Yale

FPGA-based Niederreiter Cryptosystem using Binary Goppa Codes

Wen Wang[1], Jakub Szefer[1], and Ruben Niederhagen[2]

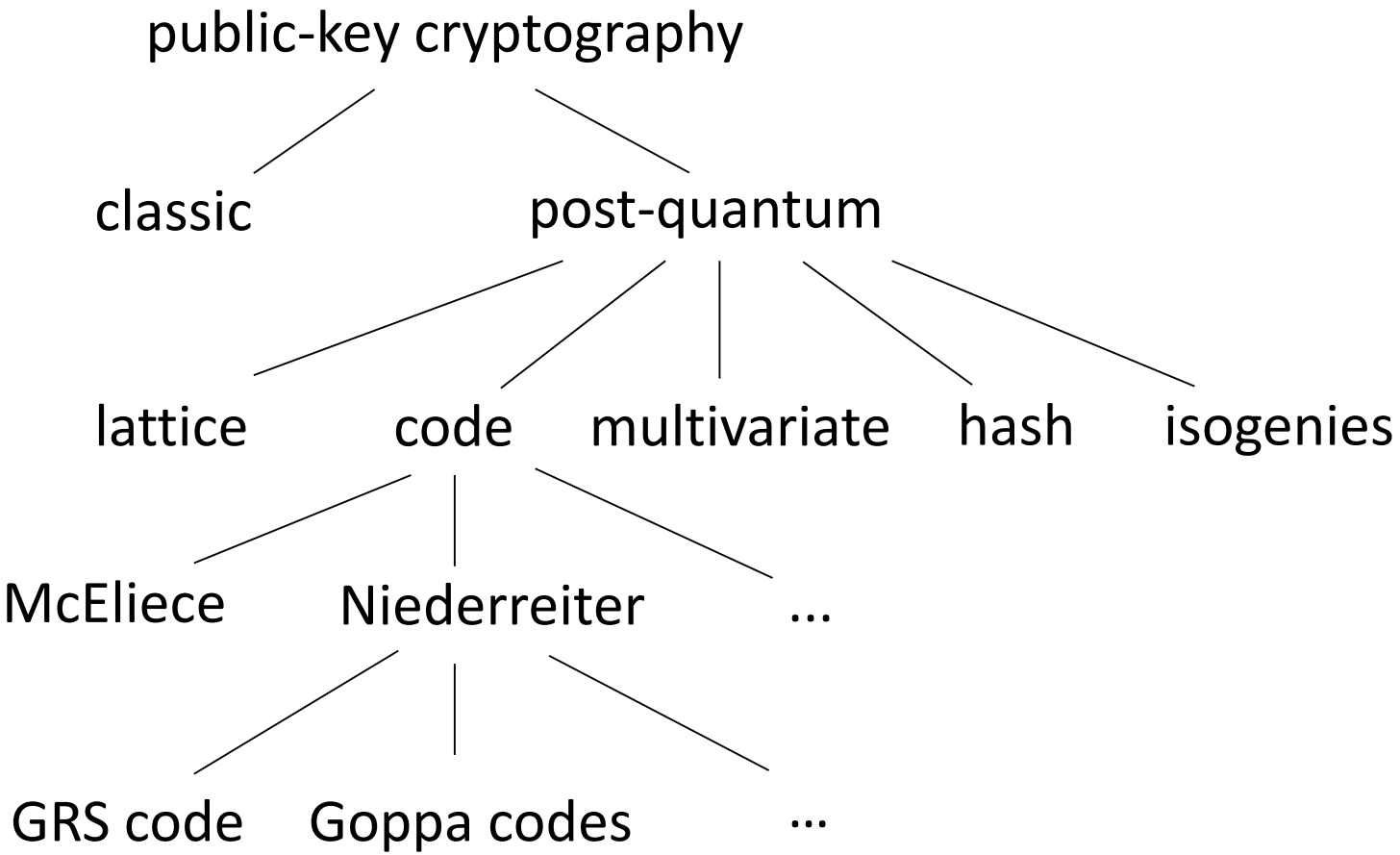
1. Yale University, USA
2. Fraunhofer Institute SIT, Germany

April 9, 2018

PQCrypto 2018 Conference

Code-Based Cryptography

Yale



Why code-based schemes in hardware?

Yale

- Code-based schemes are well-understood:
  - Long history of research
  - Security parameters widely accepted
- Code-based schemes are expensive in software.
- Parameterized hardware design can make them more practical:
  - High-throughput scenario: web-server…
  - Resource-constraint scenario: embedded devices, smart cards, …

Coding Theory Review

Yale

Binary Goppa code

- degree-t Goppa polynomial $g(x) \in GF(2^m)[x]$
- code locator $L = \{\alpha_0, \ldots, \alpha_{n-1}\},\ g(\alpha_i) \neq 0,\ \alpha_i \in GF(2^m)$
- can be defined by a parity check matrix H, e.g.,

$$C = \{c \mid Hc = 0\}$$

Niederreiter encrypt

- e: error vector of weight t
- compute syndrome $S = He$

Niederreiter decrypt

- compute e given the syndrome S and secret key $(g(x), L)$

Niederreiter Crypto System
Hardware Design
Building Blocks

Hardware Modules

Yale

- Finite field arithmetic in $GF(2^m)$.
- Polynomial arithmetic in $GF(2^m)[x]/f$.
- Merge-sort for generating a random permutation.
- Additive FFT for polynomial evaluation.
- Gaussian elimination.
- Berlekamp Massey for decoding.
- …

Hardware Modules

Yale

- Finite field arithmetic in $GF(2^m)$.
- Polynomial arithmetic in $GF(2^m)[x]/f$.
- Merge-sort for generating a random permutation.
- Additive FFT for polynomial evaluation.
- Gaussian elimination.
- Berlekamp Massey for decoding.
- …

Random Permutation

Yale

Approach: permute list of 2^m elements, pick the first n elements

- Option 1: Use shuffling algorithm

  Randomly swap two elements at a time.

  – Non-biased implementations are not constant time or need floating-point arithmetic

- Option 2: Use a constant-time sorting algorithm

  Sample 2^m random 32-bit values $r_i, i \in [0, 2^m - 1]$

  Generate a list of tuples $\{(r_0, 0), (r_1, 1), \ldots, (r_{2^m-1}, 2^m - 1)\}$

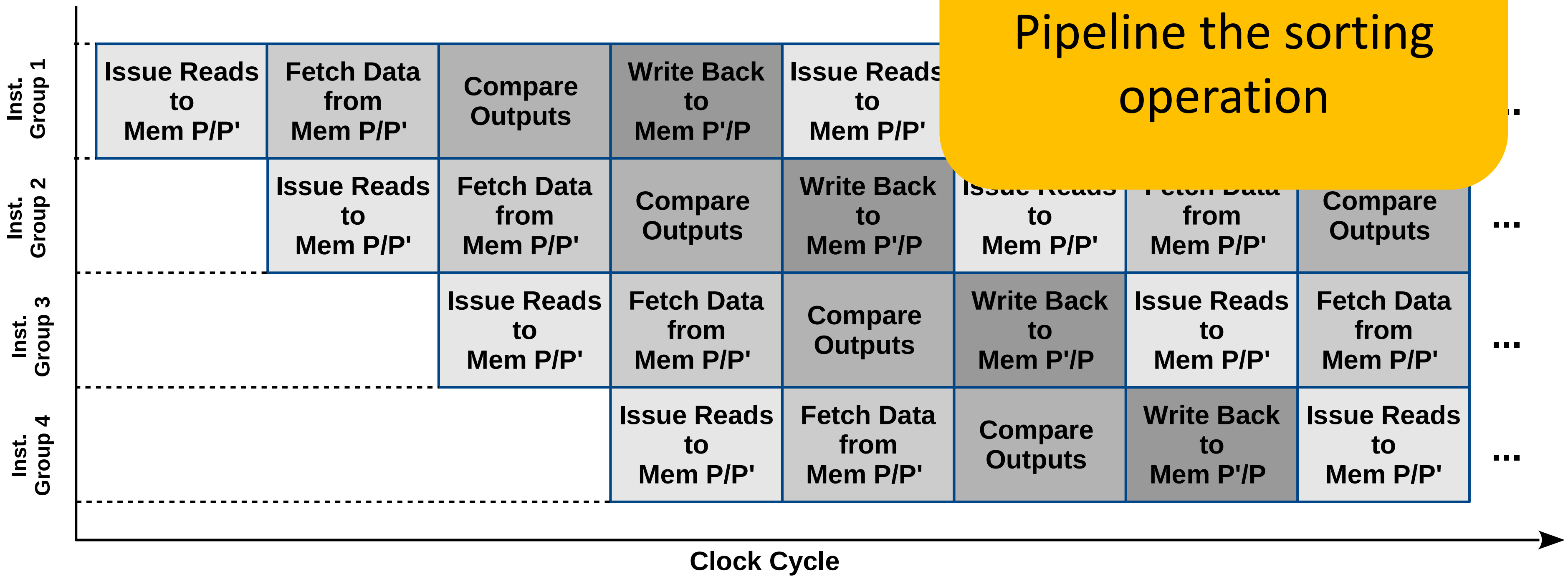
  Sort list by the first element

  Obtain the permutation by reading the second elements

  – Constant time but requires more logic and more cycles

Hardware advantage:

Pipeline the sorting operation



Computing Error Locator Polynomial

Yale

Approach: efficient decoding algorithm

- Option 1: Patterson algorithm
  - Specific for binary Goppa codes
  - Not constant time, hard to protect against timing side-channel attacks
- Option 2: Berlekamp-Massey algorithm
  - General alternant codes
  - Constant time

Constant-time Berlekamp-Massey algorithm[Xu'90]

Yale

Input: Public security parameter t, syndrome polynomial $S(x)$.
Output: Error locator polynomial $\sigma(x)$.

1: **Initialize**: $\sigma(x) = 1, \beta(x) = x, l = 0, \delta = 1$.
2: **for** k from 0 to $2t - 1$ **do**
3: $\quad d = \sum_{i=0}^{t} \sigma_i S_{k-i}$
4: $\quad$ **if** $d = 0$ or $k < 2l$:
5: $\quad\quad \{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\beta(x), l, \delta\}$.
6: $\quad$ **else:**
7: $\quad\quad \{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\sigma(x), k - l + 1, d\}$.
8: **Return** the error locator polynomial $\sigma(x)$.

[Xu'90]: Implementation of Berlekamp-Massey algorithm without inversion

Constant-time Berlekamp-Massey algorithm

Yale

Input: Public security parameter t, syndrome polynomial $S(x)$.
Output: Error locator polynomial $\sigma(x)$.

1: **Initialize**: $\sigma(x) = 1, \beta(x) = x, l = 0, \delta = 1$.
2: **for** k from 0 to $2t - 1$ **do**
3: $d = \sum_{i=0}^{t} \sigma_i S_{k-i}$
4: **if** $d = 0$ or $k < 2l$:
5: $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\beta(x), l, \delta\}$.
6: **else:**
7: $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\sigma(x), k - l + 1, d\}$.
8: **Return** the error locator polynomial $\sigma(x)$.

entry_sum

Constant-time Berlekamp-Massey algorithm

Yale

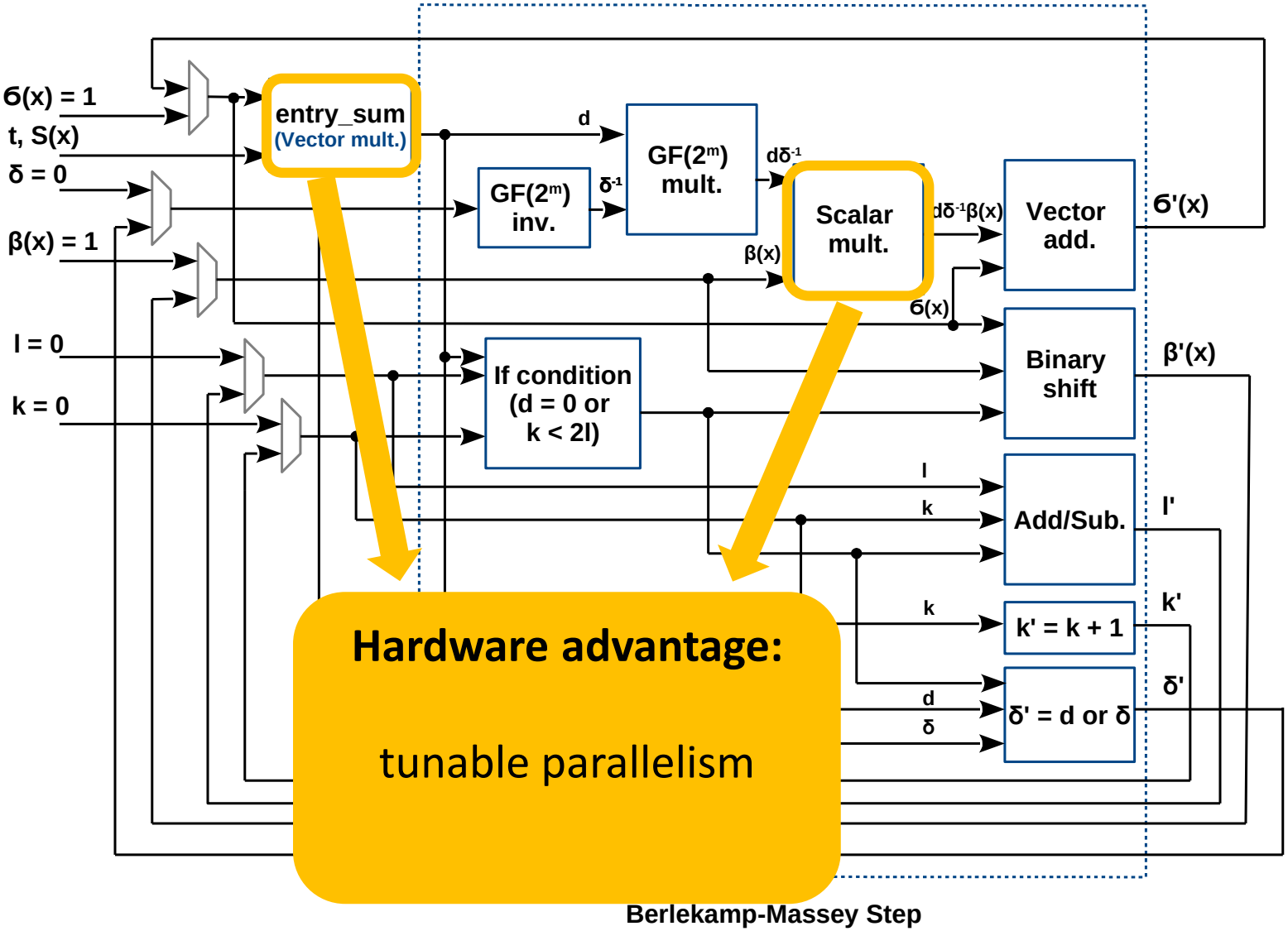
Input: Public security parameter t, syndrome polynomial $S(x)$.
Output: Error locator polynomial $\sigma(x)$.

1: **Initialize**: $\sigma(x) = 1, \beta(x) = x, l = 0, \delta = 1$.
2: **for** k from 0 to $2t - 1$ **do**
3: $d = \sum_{i=0}^{t} \sigma_i S_{k-i}$
4: **if** $d = 0$ or $k < 2l$:
5: $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\beta(x), l, \delta\}$.
6: **else:**
7: $\{\sigma(x), \beta(x), l, \delta\} = \{\sigma(x) - d\delta^{-1}\beta(x), x\sigma(x), k - l + 1, d\}$.
8: **Return** the error locator polynomial $\sigma(x)$.

entry_sum

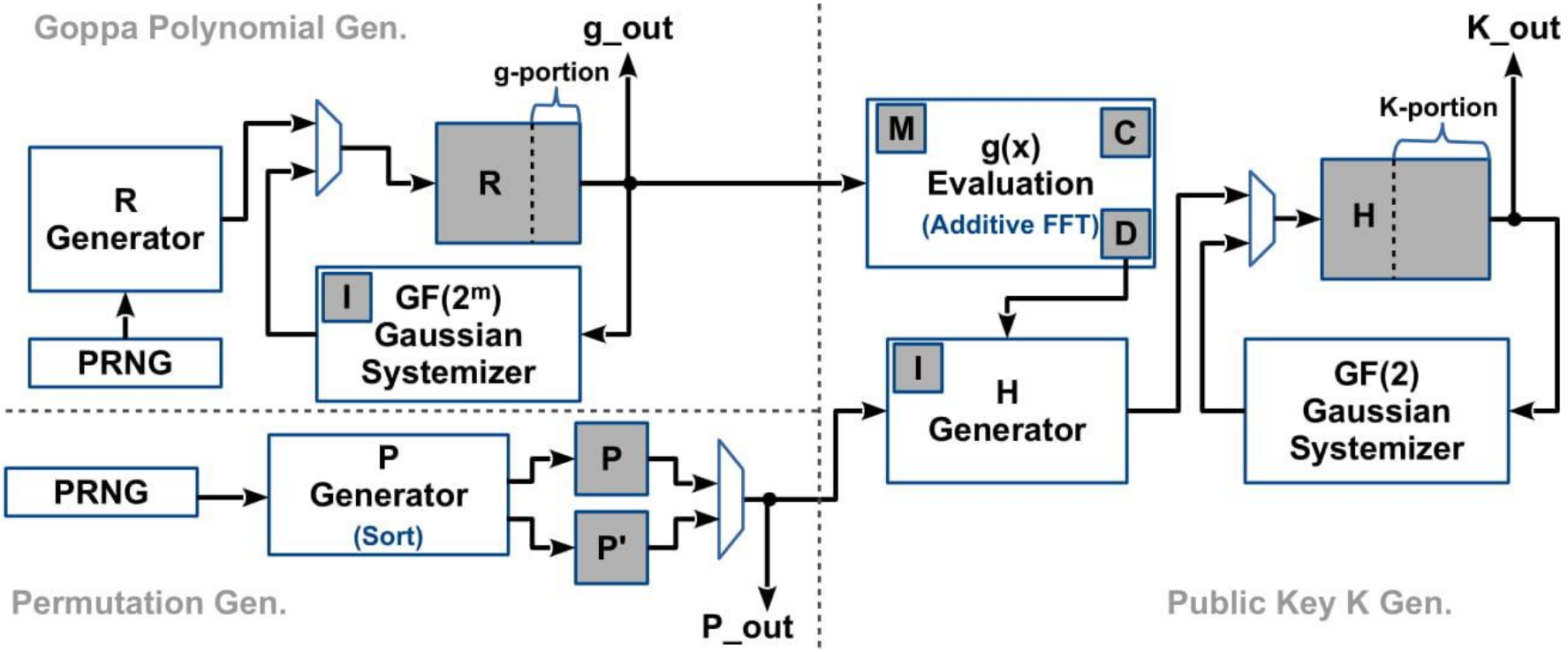
Berlekamp-Massey Step

Be


б(x) = 1
t, S(x)
δ = 0
β(x) = 1
l = 0
k = 0
entry_sum
(Vector mult.)
d
GF(2^m) inv.
δ^-1
GF(2^m) mult.
dδ^-1
Scalar mult.
β(x)
dδ^-1β(x)
Vector add.
б'(x)
б(x)
If condition (d = 0 or k < 2l)
Binary shift
β'(x)
l
k
Add/Sub.
l'
k
k' = k + 1
k'
d
δ
δ' = d or δ
δ'
Hardware advantage:
tunable parallelism
Berlekamp-Massey Step

Niederreiter Crypto System
Full Hardware design

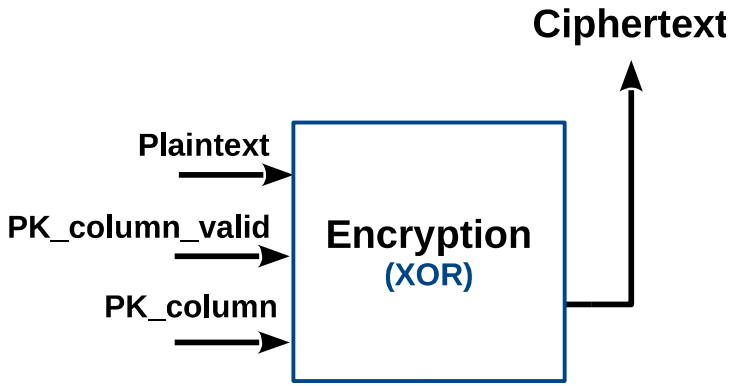
Hardware Design: Key Generator



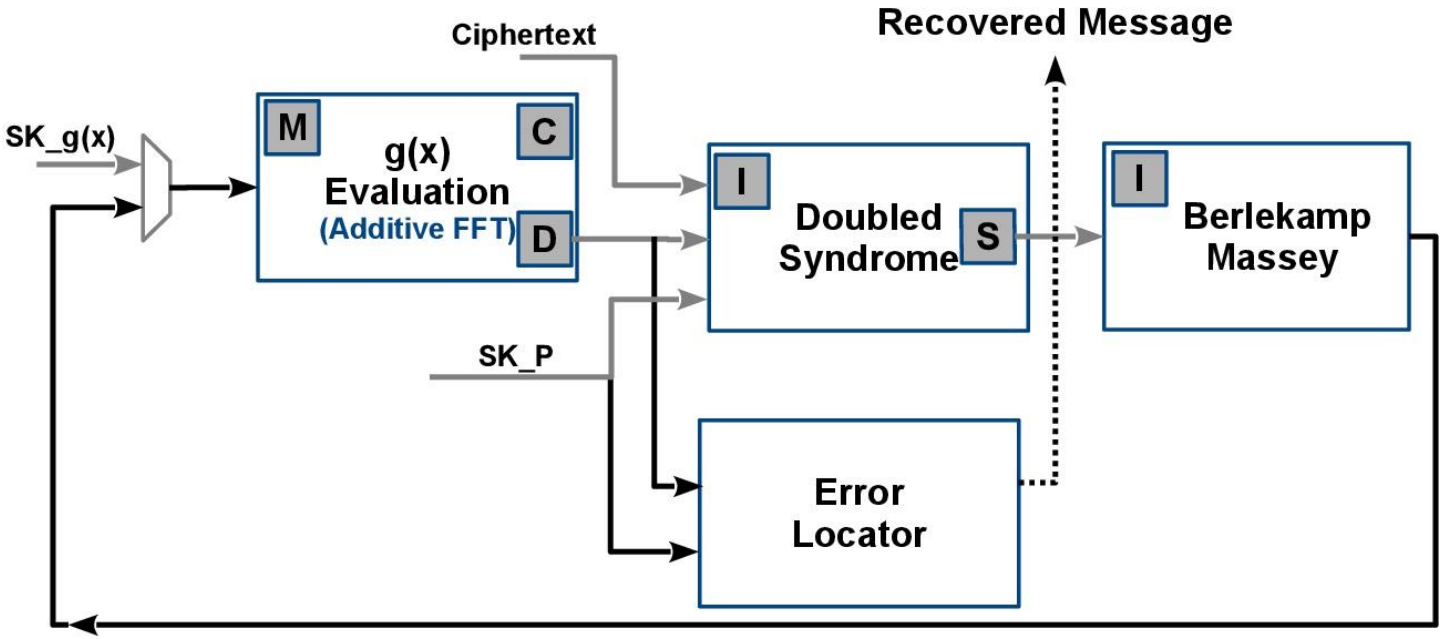
Hardware Design: Encryption and Decryption

Yale

Encryption



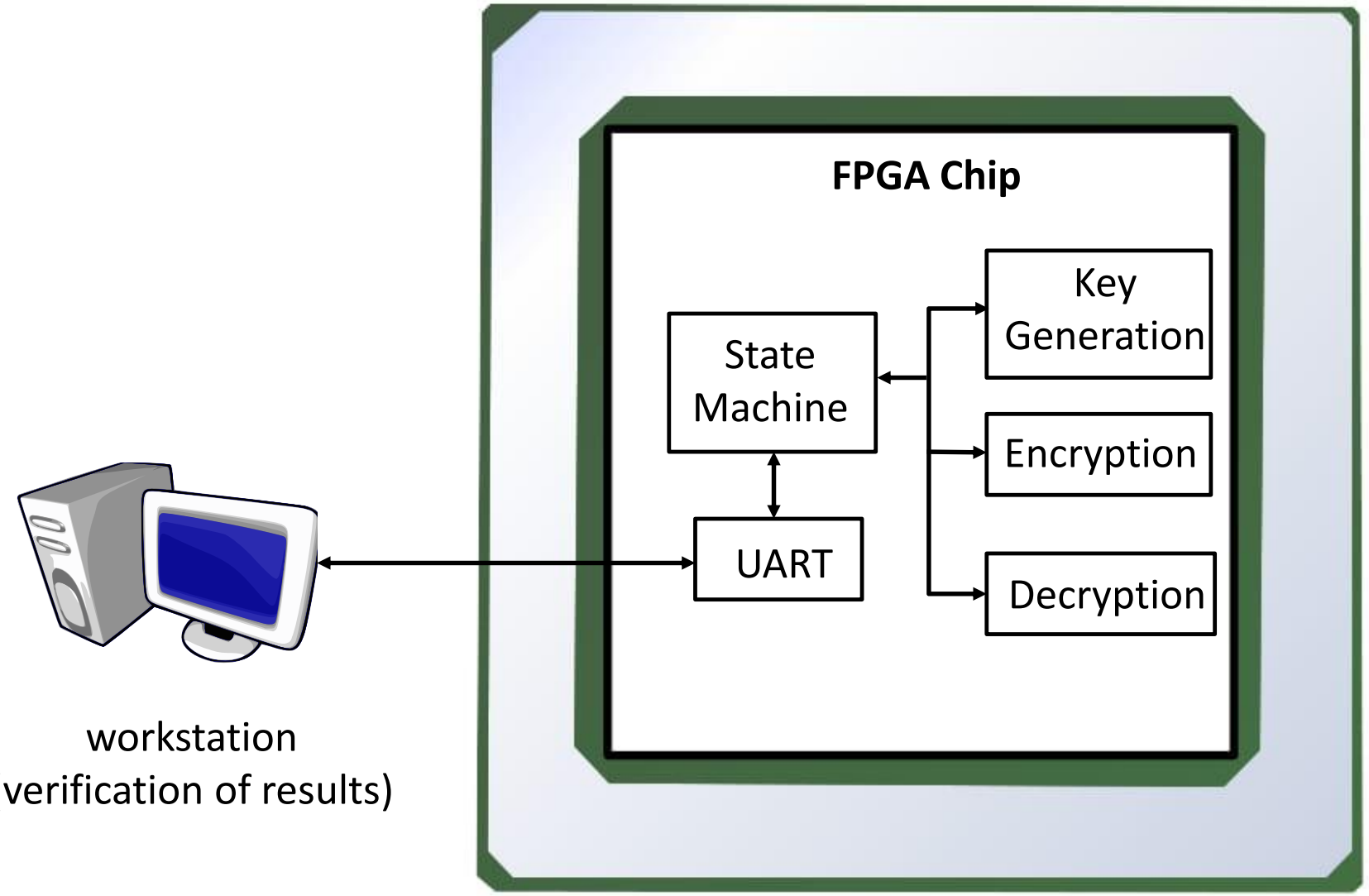
Decryption



Niederreiter Crypto System Evaluation

Testing Setup

Yale



Parameters

Yale

- Code generation scripts
- All security parameters (m, t, n) can be freely chosen.
- Performance parameters for controlling parallelism:
  - Compact, low-area design for smart cards, embedded systems, …
  - Large, high-performance design for server accelerator, …

Performance

Yale

For $m = 13, t = 119$ and $n = 6960$ (at least classical 256-bit security)

Performance for the entire Niederreiter cryptosystem (i.e., key generation, encryption and decryption) including the serial IO interface when synthesized for the Stratix V (5SGXEA7N) FPGA.

Case	Cycles		Logic	Mem.	Reg.	Fmax
	KeyGen.	Dec.				
area	11,121,214	34,492	53,447 (23%)	907 (35%)	118,243	245 MHz
bal.	3,062,936	22,768	70,478 (30%)	915 (36%)	146,648	251 MHz
time	966,400	17,055	121,806 (52%)	961 (38%)	223,232	248 MHz

Performance

Yale

Comparison with related work:

Design	Cycles			Logic	Freq. (MHz)	Mem.	Time (ms)		
	KeyGen.	Dec.	Enc.				Gen.	Dec.	Enc.
m = 11, t = 50, n = 2048, Virtex 5 LX110									
Shoufan et al.	14,670,000	210,300	81,500	14,537 (84%)	163	75	90.00	1.29	0.50
This design	1,503,927	5,864	1,498	6,660 (38%)	180	68	8.35	0.03	0.01
m = 12, t = 66, n = 3307, Virtex 6 LX240									
Massolino et al.	—	28,887	—	3307	162	15	—	0.18	—
This design	—	10,228	—	6571	267	23	—	0.04	—
m = 13, t = 128, n = 8192, Hawell vs. Stratix V									
Chou	1,236,054,840	343,344	289,152	—	4,000	—	309.0	0.09	0.07
This design	1,173,750	17,140	6,528	129,059 (54%)	231	1,126	5.08	0.07	0.07

(Logic is given in “Slices” for Xilinx Virtex FPGAs and in “ALMs” for Altera Stratix FPGAs.)

Conclusion

Yale

- First constant-time hardware-based Niederreiter cryptosystem with over 256-bit security level.
- 60 x faster compared to the fastest-to-date software implementation.
- Fully parameterized design for different use cases.

Thank you for your attention!

Key Generator Algorithm

Yale

Input: System parameters: m, t and n.
Output: Private key $(g(x), (\alpha_0, \alpha_1, \dots, \alpha_{n-1}))$ and the public key K.

1: Choose a random sequence $(\alpha_0, \alpha_1, \dots, \alpha_{n-1})$ of n distinct elements in $GF(2^m)$.
2: Choose a random polynomial $g(x)$ such that $g(\alpha) \neq 0$ for all $\alpha \in (\alpha_0, \alpha_1, \dots, \alpha_{n-1})$.
3: Compute the $t \times n$ parity check matrix

$$H = \begin{pmatrix} 1/g(\alpha_0) & 1/g(\alpha_1) & \cdots & 1/g(\alpha_{n-1}) \\ \alpha_0/g(\alpha_0) & \alpha_1/g(\alpha_1) & \cdots & \alpha_{n-1}/g(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{t-1}/g(\alpha_0) & \alpha_1^{t-1}/g(\alpha_1) & \cdots & \alpha_{n-1}^{t-1}/g(\alpha_{n-1}) \end{pmatrix}$$

4: Transform H to a $mt \times n$ binary parity check matrix H'.
5: Transform H' into its systematic form $[\mathbb{I}_{mt}|K]$.
6: **Return** the private key $(g(x), (\alpha_0, \alpha_1, \dots, \alpha_{n-1}))$ and the public key K.

Encryption Algorithm

Yale

Input: Plaintext e, public key K.
Output: Ciphertext c.

1: Compute $c = [\mathbb{I}_{mt}|K] \times e$.
2: **Return** the ciphertext c.

Decryption Algorithm

Yale

Input: Ciphertext c, secret key $(g(x), (\alpha_0, \alpha_1, \ldots, \alpha_{n-1}))$.
Output: Plaintext e.

1: Compute the double-size $2t \times n$ parity check matrix

$$H^2 = \begin{pmatrix} 1/g^2(\alpha_0) & 1/g^2(\alpha_1) & \cdots & 1/g^2(\alpha_{n-1}) \\ \alpha_0/g^2(\alpha_0) & \alpha_1/g^2(\alpha_1) & \cdots & \alpha_{n-1}/g^2(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1}/g^2(\alpha_0) & \alpha_1^{2t-1}/g^2(\alpha_1) & \cdots & \alpha_{n-1}^{2t-1}/g^2(\alpha_{n-1}) \end{pmatrix}$$

2: Transform H^2 to a $2mt \times n$ binary parity check matrix $H'^{(2)}$.
3: Compute the double-size syndrome: $S^2 = H'^{(2)} \times (c\,|0)$.
4: Compute the error-locator polynomial $\sigma(x)$ from S^2.
5: Evaluate the error-locator polynomial $\sigma(x)$ at $(\alpha_0, \alpha_1, \ldots, \alpha_{n-1})$.
6: **Return** the plaintext e.