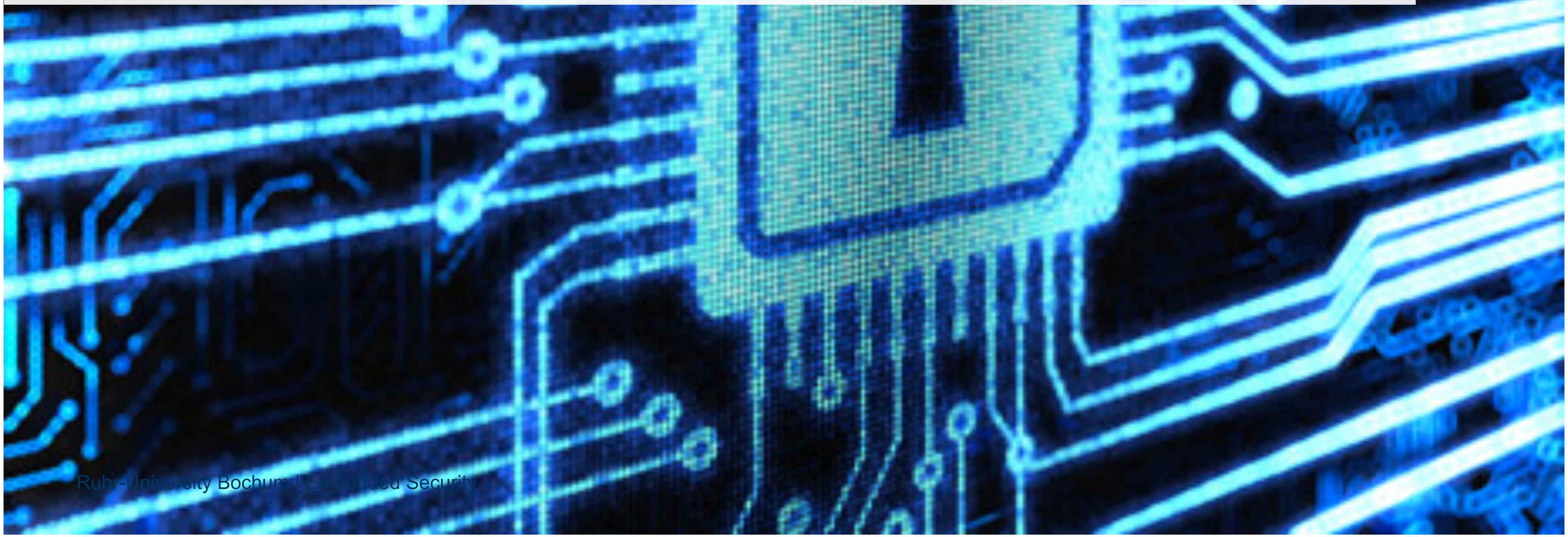
RUHR-UNIVERSITÄT BOCHUM

Towards One Cycle per Bit Asymmetric Encryption: Code-Based Cryptography on Reconfigurable Hardware

Stefan Heyse, Tim Güneysu

CHES 2012 – Leuven, Belgium

11.09.2012



Outline

- **Introduction**
- Background in code based crypto
- McEliece vs. Niederreiter
- Our implementation
- Results and conclusion

Introduction

- We need alternatives to classical schemes for larger diversification and to resist (possible?) quantum computer attacks
- Nearly all alternative PKCS are hindered by large keys
- Already shown that they can be fast
- **How fast can we get?**
- **Is McEliece or Niederreiter faster** (in standard scenario)?

Outline

- Introduction
- **Background in code based crypto**
- McEliece vs. Niederreiter
- Our implementation
- Results and conclusion

Goppa Codes

- Subgroup of error correcting code
- Belongs to the huge family of alternant codes
- Can be described by **Goppa polynomial g(z)** of degree s and a list of field elements called **support L**.

$$g(z) = \sum_{i=0}^{t} g_i z^i \in \mathbb{F}_{2^m}[z] \qquad \mathcal{L} = \{\alpha_0, \cdots, \alpha_{n-1}\} \qquad \alpha_i \in \mathbb{F}_{2^m}$$

Parity check matrix of Goppa Codes

- By evaluation g(z) in the elements of the support L we can construct the **parity check matrix H** as

$$H = \begin{pmatrix} \frac{g_s}{g(\alpha_0)} & \frac{g_s}{g(\alpha_1)} & \cdots & \frac{g_s}{g(\alpha_{n-1})} \\ \frac{g_{s-1}+g_s\cdot\alpha_0}{g(\alpha_0)} & \frac{g_{s-1}+g_s\cdot\alpha_0}{g(\alpha_1)} & \cdots & \frac{g_{s-1}+g_s\cdot\alpha_0}{g(\alpha_{n-1})} \\ \vdots & \ddots & & \vdots \\ \frac{g_1+g_2\cdot\alpha_0+\cdots+g_s\cdot\alpha_0^{s-1}}{g(\alpha_0)} & \frac{g_1+g_2\cdot\alpha_0+\cdots+g_s\cdot\alpha_0^{s-1}}{g(\alpha_1)} & \cdots & \frac{g_1+g_2\cdot\alpha_0+\cdots+g_s\cdot\alpha_0^{s-1}}{g(\alpha_{n-1})} \end{pmatrix}$$

Generator matrix of Goppa Codes

- Bringing **H** to systematic form H=(Q|ID) (by Gauss) we can derive the **generator matrix G** as $G=(ID|-Q^T)$
- $G*H^T = 0$
- m*G=c is code word of the goppa code
- m*G+e = c+e is code word with errors (up to t errors can be corrected)
- For binary Goppa codes t=s=degree of g(z), else t=floor(s/2)

- $c*H^T=syn(z)$ called **syndrome**, because it only depends on the error **e**
- If syn(z) ≠ 0 decoding algorithm (Patterson,Berlekamp-Massey,...) gives you corrected codeword and the error.

Outline

- Introduction
- Background in code based crypto
- **McEliece vs. Niederreiter**
- Our implementation
- Results and conclusion

McEliece vs. Niederreiter I

- **Classical McEliece**
  - Public key $G' = S*G*P$
  - Secret key (corresponding parity check matrix **H** defined by Goppa polynomial **g(z)** and support **L**)
  - **Decryption**
    - $c' = c*P^{-1}$
    - Decode c' to m'
    - $m = m'*S^{-1}$

- **Modern McEliece**
  - Public key **G'** in systematic form
  - Secret key (corresponding parity check matrix **H** defined by Goppa polynomial **g(z)** and
    - $c = m*G'+e$
  - **Decryption**
    - Decode directly c to m
    - S can be omitted
    - P merged into decoding algorithm

DO NOT USE MCELIECE THIS WAY.
YOU NEED a CCA2 SECURE CONVERSION!

McEliece vs. Niederreiter II

- **Classical Niederreiter**
  - Public key $H'=M*H*P$
  - Secret key (Goppa polynomial **g(z)** and support **L**)
  - **Encryption**
  - **Decryption**
    - $c'=M^{-1}*c$
    - Decode c' to e'
    - $e=P^{-1}*e'$
    - Convert e to m

YOU CAN USE NIEDERREITER LIKE THIS.

- **Modern Niederreiter**
  - Public key $H'=M*H$ in systematic form
  - Secret key (Goppa polynomial **g(z)** and permuted support **L**)
  - **Encryption**
    - $c=H'*e$
  - **Decryption**
    - $c'=M^{-1}*c$
    - Decode c' directly to e
    - Convert e to m

Security parameters

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)	$(6624, 5129, 115), 117$	$7,488$	$(1.45 \mid 84 \mid 2,183)$

Public key is a (n-k)*k bit matrix (only non-identity part)

McEliece vs. Niederreiter: existing work

- McEliece (using binary Goppa codes)
  - PC (HyMES ‘08) : 140 cycles/bit enc 2714 cycles/bit dec
  - µC (CHES’09) : 7200 cycles/bit enc 11300 cycles/bit dec
  - FPGA (ASAP’09) : 160 cycles/bit enc 446 cycles/bit dec
- Niederreiter
  - PC : (there is one-> seg fault)
  - µC (PQCrypto‘11) : 267 cycles/bit enc 30000 cycles/bit dec
  - FPGA : (only for signature scheme: 0.86s/sig)

Outline

- Introduction
- Background in code based crypto
- McEliece vs. Niederreiter
- **Our implementation**
- Results and conclusion

Niederreiter encryption

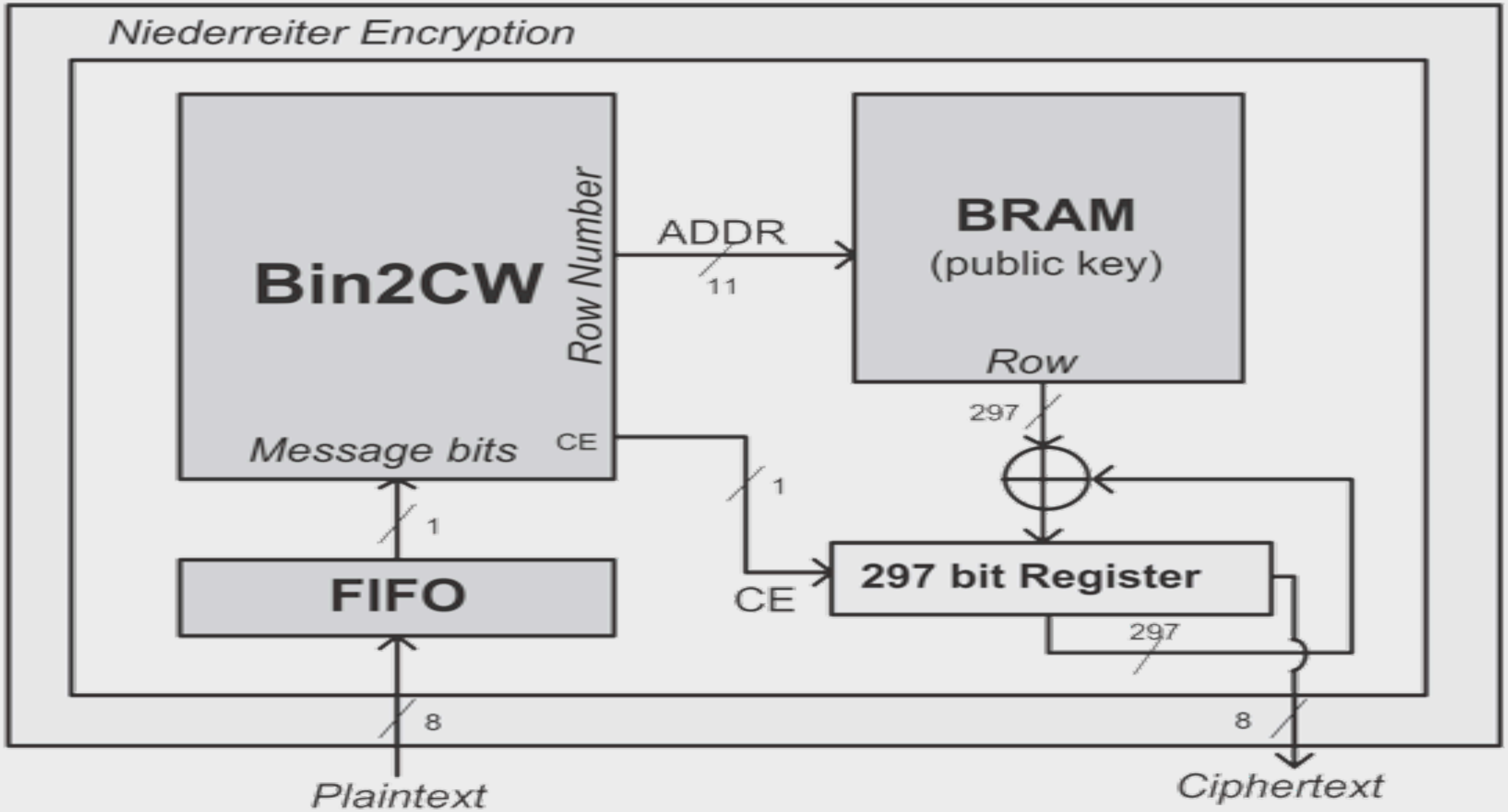
- c=H'*e is just a XOR of t=27 out of 2048 rows of H'
- Hard part is "computational expensive" mapping of m to e
- Error e is so called **constant weight word** of length n=2048 and hamming weight t=27

Algorithm 3 Encode Binary String

Input: n, t, binary stream B
Output: $\Delta[0, \ldots, t-1]$

```
δ = 0, index = 0
while t ≠ 0 do
    if n ≤ t then
        Δ[index++] = δ
        n− = 1, t− = 1, δ = 0
    end if
    u ← uTable[n, t]
    d ← (1 << u)
    if read(B, 1) = 1 then
        n− = d, δ+ = d
    else
        i ← read(B, u)
        Δ[index++] = δ + i
        δ = 0, t− = 1, n− = (i + 1)
    end if
end while
```

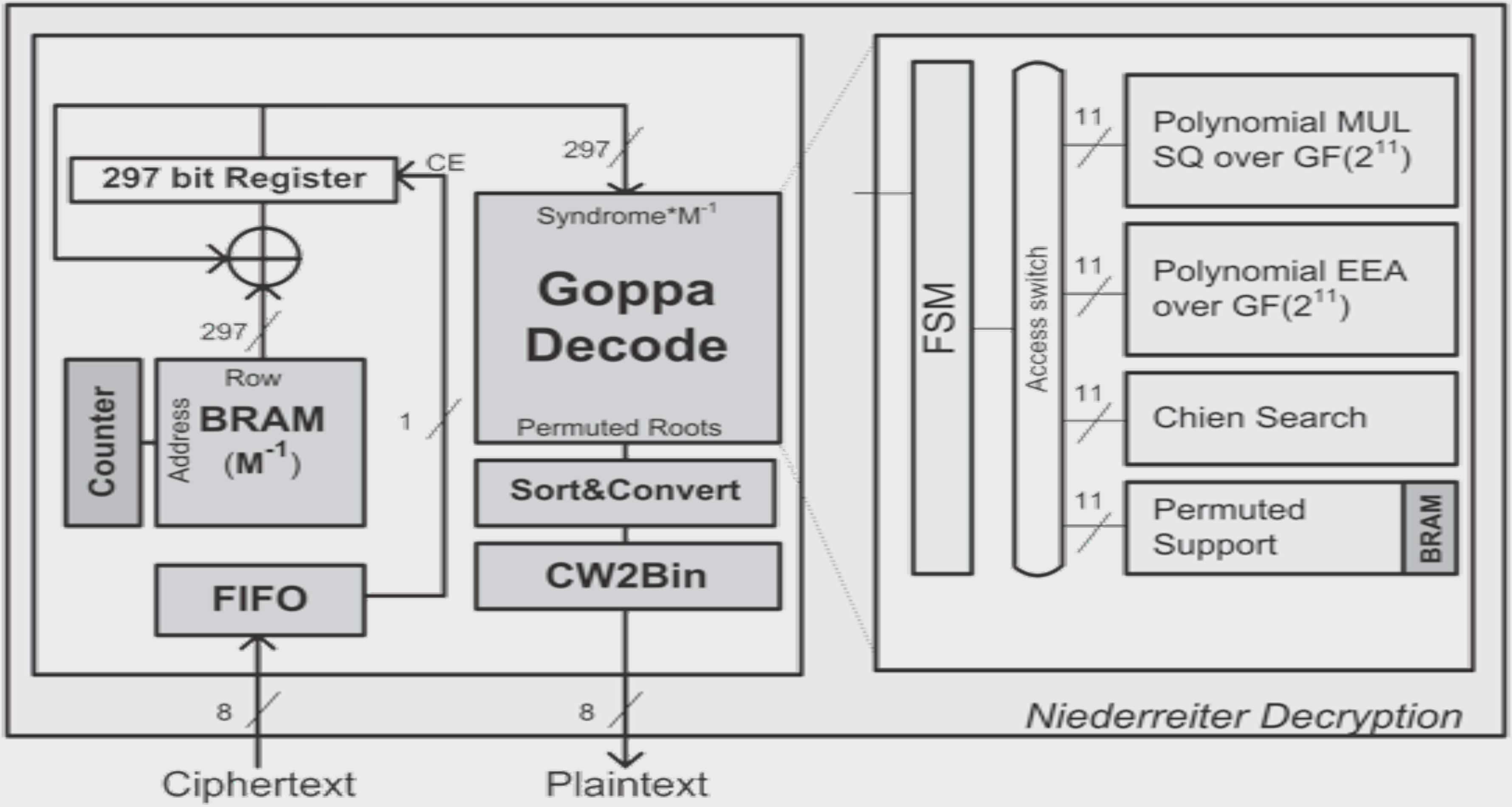
Hardware architecture for encryption



Niederreiter decryption

- Far more complex than encryption
- Multiplication with M^{-1} also just binary XOR of ~(n-k)/2 rows
- Uses Patterson algorithm for Goppa decoding
- Involved root searching is done with parallel Chien search in $3*2^m$ clock cycles

Hardware architecture for decryption



Outline

- Introduction
- Background in code based crypto
- McEliece vs. Niederreiter
- Our implementation
- **Results and conclusion**

Results

Scheme	Platform	Freq	Time/Op	Cycles/byte
This work [enc]	Virtex6-LX240T	300 MHz	0.66 µs	8.3
This work [dec]	Virtex6-LX240T	250 MHz	58.78 µs	612
McEliece [enc] [14]	Spartan3-AN1400	150 MHz	1070 µs	768
McEliece [dec] [14]	Spartan3-AN1400	85 MHz	21,610 µ	8788
This work enc	Spartan3-2000	150 MHz	1.32 µs	8.3
This work dec	Spartan3-2000	95 MHz	154 µs	612
McEliece [enc] [38]	Virtex5-LX110T	163 MHz	500 µs	389
McEliece [dec] [38]	Virtex5-LX110T	163 MHz	1400 µs	1091
This work [enc]	Virtex5-LX50T	250 MHz	0.793 µs	8.2
This work [dec]	Virtex5-LX50T	180 MHz	81 µs	612
ECC-P160 [17]	Spartan-3 1000-4	40 MHz	5.1 ms	10,200
ECC-K163 [17]	Virtex-II	128 MHz	35.75 µs	224.6
RSA-1024 random [18]	Spartan-3A	133 MHz	48.54 ms	50,436
RSA-1024 random [18]	Spartan-6	187 MHz	34.48 ms	50,373
RSA-1024 random [18]	Virtex-6	339 MHz	19.01 ms	59,258
NTRU encryption [1]	Virtex 1000EFG860	50 MHz	5 µs	8.3

Results

- Encryption of 192 bits in ~200 clock cycles means **~1 cycle/bit**
- **800** times faster than McEliece
- **4000** times faster than ECC
- Forget RSA
- Typical scenario would require a **774 GByte/sec** interface for public keys

- Decryption in 14,500 clock cycles means **~75 cycles/bit**
- **140** times faster than McEliece
- **30** times faster than ECC

Future work

- General alternant decoding (smaller and faster, despite we a working with twice as large polynomials?)
- Quasi dyadic (Goppa/Srivastava) codes in hardware
- Non typical scenario of encryption huge amounts of data with PKS (Niederreiter vs. McEliece)

RUHR-UNIVERSITÄT BOCHUM
Towards One Cycle per Bit Asymmetric Encryption: Code-Based Cryptography on Reconfigurable Hardware
Stefan Heyse, Tim Güneysu

Towards One Cycle per Bit Asymmetric Encryption: Code-Based Cryptography on Reconfigurable Hardware

Stefan Heyse, Tim Güneysu

CHES 2012 – Leuven, Belgium

11.09.2012

RUB

Thank you for your attention!

Any Questions?