts)
- 2.7sec@32Mhz ☹

	Platform	SRAM	Flash	Cycles/Op	Cycles/byte
[enc]	ATxmega256	606 Byte	3,705 Byte	37,440,137	62,400
[enc unrolled]	ATxmega256	606 Byte	5,496 Byte	26,767,463	44,612
[dec]	ATxmega256	198 Byte	2,218 Byte	86,874,388	146,457

5. Microcontroller Comparison

- Much smaller than previous McEliece implementations
- Faster and smaller than RSA
- Time/op not as good as 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

1. Motivation
2. Background
3. Efficient Decoding of MDPC Codes
4. Implementing QC-MDPC McEliece
5. Results
6. **Conclusions**

6. Conclusions

- Proposed optimized decoders and showed their advantage over existing decoders
- High throughput FPGA and low memory footprint microcontroller implementations of QC-MDPC McEliece with **practical** key sizes
- Provided another incentive for further cryptanalytical investigation of QC-MDPC codes to establish confidence in the scheme
- Smaller Keys for Code-based Cryptography: QC-MDPC McEliece Implementations on Embedded Devices, Stefan Heyse, Ingo von Maurich, Tim Güneysu, *Workshop on Cryptographic Hardware and Embedded Systems, CHES 2013*, Santa Barbara, August 20-23, 2013, to appear.
- Paper & source code (C and VHDL) available at

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

RUHR-UNIVERSITÄT BOCHUM

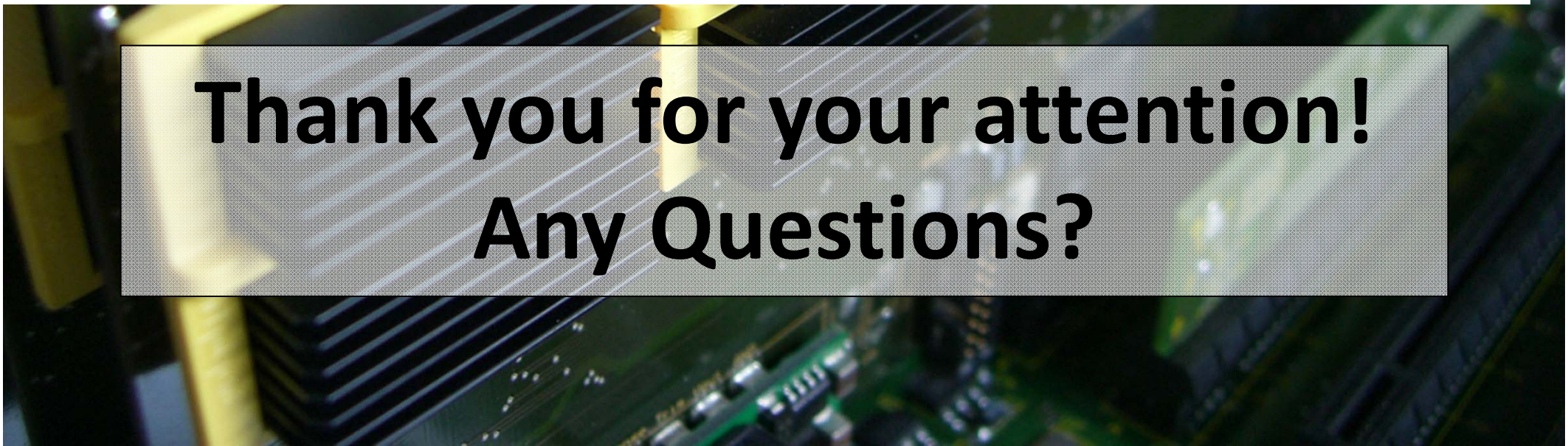
RUB

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

4th Code-based Cryptography Workshop 2013, Rocquencourt, France

Stefan Heyse, Ingo von Maurich and Tim Güneysu
Horst Görtz Institute for IT-Security, Ruhr University Bochum, Germany

June 10, 2013



References

[BBC08] M. Baldi, M. Bodrato, and F. Chiaraluce. A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes. In R. Ostrovsky, R. D. Prisco, and I. Visconti, editors, SCN, volume 5229 of Lecture Notes in Computer Science, pages 246–262. Springer, 2008.

[BC07] M. Baldi and F. Chiaraluce. Cryptanalysis of a New Instance of McEliece Cryptosystem Based on QC-LDPC Codes. In Information Theory, ISIT '07. IEEE International Symposium on, pages 2591–2595, 2007.

[BCG06] M. Baldi, F. Chiaraluce, and R. Garello. On the Usage of Quasi-Cyclic Low-Density Parity-Check Codes in the McEliece Cryptosystem. In Communications and Electronics, ICCE '06. First International Conference on, pages 305–310, 2006.

[BCG+07] M. Baldi, F. Chiaraluce, R. Garello, and F. Mininni. Quasi-Cyclic Low-Density Parity-Check Codes in the McEliece Cryptosystem. In Communications, ICC '07. IEEE International Conference on, pages 951–956, 2007.

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

[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/

References

[MRS00] C. Monico, J. Rosenthal, and A. Shokrollahi. Using Low Density Parity Check Codes in the McEliece Cryptosystem. In Information Theory, IEEE International Symposium on, page 215, 2000.

[OTD10] A. Otmani, J.-P. Tillich, and L. Dallot. Cryptanalysis of Two McEliece Cryptosystems Based on Quasi-Cyclic Codes. Mathematics in Computer Science, 3(2):129–140, 2010.