

Complete and Improved FPGA Implementation of Classic McEliece¹

Po-Jen Chen^{1,2} Tung Chou² Sanjay Deshpande³ Norman Lahr⁴
Ruben Niederhagen⁵ Jakub Szefer³ Wen Wang³

¹GIEE, National Taiwan University, Taipei, Taiwan

²CITI, Academia Sinica, Taipei, Taiwan

³CASLAB, Department of Electrical Engineering, Yale University, New Haven, US

⁴ACE, Fraunhofer SIT, Darmstadt, Germany

⁵IMADA, University of Southern Denmark, Odense, Denmark

– CHES 2022 –

Contribution

- Specification-compliant FPGA implementation
 - ▶ a constant-time implementation
 - ▶ provide compile-time parameters (security, performance)

Contribution

- Specification-compliant FPGA implementation
 - ▶ a constant-time implementation
 - ▶ provide compile-time parameters (security, performance)
- Main building blocks
 - ▶ SEEDKEYGEN: SHAKE256, KEYGEN, and \mathbb{F}_2 systemizer
 - ▶ ENCAP: SHAKE256, FIXEDWEIGHT, and ENCODE
 - ▶ DECAP: SHAKE256, DECODE, and FIELDORDERING

SEEDEDKEYGEN – \mathbb{F}_2 Systemizer

- Motivation

- ▶ the time-consuming Gaussian elimination

parameter set	#rows	#cols	matrix size (KB)
mceliece348864	768	3488	327
mceliece460896	1248	4608	702
mceliece6688128	1664	6688	1359
mceliece6960119	1677	6960	1425
mceliece8192128	1664	8192	1664

SEEDEDKEYGEN – \mathbb{F}_2 Systemizer

- Motivation

- ▶ the time-consuming Gaussian elimination
- ▶ the high probability of failed public-key generation

parameter set	#rows	#cols	matrix size (KB)
mceliece348864	768	3488	327
mceliece460896	1248	4608	702
mceliece6688128	1664	6688	1359
mceliece6960119	1677	6960	1425
mceliece8192128	1664	8192	1664

SEEDEDKEYGEN – \mathbb{F}_2 Systemizer

- Motivation

- ▶ the time-consuming Gaussian elimination
- ▶ the high probability of failed public-key generation

parameter set	#rows	#cols	matrix size (KB)
mceliece348864	768	3488	327
mceliece460896	1248	4608	702
mceliece6688128	1664	6688	1359
mceliece6960119	1677	6960	1425
mceliece8192128	1664	8192	1664

- Three algorithmic systemizer variants

- ▶ hybrid early-abort (HEA)
- ▶ single-pass early-abort (SPEA)
- ▶ dual-pass early-abort (DPEA)

SEEDEDKEYGEN – \mathbb{F}_2 Systemizer

- Motivation

- ▶ the time-consuming Gaussian elimination
- ▶ the high probability of failed public-key generation

parameter set	#rows	#cols	matrix size (KB)
mceliece348864	768	3488	327
mceliece460896	1248	4608	702
mceliece6688128	1664	6688	1359
mceliece6960119	1677	6960	1425
mceliece8192128	1664	8192	1664

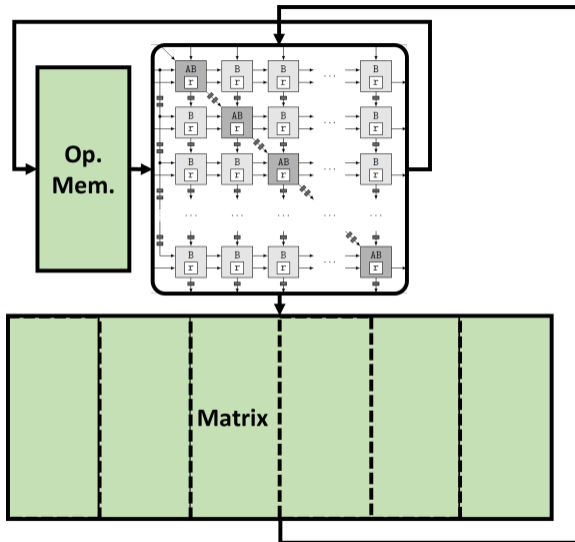
- Three algorithmic systemizer variants

- ▶ hybrid early-abort (HEA)
- ▶ single-pass early-abort (SPEA)
- ▶ dual-pass early-abort (DPEA)

- Improvement

- ▶ 2.2× to 2.6× in average runtime
- ▶ 1.7× to 2.4× in time-area efficiency

SEEDKEYGEN – Previous Systemizer Designs [WSN16]²



²a scalable processor array size, a combinational systolic line architecture

SEEDKEYGEN – Dual-pass & Single-pass approaches

- Dual-pass [SWM⁺10]
 - ▶ forward elimination → upper-triangular form
 - ▶ back-substitute elimination → systematic form

SEEDKEYGEN – Dual-pass & Single-pass approaches

- Dual-pass [SWM⁺10]
 - ▶ forward elimination → upper-triangular form
 - ▶ back-substitute elimination → systematic form
- Single-pass [WSN16]
 - ▶ forward and backwards elimination in one single pass

SEEDEDKEYGEN – Dual-pass & Single-pass approaches

- Dual-pass [SWM⁺10]
 - ▶ forward elimination → upper-triangular form
 - ▶ back-substitute elimination → systematic form
- Single-pass [WSN16]
 - ▶ forward and backwards elimination in one single pass
- Summary
 - ▶ dual-pass: less time to detect error
 - ▶ single-pass: simpler logic, efficient for single systemization

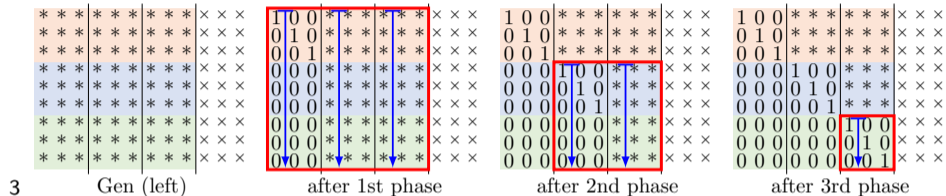
SEEDKEYGEN – Dual-pass & Single-pass approaches

- Dual-pass [SWM⁺10]
 - ▶ forward elimination → upper-triangular form
 - ▶ back-substitute elimination → systematic form
- Single-pass [WSN16]
 - ▶ forward and backwards elimination in one single pass
- Summary
 - ▶ dual-pass: less time to detect error
 - ▶ single-pass: simpler logic, efficient for single systemization
- Things to consider
 - ▶ early-abort strategies
 - ▶ overheads of generating matrices
 - ▶ dimensions of input matrices
 - ▶ choices for the column-block size

SEEDKEYGEN – Hybrid Early-abort Systemizer (HEA)

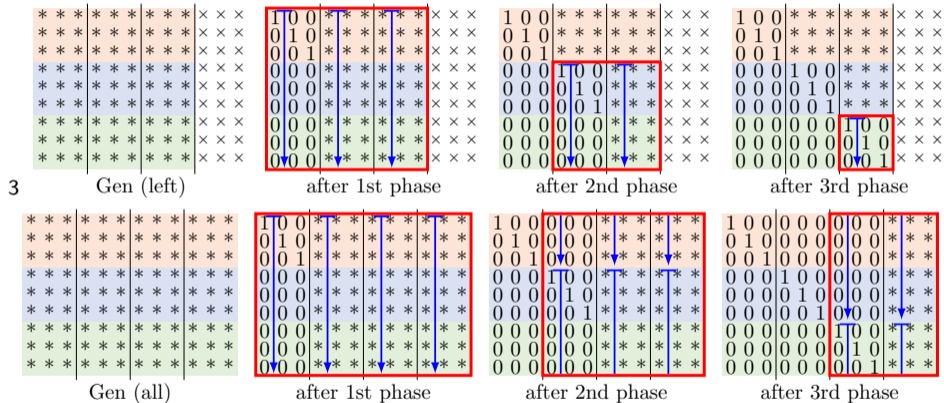
- Hybrid
 - ▶ dual-pass → upper-triangular matrix (left)
 - ▶ single-pass → systematic matrix (all)

SEEDKEYGEN – Hybrid Early-abort Systemizer (HEA)



³an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

SEEDKEYGEN – Hybrid Early-abort Systemizer (HEA)

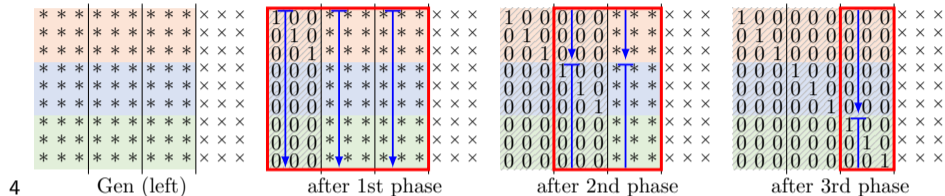


³an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

SEEDEDKEYGEN – Single-pass Early-abort Systemizer (SPEA)

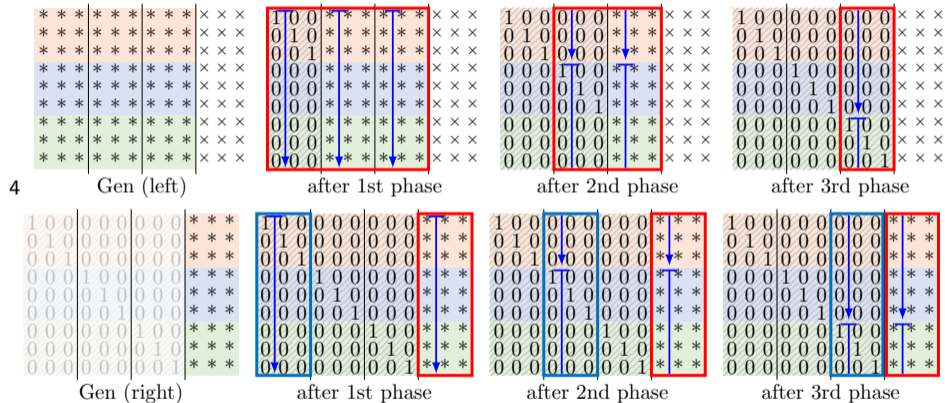
- HEA:
 - ▶ re-generates and re-eliminates the left square part of the matrix.
- SPEA:
 - ▶ stores and reuses operations from the first check.

SEEDEDKEYGEN – Single-pass Early-abort Systemizer (SPEA)



⁴an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

SEEDKEYGEN – Single-pass Early-abort Systemizer (SPEA)

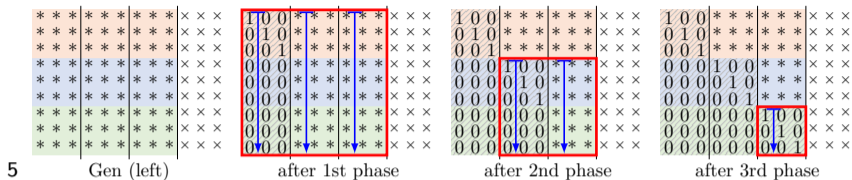


⁴an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

SEEDKEYGEN – Dual-pass Early-abort Systemizer (DPEA)

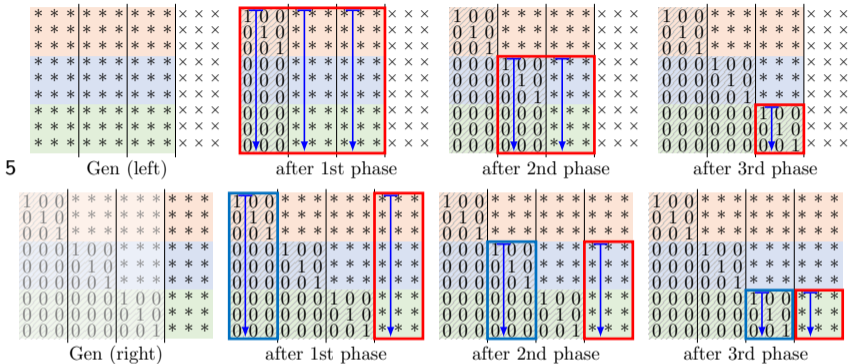
- Left square part of the matrix \hat{H}^L
 - ▶ forward elimination
- Right part of the matrix \hat{H}^R
 - ▶ forward elimination (stored operations)
 - ▶ backwards elimination

SEEDKEYGEN – Dual-pass Early-abort Systemizer (DPEA)



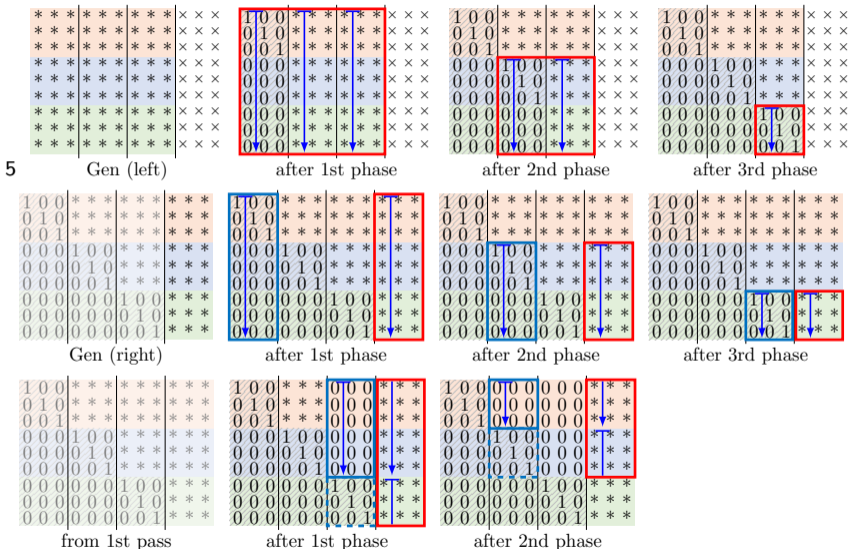
⁵an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

SEEDKEYGEN – Dual-pass Early-abort Systemizer (DPEA)



⁵an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

SEEDEDKEYGEN – Dual-pass Early-abort Systemizer (DPEA)



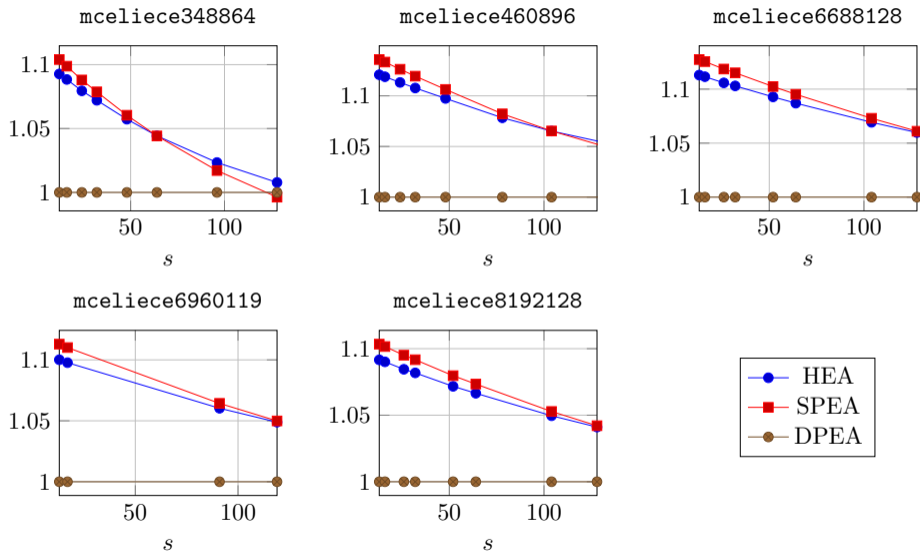
⁵an $(n - k) \times n = 9 \times 12$ matrix with a column-block size of $s = 3$

Performance of \mathbb{F}_2 Systemizer⁶

Method	s	Systemizer								Public Key Generation			
		Cycles				Resources				Cycles			
		Check (kcy.)	Finish (kcy.)	Average (kcy.)	F_{\max} (MHz)	Avg. Time (ms)	Area (LUT ^L)	Memory (FF) (BR)		Time \times Area	1st Gen. (kcy.)	2nd Gen. (kcy.)	Average (kcy.)
[WSN16]	16	7,525	—	25,580	188	136.1	877	843	97	119.4 $\times 10^3$	201.4	—	26,270
[WSN16]	32	1,961	—	6,667	191	34.91	1,757	2,528	98	61.33 $\times 10^3$	102.5	—	7,016
[WSN16]	64	535.8	—	1,822	220	8.281	5,206	8,988	132	43.11 $\times 10^3$	53.46	—	2,004
[WSN16]	128	157.6	—	535.9	215	2.492	18,080	34,223	142.5	45.06 $\times 10^3$	29.01	—	634.5
[WSN18]	16	7,534	—	25,620	166	154.3	1,107	1,056	97.5	170.8 $\times 10^3$	201.4	—	26,300
[WSN18]	32	1,963	—	6,675	172	38.81	2,381	2,825	99	92.41 $\times 10^3$	102.5	—	7,024
[WSN18]	64	536.4	—	1,824	171	10.67	7,435	9,537	133.5	79.30 $\times 10^3$	53.46	—	2,006
[WSN18]	128	157.8	—	536.4	175	3.065	26,393	35,267	149.5	80.89 $\times 10^3$	29.01	—	635.0
HEA	16	611.8	7,173	9,253	150	61.69	1,139	1,001	96.5	70.26 $\times 10^3$	44.35	201.4	9,606
HEA	32	160.0	1,800	2,344	151	15.52	2,154	2,864	97	33.44 $\times 10^3$	22.56	102.5	2,523
HEA	64	44.63	459.4	611.1	155	3.942	5,967	9,678	130	23.52 $\times 10^3$	11.66	53.46	704.2
HEA	128	14.51	120.6	169.9	181	0.9389	19,538	35,605	132	18.34 $\times 10^3$	6.216	29.01	220.1
SPEA	16	906.5	6,307	9,389	153	61.37	1,300	1,012	97	79.78 $\times 10^3$	44.35	157.1	9,697
SPEA	32	233.6	1,588	2,383	140	17.02	2,523	2,876	97.5	42.94 $\times 10^3$	22.56	79.90	2,539
SPEA	64	62.93	408.7	622.6	160	3.891	6,538	9,732	130.5	25.44 $\times 10^3$	11.66	41.80	704.1
SPEA	128	18.99	109.1	173.7	173	1.004	20,969	35,709	132.5	21.05 $\times 10^3$	6.216	22.79	217.6
DPEA	16	611.8	6,438	8,518	140	60.84	1,527	1,065	97	92.90 $\times 10^3$	44.35	157.1	8,825
DPEA	32	160.0	1,653	2,197	139	15.81	2,600	2,964	97.5	41.1 $\times 10^3$	22.56	79.90	2,354
DPEA	64	44.63	441.1	592.9	150	3.952	6,752	9,846	130.5	26.69 $\times 10^3$	11.66	41.80	674.3
DPEA	128	14.51	125.1	174.5	152	1.148	20,545	35,926	132.5	23.58 $\times 10^3$	6.216	22.79	218.4

LUT^L = Lut as logic, FF = flip-flop, BR = BRAM

Cycle Count Comparison for Large Parameter Sets



SEEDEDKEYGEN – Secret Key Format

Algorithm SEEDEDKEYGEN(δ) algorithm (using PRNG G) [ABC⁺20, Sect. 2.4.3].

- 1: Compute $E = G(\delta)$, a string of $n + \sigma_2 q + \sigma_1 t + \ell$ bits.
- 2: Define δ' as the last ℓ bits of E .
- 3: Define s as the first n bits of E .
- 4: Compute $\alpha_1, \dots, \alpha_q$ from the next $\sigma_2 q$ bits of E by the FIELDORDERING algorithm.
If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.
- 5: Compute g from the next $\sigma_1 t$ bits of E by the IRREDUCIBLE algorithm.
If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.
- 6: Define $\Gamma = (g, \alpha_1, \alpha_2, \dots, \alpha_n)$. (Note that $\alpha_{n+1}, \dots, \alpha_q$ are not used here.)
- 7: Compute $(T, c_{n-k-\mu+1}, \dots, c_{n-k}, \Gamma') \leftarrow \text{MATGEN}(\Gamma)$.
If this fails, set $\delta \leftarrow \delta'$ and restart the algorithm.
- 8: Write Γ' as $(g, \alpha'_1, \alpha'_2, \dots, \alpha'_n)$.
- 9: Output T as public key and $(\delta, c, g, \alpha, s)$ as private key, where $c = (c_{n-k-\mu+1}, \dots, c_{n-k})$ and $\alpha = (\alpha'_1, \dots, \alpha'_n, \alpha_{n+1}, \dots, \alpha_q)$.

- Default secret key format: $(\delta, c, g, \alpha, s) \Rightarrow$ Compressed secret key: (δ, c, g) [ABC⁺20, Sect. 2.2.4]
 - ▶ expand δ and compute α using FIELDORDERING
 - ▶ reduce the key size by a large factor

ENCAP – FIXEDWEIGHT

Algorithm FIXEDWEIGHT algorithm [ABC⁺20, Sect. 2.4.4].

- 1: Generate $\sigma_1\tau$ uniform random bits $b_0, b_1, \dots, b_{\sigma_1\tau-1}$.
 - 2: Define $d_j = \sum_{i=0}^{m-1} b_{\sigma_1j+i}2^i$ for each $j \in \{0, 1, \dots, \tau-1\}$.
 - 3: Define a_0, a_1, \dots, a_{t-1} as the first t entries in $d_0, d_1, \dots, d_{\tau-1}$ in the range $\{0, 1, \dots, n-1\}$. If there are fewer than t such entries, restart the algorithm.
 - 4: If a_0, a_1, \dots, a_{t-1} are not all distinct, restart the algorithm.
 - 5: Define $e = (e_0, e_1, \dots, e_{n-1}) \in \mathbb{F}_2^n$ as the weight- t vector such that $e_{a_i} = 1$ for each i .
 - 6: Return e .
-

- An error-vector generation module

- ▶ use SHAKE module (PRNG) as a seed generator
- ▶ generate extra 512 bits than the required support in case of process failures

ENCAP – ENCODE

- Input
 - ▶ an error vector e
 - ▶ a public key T
- Output
 - ▶ a ciphertext C_0

ENCAP – ENCODE

- [WSN18] stores the public key in column-major format inside the `Encode` module
 - ▶ additional efforts (large registers) for key transformation

ENCAP – ENCODE

- [WSN18] stores the public key in column-major format inside the `Encode` module
 - ▶ additional efforts (large registers) for key transformation
 - ▶ a fixed full-width implementation

DECAP – ReEncrypt Module

- After the error vector e is recovered...
 - ▶ the hamming weight of e is t
 - ▶ $Hv = He$

DECAP – ReEncrypt Module

- After the error vector e is recovered...
 - ▶ the hamming weight of e is t
 - ▶ $Hv = He$
- Scan e and packs the indexes of the nonzero bits to `error_bits_indexes`

DECAP – ReEncrypt Module

- After the error vector e is recovered...
 - ▶ the hamming weight of e is t
 - ▶ $Hv = He$
- Scan e and packs the indexes of the nonzero bits to `error_bits_indexes`
- $H^{(2)}v = H^{(2)}e$ [ABC⁺20, Sect. 2.2.4]
 - ▶ $H^{(2)}v$: a sub-module within the Decode module
 - ▶ reuse the sub-module with `error_bits_indexes` for $H^{(2)}e$

Comparison between Code-based Schemes⁷

Design	Resources										
	Logic		Memory		F	Encap	Decap	KeyGen			
	(LUT)	(DSP)	(FF)	(BR)	(MHz)	(Mcyc.)	(ms)	(Mcyc.)	(ms)	(Mcyc.)	(ms)
	mceliece348864 (our design)										
<i>LW</i>	23,890	5	45,658	138.5	112	0.13	1.1	0.17	1.5	8.88	79.2
<i>HS</i>	40,018	4	61,881	177.5	113	0.03	0.3	0.10	0.9	0.97	8.6
	BIKE - L1 [RBMG20]										
<i>LW</i>	12,868	7	5,354	17.0	121	0.20	1.2	1.62	13.3	2.67	21.9
<i>HS</i>	52,967	13	7,035	49.0	96	0.01	0.1	0.19	1.9	0.26	2.6
	HQC - L1 (HLS design) [AAB⁺20]										
<i>LW</i>	8,900	0	6,400	14.0	132	1.50	11.4	2.10	15.9	0.63	4.7
<i>HS</i>	20,000	0	16,000	12.5	148	0.09	0.6	0.19	1.3	0.04	0.3

LW = LightWeight, *HS* = HighSpeed, FF = flip-flop, F = F_{\max} , BR = BRAM

⁷NIST security level 1, Xilinx Artix 7 (xc7a200t) FPGA

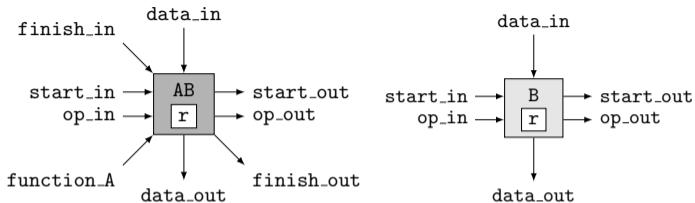
In the end...

- Errata for ePrint (ia.cr/2022/412)
 - ▶ a clarification on the choice of parameter s in Section 5.2 on page 28
 - ▶ some typos
- Released source code:
<https://caslab.csl.yale.edu/code/pqc-classic-mceliece/>

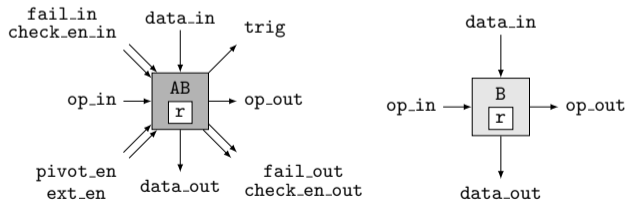
Reference

- [AAB⁺20] Carlos Aguilar Melchor, Nicolas Aragon, Slim Bettaieb, Loïc Bidoux, Olivier Blazy, Jean-Christophe Deneuville, Philippe Gaborit, Edoardo Persichetti, Gilles Zémor, and Jurjen Bos.
HQC.
Technical report, National Institute of Standards and Technology, 2020.
available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [ABC⁺20] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang.
Classic McEliece.
Technical report, National Institute of Standards and Technology, 2020.
available at <https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions>.
- [RBMG20] Jan Richter-Brockmann, Johannes Mono, and Tim Güneysu.
Folding BIKE: Scalable hardware implementation for reconfigurable devices.
Cryptology ePrint Archive, Report 2020/897, 2020.
<https://eprint.iacr.org/2020/897>.
- [SWM⁺10] Abdulhadi Shoufan, Thorsten Wink, Gregor Molter, Sorin A. Huss, and Eike Kohnert.
A novel cryptoprocessor architecture for the McEliece public-key cryptosystem.
IEEE Trans. Computers, 59(11):1533–1546, 2010.
- [WSN16] Wen Wang, Jakub Szefer, and Ruben Niederhagen.
Solving large systems of linear equations over GF(2) on FPGAs.
In Peter M. Athanas, René Cumplido, Claudia Feregrino, and Ron Sass, editors, *ReConfigurable Computing and FPGAs - International Conference, ReConFig 2016*, pages 1–7. IEEE, November 2016.
- [WSN18] Wen Wang, Jakub Szefer, and Ruben Niederhagen.
FPGA-based Niederreiter cryptosystem using binary Goppa codes.
In Tanja Lange and Rainer Steinwandt, editors, *Post-Quantum Cryptography - 9th International Conference, PQCrypto 2018*, volume 10786 of *LNCS*, pages 77–98. Springer, April 2018.

In-/Out-put Ports of Processor Elements

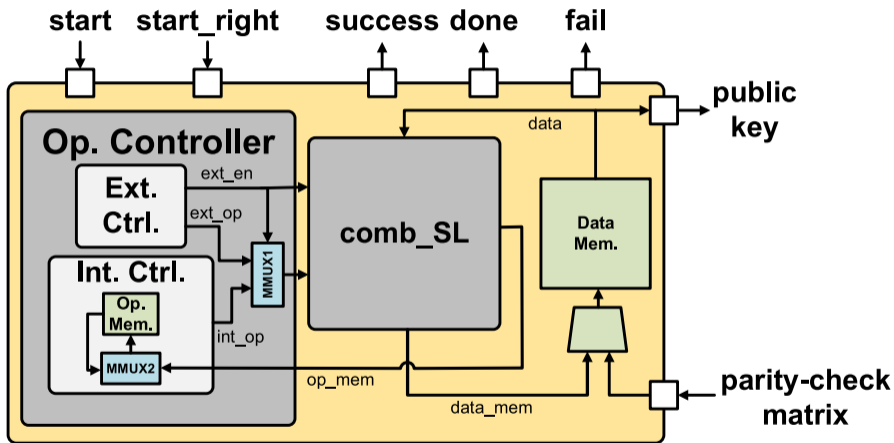


Processor elements from [WSN16]

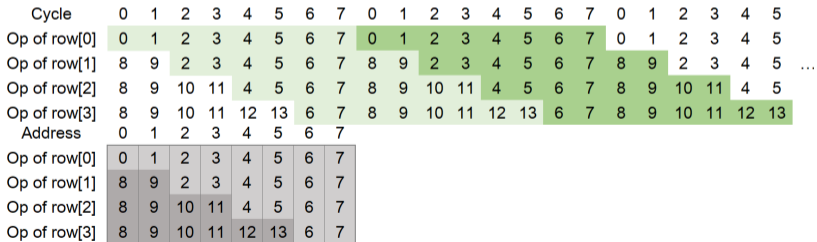
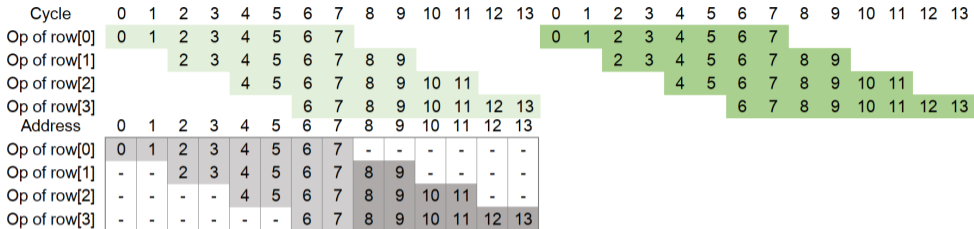


Processor elements in this work

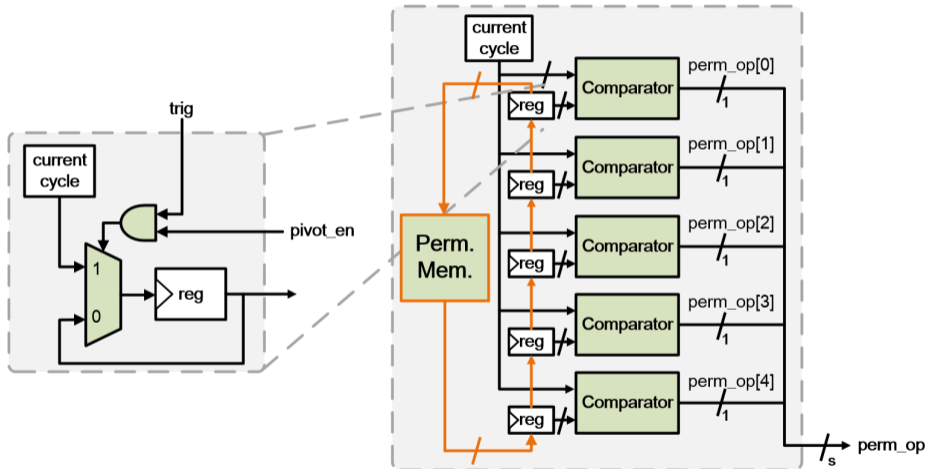
Hardware Structure of the HEA Systemizer



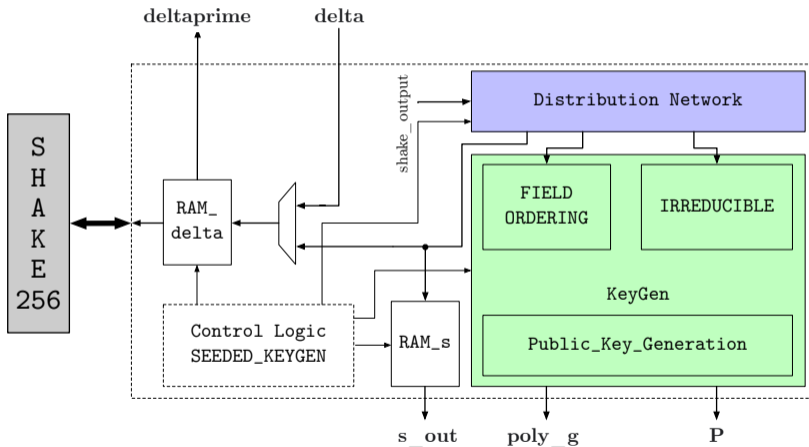
Overlapping Steps for Reducing Time/BRAM



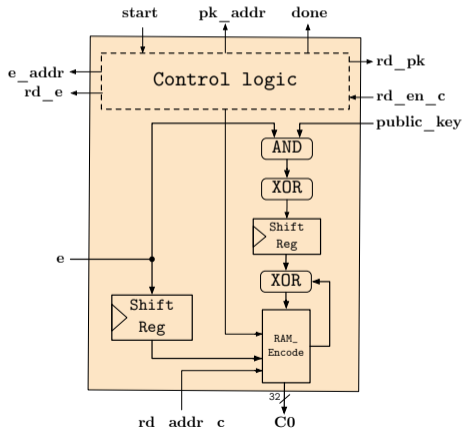
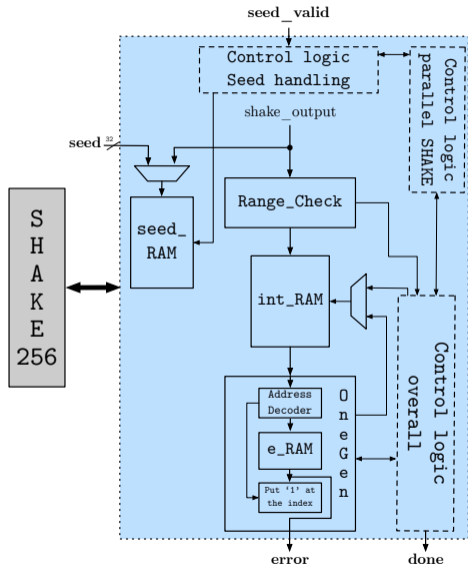
Reducing Memory Demand for the Swap Operations



Hardware Design of SeededKeyGen Module Interfaced with SHAKE256 module



Hardware designs of the FixedWeight and Encode Modules

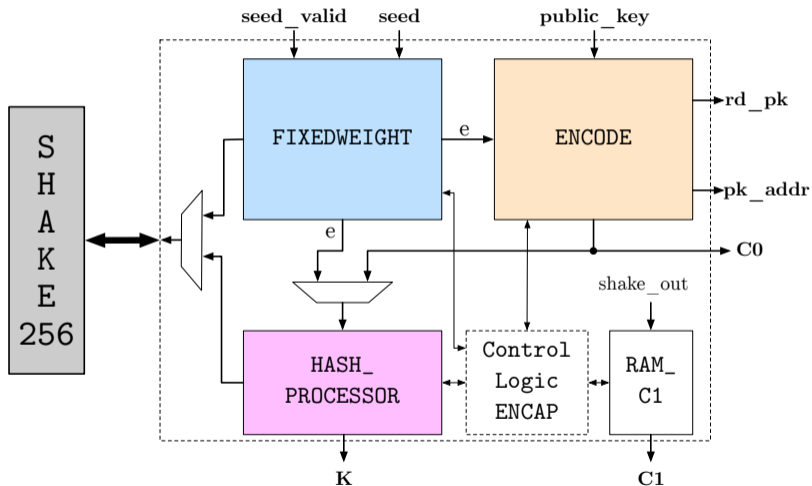


Comparison for Encode Modules

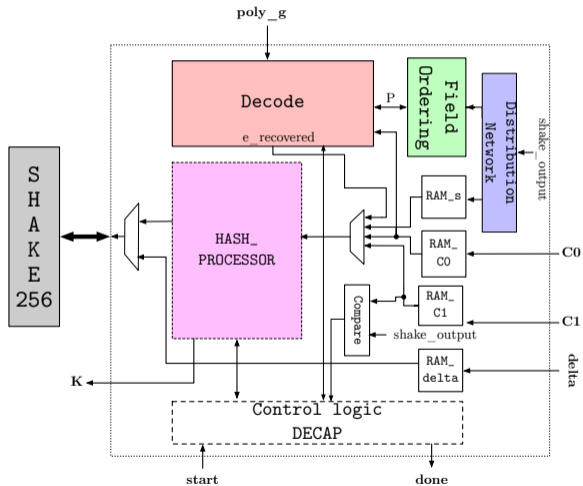
Param. Set	Resources						
	Area (LUT ^L)	Memory		Cycles	Freq. (MHz)	Time (us)	Time × Area
		(FF)	(BR)				
32-bit design (Our Design)							
mceliece348864	139	167	1	66,053	337	195.9	15.87×10^3
mceliece460896	144	173	1	132,293	329	401.9	35.37×10^3
mceliece6688128	150	175	1	262,917	337	780.3	71.01×10^3
mceliece6960119	160	196	1	264,542	319	828.6	72.08×10^3
mceliece8192128	145	176	1	341,125	335	1,019	83.55×10^3
160-bit design (Our Design)							
mceliece348864	313	321	1	13,289	197	67.46	10.55×10^3
mceliece460896	313	326	1	27,461	201	136.5	21.02×10^3
mceliece6688128	322	393	1	54,917	196	279.9	47.03×10^3
mceliece6960119	333	350	1	54,150	190	284.6	47.24×10^3
mceliece8192128	320	394	1	69,893	199	351.8	54.89×10^3
Full-width implementation [WSN18]							
mceliece348864	4,267	3,504	0	2,720	312	8.718	37.20×10^3
mceliece460896	5,866	4,624	0	3,360	330	10.18	59.73×10^3
mceliece6688128	8,365	6,705	0	5,024	322	15.60	130.5×10^3
mceliece6960119	8,519	6,977	0	5,413	310	17.46	148.8×10^3
mceliece8192128	9,869	8,209	0	6,528	321	20.33	200.7×10^3

LUT^L = LUT as logic, FF = flip-flop, BR = BRAM

Hardware Design of Encap Module Interfaced with SHAKE256 Module



Hardware Design of Decap Module Interfaced with SHAKE256 Module



Comparison for SeededKeyGen Module⁸

Param. Set	Resources								
	Area	Memory				ACC	F _{max}	Time	T×A
	(LUT ^L)	(LUT ^M)	(FF)	(BR)	(DSP)	(Mcyc.)	(MHz.)	(ms)	
Xilinx Artix 7 (xc7a200t)									
mceliece348864	25,532	290	37,185	165.0	4	1.0	142	6.8	0.17
mceliece460896	44,644	515	66,869	271.5	4	1.7	147	11.8	0.53
Xilinx Zynq UltraScale+ (xczu49dr)									
mceliece348864	25,119	344	37,245	112.5	4	1.0	186	5.2	0.13
mceliece460896	44,631	577	66,894	234.5	4	1.7	178	9.7	0.43
mceliece6688128	58,881	408	89,174	365.0	4	2.8	164	17.4	1.02
mceliece6960119	55,489	579	85,662	369.0	4	2.7	155	17.2	0.95
mceliece8192128	59,127	407	89,200	425.0	4	3.1	158	19.3	1.14

LUT^L = LUT as logic, LUT^M = LUT as memory, FF = flip-flop, BR = BRAM, ACC = average clock cycles, T×A = Time×Area

⁸Xilinx Artix 7 and Xilinx Zynq UltraScale+ FPGAs

Comparison for Encap Module⁹

Parameter Set	Resources				Cycles (kcyc.)	Freq. (MHz)	Time (μ s)	T \times A
	Area (LUT ^L)	Memory (LUT ^M) (FF) (BR)						
Encap with column-block size = 32-bits								
mceliece348864	679	76	423	4	67.98	215	316.2	214.7×10^3
mceliece460896	713	64	427	4	135.6	219	619.1	441.4×10^3
mceliece6688128	731	90	446	4	267.9	204	1,313	959.9×10^3
mceliece6960119	809	116	482	4	268.9	217	1,239	$1,002 \times 10^3$
mceliece8192128	718	90	414	4	344.8	204	1,690	$1,214 \times 10^3$
Encap with column-block size = 160-bits								
mceliece348864	1,110	76	577	7.5	15.75	174	90.52	100.5×10^3
mceliece460896	1,209	90	591	7.5	28.19	144	195.8	236.7×10^3
mceliece6688128	1,190	90	664	7.5	32.55	142	229.2	272.8×10^3
mceliece6960119	1,240	116	636	7.5	58.50	147	398.0	493.5×10^3
mceliece8192128	1,181	90	677	7.5	70.02	146	479.6	566.4×10^3

LUT^L = LUT as logic, LUT^M = LUT as memory, FF = flip-flop, BR = BRAM, T \times A = Time \times Area

⁹Xilinx Artix 7 FPGA

Comparison for Decap Module¹⁰

Parameter Set	Resources							
	Area	Memory			Cycles (kcyc.)	Freq. (MHz)	Time (ms)	T×A
	(LUT ^L)	(LUT ^M)	(FF)	(BR)				
mceliece348864	15,557	314	29,984	34.5	100.2	180	0.56	8.711×10^3
mceliece460896	24,698	540	46,509	70.5	201.7	176	1.15	28.36×10^3
mceliece6688128	25,848	330	54,527	52.5	216.0	175	1.24	31.96×10^3
mceliece6960119	29,546	546	58,126	70.5	210.9	171	1.23	36.36×10^3
mceliece8192128	26,633	330	59,048	52.5	219.1	174	1.26	33.43×10^3

LUT^L = LUT as logic, LUT^M = LUT as memory, FF = flip-flop, BR = BRAM, T×A = Time×Area

¹⁰Xilinx Artix 7 (xc7a200t) FPGA

Future Work

- Systemizers for the semi-systematic public key generation
- Side-channel attacks on Classic McEliece