Hardware Implementation of the Code-based Key Encapsulation Mechanism using Dyadic GS Codes (DAGS)

Viet Dang and Kris Gaj

ECE Department

George Mason University

Fairfax, VA, USA

Introduction to DAGS

- The first KEM using quasi-dyadic approach for Generalized Srivastava codes
- Achieve IND-CCA security by applying recent framework in Hofheinz et al.
- “Shortish” public and private keys
- Relatively efficient encapsulation and decapsulation algorithm

DAGS Sizes

Parameter Set	Public Key Size (in bytes)	Private Key Size (in bytes)	Ciphertext Size (in bytes)
DAGS_1	6,760	2,496	552
DAGS_3	8,448	3,648	944
DAGS_5	11,616	6,336	1,616

DAGS Key Encapsulation Mechanism

Alice **Bob**

KEM.KeyGen → (sk,pk)

pk →

KEM.Encaps(pk) → (K,C)

← **C**

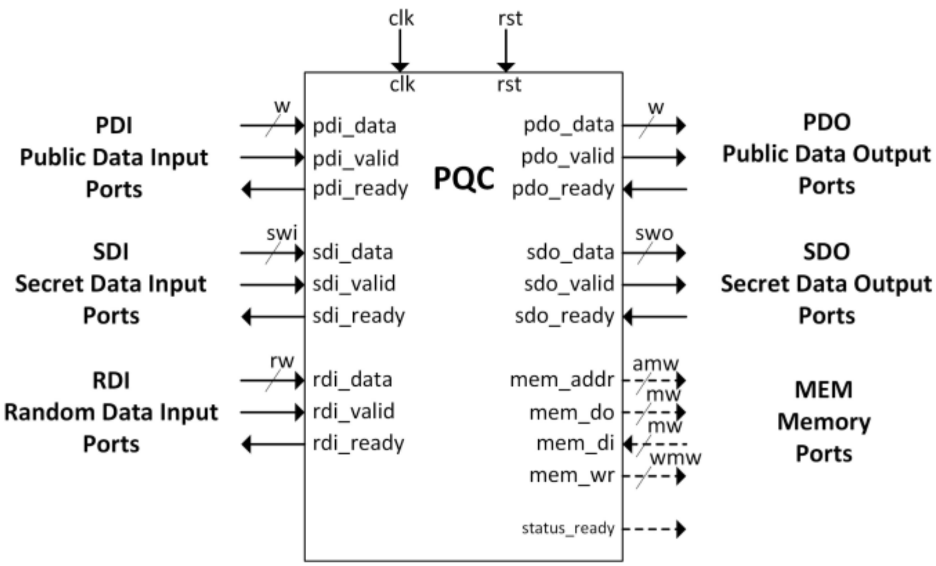
KEM.Decaps(sk) → K

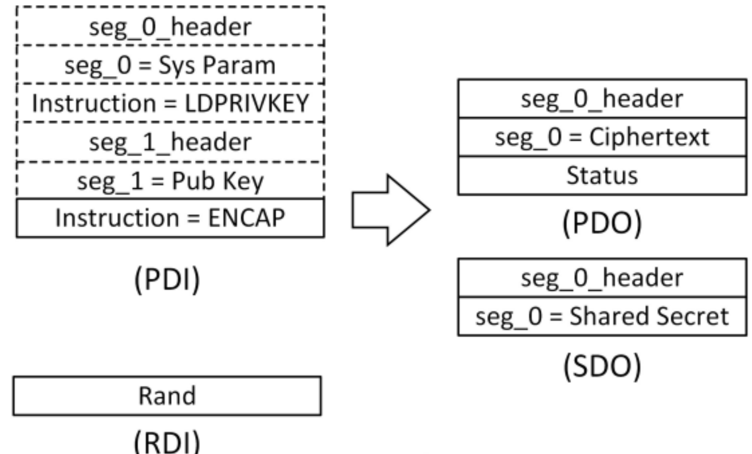
Shared Key := K

Design Methodology

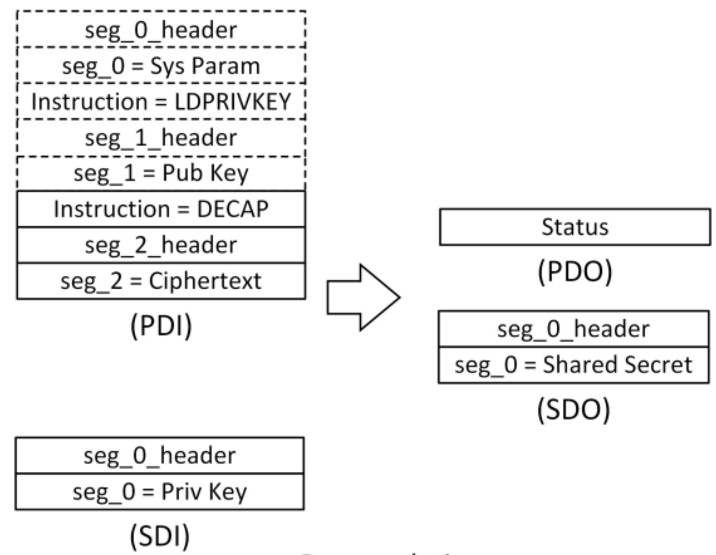
- Optimization for speed
  - Minimum latency
  - Maximum number of operations per second
- Key generation performed externally, e.g., in software
- No countermeasures against side-channel attacks
- Full compliance with the latest DAGS specification
- Single module for both Encapsulation and Decapsulation

GMU Hardware API





Encapsulation



Decapsulation

Design Methodology

- Language: VHDL
- Approach: Manual design based on specification & reference software implementation
- Verification: Simulation using test vectors generated using reference software implementation
- Simulator: Vivado Simulator
- Synthesis & Implementation: Vivado ver.2017.2
- Target:
  - FPGA Family: Xilinx Kintex-7 UltraSCALE
  - Device: XCKU035-FFVA1156
  - Technology: 20nm CMOS
- FPGA Tool Option Optimization: Minerva (developed by GMU)

DAGS parameters

	Description	DAGS_3	DAGS_5
n	Code length	1216	2112
k	Code dimension	512	704
w	Number of errors	176	352
l	Shared secret length	64	64
F_q	Base Field/ Subfield	F_{2^6}	F_{2^6}
F_{q^m}	Extension Field	$F_{2^{12}}$	$F_{2^{12}}$

Multiplication in Extension Field

- Reduce extension field multiplication to base field multiplication

$p, q \epsilon GF(2^{12}), a_1, b_1, a_2, b_2 \epsilon GF(2^6)$

$$\begin{cases} p = a_1 x + b_1 \\ q = a_2 x + b_2 \end{cases}$$

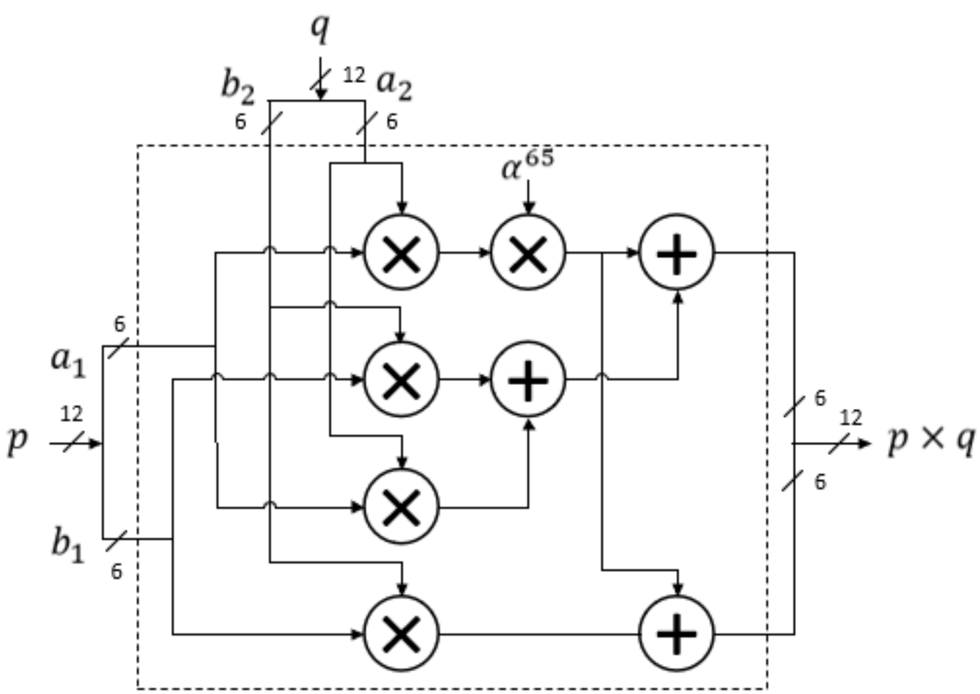
$$p \times q = (a_1 x + b_1)(a_2 x + b_2) \, mod \, x^2 + \alpha^{65} x + \alpha^{65} =$$

$$= x(a_1 a_2 \alpha^{65} + a_1 b_2 + a_2 b_1) + a_1 a_2 \alpha^{65} + b_1 b_2$$

$\alpha^{65} = \gamma : a\ primitive\ element\ in\ base\ field$

Multiplication in Extension Field

- $p \times q = x(a_1 a_2 \alpha^{65} + a_1 b_2 + a_2 b_1) + a_1 a_2 \alpha^{65} \;\; + b_1 b_2$



Resources used:

4 MUL, 1 CMUL, 3 ADD

Critical path:

1 MUL + 1 CMUL + 1 ADD

Direct Inversion in Extension Field

- Direct inversion: reduces extension field inversion to subfield inversion.

$p, q \in GF(2^{12}), a_1, b_1, a_2, b_2 \in GF(2^6)$
$q = p^{-1}$

$$\begin{cases} p = a_1 x + b_1 \\ q = a_2 x + b_2 \end{cases}$$

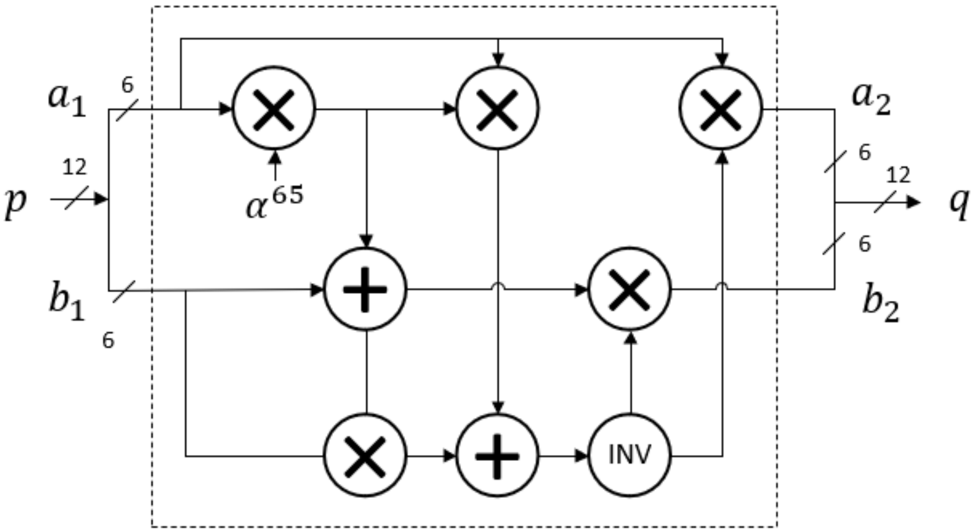
$$p \times q = (a_1 x + b_1)(a_2 x + b_2) \; mod \; x^2 + \alpha^{65} x + \alpha^{65} = 1$$

$$\begin{cases} a_2 = a_1 \left(\alpha^{65} a_1^2 + b_1(\alpha^{65} a_1 + b_1)\right)^{-1} \\ b_2 = (\alpha^{65} a_1 + b_1) \left(\alpha^{65} a_1^2 + b_1(\alpha^{65} a_1 + b_1)\right)^{-1} \end{cases}$$

Direct Inversion in Extension Field

$$a_2 = a_1 \left(\alpha^{65} a_1^2 + b_1(\alpha^{65} a_1 + b_1)\right)^{-1}$$

$$b_2 = (\alpha^{65} a_1 + b_1) \left(\alpha^{65} a_1^2 + b_1(\alpha^{65} a_1 + b_1)\right)^{-1}$$



Resources used:

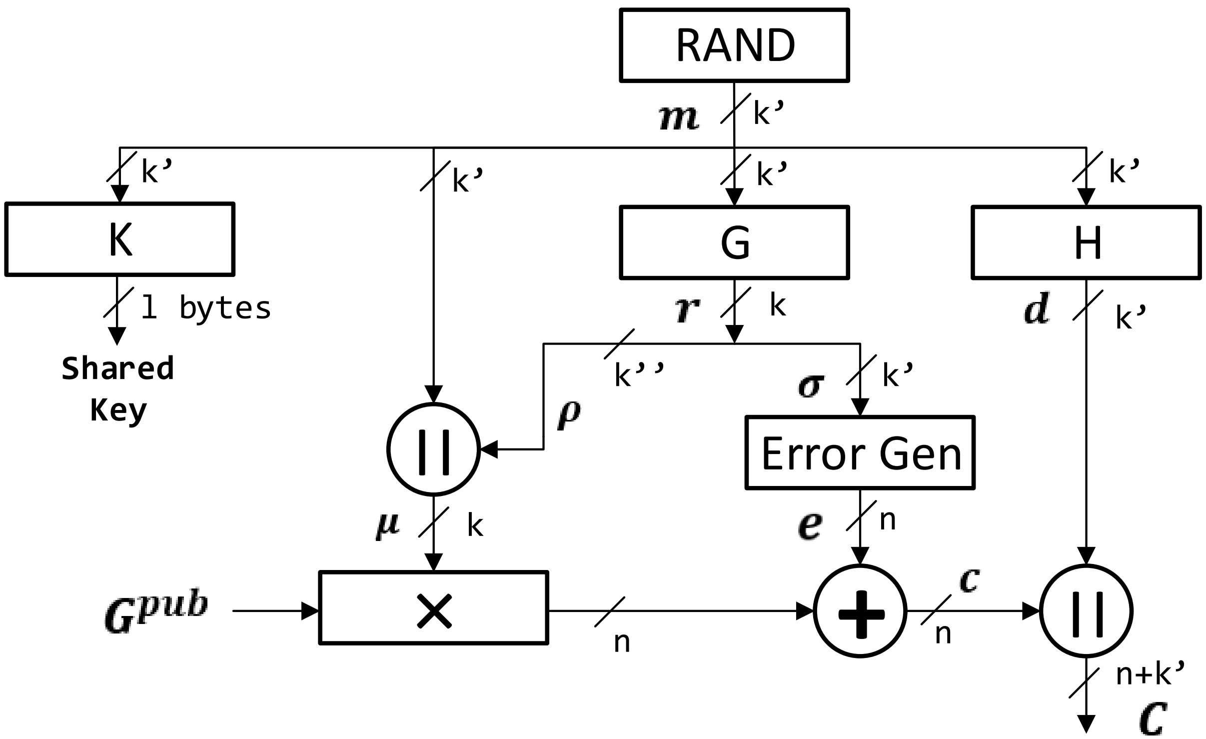
1 INV, 3 MUL, 1 CMUL, 2 ADD

Critical path:

1 CMUL + 1 ADD +1 MUL

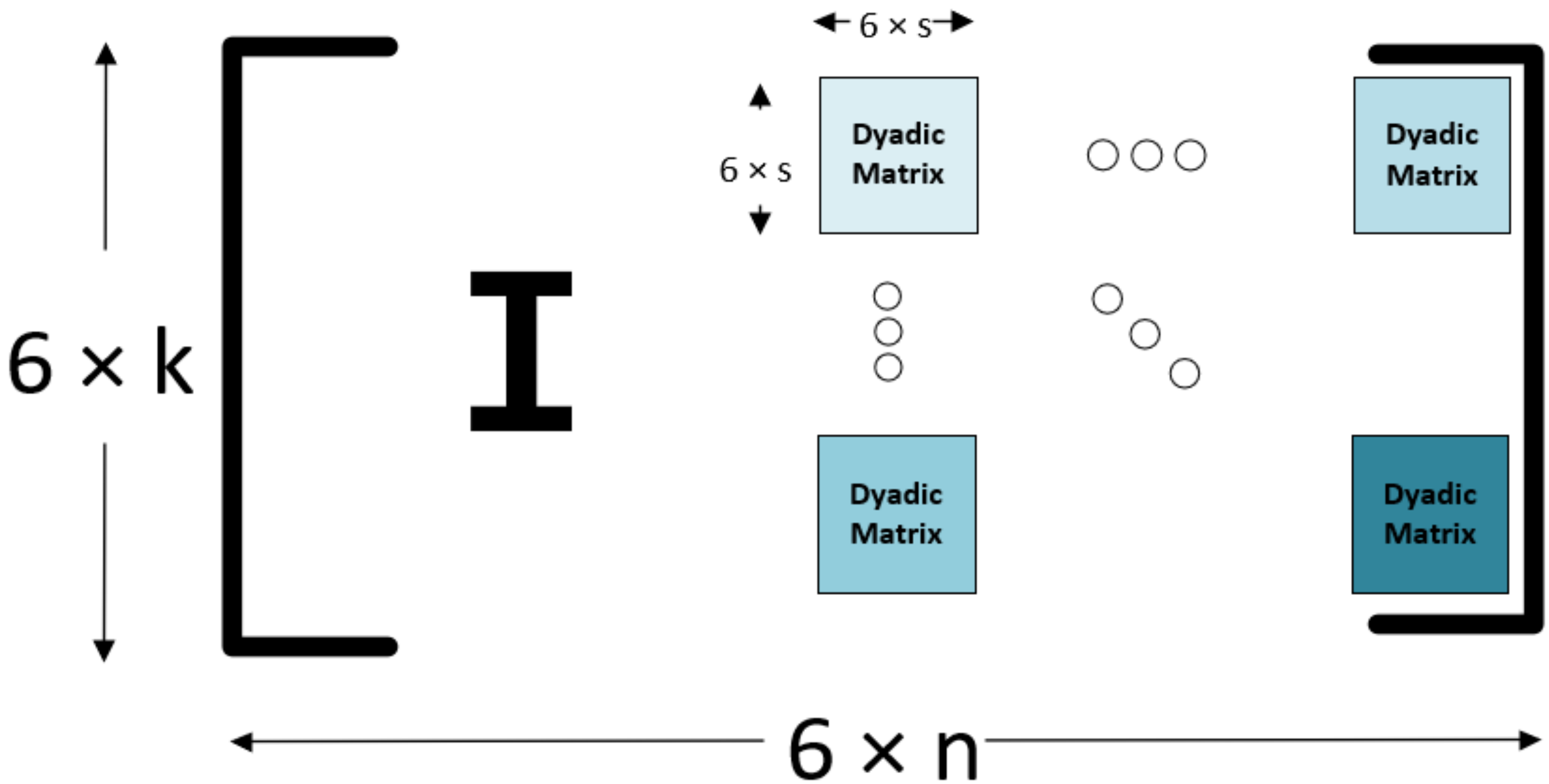
+ 1 ADD + 1 INV + 1 MUL

Encapsulation

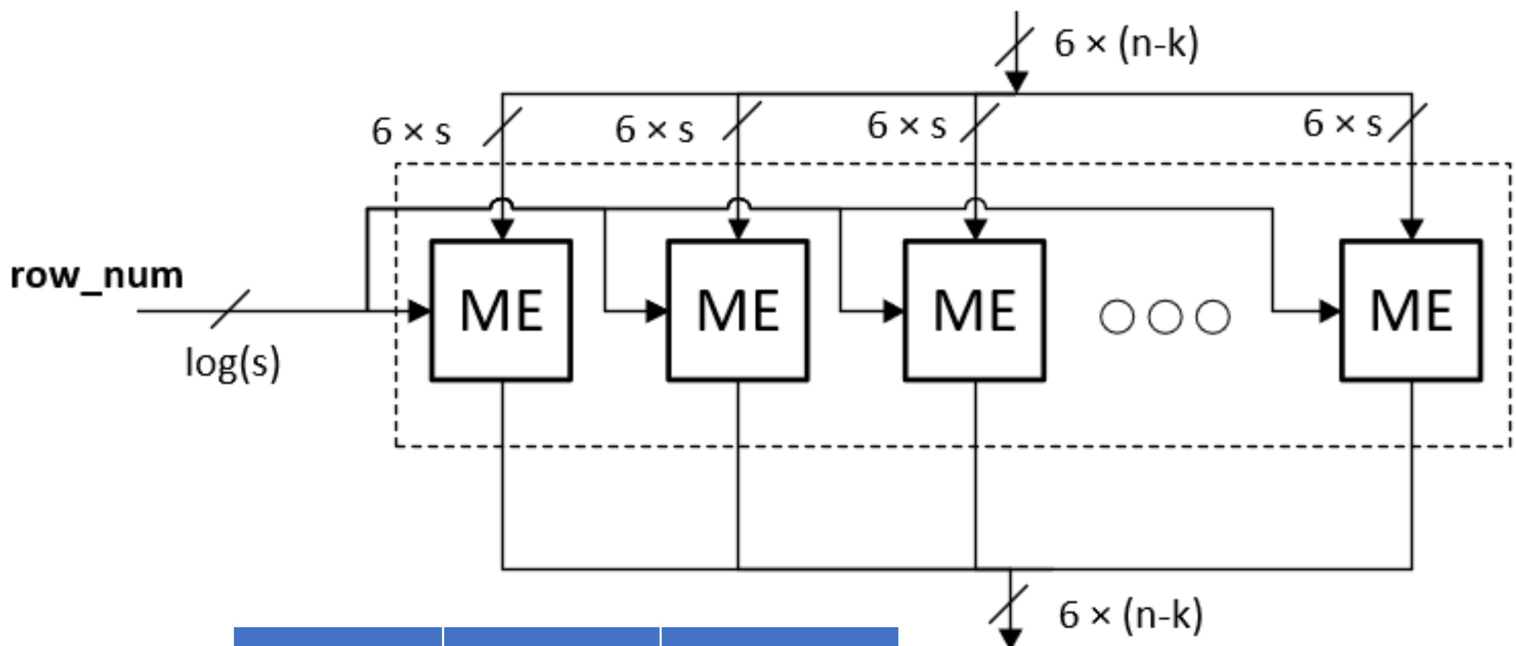


	DAGS_3	DAGS_5
n	1216	2112
k	512	704
k’	43	43
k’’	469	661
w	176	352
l	64	64
G, H	SHA-3 Extendable Output Function	
K	SHA3-512 hash function	

Generator Matrix G^{pub}



G^{pub} generator



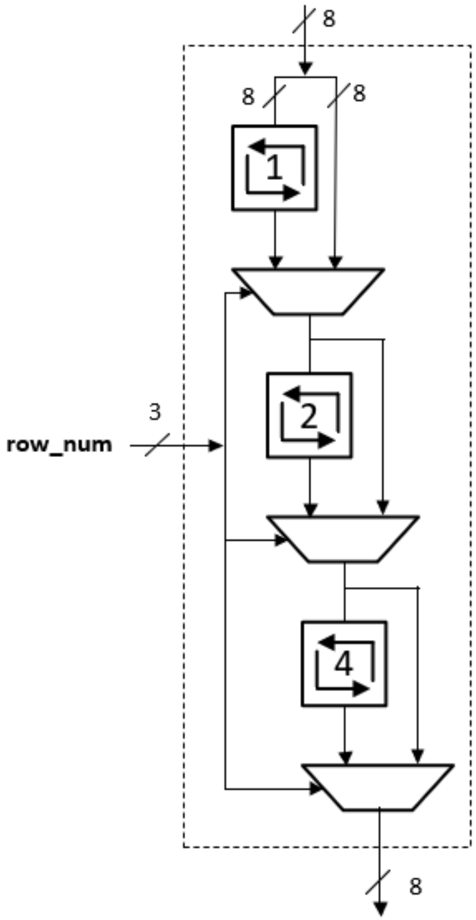
	DAGS_3	DAGS_5
n	1216	2112
s	2^5	2^6
k	512	704

ME: Matrix Expander

Dyadic Matrix Example

- $\mathrm{M}[i][j] = S[i \oplus j]$
- S = $\{S[0], S[1], S[2], S[3], S[4], S[5], S[6], S[7]\}$
- M = $\begin{bmatrix} S[0] & S[1] & S[2] & S[3] & S[4] & S[5] & S[6] & S[7] \\ S[1] & S[0] & S[3] & S[2] & S[5] & S[4] & S[7] & S[6] \\ S[2] & S[3] & S[0] & S[1] & S[6] & S[7] & S[4] & S[5] \\ S[3] & S[2] & S[1] & S[0] & S[7] & S[6] & S[5] & S[4] \\ S[4] & S[5] & S[6] & S[7] & S[0] & S[1] & S[2] & S[3] \\ S[5] & S[4] & S[7] & S[6] & S[1] & S[0] & S[3] & S[2] \\ S[6] & S[7] & S[4] & S[4] & S[2] & S[3] & S[0] & S[1] \\ S[7] & S[6] & S[5] & S[5] & S[3] & S[2] & S[1] & S[0] \end{bmatrix}$

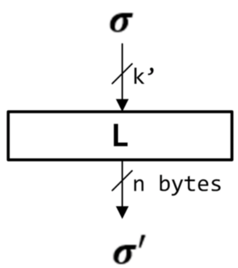
Dyadic Matrix Expander Example

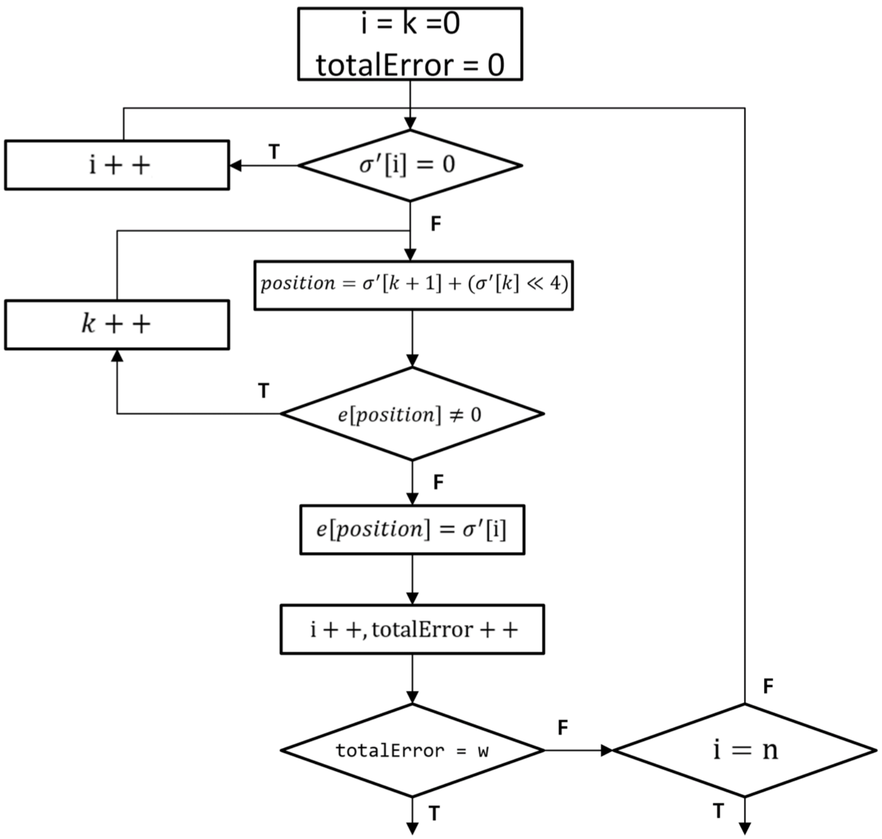


In: $\{S[0], S[1], S[2], S[3], S[4], S[5], S[6], S[7]\}$

Out: $\{S[2], S[3], S[0], S[1], S[6], S[7], S[4], S[5]\}$

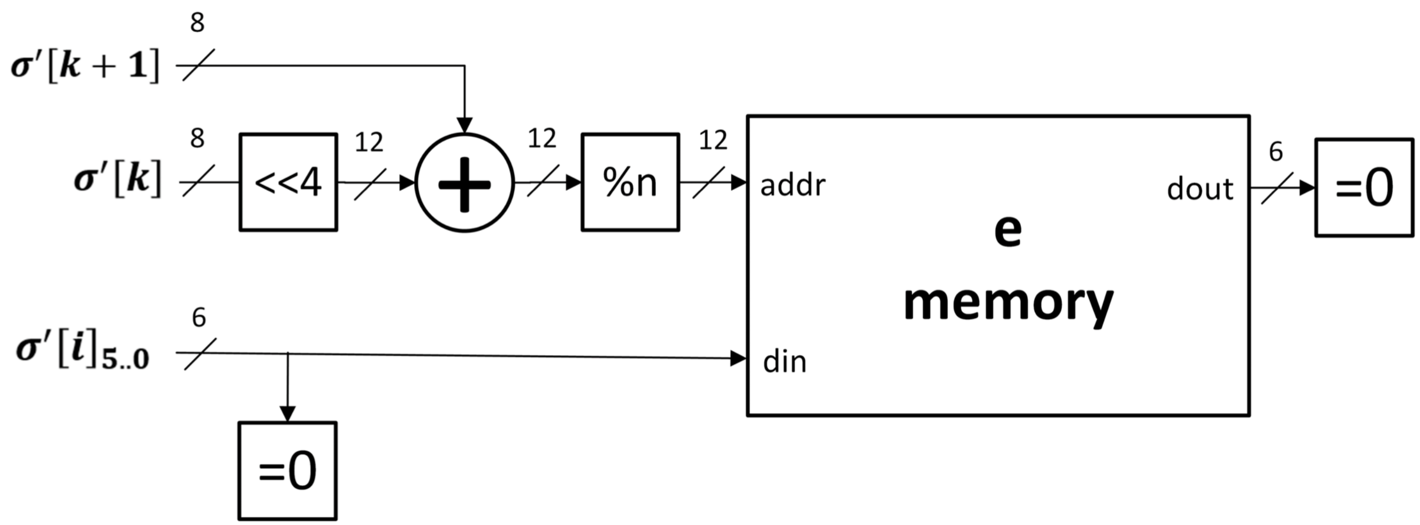
Error Generation





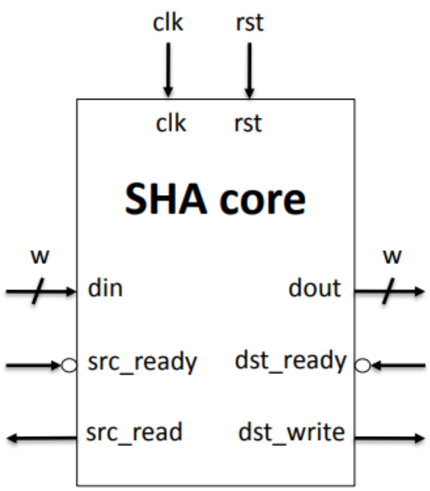
	DAGS_3	DAGS_5
n	1216	2112
k'	43	43
w	176	352
L	SHA-3 Extendable Output Function	

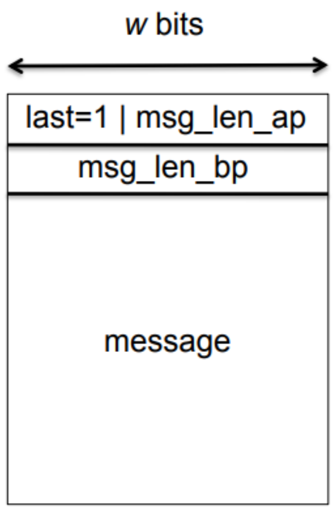
Error Generator



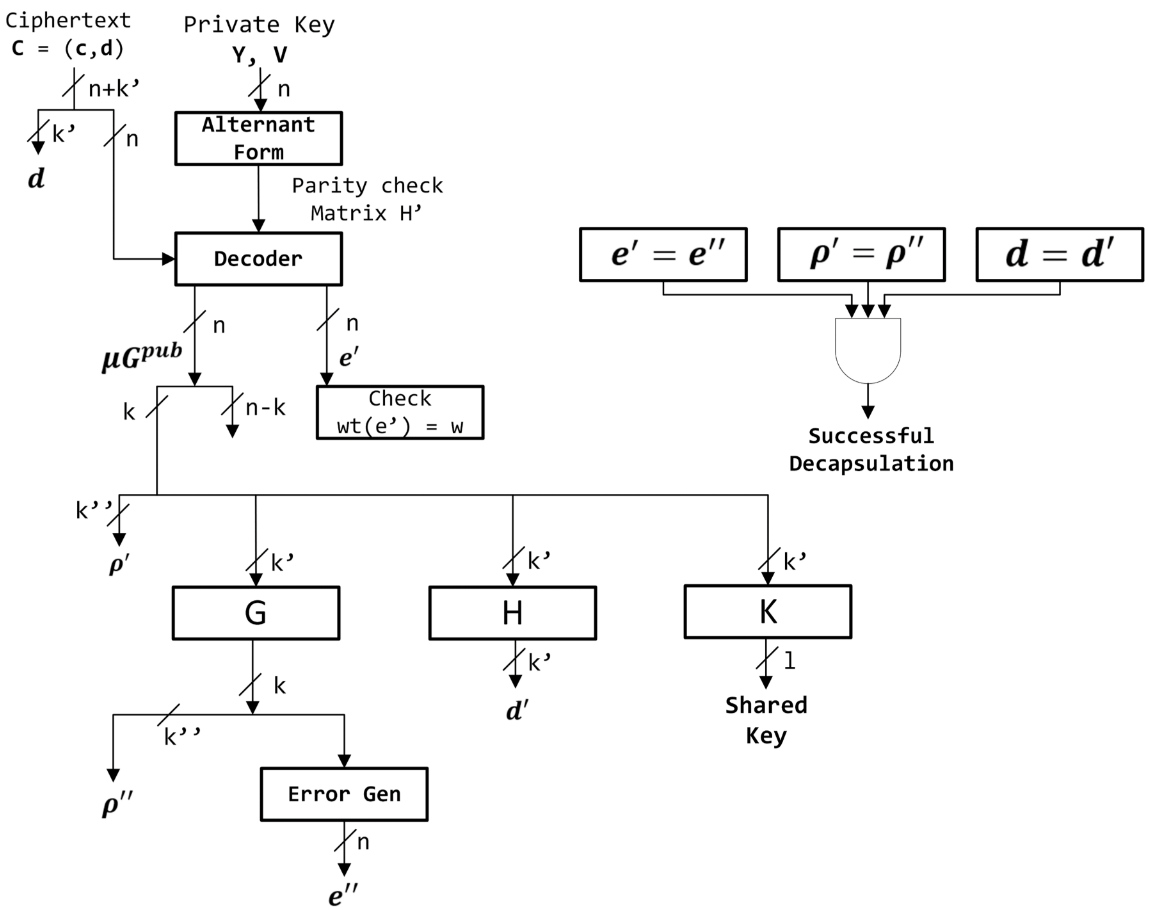
Extendable-Output Function

- SHAKE: based on Keccak hash function
- Generalization of a cryptographic hash function with arbitrary output length.
- Modified Basic Iterative with Padding Design from GMU.





Decapsulation



	DAGS_3	DAGS_5
n	1216	2112
k	512	704
k'	43	43
k''	469	661
w	176	352
l	64	64
G, H	SHA-3 Extendable Output Function	
K	SHA3-512 hash function	

Alternant Decoding

- Calculate syndrome polynomial $S(z)$ from ciphertext and 2 vectors ($Y\ and\ V$) from private key
- Apply Extended Euclidean Algorithm to solve key equation, get $\sigma(z)$, $\omega(z)$
- Evaluate $\sigma(z)$ to get error position
- Evaluate $\omega(z)$ to get error value

Private Key

- Y = $[\,y_0 \;\; y_1 \;\; y_2 \;\; \dots \; y_{n-1}]$
- V = $[v_0 \;\; v_1 \;\; v_2 \;\; \dots \;\; v_{n-1}]$
- $$H' = \begin{pmatrix} 1 & 1 & \dots & 1 \\ v_0 & v_1 & \dots & v_{n-1} \\ v_0^2 & v_1^2 & \dots & v_{n-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_0^{st-1} & v_1^{st-1} & \dots & v_{n-1}^{st-1} \end{pmatrix} \times \begin{pmatrix} y_0 & 0 & \dots & 0 \\ 0 & y_1 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & y_{n-1} \end{pmatrix}$$

	DAGS_3	DAGS_5
n	1216	2112
s	2^5	2^6
t	11	11

Syndrome Calculation

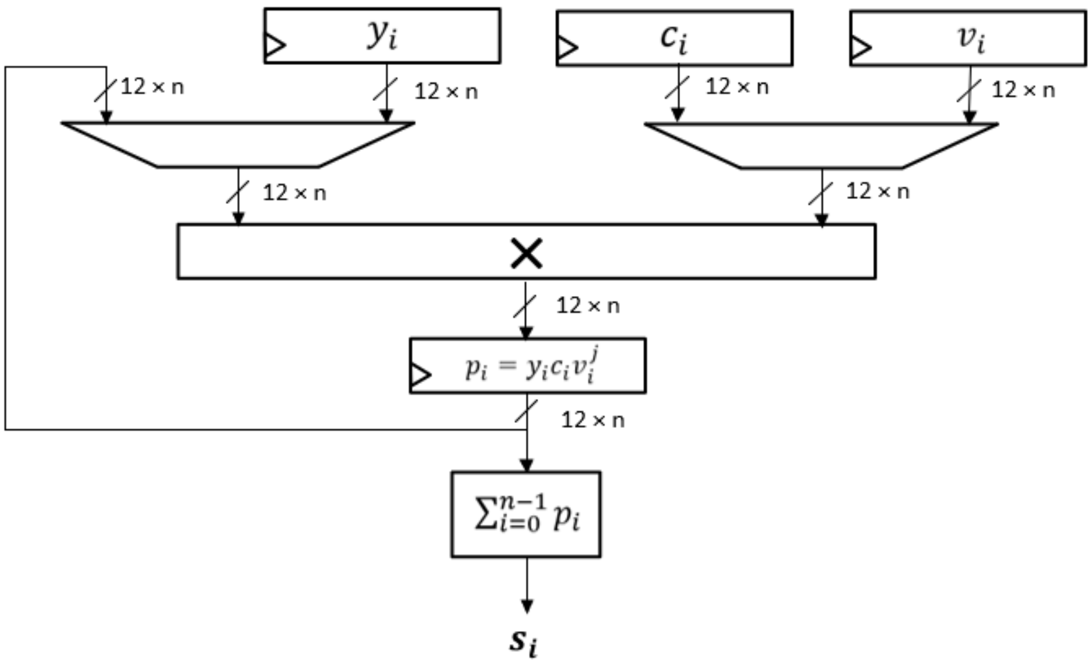
- $S = H' \times c = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ v_0 & v_1 & \cdots & v_{n-1} \\ v_0^2 & v_1^2 & \cdots & v_{n-1}^2 \\ \vdots & \vdots & \ddots & \vdots \\ v_0^{st-1} & v_1^{st-1} & \cdots & v_{n-1}^{st-1} \end{pmatrix} \times \begin{pmatrix} y_0 & 0 & \cdots & 0 \\ 0 & y_1 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & y_{n-1} \end{pmatrix} \times \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ \vdots \\ c_{n-1} \end{pmatrix}$

- $S = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{st-1} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n-1} y_i c_i \\ \sum_{i=0}^{n-1} y_i c_i \, v_i \\ \sum_{i=0}^{n-1} y_i c_i \, v_i^2 \\ \vdots \\ \sum_{i=0}^{n-1} y_i c_i \, v_i^{st-1} \end{pmatrix}$

- $S(x) = s_{st-1} z^{st-1} + s_{st-2} z^{st-2} + \ \ldots + s_2 z^2 + s_1 z + s_0$

Syndrome Calculation

- $$S = \begin{pmatrix} s_0 \\ s_1 \\ s_2 \\ \vdots \\ s_{st-1} \end{pmatrix} = \begin{pmatrix} \sum_{i=0}^{n-1} y_i c_i \\ \sum_{i=0}^{n-1} y_i c_i v_i \\ \sum_{i=0}^{n-1} y_i c_i v_i^2 \\ \vdots \\ \sum_{i=0}^{n-1} y_i c_i v_i^{st-1} \end{pmatrix}$$



	DAGS_3	DAGS_5
n	1216	2112
s	2^5	2^6
t	11	11

Solving key equation

- Find
  - $\sigma(z)$: error locator polynominal
  - $\omega(z)$: error evaluator polynomial
- Key Equation: $\mathrm{r}(z) = S(z) \times u(z)\ mod\ z^{st}$

  with $\deg(\mathrm{r}(z)) \leq \frac{st}{2}$ and $\deg(\mathrm{u}(z)) \leq \frac{st}{2} - 1$ with $\frac{st}{2} =$ w
- Calculate $\sigma(z) = \delta \times r(z)$ and $\omega'(z) = \delta \times \mathrm{u}(z)$

  with $\delta = r(0) = r_0$

	DAGS_3	DAGS_5
w	176	352
s	2^5	2^6
t	11	11

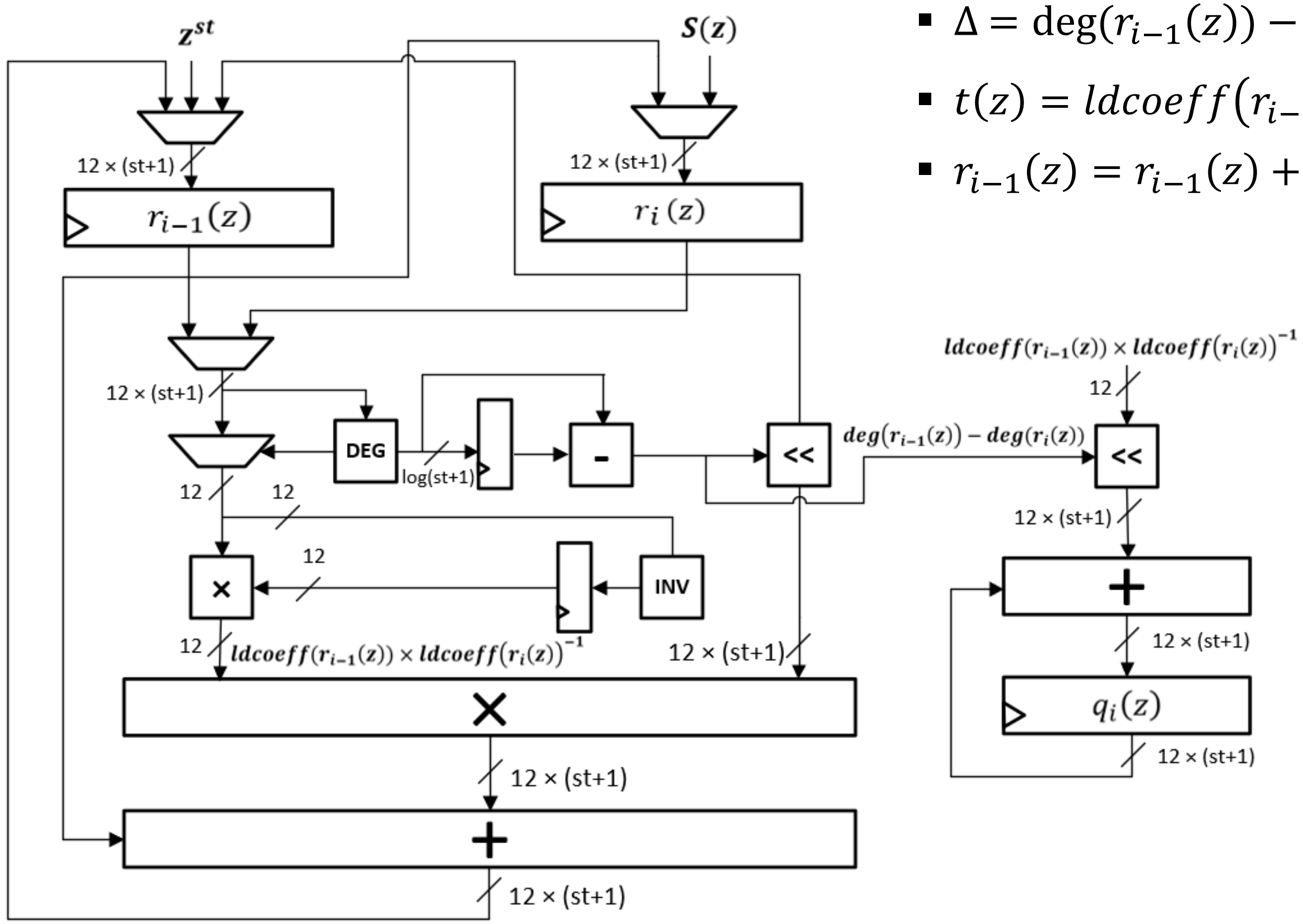
Extended Euclidean Algorithm

i	$q(z)$	$r(z)$	$u(z)$
-2		$r_{-2}(x) = z^{st}$	$u_{-2}(z) = 0$
-1	$q_1(x) = \frac{z^{st}}{S(z)}$	$r_{-1}(z) = S(z)$	$u_{-1}(z) = 1$
0	$q_0(z)$	$r_0(z)$	$u_0(z)$
…	…	…	…

- $q_i(z) = \frac{r_{i-1}(z)}{r_i(z)}$
- $r_{i+1}(z) = r_{i-1}(z) + q_i(z) r_i(z)$
- $u_{i+1}(z) = u_{i-1}(z) + q_i(z) u_i(z)$
- Termination:

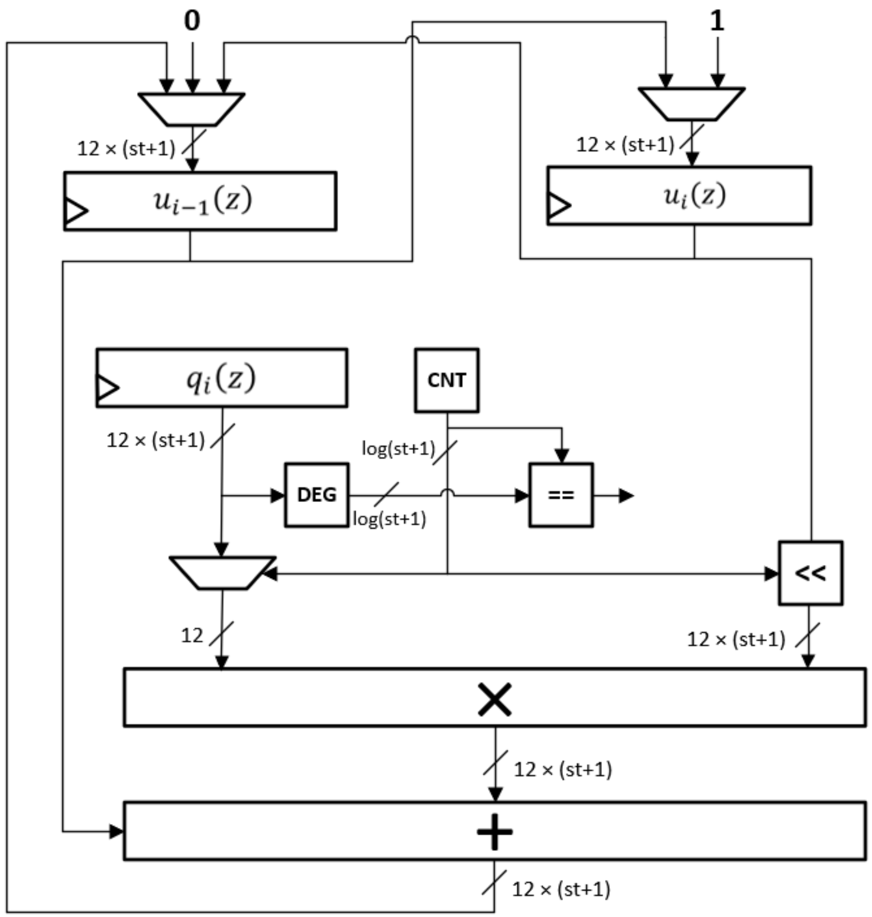
$$\deg(\mathrm{r}_{\mathrm{i}-1}(z)) \geq \frac{st}{2} \text{ and } \deg(r_i(z)) \leq \frac{st}{2} - 1 \text{ with } \frac{st}{2} = \mathrm{w}$$

Polynomial Division



- $\Delta = \deg(r_{i-1}(z)) - \deg(r_i(z))$
- $t(z) = ldcoeff\big(r_{i-1}(z)\big) \times ldcoeff\big(r_i(z)\big)^{-1} \times \Delta$
- $r_{i-1}(z) = r_{i-1}(z) + r_i(z) \times t(z)$

Polynomial Multiplication



- $u_{i-1}(z) = u_{i-1}(z) + u_i(z) \times q_{i,j} \times z^j$

Polynomial Evaluation – Root Finding

- Apply Chien search to evaluate $\sigma(x)\ and\ \omega(x)$

$$\sigma(z) = \sigma_0 + \sigma_1 z + \sigma_2 z^2 + \ \dots + \sigma_{st/2} z^{st/2}$$

$$\sigma(\alpha^i) = \sigma_0 + \sigma_1(\alpha^i) + \sigma_2(\alpha^i)^2 \ + \ \dots + \sigma_{st/2}(\alpha^i)^{st/2}$$

$$= \gamma_{i,0} + \gamma_{i,1} + \gamma_{i,2} + \ \dots + \gamma_{i,st/2}$$

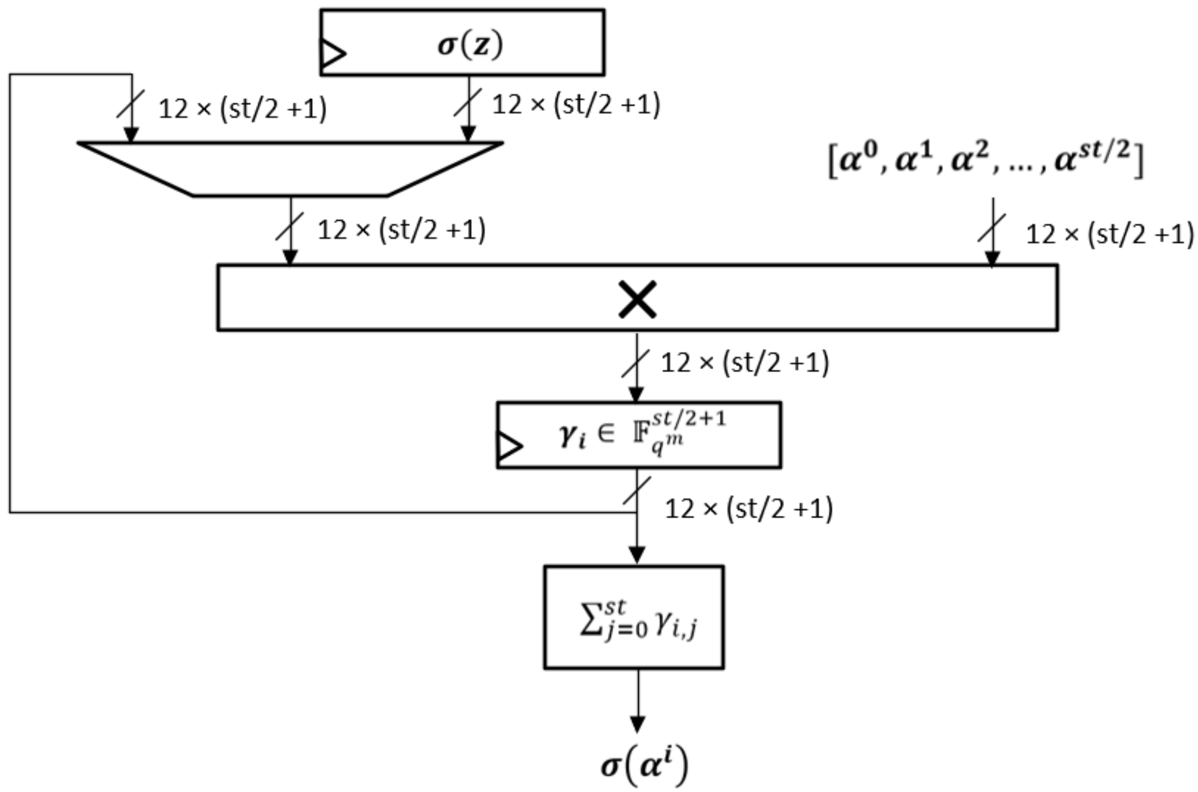
$$\sigma(\alpha^{i+1}) = \sigma_0 + \sigma_1(\alpha^{i+1}) + \sigma_2(\alpha^{i+1})^2 \ + \ \dots + \sigma_{st/2}(\alpha^{i+1})^{st/2}$$

$$= \sigma_0 + \sigma_1 \alpha^i \alpha + \sigma_2(\alpha^i)^2 \alpha^2 \ + \ \dots + \sigma_{st/2}(\alpha^i)^{st/2} \alpha^{st/2}$$

$$= \gamma_{i,0} + \gamma_{i,1}\alpha + \gamma_{i,2}\alpha^2 \ + \ \dots + \gamma_{i,st/2}\alpha^{st/2}$$

Polynomial Evaluation – Root Finding

- Chien search



Get Error Position and Value

- Error locator polynomial:

$$\sigma(z) = \prod_{i=1}^{st/2} (1 - L_i z)$$

- Evaluate error evaluator polynomial and get error value:

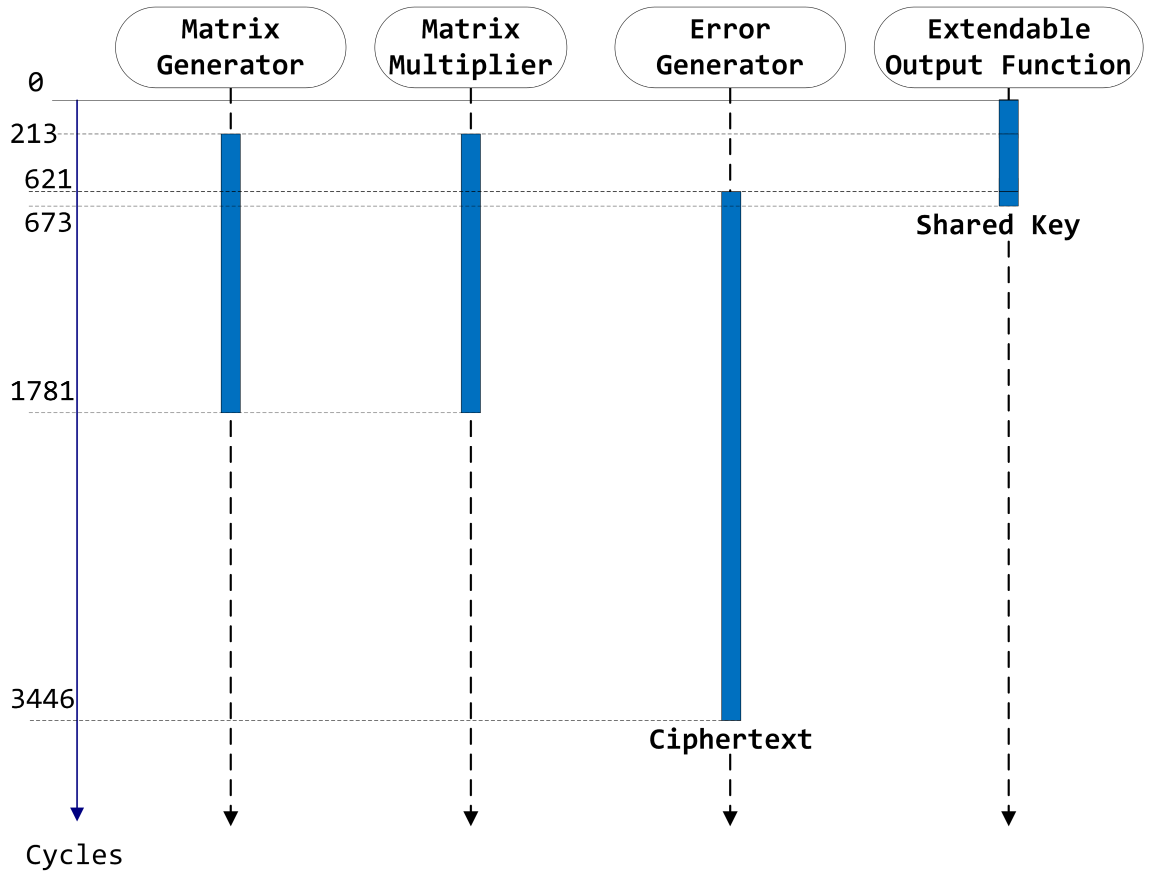
$$ErrVal_i = \frac{\omega(V_i^{-1})}{Y_i \times \prod_{j \neq i}(1 - V_j \times V_i^{-1})} \ (i \ and \ j \ in \ range \ (0 \ to \ st/2))$$

Tentative Result for DAGS_3

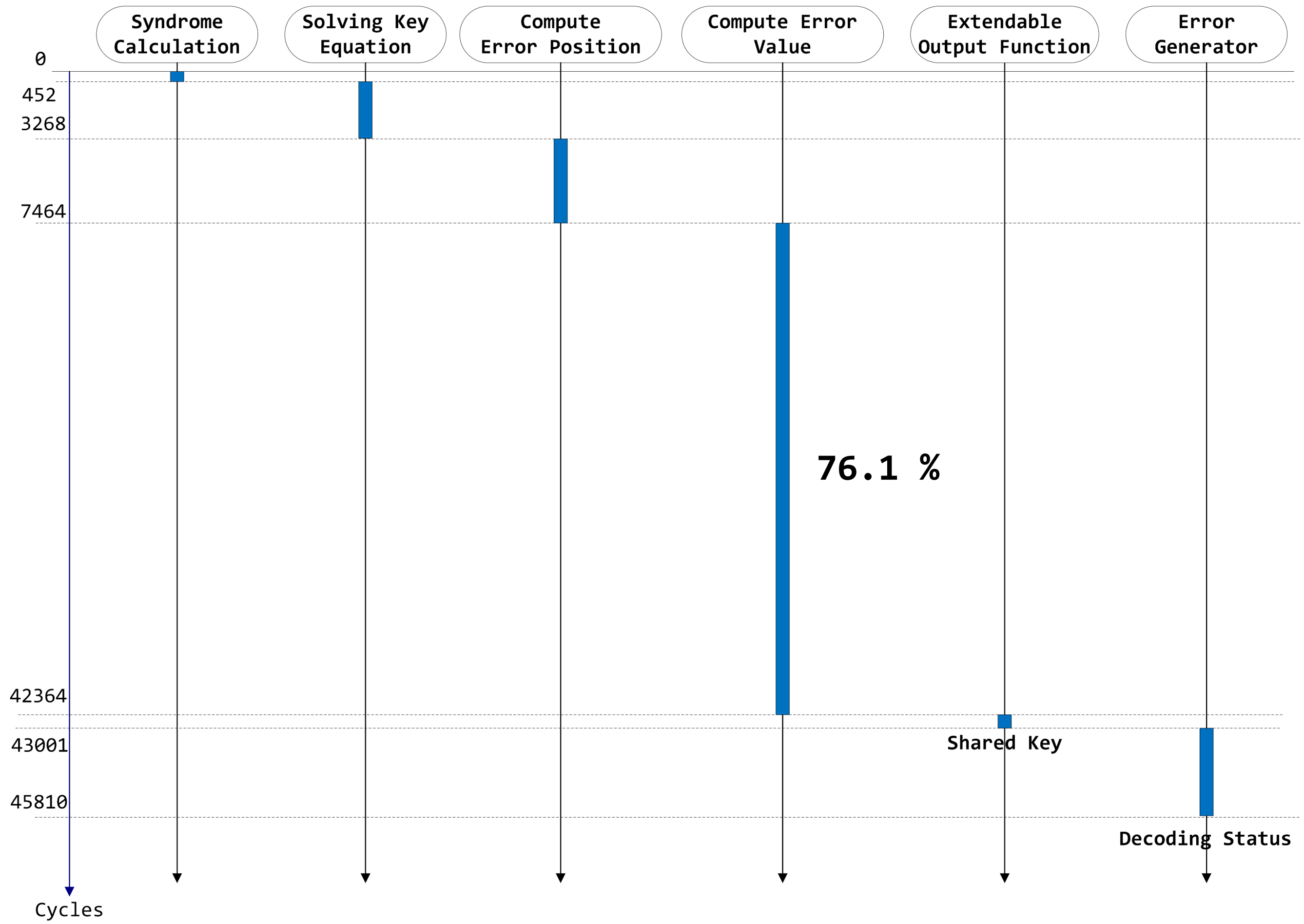
	Software	Hardware	Speed Up
Encapsulation	3,660,663 ns (**8,419,526 cycles**)	78,318 ns **(3,446 cycles)**	×**46.7**
Decapsulation	30,784,052 ns (**70,803,320 cycles**)	1,034,085 ns **(45,810 cycles)**	× **29.7**

- Software: Processor x64 Intel core i5-5300U@2.30GHz with 16GiB of RAM compiled with GCC version 6.3.020170516
- Hardware: maximum frequency 43.2 MHz.

Timing Analysis of Encapsulation



Timing Analysis of Decapsulation



Implementation Results DAGS_3

Algorithm	LUTs	FFs	Block Rams
DAGS_3	189,213 (70.3%)	99,210 (16.0%)	3

Blocks	LUTS	FFs	Block Rams
Encoder/Decoder	142,724	65,002	2
Error Gen	17,069	17,058	1
Matrix Gen	20,453	4,399	0

Conclusions

- First hardware implementation of DAGS scheme
- Fully compliant with the PQC Hardware API
- Hardware vs Software speed up of 46.7 times for encapsulation and 29.7 times for decapsulation
- Needs improvement in maximum clock frequency and area
- Needs to be constant-time
- Our VHDL code will soon be made available as open-source

Acknowledgments

- Owners, inventors, developers and submitters of DAGS

Gustavo Banegas[1], Paulo S. L. M. Barreto[2], Brice Odilon Boidje[3], Pierre-Louis Cayrel[4], Gilbert Ndollane Dione[3], Kris Gaj[7], Cheikh Thiécoumba Gueye[3], Richard Haeussler[7], Jean Belo Klamti[3], Ousmane N'diaye[3], Duc Tri Nguyen[7], Edoardo Persichetti[5], and Jefferson E. Ricardini[6]

[1] Technische Universiteit Eindhoven, The Netherlands
[2] University of Washington Tacoma, USA
[3] Université Cheikh Anta Diop, Dakar, Senegal
[4] Laboratoire Hubert Curien, Saint-Etienne, France
[5] Florida Atlantic University, USA
[6] Universidade de São Paulo, Brazil
[7] George Mason University, USA

- Dr.Patrick Baier

Thank you!

Questions?