COSIC

Efficient and First-Order DPA Resistant Implementations of KECCAK

Begül Bilgin, Joan Daemen, Ventzislav Nikov,
Svetla Nikova, Vincent Rijmen, and Gilles Van Assche

KU LEUVEN NXP ST

CARDIS 2013, Berlin

KECCAK

Selected by NIST as SHA-3

Software Hardware

Slice Based Hybrid Lane Based

Low Area High Speed Uses RAM Reg. only

Hashing Encryption MAC AE PRNG

KECCAK

Selected by NIST as SHA-3

Software

Hardware

Slice Based

Hybrid

Lane Based

Low Area

High Speed

Uses RAM

Reg. only

Hashing

Encryption MAC AE PRNG

Secret key or internal state

Needs to be secured against DPA

KECCAK

Countermeasures

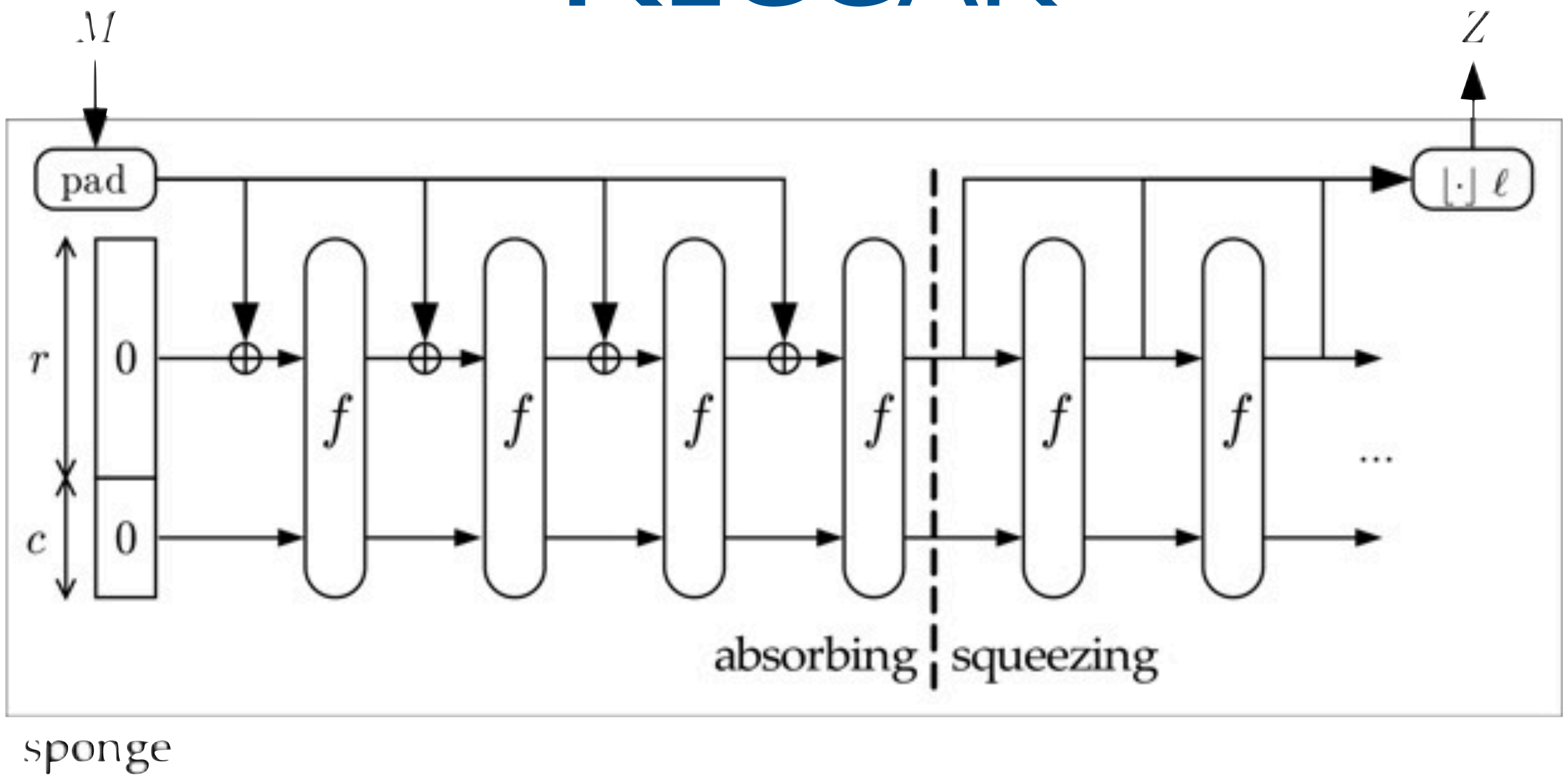
- So far [BDPV]:
  - Software: 2-share masking
  - Hardware: 3-share masking with threshold imp. (TI)

    This version does not satisfy all the TI properties
- This work:
  - 3-share TI with injection of fresh randomness
  - 4-share TI

Outline

- KECCAK
  - Architecture
  - Plain implementation
- Threshold Implementation
  - Properties
  - χ-function
  - f-function
- Performance results and comparison

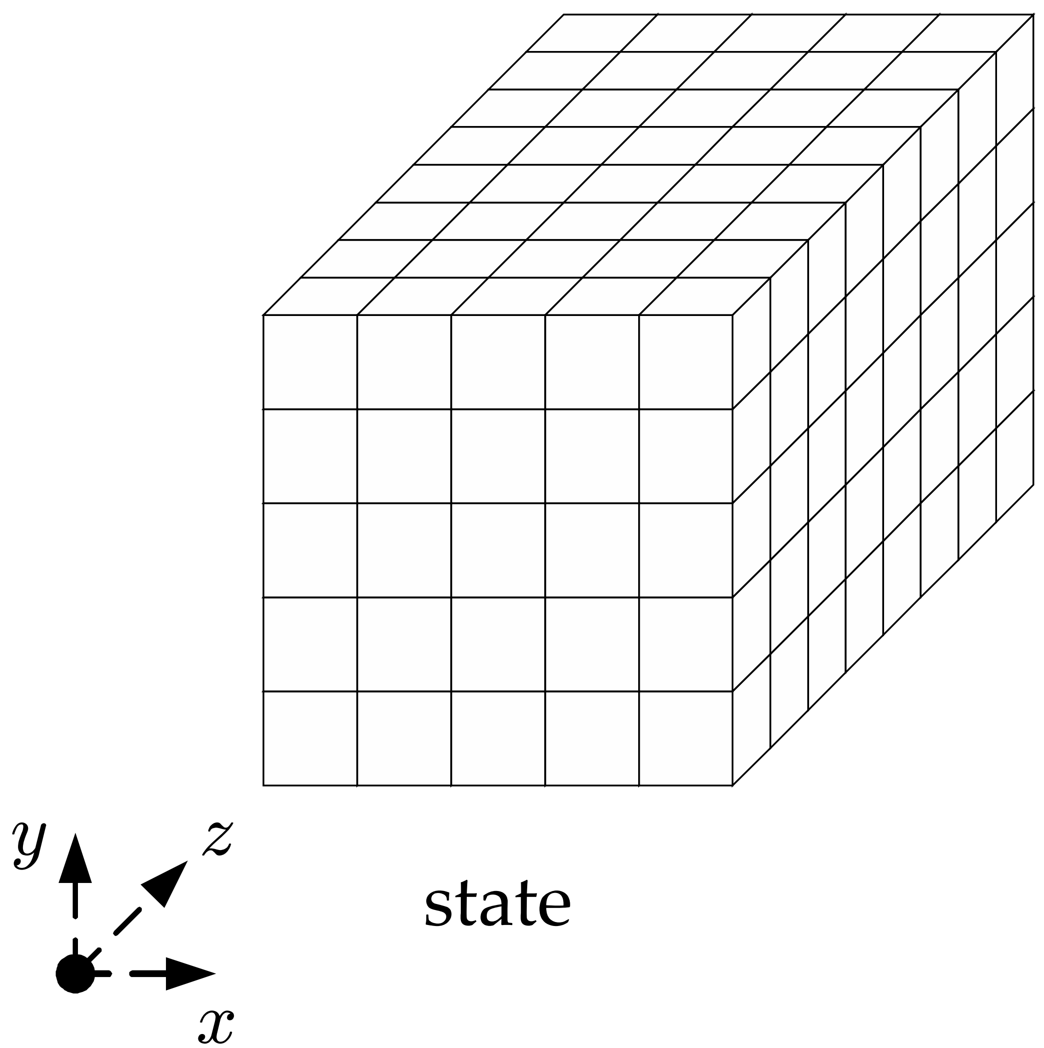
KECCAK



- b=r+c bits permutation KECCAK-*f*
- 7 versions: b∈{25, 50, 100, 200, 400, 800, 1600}
- SHA-3 instance: b=1600

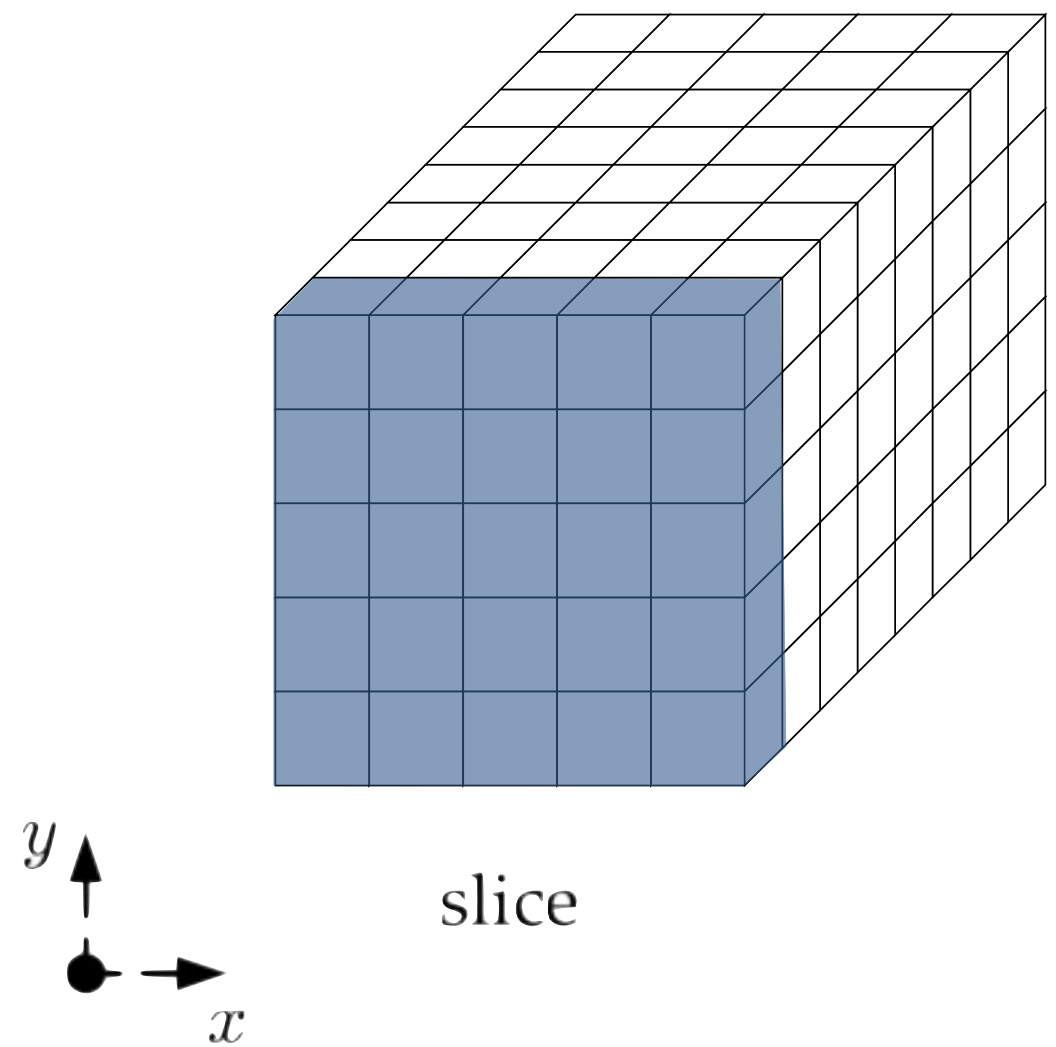
KECCAK

f-permutation



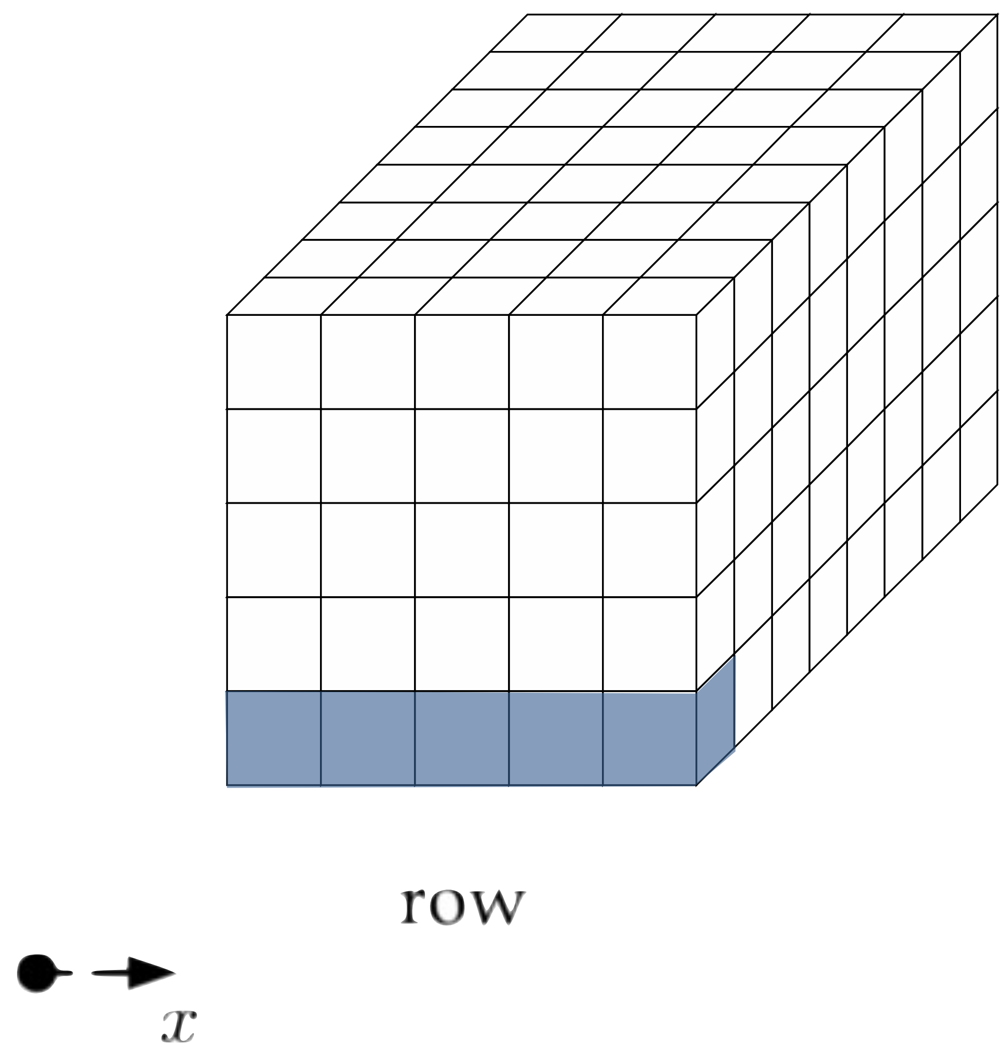
KECCAK

f-permutation



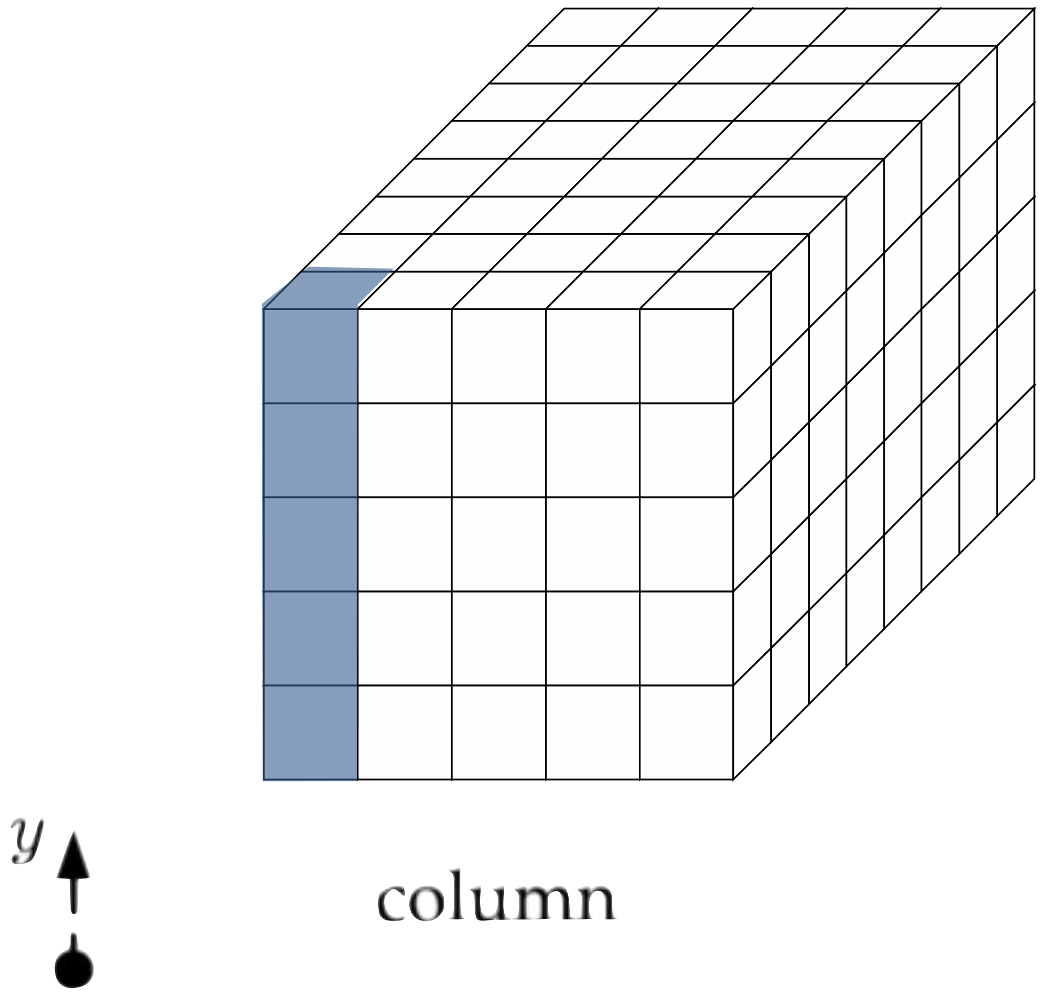
KECCAK

f-permutation



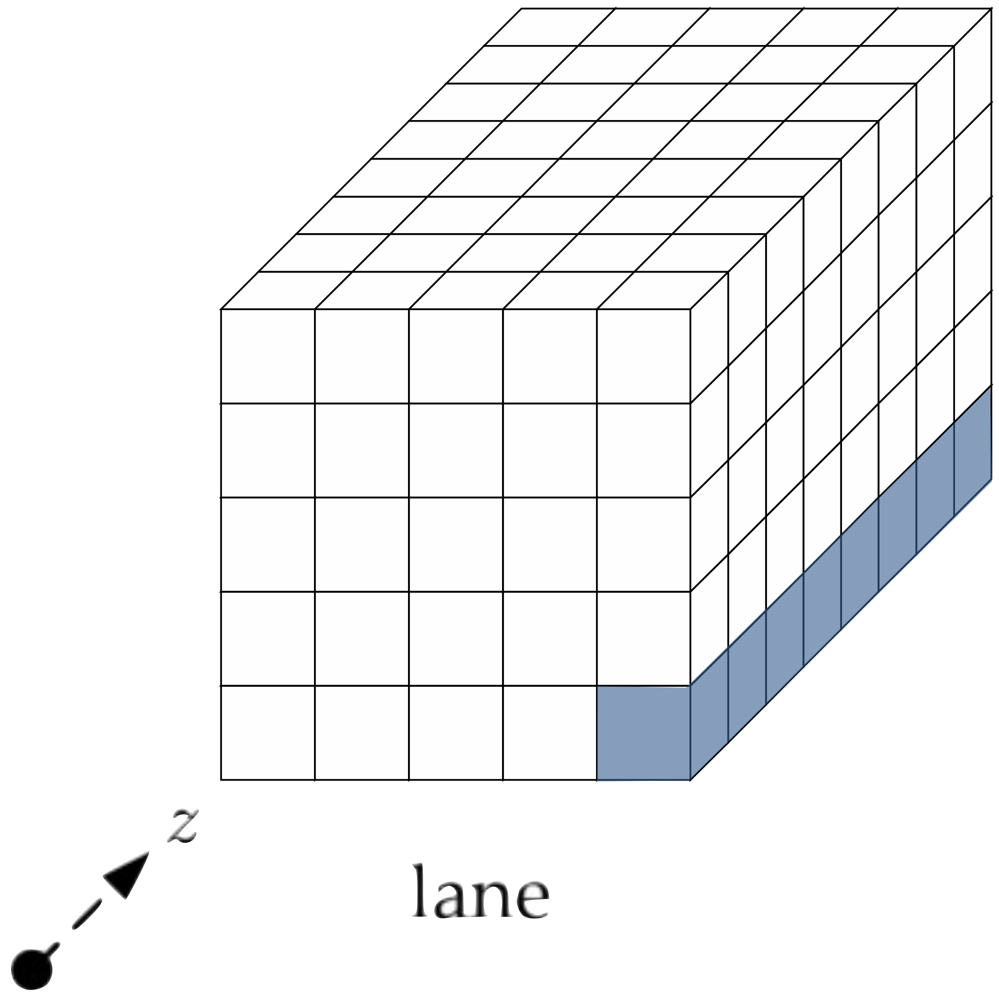
KECCAK

f-permutation



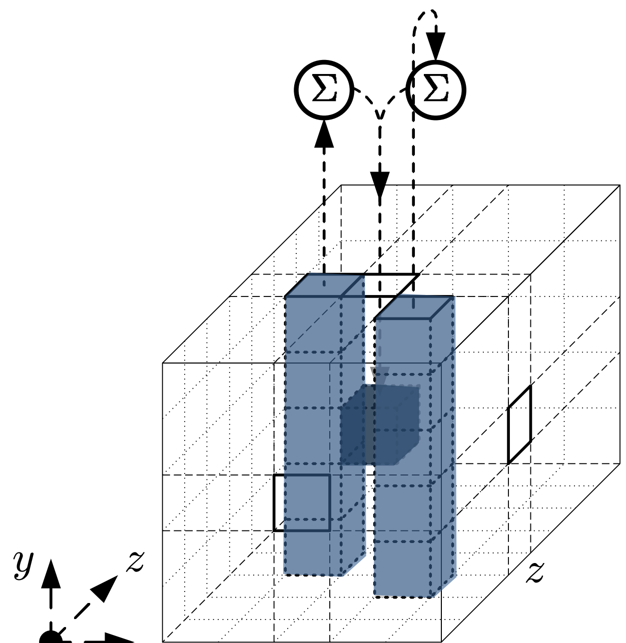
Keccak

f-permutation

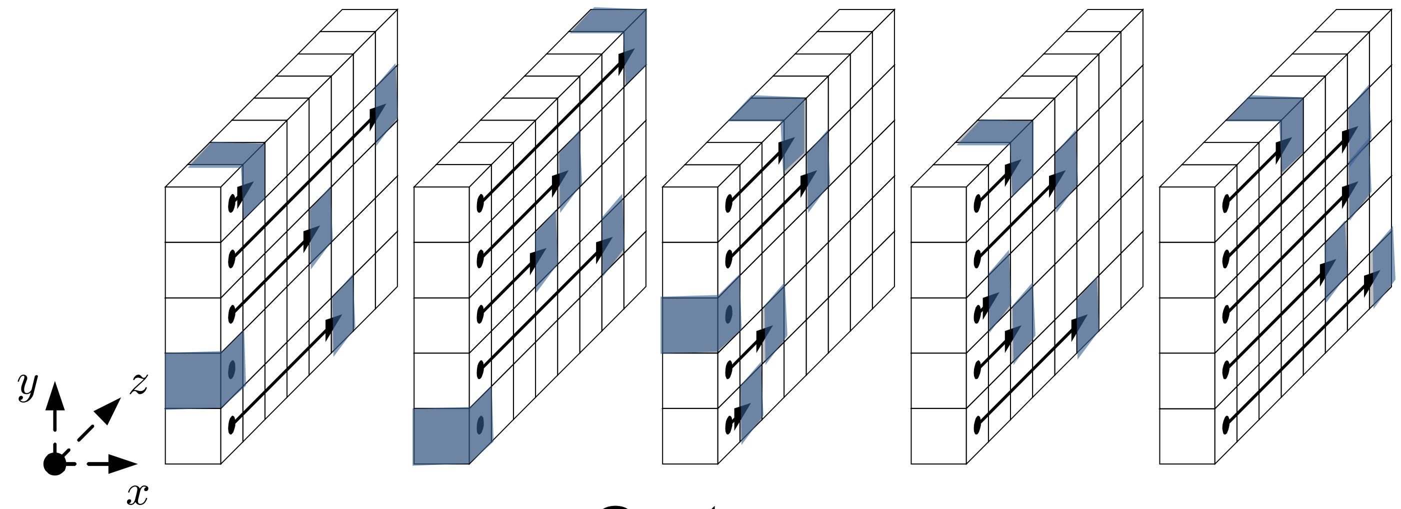


KECCAK

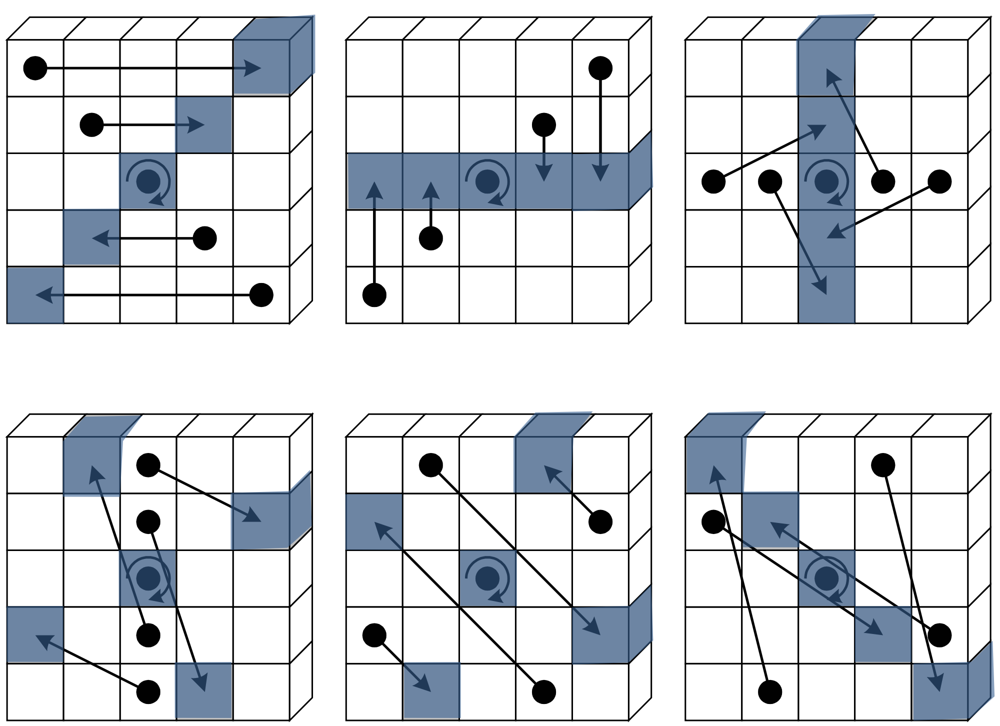
f-permutation



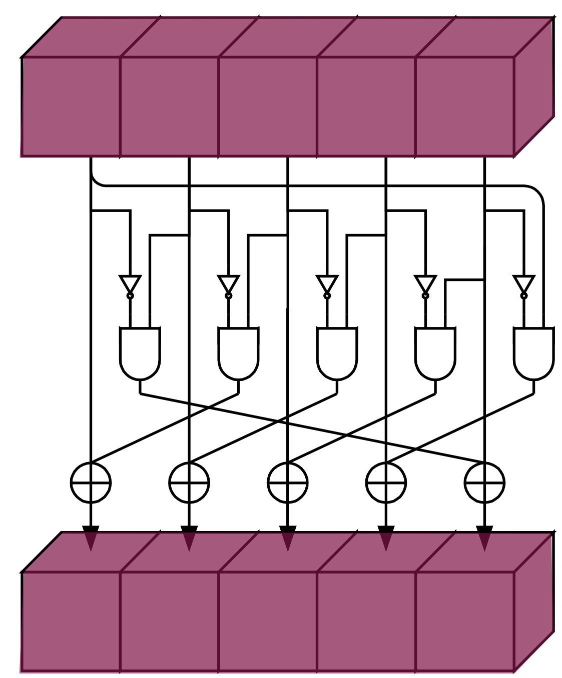
θ step



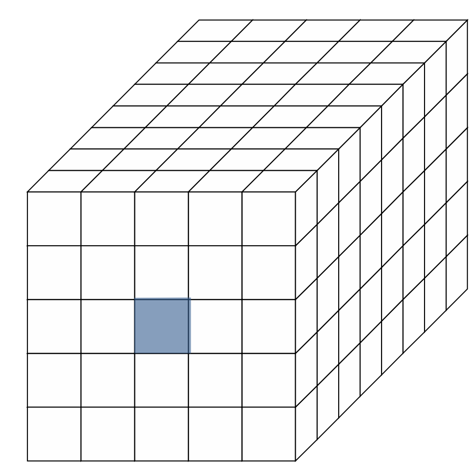
ρ step



π step



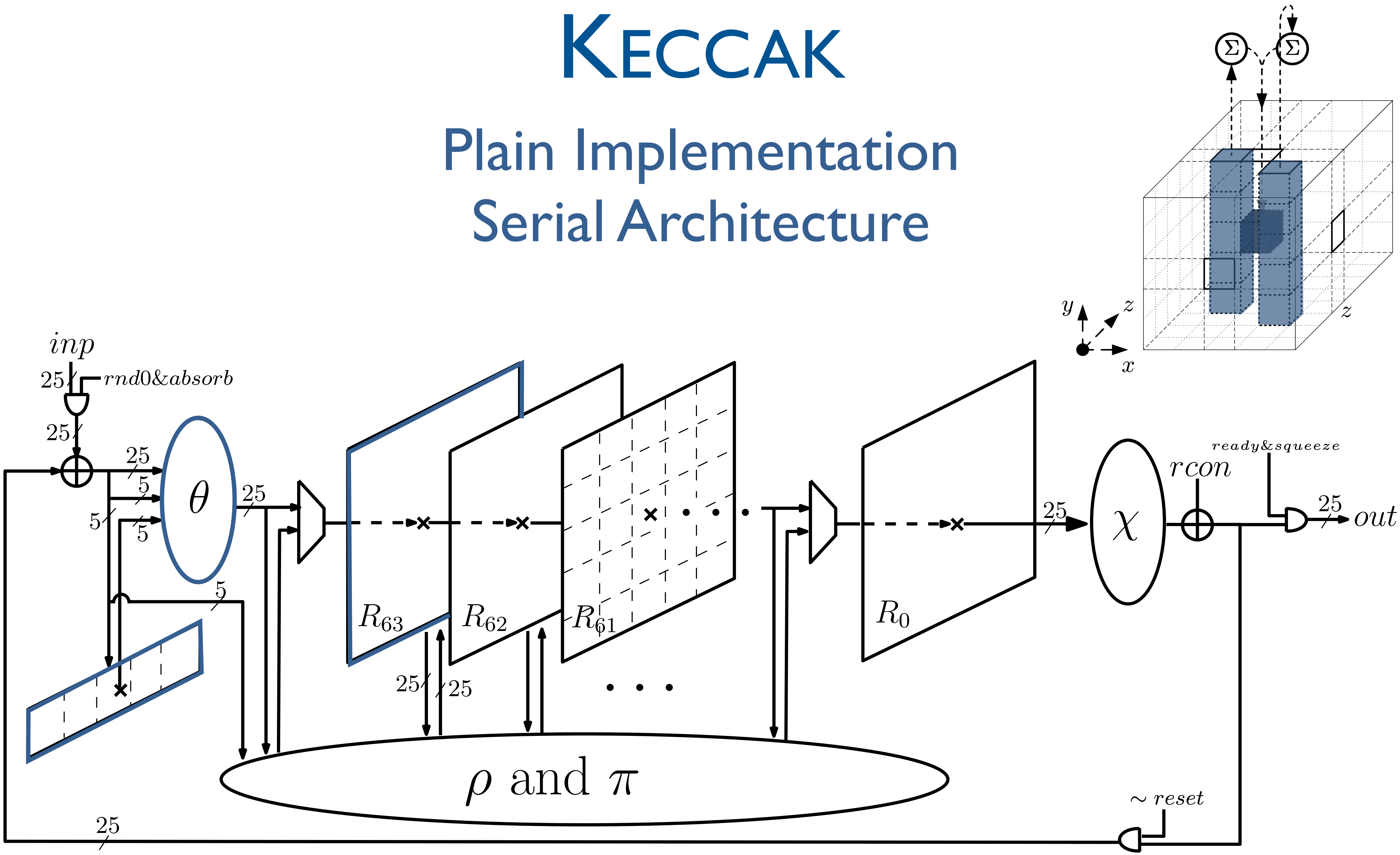
χ step



ι step

Keccak

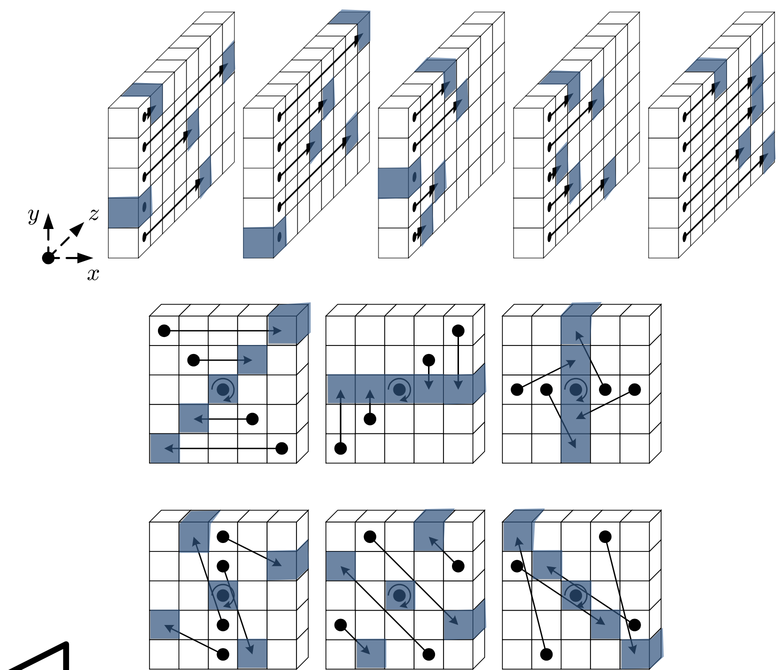
Plain Implementation
Serial Architecture

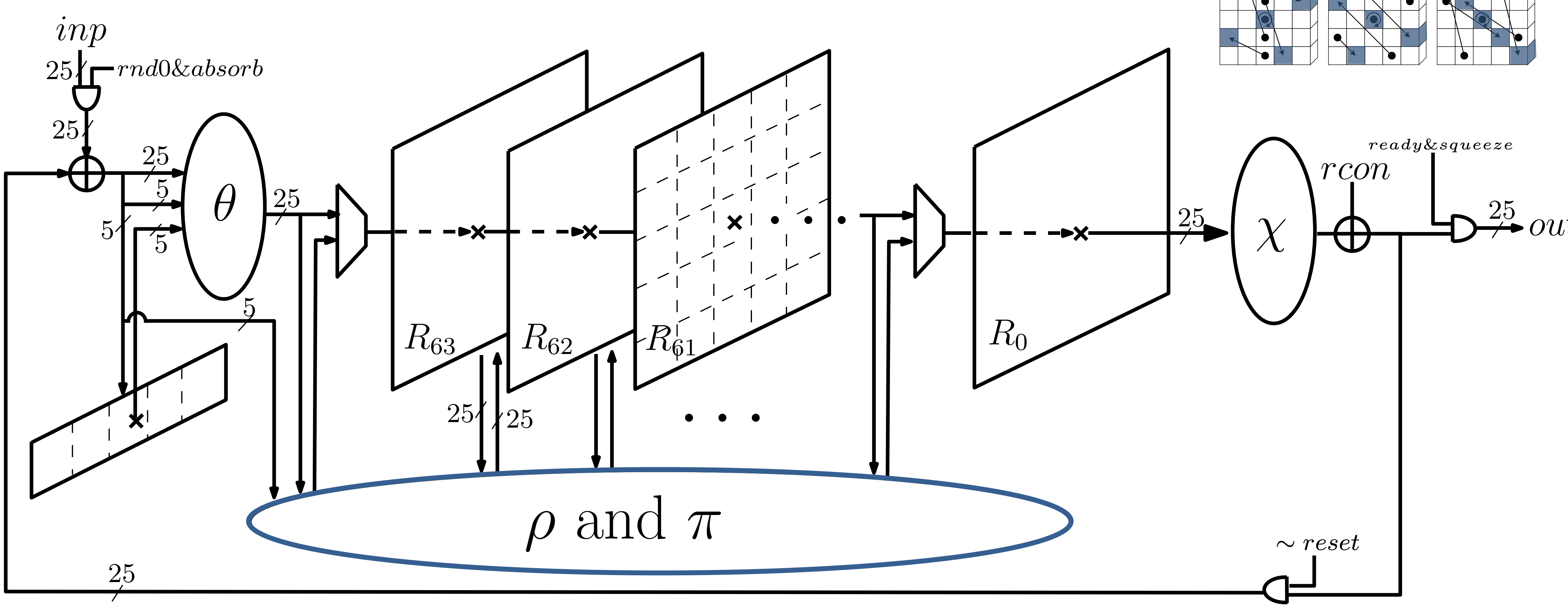


~ 170GE

KECCAK

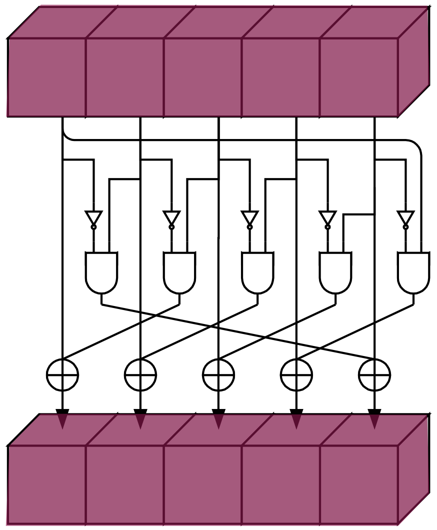
Plain Implementation
Serial Architecture

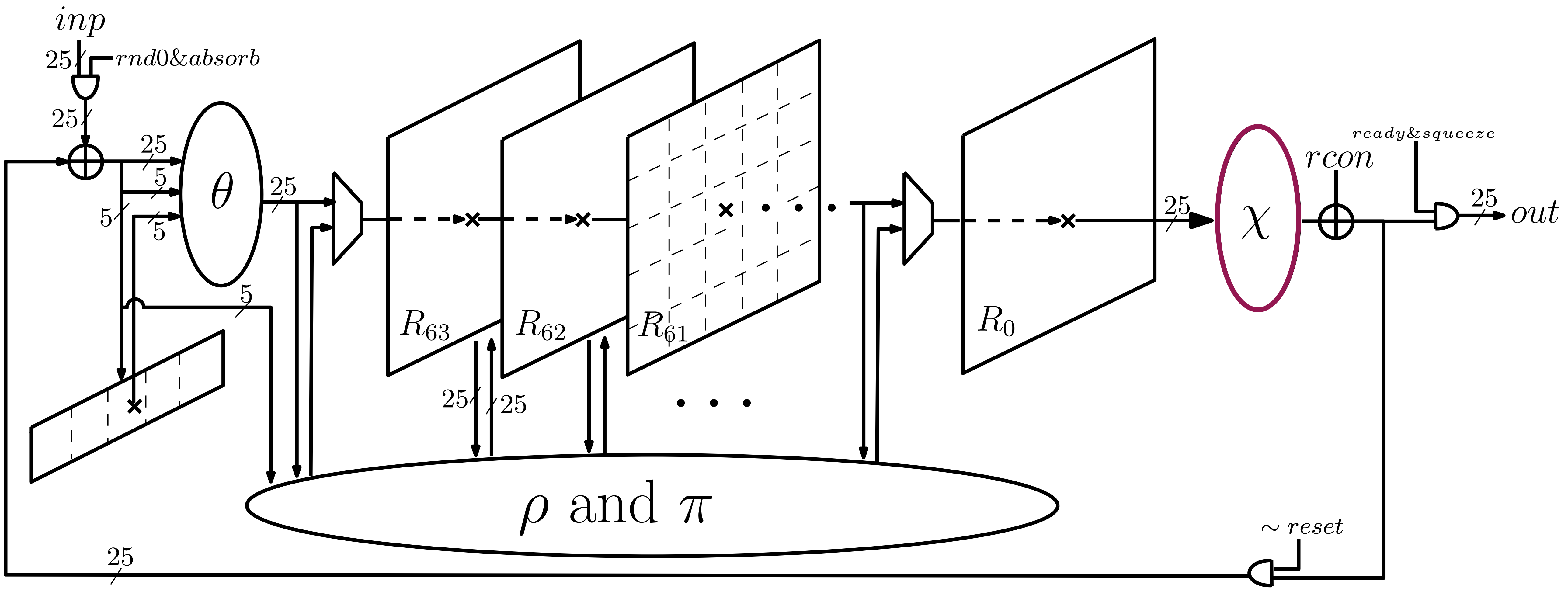




KECCAK

Plain Implementation
Serial Architecture

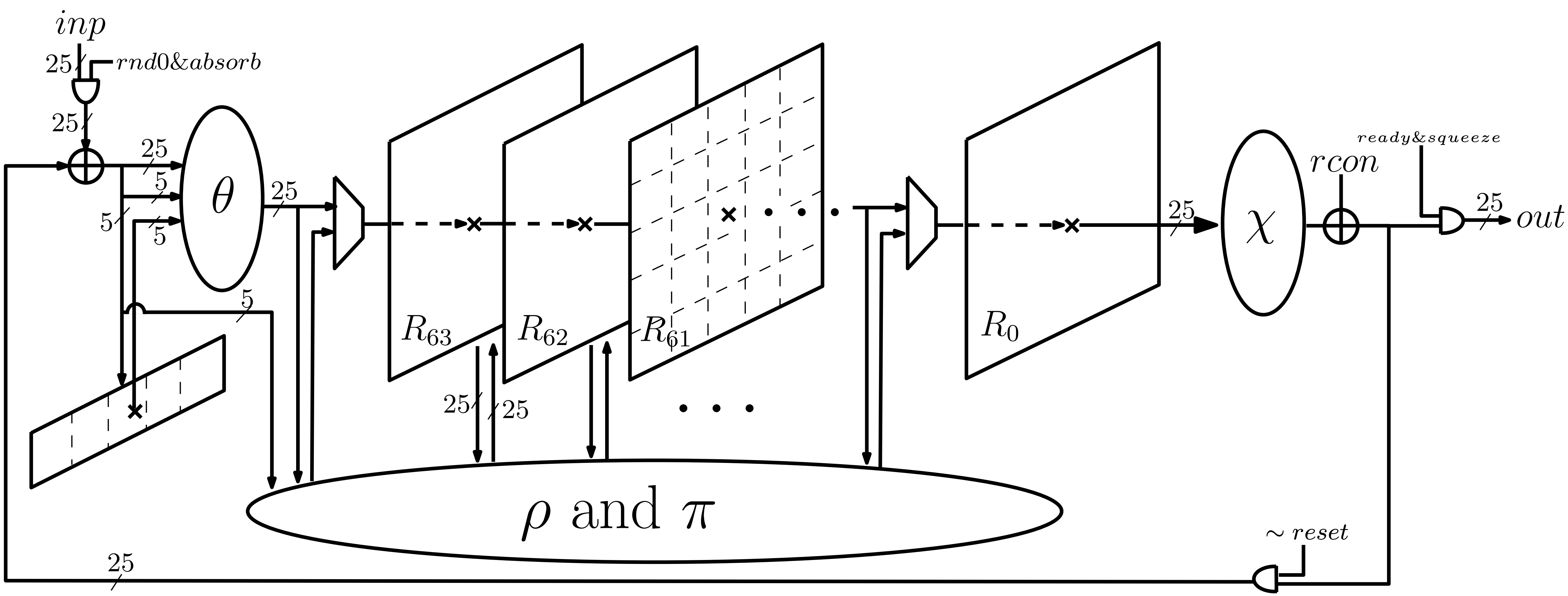




~ 110 GE

KECCAK

Plain Implementation
Serial Architecture



State only ~ 10kGE

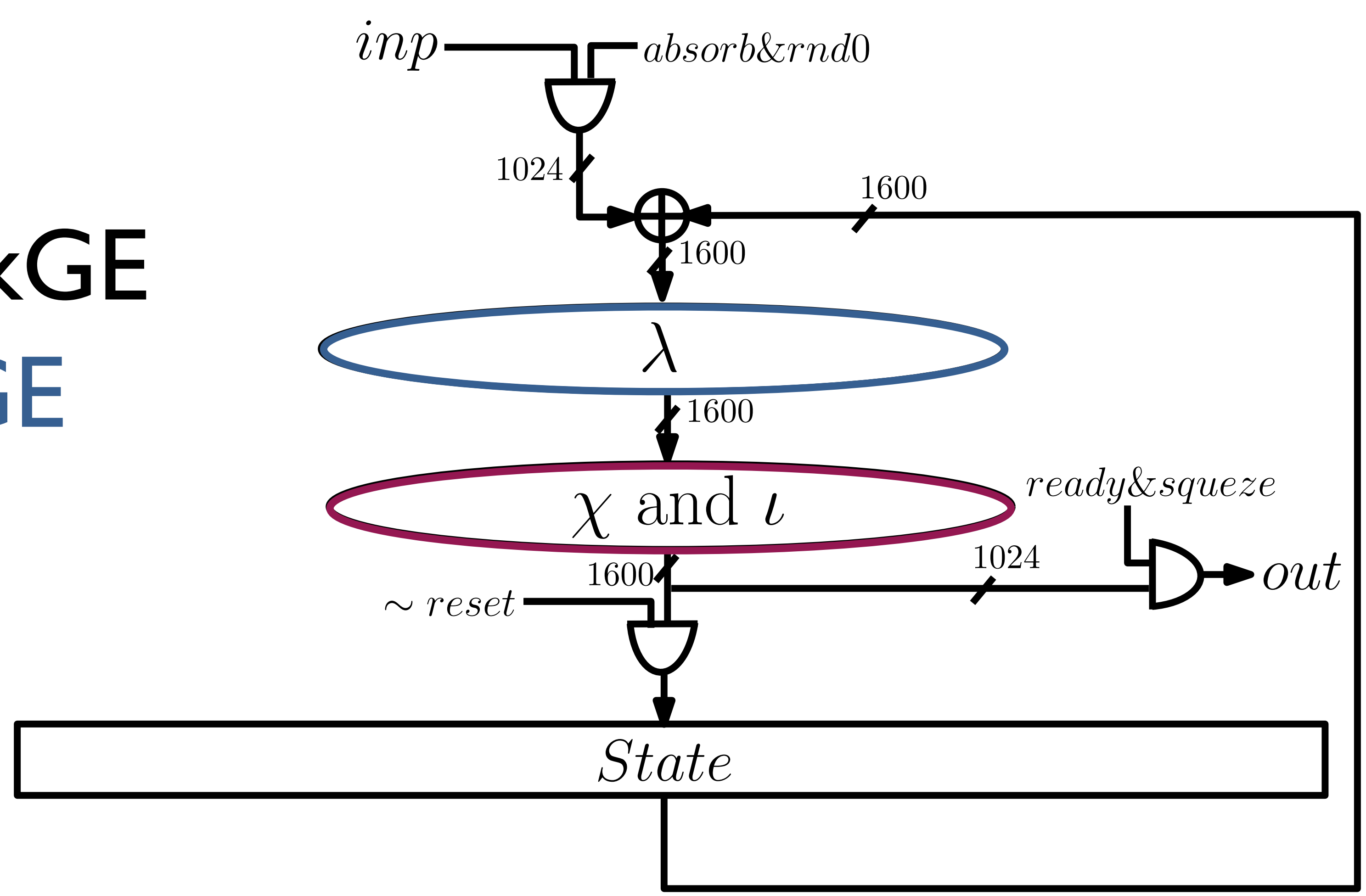
KECCAK

Plain Implementation
Parallel Architecture

State ~ 9kGE

λ ~ 9.3kGE

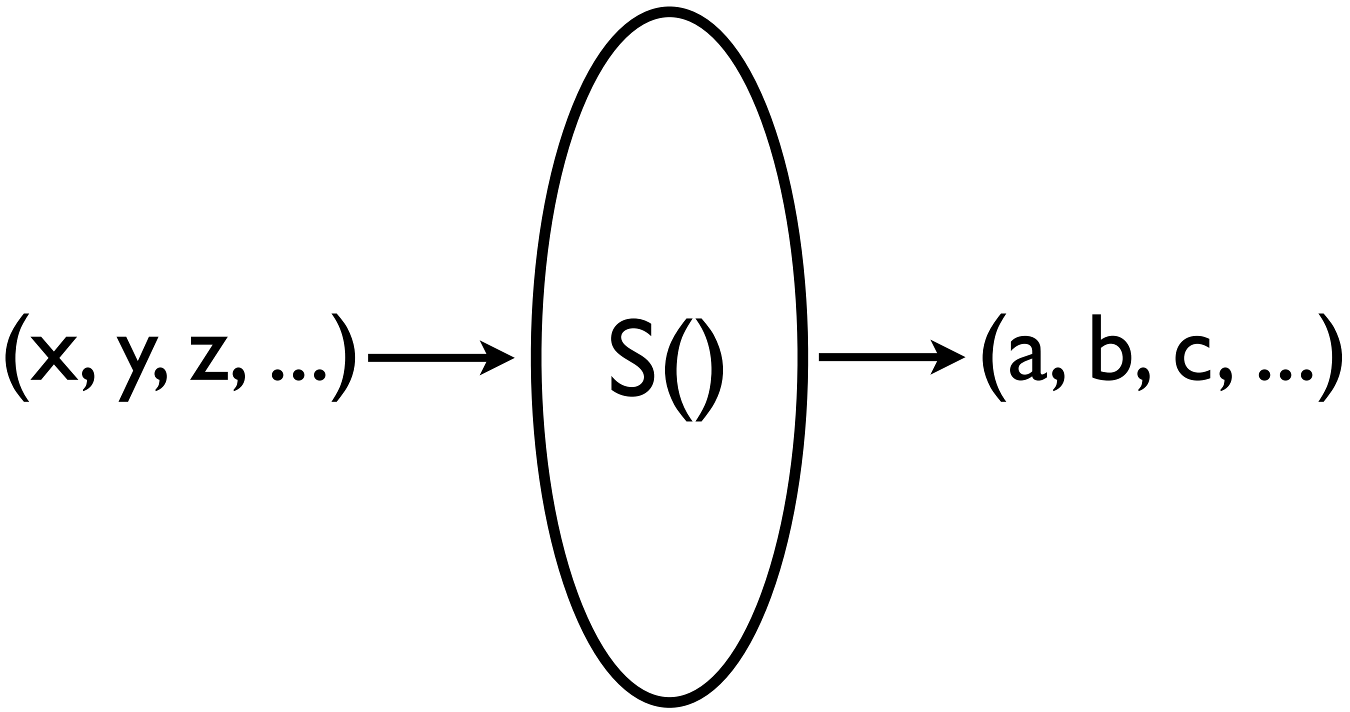
χ ~ 7kGE



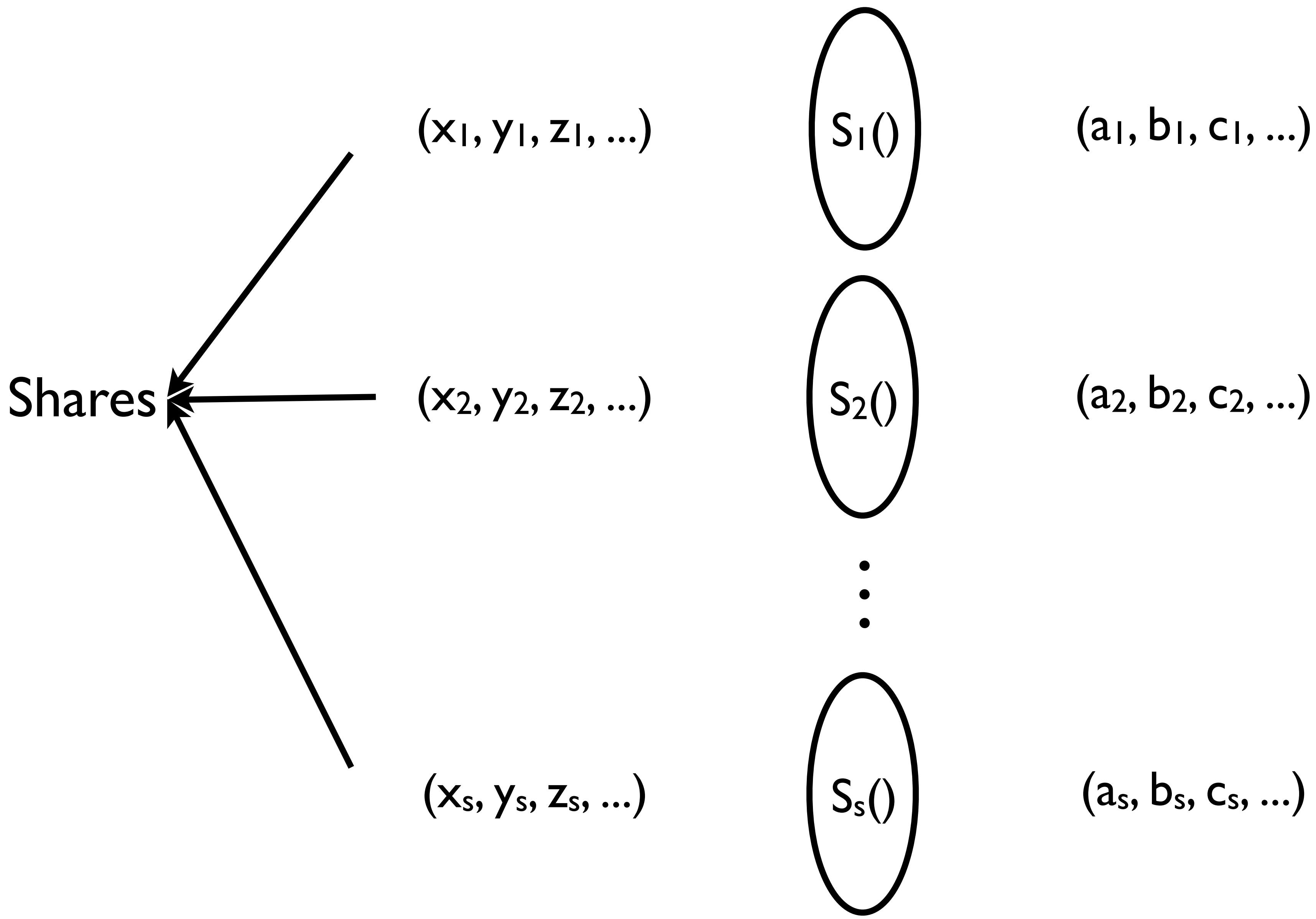
Threshold Implementations

- Kind of a Boolean Masking
- Provably secure against 1st order DPA
- Based on Secret Sharing and Multi Party Computations

Threshold Implementations



Threshold Implementations

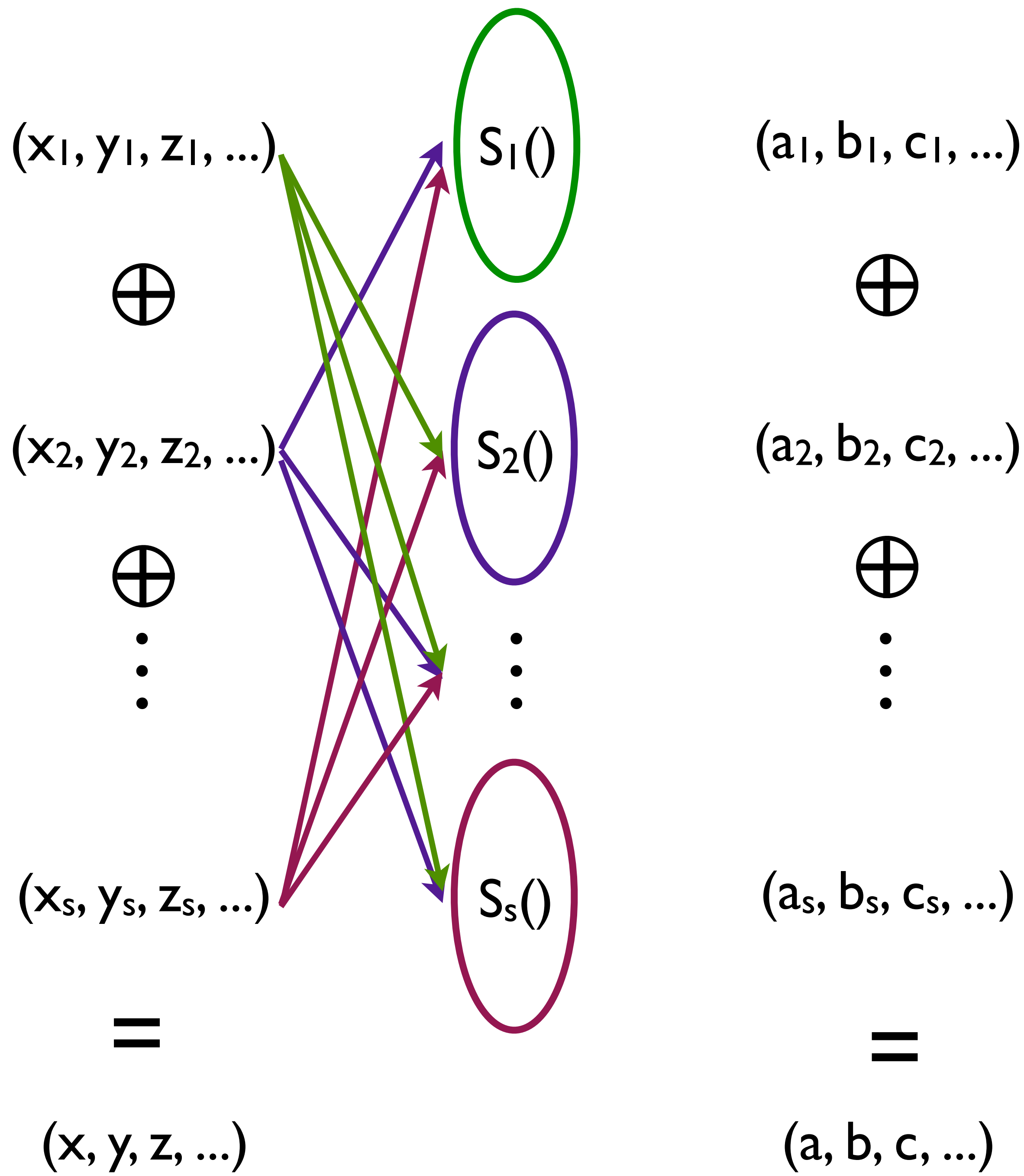


Threshold Implementations

$(x_1, y_1, z_1, \ldots)$	$S_1()$	$(a_1, b_1, c_1, \ldots)$
$\oplus$		$\oplus$
$(x_2, y_2, z_2, \ldots)$	$S_2()$	$(a_2, b_2, c_2, \ldots)$
$\oplus$		$\oplus$
$\vdots$	$\vdots$	$\vdots$
$(x_s, y_s, z_s, \ldots)$	$S_s()$	$(a_s, b_s, c_s, \ldots)$
$=$		$=$
$(x, y, z, \ldots)$		$(a, b, c, \ldots)$

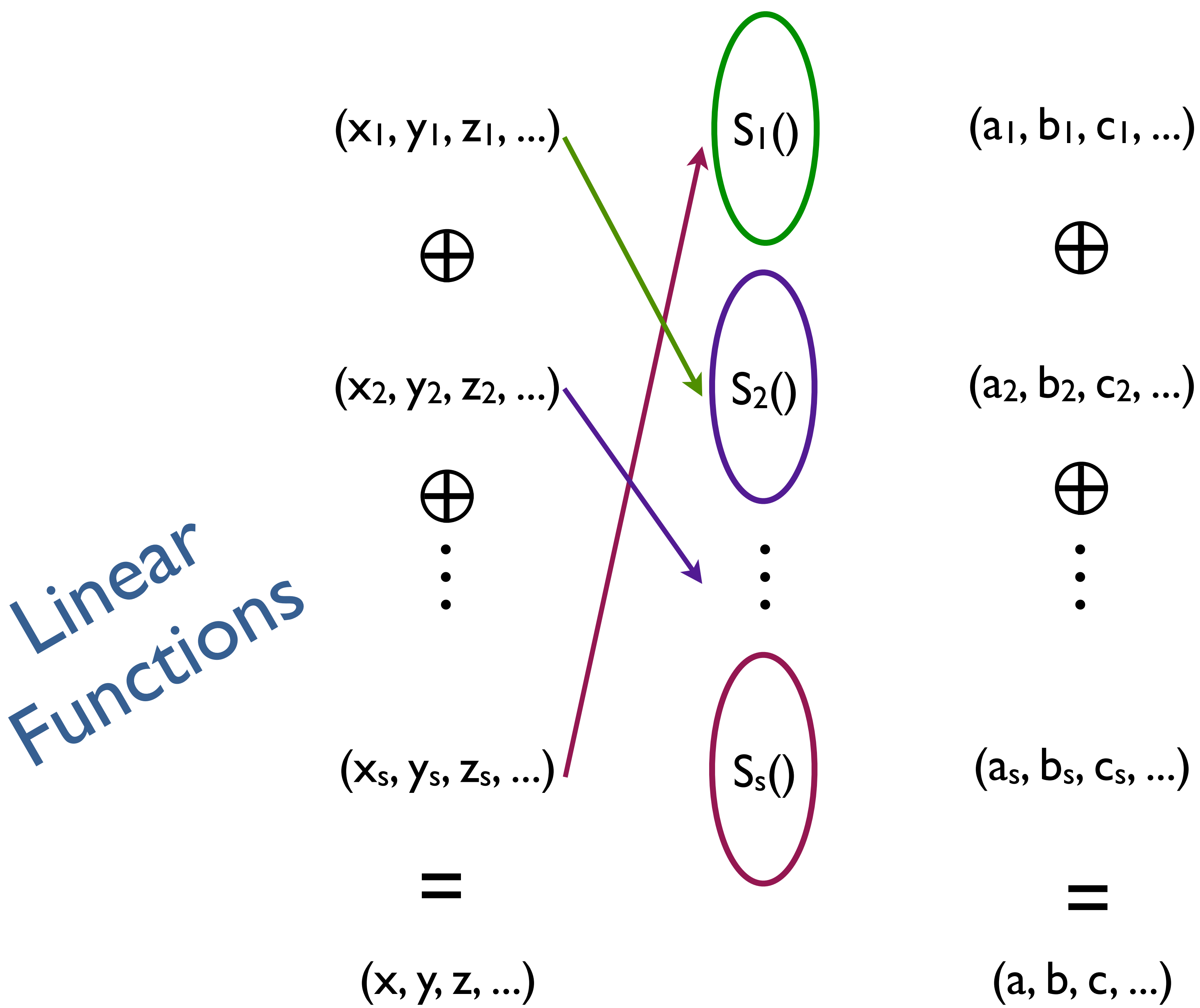
Correct

Threshold Implementations



Correct, Non-complete

Threshold Implementations



Correct, Non-complete

Threshold Implementations

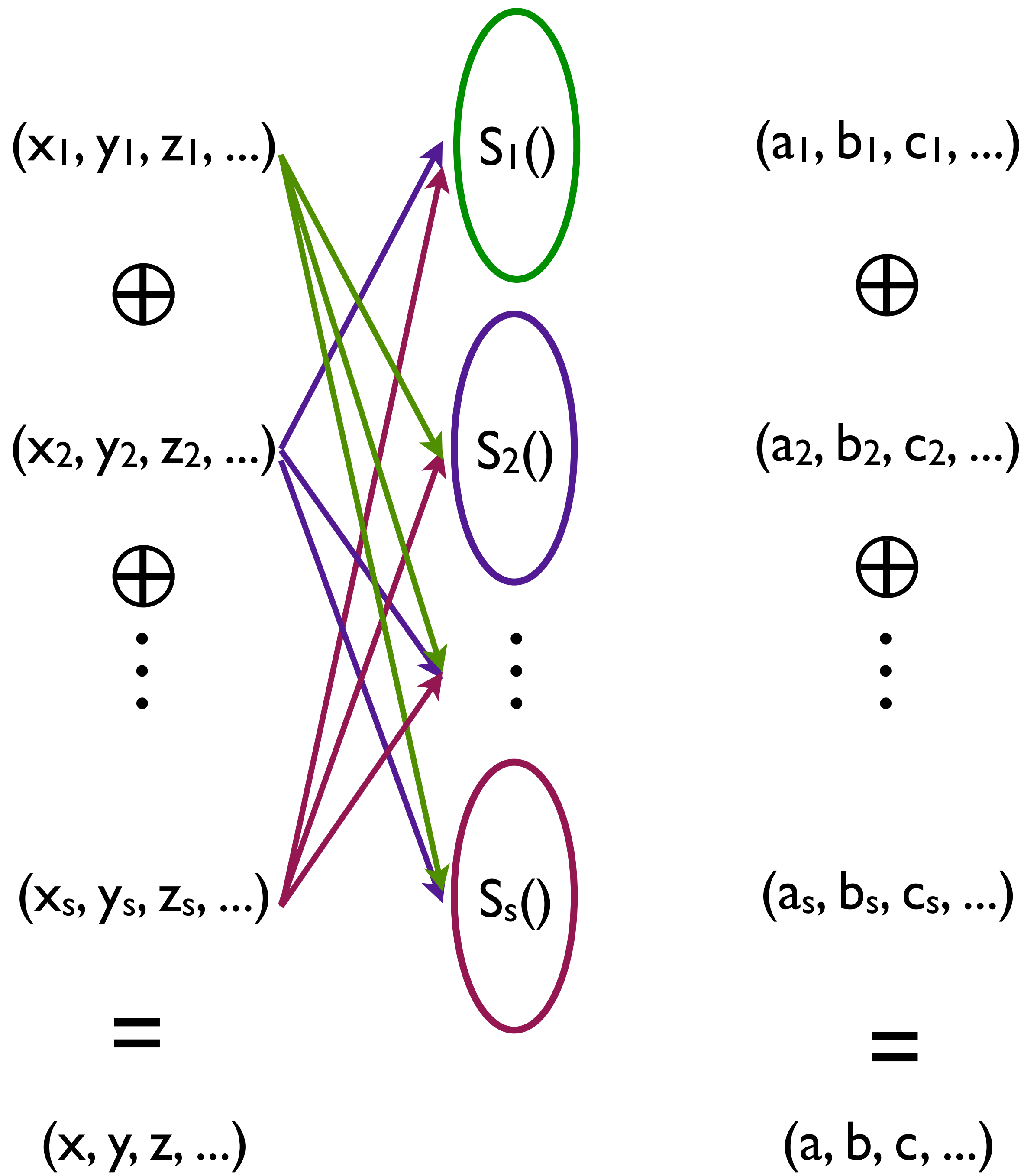
Non-completeness

$$S(x, y, z) \quad = \quad x + yz$$

$$\begin{aligned} S_1 &= x_2 + y_2 z_2 + y_2 z_3 + y_3 z_2 \\ S_2 &= x_3 + y_3 z_3 + y_3 z_1 + y_1 z_3 \\ S_3 &= x_1 + y_1 z_1 + y_1 z_2 + y_2 z_1 \end{aligned}$$

To protect a function with degree *d*, at least *d+1* shares are required

Threshold Implementations



Correct, Non-complete, Uniform

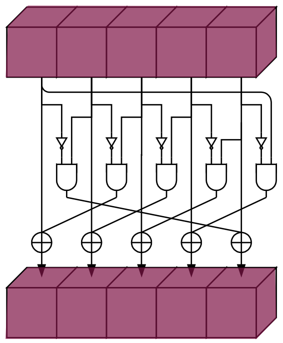
Threshold Implementations

Uniformity

If unshared function is a permutation,
the shared function should also be a permutation

If the masking of x is uniform and the circuit S *is non-complete, then any single component function of* S *does not leak information on x.*

Threshold Implementations



χ function

$$x_i' \leftarrow x_i + (x_{i+1} + 1)\, x_{i+2}$$

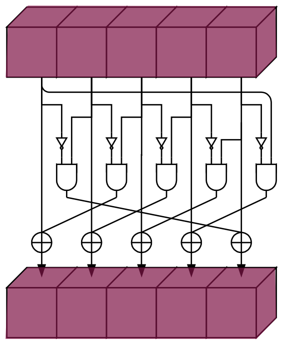
$$A_i' \leftarrow \chi_i'(B,C) \triangleq B_i + (B_{i+1}+1)B_{i+2} + B_{i+1}C_{i+2} + B_{i+2}C_{i+1},$$
$$B_i' \leftarrow \chi_i'(C,A) \triangleq C_i + (C_{i+1}+1)C_{i+2} + C_{i+1}A_{i+2} + C_{i+2}A_{i+1},$$
$$C_i' \leftarrow \chi_i'(A,B) \triangleq A_i + (A_{i+1}+1)A_{i+2} + A_{i+1}B_{i+2} + A_{i+2}B_{i+1}.$$

Not uniform

1. Inject fresh randomness to preserve uniformity
2. Find a uniform sharing

Threshold Implementations

χ function



$$x_i' \leftarrow x_i + (x_{i+1} + 1)\, x_{i+2}$$

$$A_i' \leftarrow \chi_i'(B,C) \triangleq B_i + (B_{i+1}+1)B_{i+2} + B_{i+1}C_{i+2} + B_{i+2}C_{i+1},$$
$$B_i' \leftarrow \chi_i'(C,A) \triangleq C_i + (C_{i+1}+1)C_{i+2} + C_{i+1}A_{i+2} + C_{i+2}A_{i+1},$$
$$C_i' \leftarrow \chi_i'(A,B) \triangleq A_i + (A_{i+1}+1)A_{i+2} + A_{i+1}B_{i+2} + A_{i+2}B_{i+1}.$$

Not uniform

1. Inject fresh randomness to preserve uniformity
2. Find a uniform sharing

Threshold Implementations

χ function

Fresh Randomness

- Standard masking [MPLPW'11]

$$
\begin{aligned}
A'_i &\leftarrow \chi'_i(B, C) + P_i + S_i, \\
B'_i &\leftarrow \chi'_i(C, A) + P_i, \\
C'_i &\leftarrow \chi'_i(A, B) + S_i,
\end{aligned}
$$

- 2 random bits per state bit
- One needs 3200 bits per round

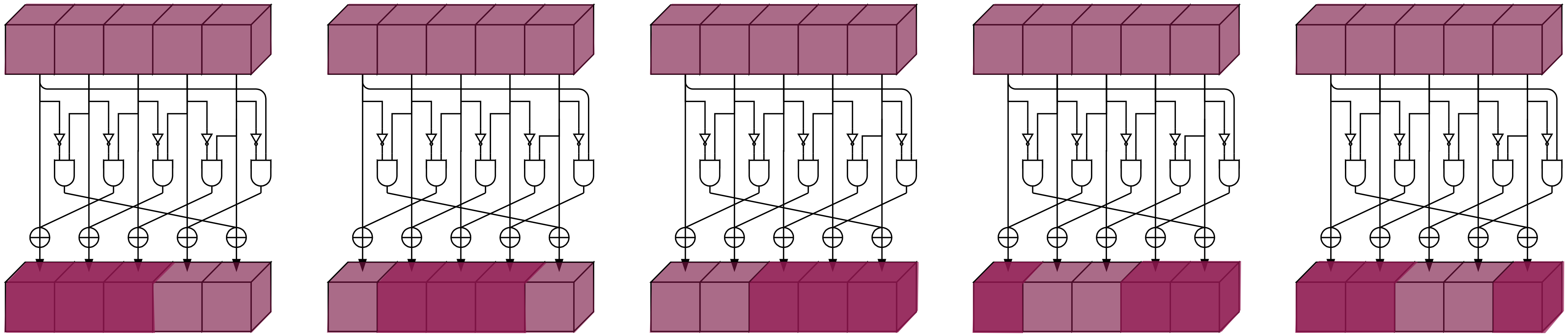
Not feasible in practice

Threshold Implementations

χ function

Fresh Randomness

For any consecutive 3 positions, the output shares are uniform



- 4 random bits per each χ operation
- 1280 bits per round

Still too much in practice

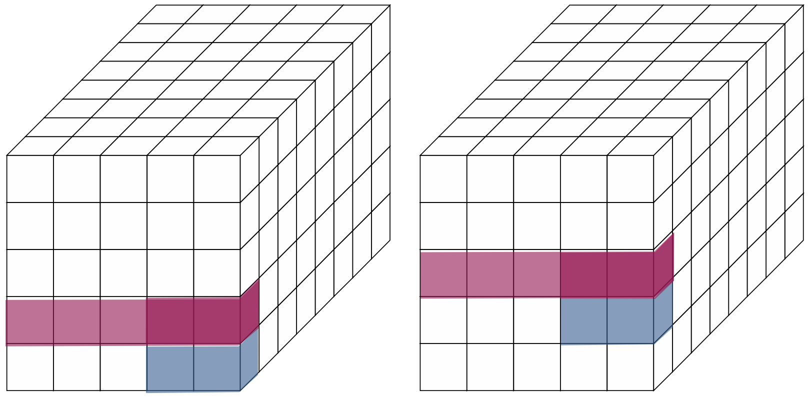
Threshold Implementations

χ function

Fresh Randomness

Make the output row *j+1* uniform by using input from row *j*

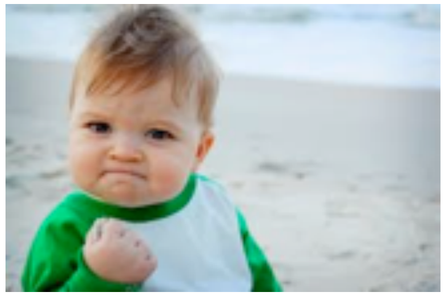
$$A_i'^{(j)} \leftarrow \chi_i'(B^{(j)}, C^{(j)}) + A_i^{(j-1)} + B_i^{(j-1)},$$
$$B_i'^{(j)} \leftarrow \chi_i'(C^{(j)}, A^{(j)}) + A_i^{(j-1)},$$
$$C_i'^{(j)} \leftarrow \chi_i'(A^{(j)}, B^{(j)}) + B_i^{(j-1)},$$



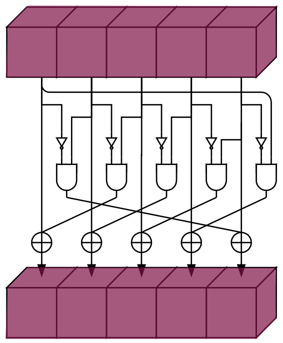
To break circular dependency, use fresh masks in one row

Detailed proof in the paper

- 4 random bits per round
- 96 bits in total for 24 rounds of KECCAK-*f*



Threshold Implementations



χ function

$$x_i' \leftarrow x_i + (x_{i+1} + 1)\, x_{i+2}$$

$$A_i' \leftarrow \chi_i'(B,C) \triangleq B_i + (B_{i+1}+1)B_{i+2} + B_{i+1}C_{i+2} + B_{i+2}C_{i+1},$$
$$B_i' \leftarrow \chi_i'(C,A) \triangleq C_i + (C_{i+1}+1)C_{i+2} + C_{i+1}A_{i+2} + C_{i+2}A_{i+1},$$
$$C_i' \leftarrow \chi_i'(A,B) \triangleq A_i + (A_{i+1}+1)A_{i+2} + A_{i+1}B_{i+2} + A_{i+2}B_{i+1}.$$

Not uniform

1. Inject fresh randomness to preserve uniformity
2. Find a uniform sharing

Threshold Implementations

χ function
Uniform Sharing

- ✗ With 3 shares with different sharing functions

  i.e. with correction terms

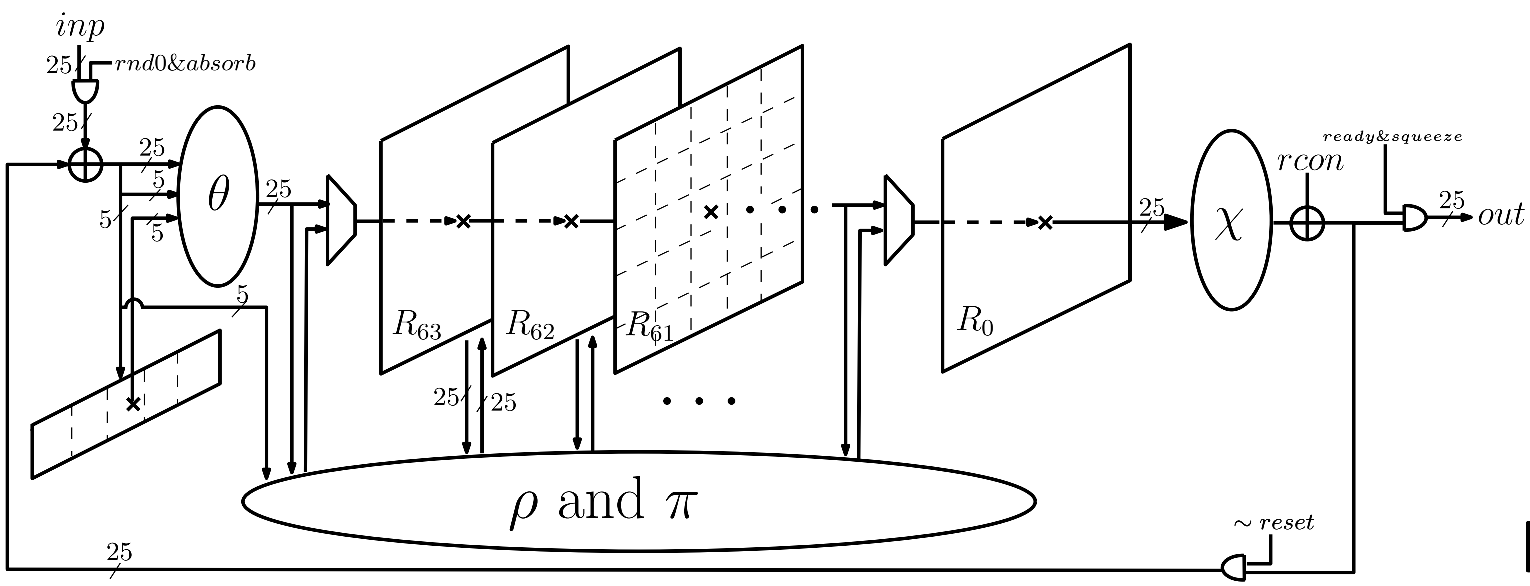
- ✓ With more shares

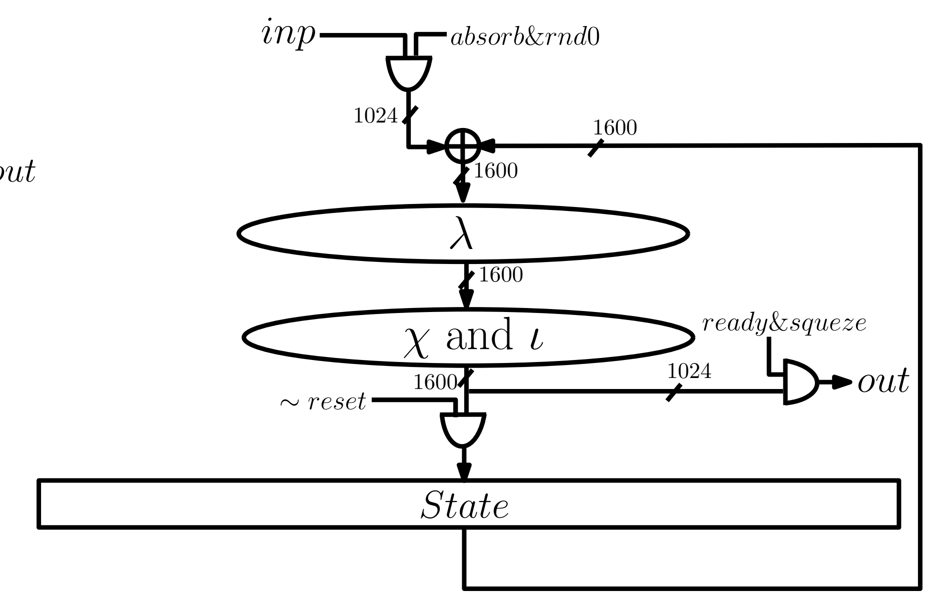
$$
\begin{aligned}
A'_i &\leftarrow B_i + B_{i+2} + ((B_{i+1} + C_{i+1} + D_{i+1})(B_{i+2} + C_{i+2} + D_{i+2})),\\
B'_i &\leftarrow C_i + C_{i+2} + (A_{i+1}(C_{i+2} + D_{i+2}) + A_{i+2}(C_{i+1} + D_{i+1}) + A_{i+1}A_{i+2}),\\
C'_i &\leftarrow D_i + D_{i+2} + (A_{i+1}B_{i+2} + A_{i+2}B_{i+1}),\\
D'_i &\leftarrow A_i + A_{i+2}, \qquad\qquad i = 0, 1, 2, 4
\end{aligned}
$$

$$
\begin{aligned}
A'_3 &\leftarrow B_3 + B_0 + C_0 + D_0 + ((B_4 + C_4 + D_4)(B_0 + C_0 + D_0)),\\
B'_3 &\leftarrow C_3 + A_0 + (A_4(C_0 + D_0) + A_0(C_4 + D_4) + A_0A_4),\\
C'_3 &\leftarrow D_3 + (A_4B_0 + A_0B_4),\\
D'_3 &\leftarrow A_3.
\end{aligned}
$$

Threshold Implementations

KECCAK-f function

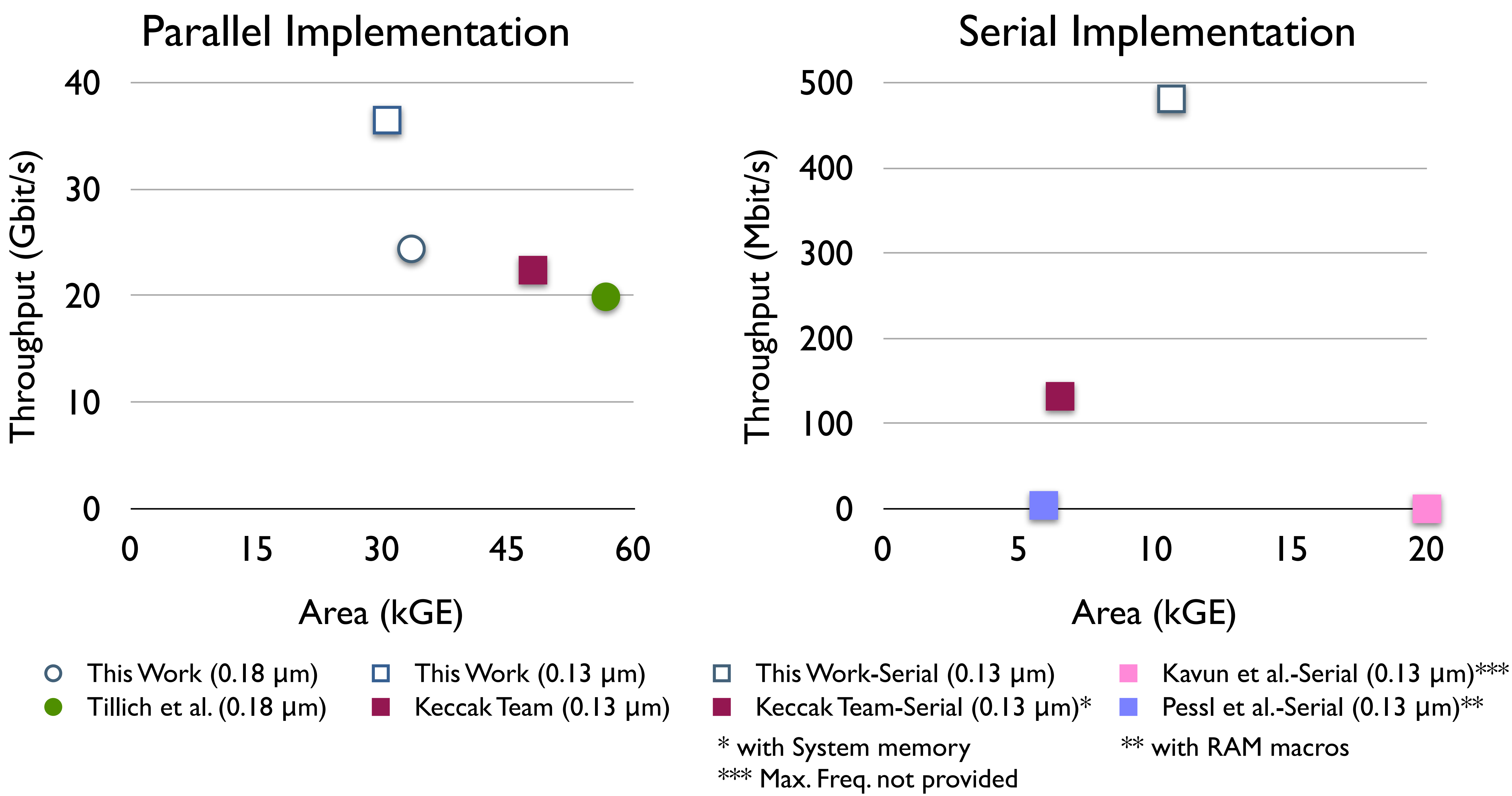




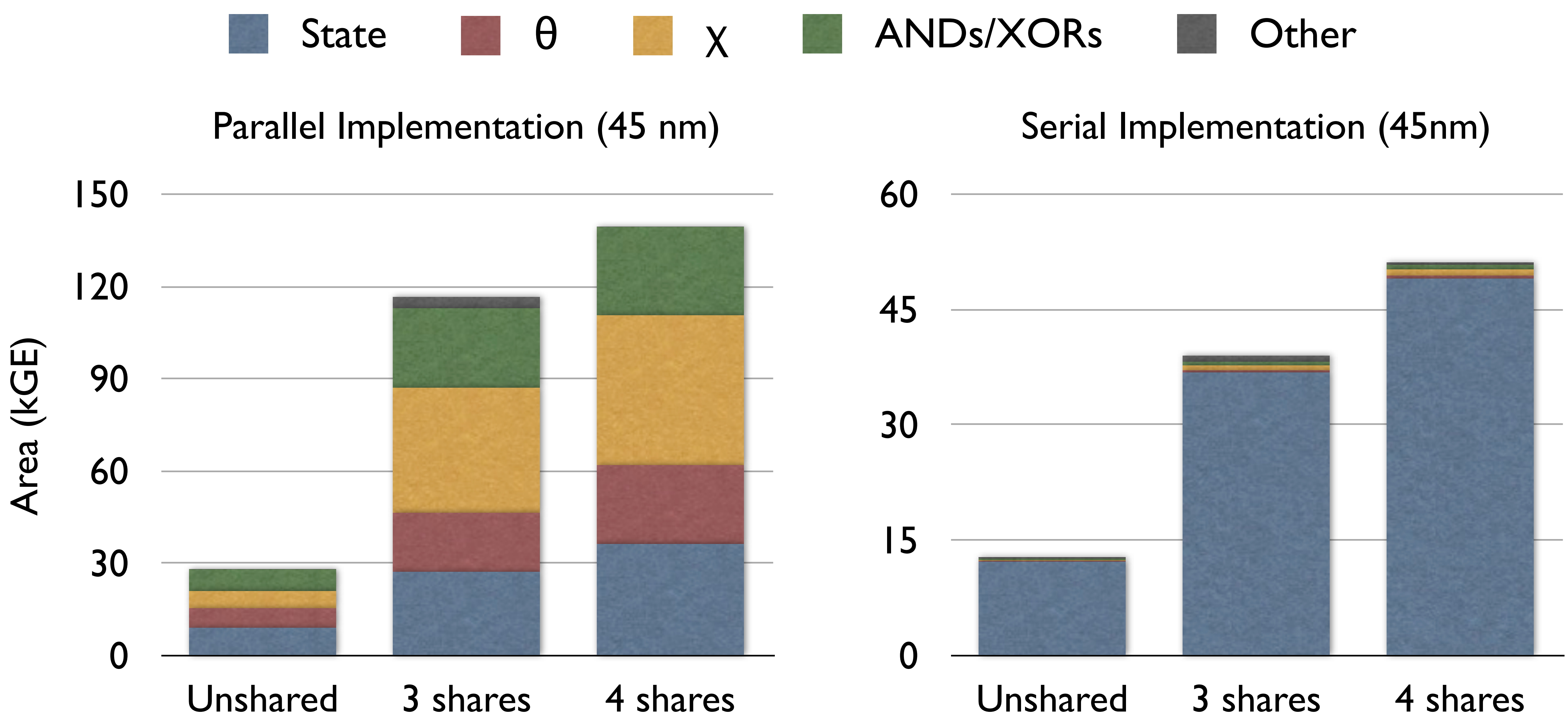
- Two threshold implementations (3 and 4 shares)
- Provide results in three different technologies

FARADAY (based on UMC 0,18µm and 0,13µm) and NANGATE (45 nm)

Performance



Performance



Conclusion

- Threshold implementation fulfilling all the properties
- 4-share TI without extra randomness
- 3-share TI with only 4 bits of randomness per round
- ? 3-shares TI without extra randomness
- ? Observable difference between non-uniform and uniform TI in practice

Thank you!

HW

Threshold Implementation

Share

λ

θ

Serial

KECCAK

Slice

?

Uniform

3

MHz

χ

Linear

Mask

Parallel

Non-linear

kGE

Re-masking

4