

Serialized Keccak Architecture for Lightweight Applications

Elif Bilge Kavun, Begül Bilgin, Tolga Yalcin

Department of Cryptography,
Institute of Applied Mathematics,
Middle East Technical University,
Ankara, Turkey

Outline

- Introduction
 - RFID
 - RFID-Hash
- SHA-3 Competition and Lightweight Cryptography
 - SHA-3 Candidates
 - High-Speed Comparison
 - Lightweight Comparison
 - Results
- **KECCAK**
 - Implementation
 - Data Flow
 - Block Diagram
 - Implementation Results
- Conclusion

RFID (Radio-Frequency IDentification)

- The use of an object (RFID tag) for identification and tracking purposes using radio waves
- Consists of (at least) 2 parts
 - Integrated Circuit: for storing and processing information, modulation and demodulation of an RF signal and specialized functions
 - Antenna: for receiving and transmitting signal
- Various areas of RFID applications
- RFID requirements:
 - Extremely low-power consumption
 - Compactness (4-5K gates for security modules)
 - Speed/throughput not so important

RFID (cont'd)

- As a result of the increase in RFID usage, security and identification problems arise
- There exists several protocols using hash functions for the privacy of consumers
- There is no RFID security standard yet
- Need of a standard for RFIDs with cryptographic module technical specifications
 - Area
 - Power consumption
 - Throughput
 - Security issues
- Present assumption: 64/128 bits sufficient for block ciphers and hash functions

SHA-3 Candidates

- **Question:** Are SHA-3 candidates suitable for RFID applications?
- **Answer:** Yes, some of them - JH, Keccak, Luffa
- **How:** After application of lightweight tweaks (and tricks) on the original algorithms

Previous Work on High-Speed Implementations

	Block (bits)	Latency (cycles)	Area (GE)	Clock Freq (MHz)	TP (Gbps)
Blake-32	512	22	45,640	170.64	3.971
BMW-256	512	1	169,737	10.46	5.358
CubeHash16/32-h	256	8	58,872	145.77	4.665
ECHO-256	1,536	97	141,489	141.84	2.246
Fugue-256	32	2	46,257	255.75	4.092
Groestl-256	512	22	58,402	270.27	6.290
Hamsi-256	32	1	58,661	173.91	5.565
JH-256	512	39	58,832	380.22	4.992
Keccak(-256)	1,088	25	56,316	487.80	21.229
Luffa-224/256	256	9	44,972	483.09	13.741
Shabal-256	512	50	54,186	320.51	3.282
SHAvite-3(256)	512	37	57,388	227.79	3.152
SIMD-256	512	36	104,166	64.93	0.924
Skein-256-256	256	10	58,611	73.52	1.882
Skein-512-512	512	10	102,039	48.87	2.502
SHA-256	512	66	19,144	302.11	2.344

Estimated Lightweight Comparison

	Rounds	Message Block (bits)	State Regs (bits)	Throughput (bits/cycle)	Efficiency (bpc/regs)
Blake	10x8=80	256	512	3.2	0.006
BMW*	16	128	528	8	0.015
CubeHash	16	128	1024	8	0.007
ECHO	8	1792	2048	224	0.1
Fugue	2	32	768	16	0.016
Groestl	10	256	768	25.6	0.05
Hamsi**	3	16	256	5.3	0.02
JH	24	128	384	5.33	0.013
KECCAK-f [400]-128	20	144	400	7.2	0.018
KECCAK-f [200]-64	18	72	200	4	0.02
Luffa	9	128	256	14.2	0.05
Shabal	50	128	352	2.56	0.007
SHAvite	42	512	1088	12.19	0.011
SIMD	36	128	1024	3.55	0.0035
Skein	72	256	512	3.55	0.007
* : Additions and Subtractions			** : S-boxes and Expansion		

Area Issues

Number of registers, hence the area is too high to be considered for a lightweight application:

- Blake
- BMW
- CubeHash
- ECHO
- Fugue
- Groestl
- SHAvite
- SIMD

Throughput Issues

Throughput is too low for the occupied area:

- Blake
- Shabal
- Skein

S-box, Expansion, Add/Sub Operations

The area of S-boxes and message expansion block is too large:

- Hamsi

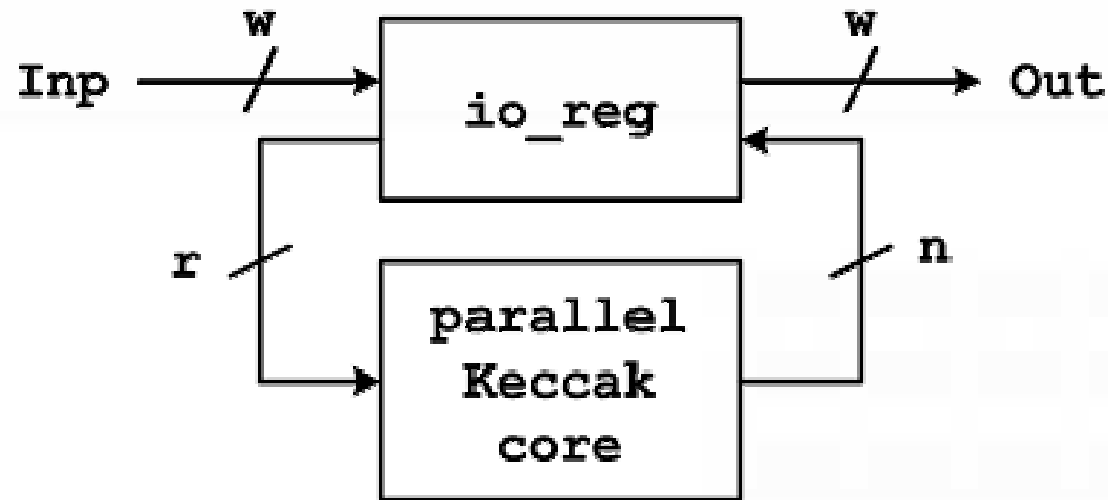
The delay of add/sub operations is too high:

- BMW
- Hamsi

Why **KECCAK**?

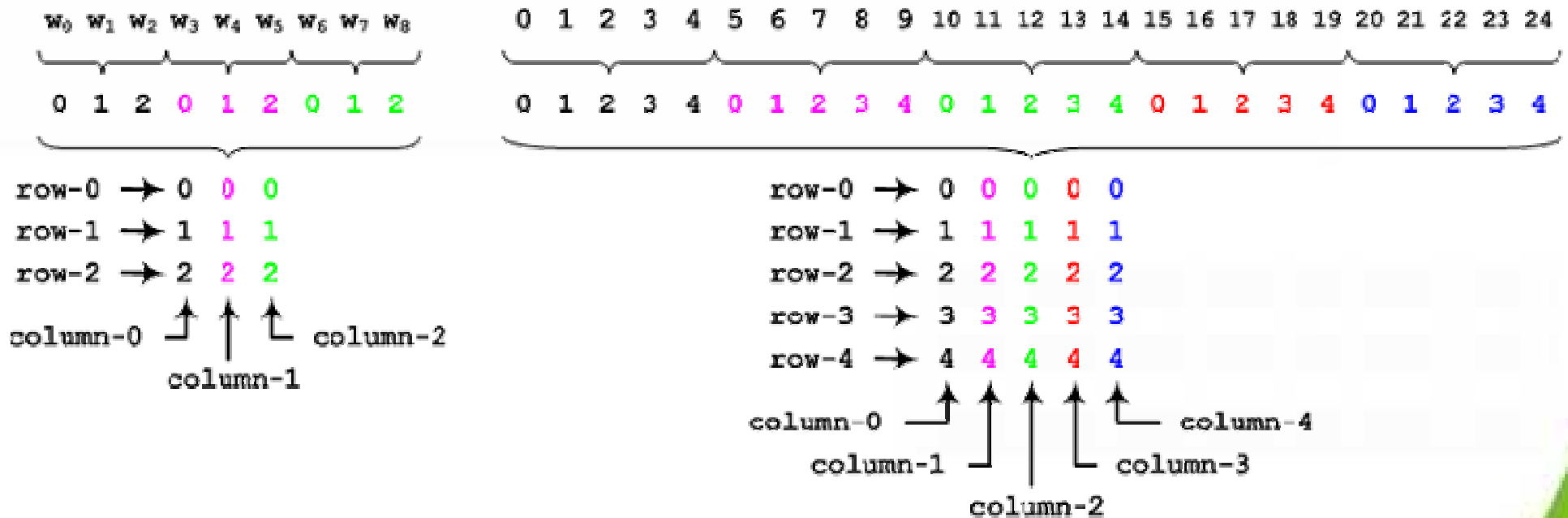
- **JH**, **KECCAK** and **Luffa** have similar performances in terms of area and throughput
- **KECCAK** and **Luffa** are sponge functions which are original in design
- For lightweight purposes we investigated **KECCAK200**, **KECCAK400** and **Luffa(w=2)**
- **KECCAK** has simpler functions which yield to shorter delays and smaller area

Parallel KECCAK Implementation



- Fully parallel straight forward implementation
- r-bit I/O register for fast data transfer
- Optimized for area
- **KECCAK- $f[1600]$ -256, $f[400]$ -128, $f[200]$ -64** implemented in parallel for comparison with the target lightweight architecture

Lightweight KECCAK Implementation

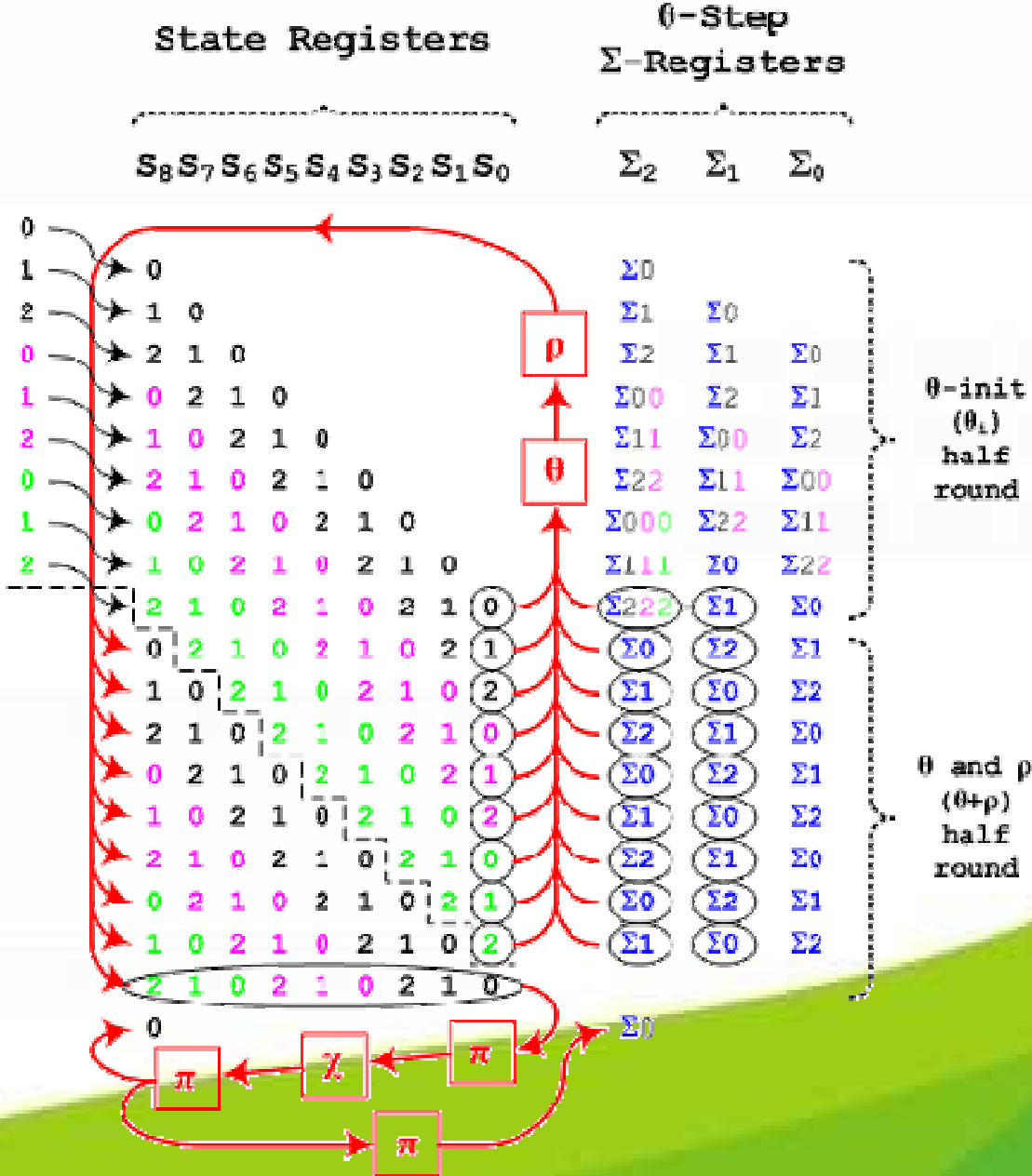


- A serial datapath design is favoured
- Data blocks processed in words
- All the step functions (except π -step) reduced to w -bits
- Operation explained for a 3×3 state matrix (instead of regular 5×5 Keccak matrix)

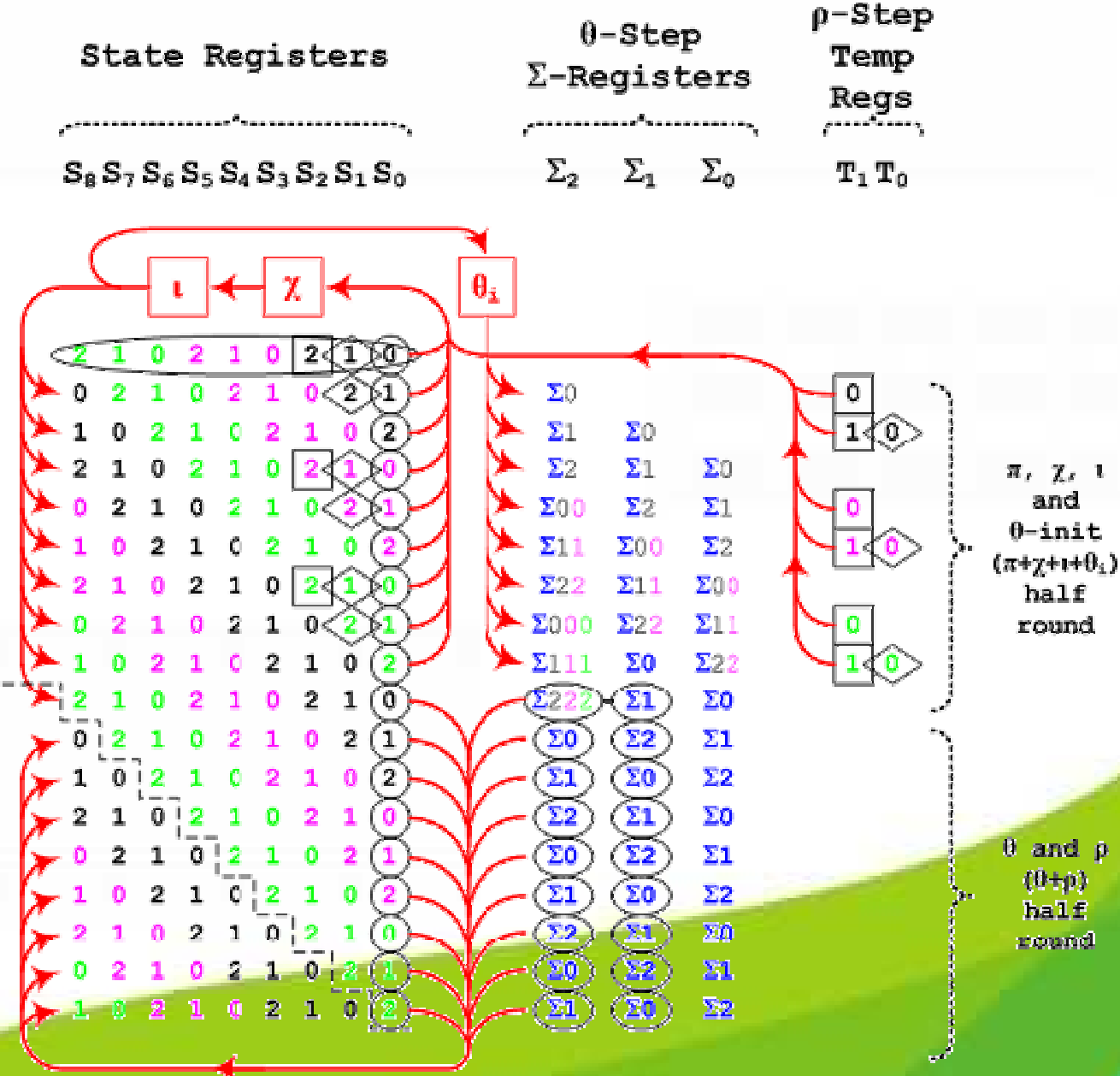
Lightweight KECCAK Data Flow

- Each round divided into 2 half rounds
- The 0th half round is for the absorption of the first data block and initialization of row sums for θ step
- It is followed by regular rounds
- The 1st half round of each round is for θ and ρ steps
- The 2nd half round is for π , χ and ι steps
- Initialization of row sums for θ step of the next round also takes place in the second half round

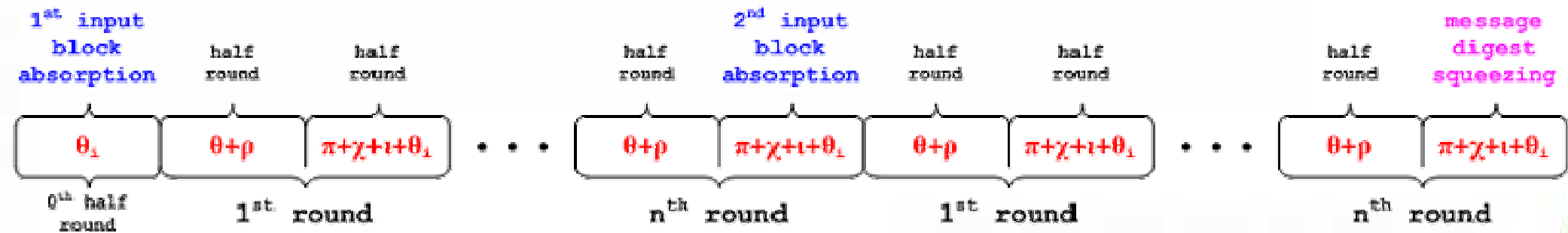
Lightweight KECCAK Data Flow (cont'd)



Lightweight KECCAK Data Flow (cont'd)

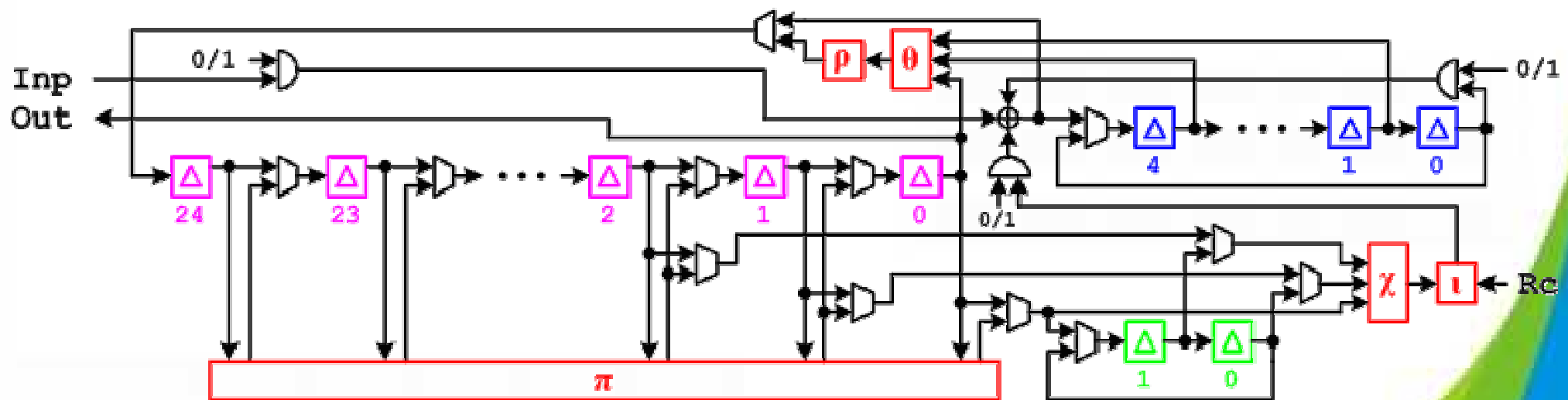


Lightweight KECCAK Data Flow (cont'd)



- Each half round takes 25 cycles to complete
- Number of rounds depends on the **KECCAK** permutation
- The second half round of every last round is also used for absorption of the next data block
- Data squeezing occurs at the second half of the last round of the last data block

Lightweight KECCAK Block Diagram



 : state registers  : Σ registers  : χ temp registers  : step blocks

Lightweight vs. Parallel

Parallel Implementations								
	Hash Output Size	Data path Size	Input Data Size	Cycles per Block	Throughput @ 250 MHz (Mbps)	Area (GE)	Efficiency (Mbps/GE)	Power Cons. (μ W/MHz)
KECCAK-f [1600]-256	256	64	1088	24	11333	47.63K	0.238	315.1
KECCAK-f [400]-128	128	16	144	20	1800	10.56K	0.170	78.1
KECCAK-f [200]-64	64	8	72	18	1000	4.9K	0.204	27.6
Lightweight Implementations								
	Hash Output Size	Data path Size	Input Data Size	Cycles per Block	Throughput @ 100 KHz (Kbps)	Area (GE)	Efficiency (bps/GE)	Power Cons. (μ W/MHz)
KECCAK-f [1600]-256	256	64	1088	1200	90.66	20.79K	4.36	44.9
KECCAK-f [800]-128	128	32	544	1100	49.45	13K	3.8	28.2
KECCAK-f [400]-128	128	16	144	1000	14.4	5.09K	2.83	11.5
KECCAK-f [200]-64	64	8	72	900	8	2.52K	3.17	5.6

KECCAK vs. Others

Function	Hash Output Size	Throughput @ 100 KHz (Kbps)	Area (GE)	Efficiency (bps/GE)
MD-5	128	83.66	8.4K	10
SHA-256	256	45.39	10.9K	4.2
MAME	256	266.67	8.1K	32.9
DM-Present-128	64	387.88	1.6K	26.91
H-Present-128	128	200	4.3K	47
C-Present-128	192	59.26	8.0K	7.4
KECCAK-f [400]-128	128	14.4	5.09	2.83
KECCAK-f [200]-64	64	8	2.52	3.17

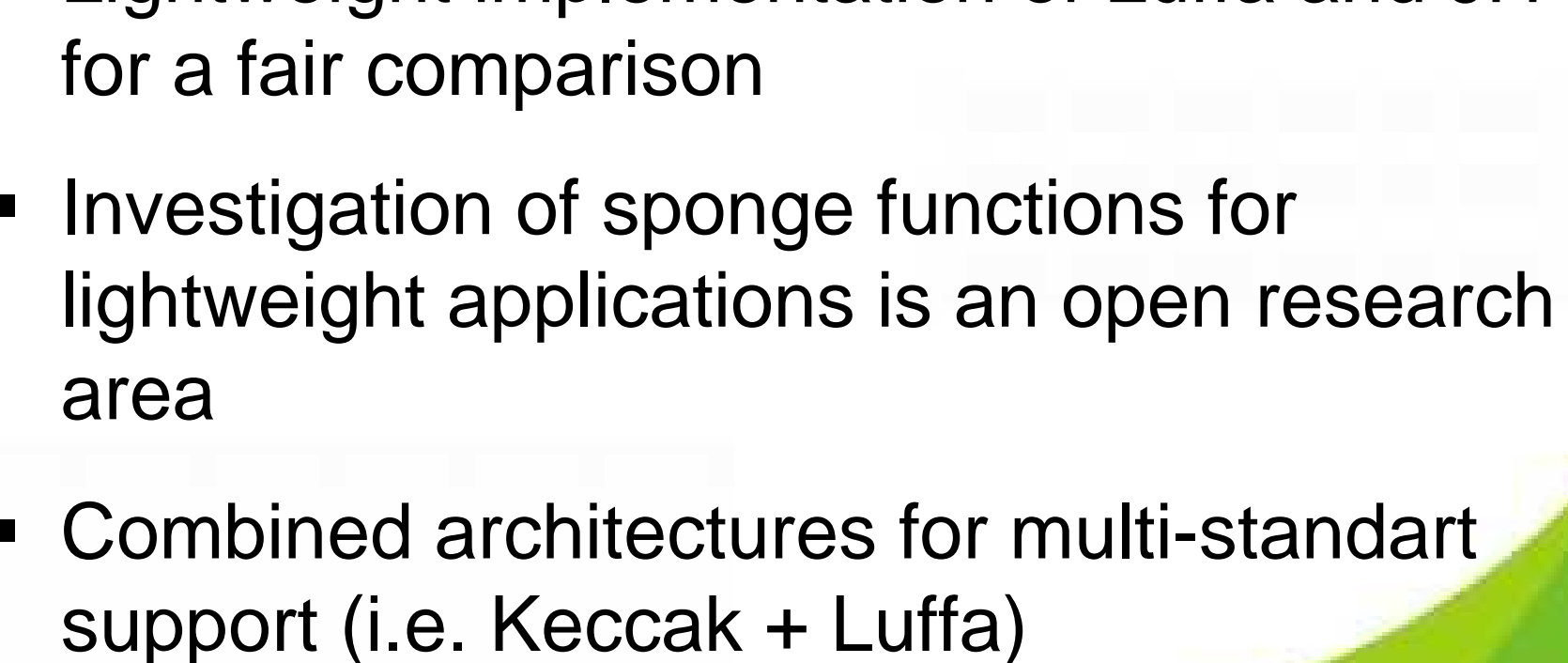
Randomness Test Results for KECCAK-f[400]

	1-Round	2-Rounds	3-Rounds	4-Rounds	5-Rounds
Frequency	0.000000	0.294749	0.911874	0.662486	0.234682
B.Frequency	0.000000	0.519962	0.000920	0.834166	0.553189
# of Runs	0.000000	0.000494	0.089943	0.373540	0.738698
Longest Run	0.000000	0.306412	0.592643	0.490267	0.926211
App. Entropy	0.000000	0.217282	0.461494	0.176800	0.103087
Cum. Sum1	0.000000	0.406529	0.298847	0.160818	0.451807
Cum. Sum2	0.000000	0.000215	0.635253	0.145432	0.067465
Serial-1	0.000000	0.264770	0.465440	0.065450	0.184705

Conclusion

- Lightweight Keccak is a feasible candidate for RFID tags and other applications with limited resources
- Area and throughput comparable to lightweight specific compression and block cipher based authentication functions (MAME, Present-MAC, etc.)
- Lightweight Keccak security seems to be adequate for target applications
- Throughput improvement still possible through row or column based processing instead of word-based processing at the cost of increased gate count and power consumption

Future Work

- Lightweight implementation of Luffa and JH for a fair comparison
 - Investigation of sponge functions for lightweight applications is an open research area
 - Combined architectures for multi-standart support (i.e. Keccak + Luffa)
- 

References

- Andrey Bogdanov, Gregor Leander, Christof Paar, Axel Poschmann, M.J.B. Robshaw, Yannick Seurin, Hash Functions and RFID Tags : Mind The Gap, CHES 2008, LNCS, Springer-Verlag, 2008.
- S. Tillich et al, High-Speed Hardware Implementations of BLAKE, BMW, CubeHash, ECHO, Fugue, Grostl, Hamsi, JH, Keccak, Luffa, Shabal, SHAvite-3, SIMD, and Skein, In Cryptography ePrint, November 2009.
- G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, Keccak specifications, version 2, submission to NIST, 2009.
- G. Bertoni, J. Daemen, M. Peeters and G. Van Assche, Keccak sponge function family main document, submission to NIST, 2009.
- H. Yoshida, D. Watanabe, K. Okeya, J. Kitahara, H. Wu, O. Kucuk, B. Preneel, MAME: A compression function with reduced hardware requirements, LNCS, volume 4727, pages 148-165. Springer, August 2007.
- M. Feldhofer, C. Rechberger, A Case Against Currently Used Hash Functions in RFID Protocols, LNCS, Volume 4277, pages 372-381. Springer, Nov 2006.

THANKS FOR LISTENING...

QUESTIONS?