Mckeycutter: A High-throughput Key Generator of Classic McEliece on Hardware

Yihong Zhu[1], Wenping Zhu[1], Chen Chen[1], Min Zhu[2],
Zhengdong Li[1], Shaojun Wei[1], Leibo Liu[1]

[1]School of Integrated Circuits, Tsinghua University.
[2]Micro Innovation Integrated Circuit Design, Wuxi, China

Outline

- Classic McEliece
- Key strategies
  - Compact GF(2) Gauss elimination hardware
  - Constant-time parallel sorters
  - Algorithm-level pipeline design
- Evaluation and Implementation
- Conclusion

Outline

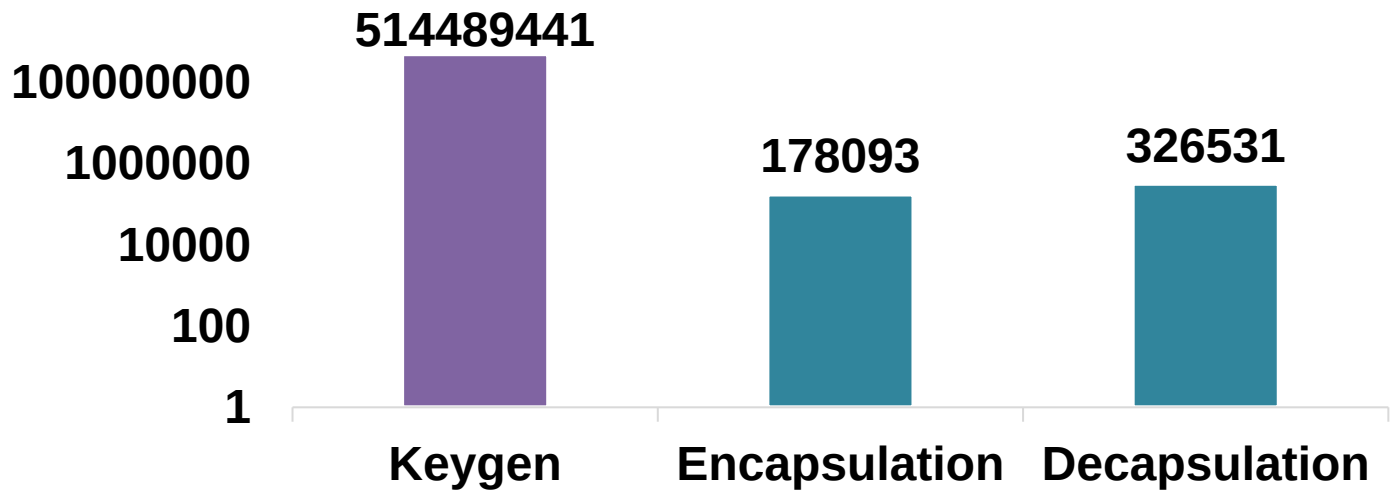
- **Classic McEliece**
- Key strategies
  - Compact GF(2) Gauss elimination hardware
  - Constant-time parallel sorters
  - Algorithm-level pipeline design
- Evaluation and Implementation
- Conclusion

Classic McEliece

" Classic McEliece has a very large public key size and fairly slow key generation. This is likely to make Classic McEliece undesirable in many common settings."

----NIST third-round report[1]

Cycles comparisons among different stages of McEliece(Intel Haswell)[2]



[1] G. Alagic, et al., "Status Report on the Third Round of the NIST Post-Quantum Cryptography Standardization Process"
[2] D. J. Berstein, et al., "Classic McEliece: conservative code-based cryptograhy," NIST 2020

Classic McEliece

Algorithm 1 Key generation algorithm in Classic McEliece [2]

Input: seed; m,t,n,$f(x)$: parameters; $(mt = m \times t)$

Output: $(g(x),(\alpha_0, \alpha_1, ..., \alpha_{n-1}))$:secret key, $\hat{T}$:public key

1: (degree-t polynomial β, 2^m 32-bit random prefixes, newseed) $\leftarrow$ PRNG(seed);

2: Generate irreducible polynomial g through calculating minimal polynomial of β;

3: Generate a random full permutation $(\alpha_0, \alpha_1, ..., \alpha_{n-1})$ through sorting random prefixes;

4: Generate the $t \times n$ matrix H calculated by g and $(\alpha_0, \alpha_1, ..., \alpha_{n-1})$;

5: Replace each entry in H with a column of bits to generate $mt \times n$ matrix T;

6: Transform matrix T into its systematic form $[I_{mt}, \hat{T}]$. If matrix T is singular, restart;

Key generation process of McEliece

Constant-time sorter: 15.2% cycles[1].

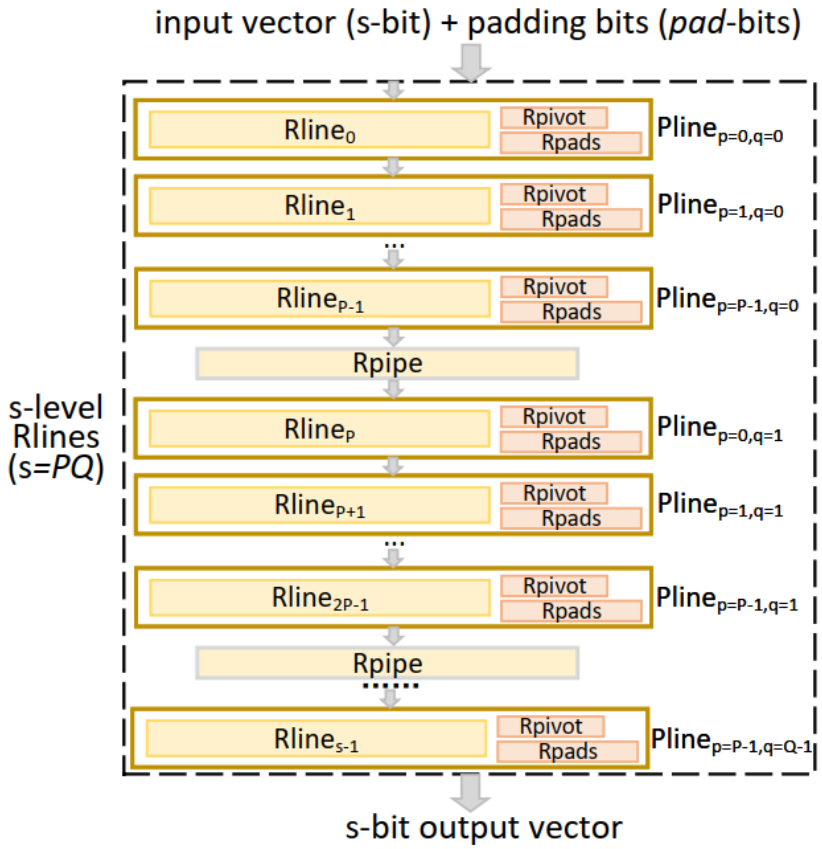
GF(2) systemizer: 76.4% cycles[1].

[1] W. Wang, et al., “FPGA-based Niederriter cryptosystem using binary goppa codes”. 2018 PQCrypto.

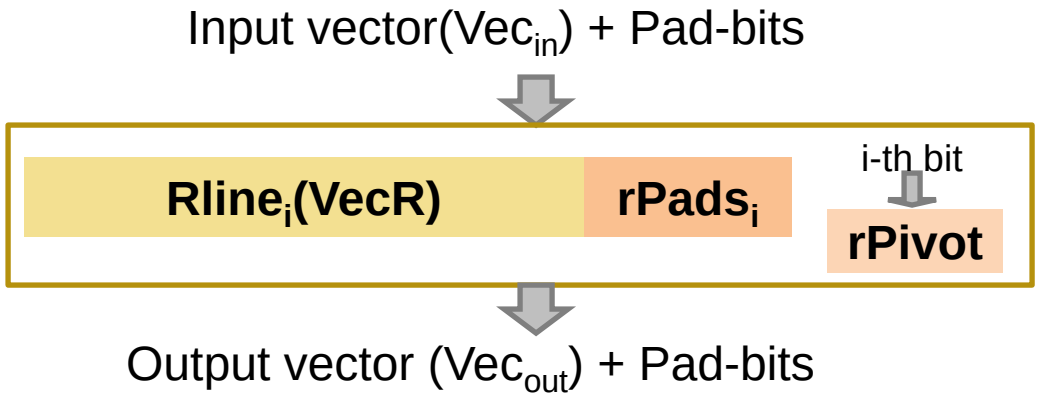
Outline

- Classic McEliece
- **Key strategies**
  - **Compact GF(2) Gauss elimination hardware**
  - Constant-time parallel sorters
  - Algorithm-level pipeline design
- Evaluation and Implementation
- Conclusion

Compact GF(2) systemizer



Line-based processing array for GF(2) Gauss elimination



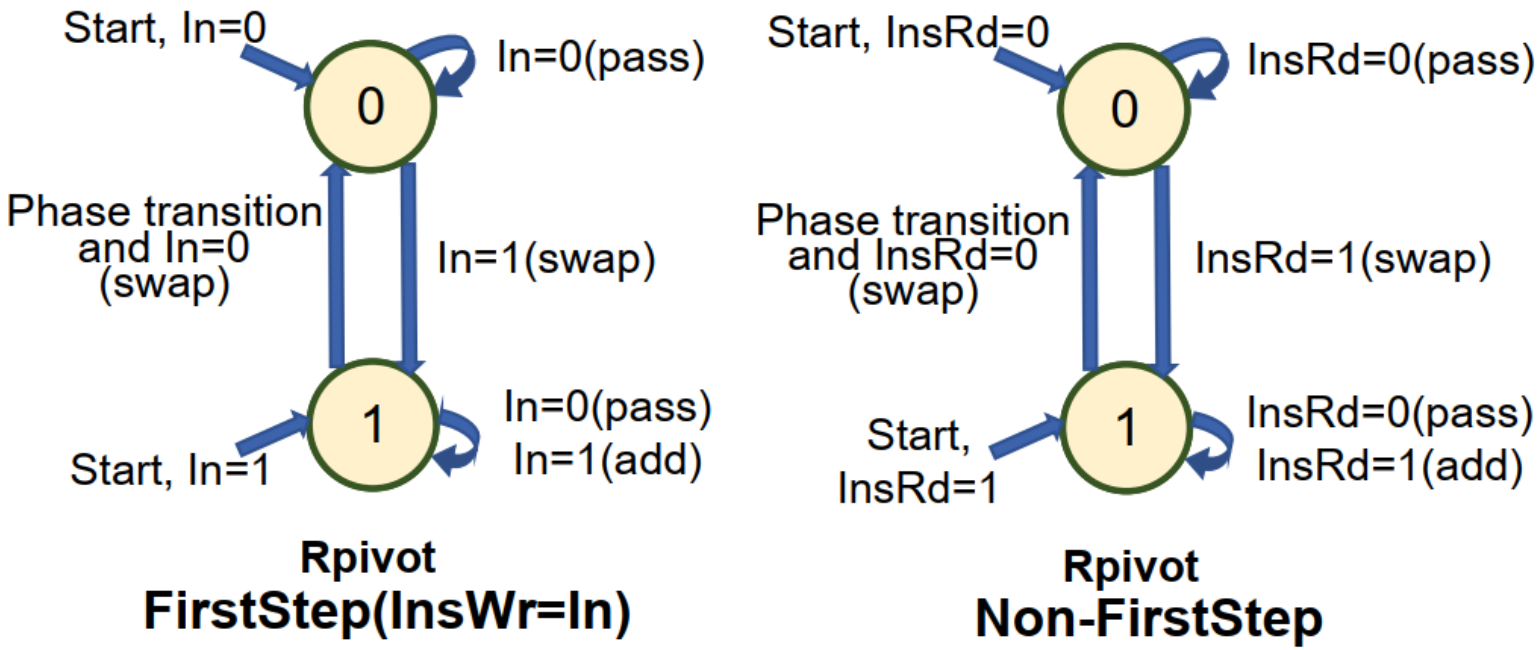
Operations:

Swap: $Vec_{out}=Vec_R$; $Vec_R=Vec_{In}$;

Pass: $Vec_{out}=Vec_{in}$; $Vec_R=Vec_R$;

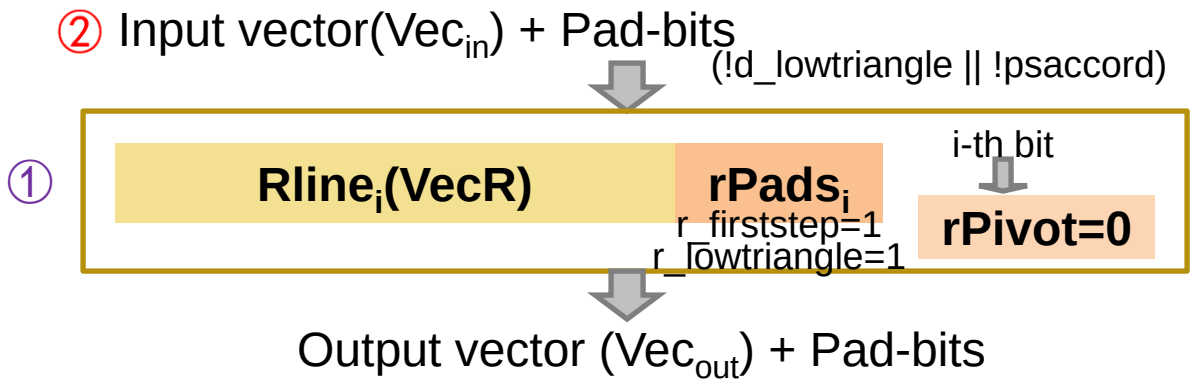
Add: $Vec_{out}=Vec_R + Vec_{in}$; $Vec_R=Vec_R$;

Compact GF(2) systemizer



Switching pattern in processing lines
(In: Diagonal bit)

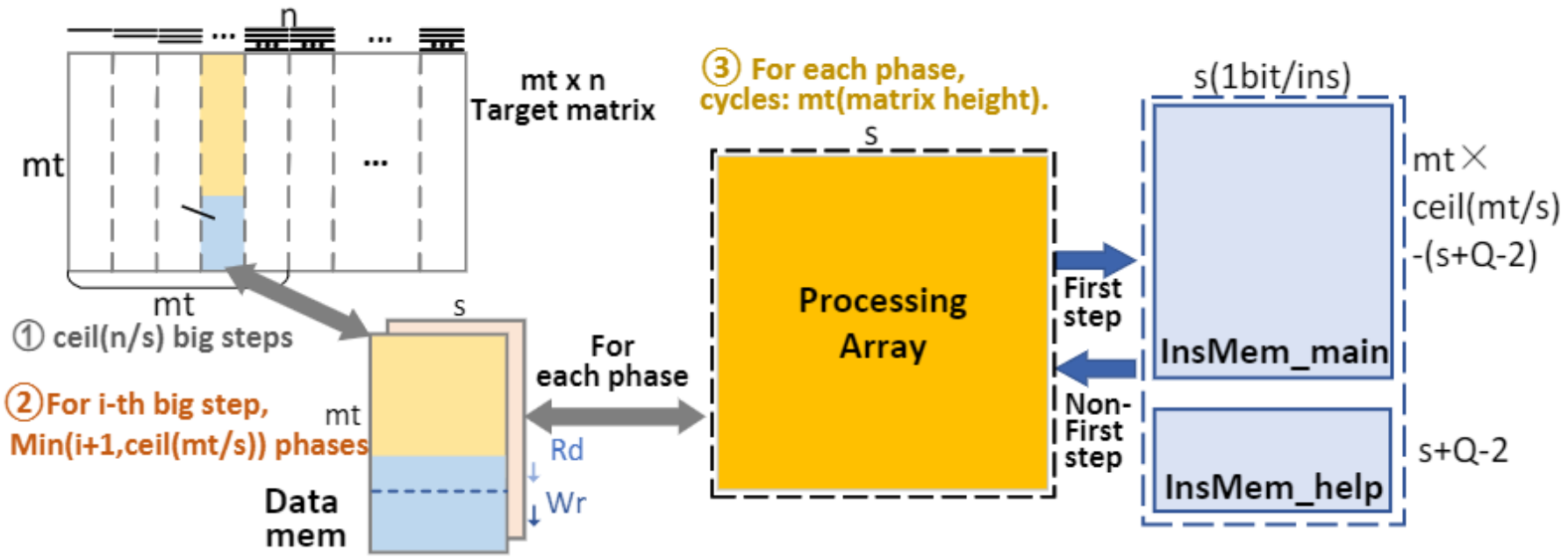
Compact GF(2) systemizer



Detect the singularity cases

The rPivot flag in Rline is not active and the data vectors belongs to the first step, while the input vector does not belong to the lower triangle part in this first step (upper triangle part or next phase).

Compact GF(2) systemizer



Efficient memory-reusage strategy

Processing array coupled with two instruction memories storing instructions from all the phases

Compact GF(2) systemizer

		Cycles in failed try	Cycles in successful try	BRAM	LUT	Flip-Flops	Freq (MHz)
mt=768 n=3488	[WSN18] [a] (s=128)	157.8k	157.8k	149.5	26393	35267	175
	[CCD+22] [a] (s=128)	14.51k	135.1k	132	19538	35606	181
	This Work [a] (s=128,Q=128)	**17.1k**	**119.8k**	**42**	**21494**	**35694**	**185**
	This Work (s=128,Q=64)	**17.0k**	**119.7k**	**28.5**	**22039**	**26931**	**284**
	This work (s=160,Q=80)	**12.6k**	**79.0k**	**29.5**	**32505**	**41279**	**278**
mt=1248 n=4608	**This Work (s=160,Q=80)**	**46.4k**	**258.1k**	**67**	**32580**	**41448**	**262**
mt=1664 n=6688	**This Work (s=160,Q=80)**	**111.9k**	**682.3k**	**103.5**	**32900**	**41445**	**259**
mt=1547 n=6960	[WSN18] [b] (s=160)	737.86k	737.86k	720(400)	40530	55675	290
	This Work (s=160,Q=80)	**87.0k**	**615.9k**	**100.5**	**32630**	**41314**	**264**
mt=1664 n=8192	**This Work (s=160,Q=80)**	**111.9k**	**865.8k**	**103.5**	**32892**	**41443**	**259**

[a] Implementations are based on Xilinx Artix 7 FPGA.
[b] The implementation results are based on Altera Straix V FPGA. The memory consumed is the same as that of 400 BRAMs.

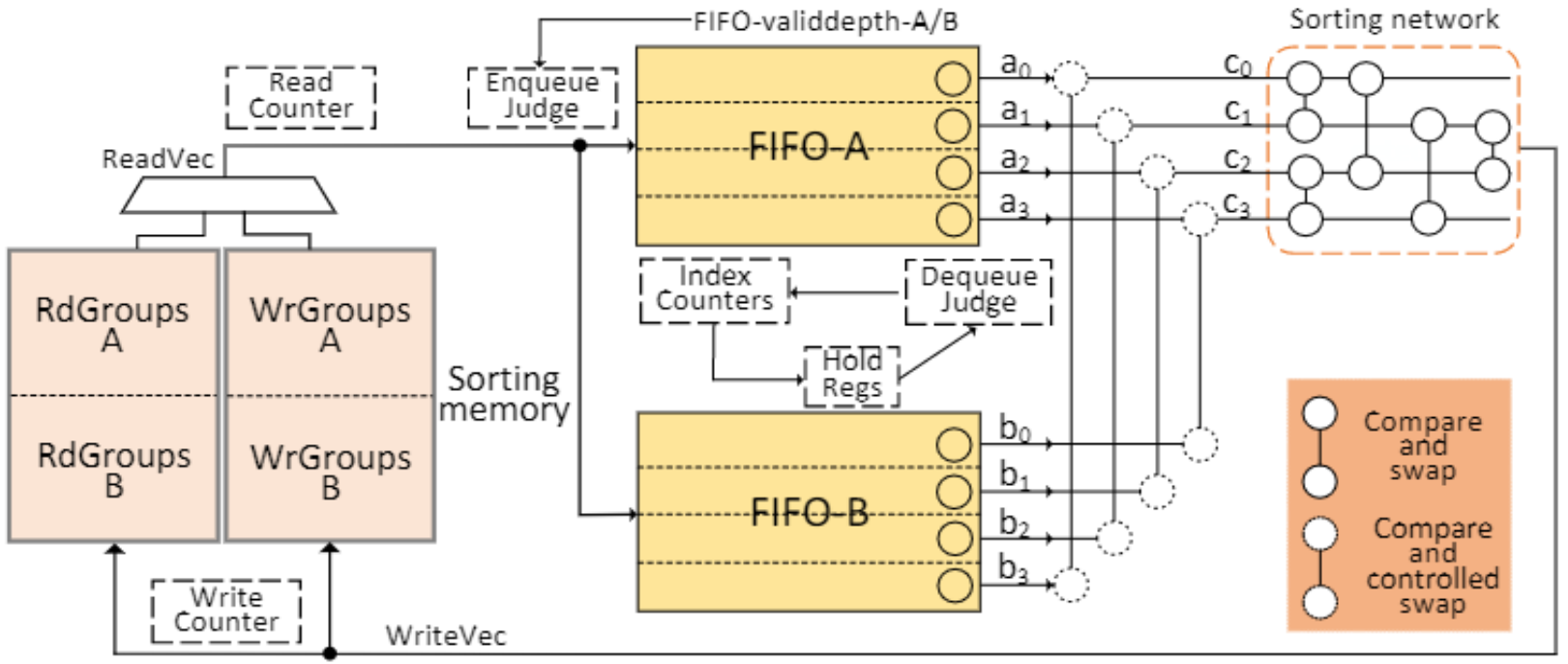
Memory comparison: quarter to one third

Cycles comparison: fewer cycles

Outline

- Classic McEliece
- **Key strategies**
  - Compact GF(2) Gauss elimination hardware
  - **Constant-time parallel sorters**
  - Algorithm-level pipeline design
- Evaluation and Implementation
- Conclusion

Constant-time parallel sorters



Double banks of data mem

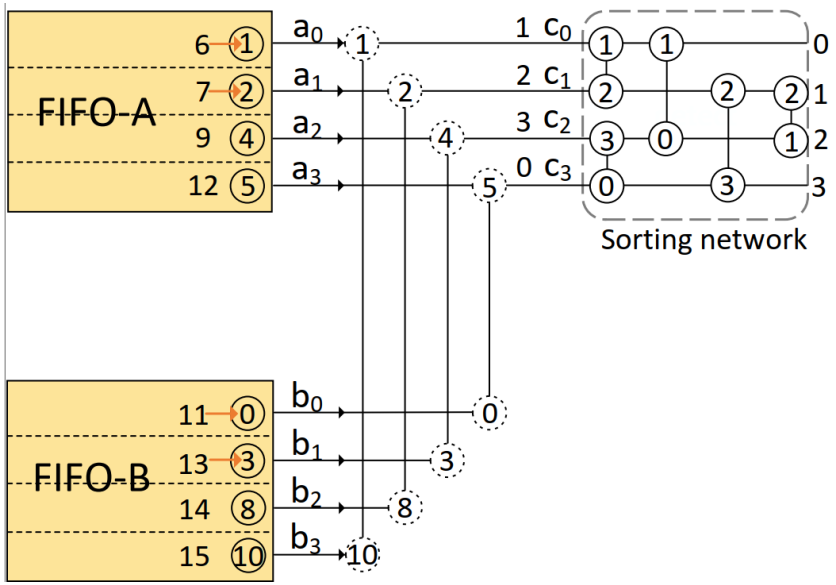
Read FIFOs

Selection network

Output sorting network

Architecture of constant-time parallel sorters

Constant-time parallel sorters



An example. (A-array: 1,2,4,5,6,7,9,12; B-array: 0,3,8,10,11,13,14,15)

$$a_i \le a_{i+1 \bmod 4} \le a_{i+2 \bmod 4} \le a_{i+3 \bmod 4};$$
$$b_j \le b_{j+1 \bmod 4} \le b_{j+2 \bmod 4} \le b_{j+3 \bmod 4};$$
$$i + j = 0 \bmod 4;$$

Proved relations among the top-elements of FIFO-A/FIFO-B (a_0,a_1,a_2,a_3: FIFO-A; b_0,b_1,b_2,b_3: FIFO-B)

Correctness:

It is guaranteed that in each cycle output vector c_i are the smallest 4 elements and the remaining array remains sorted after shifting.

Constant-time property:

It is guaranteed that there is no overflow or excessive consumption.

Constant-time parallel sorters

	LUTs	Flip-Flops	BRAM	Cycles	Freq
[5]	583	411	20	147505	250M
[8]	2533	1589	33	26646	250M
This work	**2510**	**3887**	**20**	**24598**	**300M**

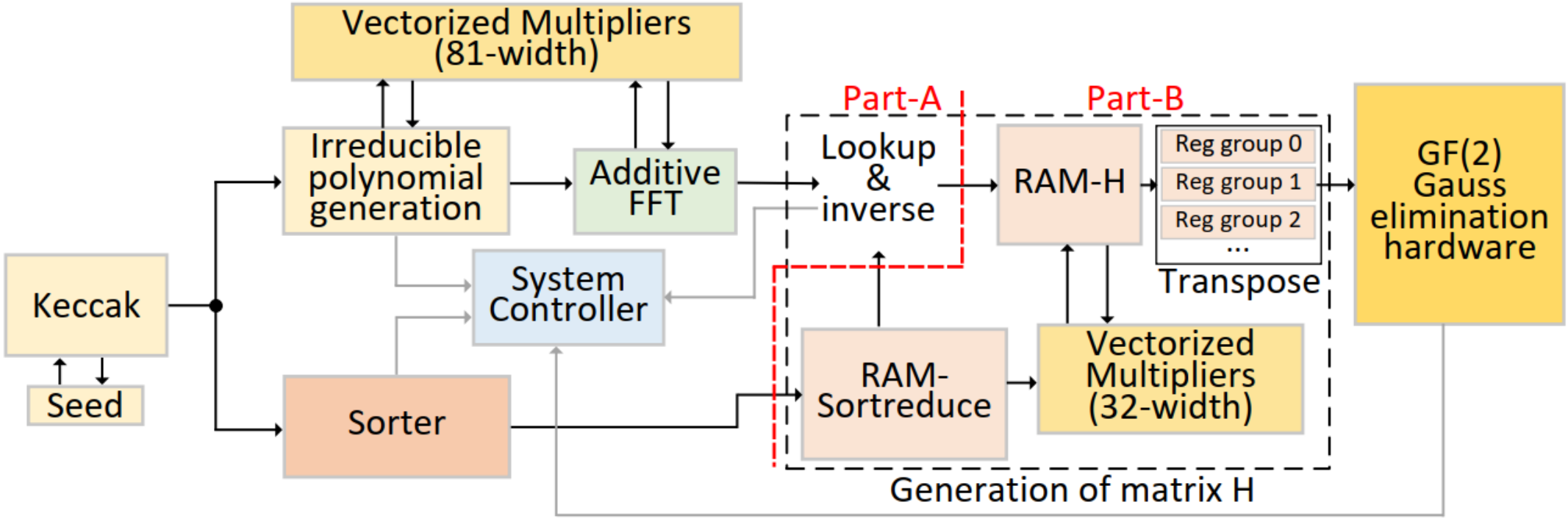
Comparison of the hardware sorter's resource usage
(sorting 8192 elements; random prefixes:32-bit; satellite: 13-bit)

Additional storage space is reduced to 80 coefficients (size of FIFOs) from several thousand.

Outline

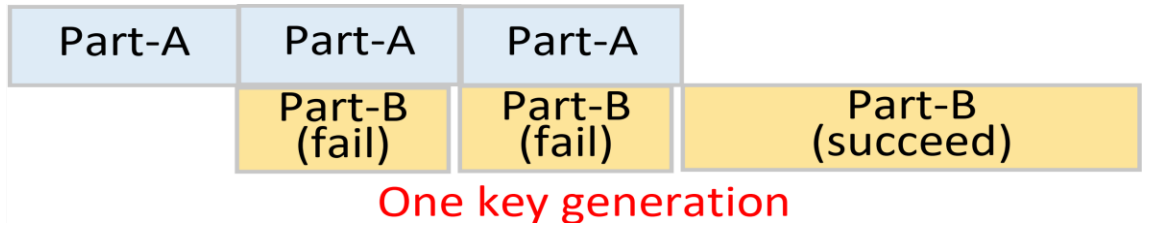
- Classic McEliece
- **Key strategies**
  - Compact GF(2) Gauss elimination hardware
  - Constant-time parallel sorters
  - **Algorithm-level pipeline design**
- Evaluation and Implementation
- Conclusion

Overall architecture

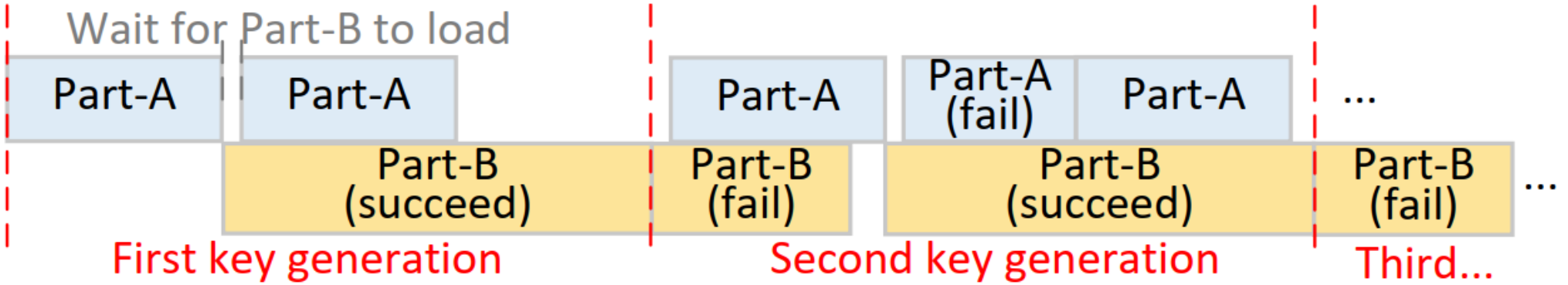


The overall architecture of Mckeycutter.
(The design is divided as Part-A and Part-B)

Algorithm-level pipeline design



Algorithm-level pipeline design
(Single mode for only one key generation job)



Algorithm-level pipeline design
(Batching mode for multiple key generation jobs)

Outline

- Classic McEliece
- Key strategies
  - Compact GF(2) Gauss elimination hardware
  - Constant-time parallel sorters
  - Algorithm-level pipeline design
- **Evaluation and Implementation**
- Conclusion

Implementation of Mckeycutter

		Cycles in a successful try	Average cycles	LUTs	Flip-Flops	BRAMs	Freq (MHz)	Time (ms)
m=12; t=64; n=3488	[CCD+22] [a]	Not given	1.0 M	25463	37245	112.5 (1.7×)	186	5.2 (8.3×)
	This Work [c]	**117.7k**	**173.4k**	**69428**	**69636**	**64.5 (1.0)**	**278**	**0.6 (1.0)**
m=13; t=96; n=4608	[CCD+22] [a]	Not given	1.7 M	45208	66894	234.5 (2.0×)	178	9.7 (6.0×)
	This Work [c]	**339.4k**	**425.4k**	**73765**	**72273**	**116 (1.0)**	**262**	**1.6 (1.0)**
m=13; t=128; n=6688	[CCD+22] [a]	Not given	2.8 M	59289	89174	365 (2.3×)	164	17.4 (4.2×)
	This Work [c]	**805.9k**	**1077.1k**	**74271**	**72418**	**158.5 (1.0)**	**259**	**4.2 (1.0)**
m=13; t=119; n=6960	[WSN17] [b]	Not given	3.1 M	112845	Not given	375 (2.7×)	225	13.8 (3.7×)
	[CCD+22] [a]	Not given	2.7 M	56068	85662	369 (2.7×)	155	17.2 (4.6×)
	This Work [c]	**714.9k**	**958.5k**	**73821**	**73512**	**137 (1.0)**	**258**	**3.7 (1.0)**
m=13; t=128; n=8192	[CCD+22] [a]	Not given	3.1 M	59534	89200	425 (2.7×)	158	19.3 (4.0×)
	This Work [c]	**991.4k**	**1236.0k**	**74110**	**72409**	**158.5 (1.0)**	**259**	**4.8 (1.0)**

[a] Processing array size: s=t. Keccak is not included.
[b] Processing array size: s=160.
[c] Processing array size: s=160; systolic stages: Q=80.

1.7x ~2.7x less BRAMs

4.0~8.3x higher throughput

9.6~14.5x less memory-time product

Implementation of Mckeycutter

	LUTs	Flip-Flops	DSP	BRAM	Freq (MHz)	Cycles	Time (ms)
BIKE-L1 [JRBG20]	52967	7035	13	49	96	259k	2.69
BIKE-L1 [RBCGG21]	25549	5462	13	34	113	190k	1.67
HQC-L1 [Mel20]	20k	16k	0	12.5	148	40k	0.27
SIKEp434 [REK22] [a]	12818	18271	195	32	250	541k	2.16
McEliece-3488 [CCD+22]	40018	61881	4	177.5	113	970k	8.58
McEliece-3488 **This work** [b]	**58208**	**63561**	**0**	**77**	**151**	**255k** [c]	**1.69**
McEliece-3488 **This work** [d]	**69428**	**69636**	**0**	**64.5**	**278**	**183k** [c]	**0.66**

[a] Implemented on Xilinx Virtex-7 FPGA.
[b] Processing arrya size: s=128; systolic stages: Q=128.
[c] Average cycles of one key generation(Single mode).
[d] Implemented on Xilinx UltraScale+ FPGA. (s=160; Q=80).

Execution speed of key generation is faster than other KEM schemes except HQC in fourth-round standardization.

Comparisons with implementations of other KEM schemes

Outline

- Classic McEliece
- Key strategies
  - Compact GF(2) Gauss elimination hardware
  - Constant-time parallel sorters
  - Algorithm-level pipeline design
- Evaluation and Implementation
- **Conclusion**

Conclusion

1, This work is applicable to the scenario where key generation time is important when a PKE or a KEM is used to provide perfect secrecy.

2, This implementation of McEliece achieves 4x improvements in throughput with 9~14x less memory-time product.

Thanks for your attention!