Radboud University

Changing of the Guards

Joan Daemen

CHES 2017

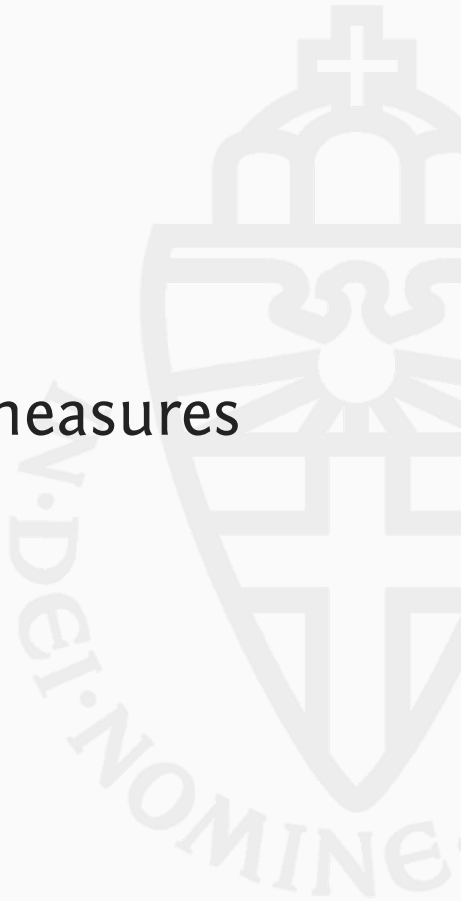
Taipei, September 26, 2017

Radboud University

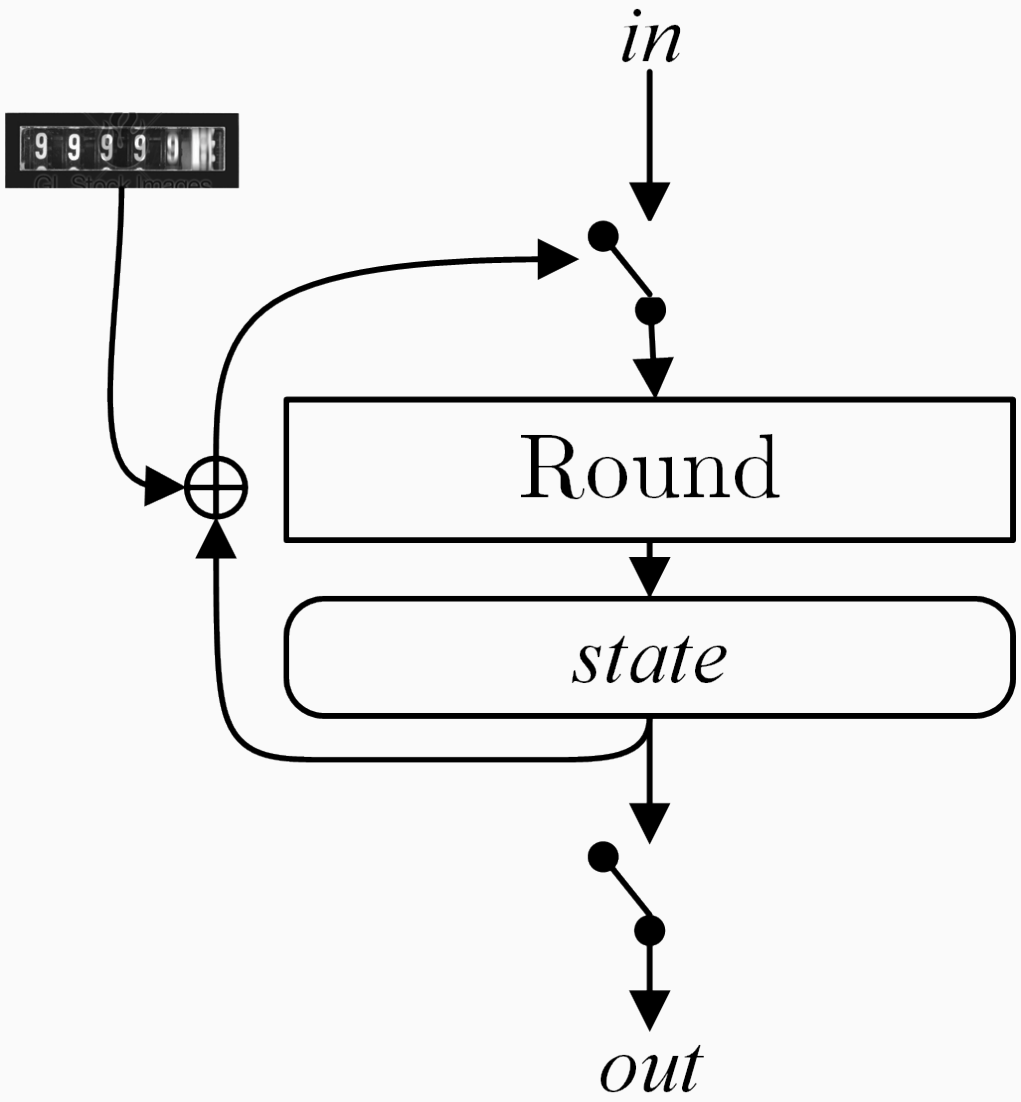
STMicroelectronics

Disclaimer

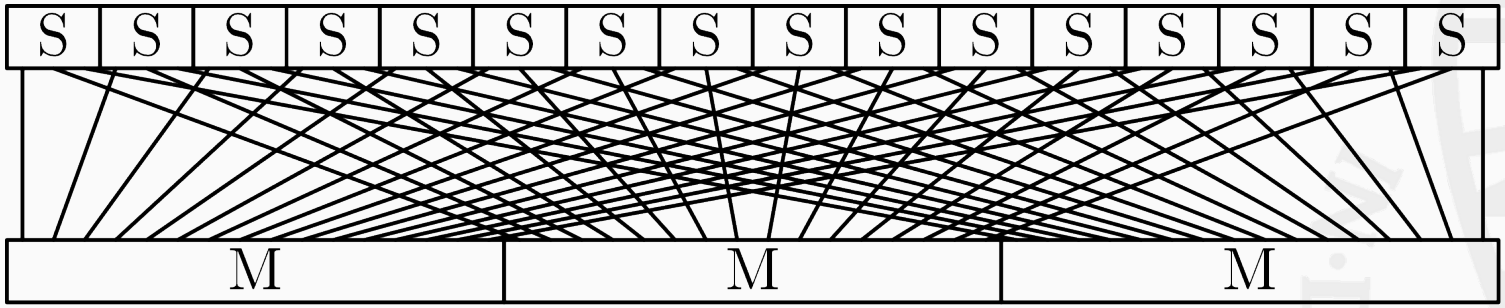
This is not a talk about higher-order countermeasures



Iterative cryptographic permutation

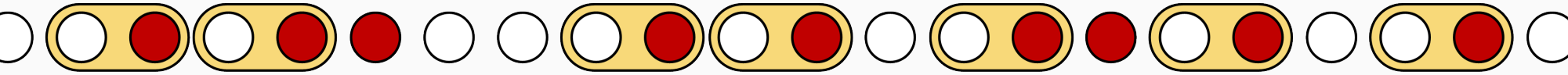


Three-stage round function: wide trail

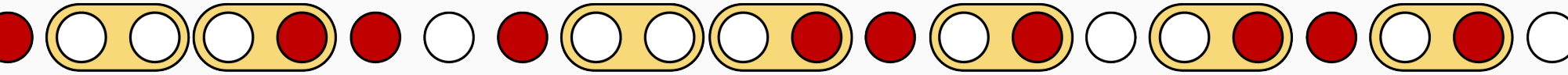


Nonlinear layer χ

Nonlinear layer χ



Nonlinear layer χ



Nonlinear layer χ

Nonlinear layer χ

```
X[i] ^= (~X[i+1]) & X[i+2]
```

Nonlinear layer χ

```
X[i] ^= (~X[i+1]) & X[i+2]
```

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

Nonlinear layer χ

```
X[i] ^= (~X[i+1]) & X[i+2]
```

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

Invertible for odd length n

Nonlinear layer χ

```
X[i] ^= (~X[i+1]) & X[i+2]
```

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

Invertible for odd length n

Used in KETJE, KEYAK, KECCAK ($n = 5$)

Nonlinear layer χ

```
X[i] ^= (~X[i+1]) & X[i+2]
```

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

Invertible for odd length n

Used in KETJE, KEYAK, KECCAK ($n = 5$)

RADIOGATUN ($n = 19$), PANAMA ($n = 17$), BASEKING,

3-WAY ($n = 3$), SUBTERRANEAN, CELLHASH ($n = 257$)

[Daemen, Govaerts, Vandewalle, WIC Benelux 1991]

Masking of χ as DPA/DEMA countermeasure

$$x_0 \leftarrow x_0 + (x_1 + 1)x_2$$

Masking of χ as DPA/DEMA countermeasure

$$x_0 \leftarrow x_0 + (x_1 + 1)x_2$$

$$a_i + b_i = x_i \text{ with } a_i \text{ random}$$

Masking of χ as DPA/DEMA countermeasure

$$x_0 \leftarrow x_0 + (x_1 + 1)x_2$$

$$a_i + b_i = x_i \text{ with } a_i \text{ random}$$

$$\begin{aligned} a_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1 b_2 \\ b_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1 a_2 \end{aligned}$$

[Daemen, Peeters, Van Assche, FSE 2000]

χ': a three-share masking of χ

$$x_0 \leftarrow x_0 + (x_1 + 1)x_2$$

$$a_i + b_i + c_i = x_i \text{ with } a_i \text{ and } b_i \text{ random}$$

χ': a three-share masking of χ

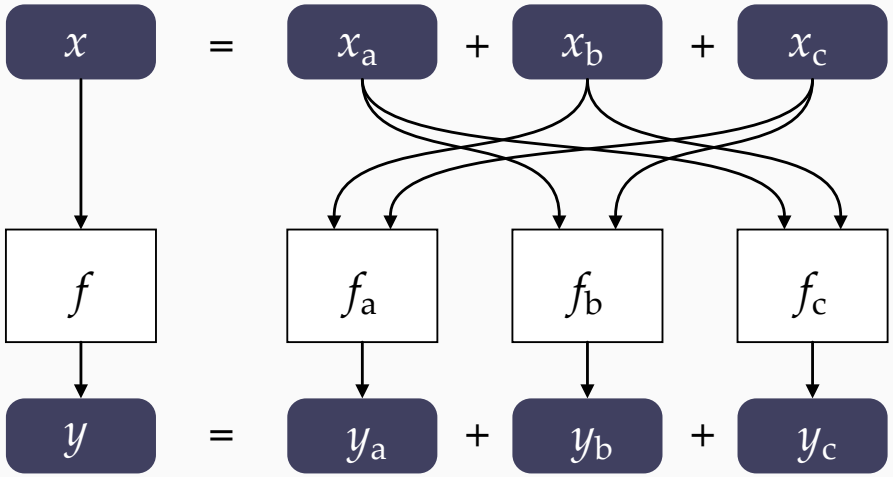
$$x_0 \leftarrow x_0 + (x_1 + 1)x_2$$

$$a_i + b_i + c_i = x_i \text{ with } a_i \text{ and } b_i \text{ random}$$

$$\begin{aligned} a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\ b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\ c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1 \end{aligned}$$

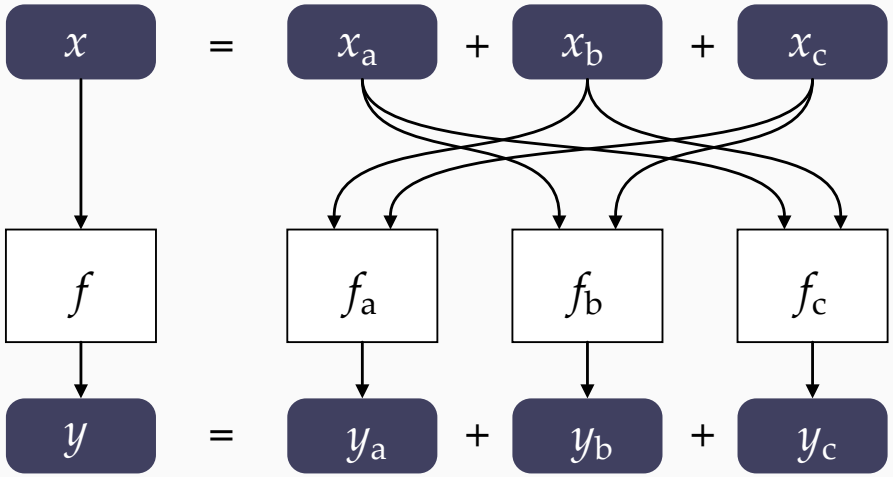
[Bertoni, Daemen, Peeters, Van Assche, 2nd SHA-3, 2010]

Threshold masking schemes [Nikova, Rijmen, Rechberger, Schläffer, '06 - '08]



Scheme at the right computes f securely against 1st order DPA if:

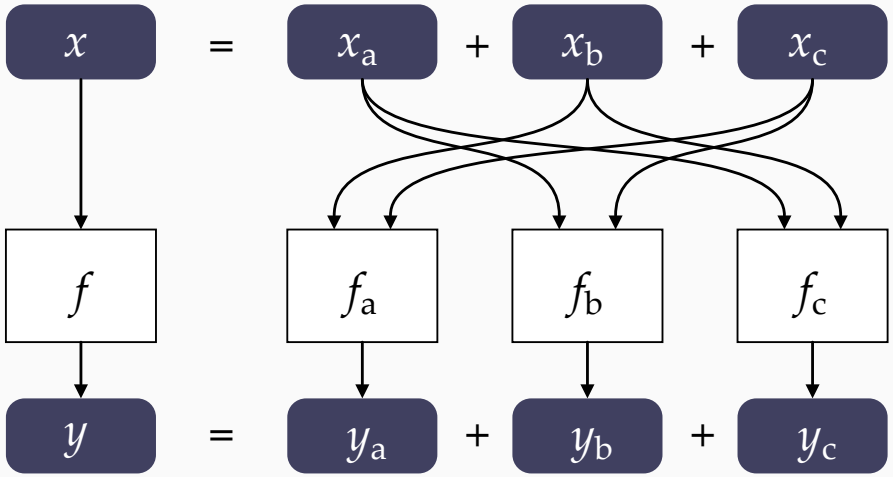
Threshold masking schemes [Nikova, Rijmen, Rechberger, Schläffer, '06 - '08]



Scheme at the right computes f securely against 1st order DPA if:

- (f_a, f_b, f_c) is a correct sharing of f

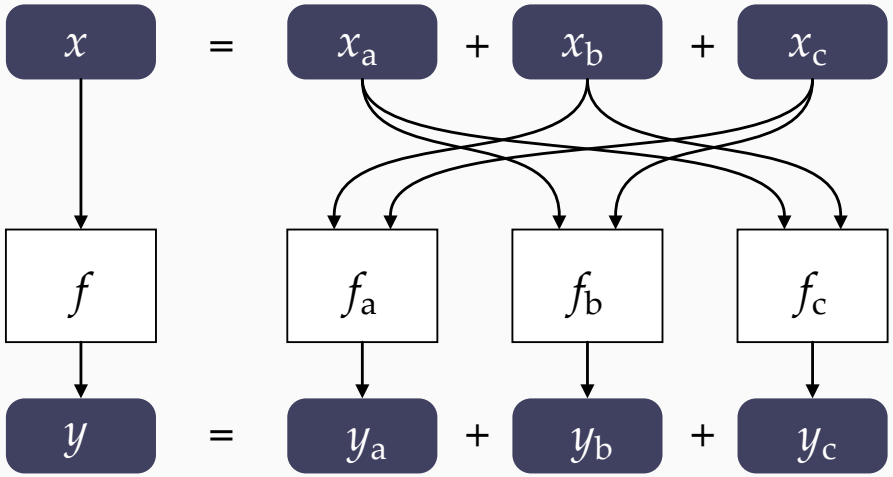
Threshold masking schemes [Nikova, Rijmen, Rechberger, Schläffer, '06 - '08]



Scheme at the right computes f securely against 1st order DPA if:

- (f_a, f_b, f_c) is a correct sharing of f
- (f_a, f_b, f_c) is incomplete: requires # shares $\geq d + 1$

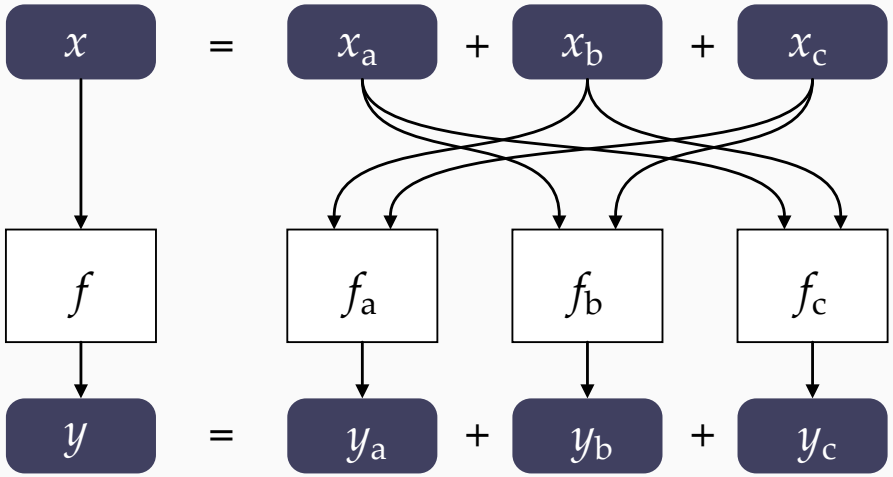
Threshold masking schemes [Nikova, Rijmen, Rechberger, Schläffer, '06 - '08]



Scheme at the right computes f securely against 1st order DPA if:

- (f_a, f_b, f_c) is a correct sharing of f
- (f_a, f_b, f_c) is incomplete: requires # shares $\geq d + 1$
- (x_a, x_b, x_c) is a uniform sharing of x:
  - all values (x_a, x_b, x_c) with $x_a + x_b + x_c = x$ equiprobable

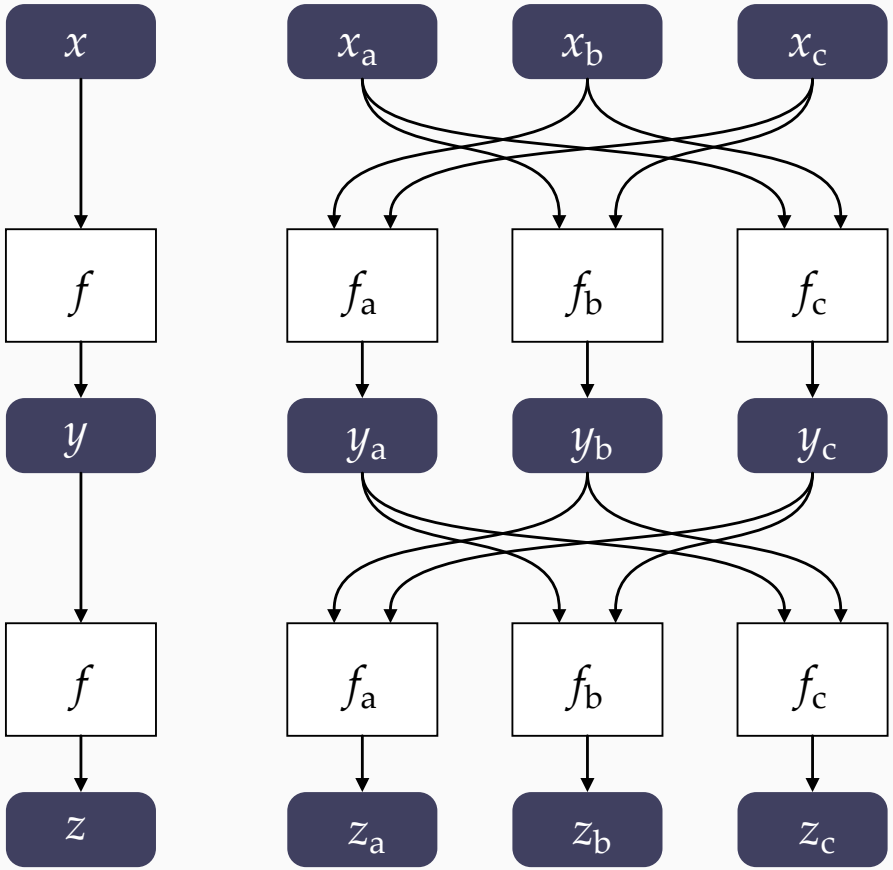
Threshold masking schemes [Nikova, Rijmen, Rechberger, Schläffer, '06 - '08]



Scheme at the right computes f securely against 1st order DPA if:

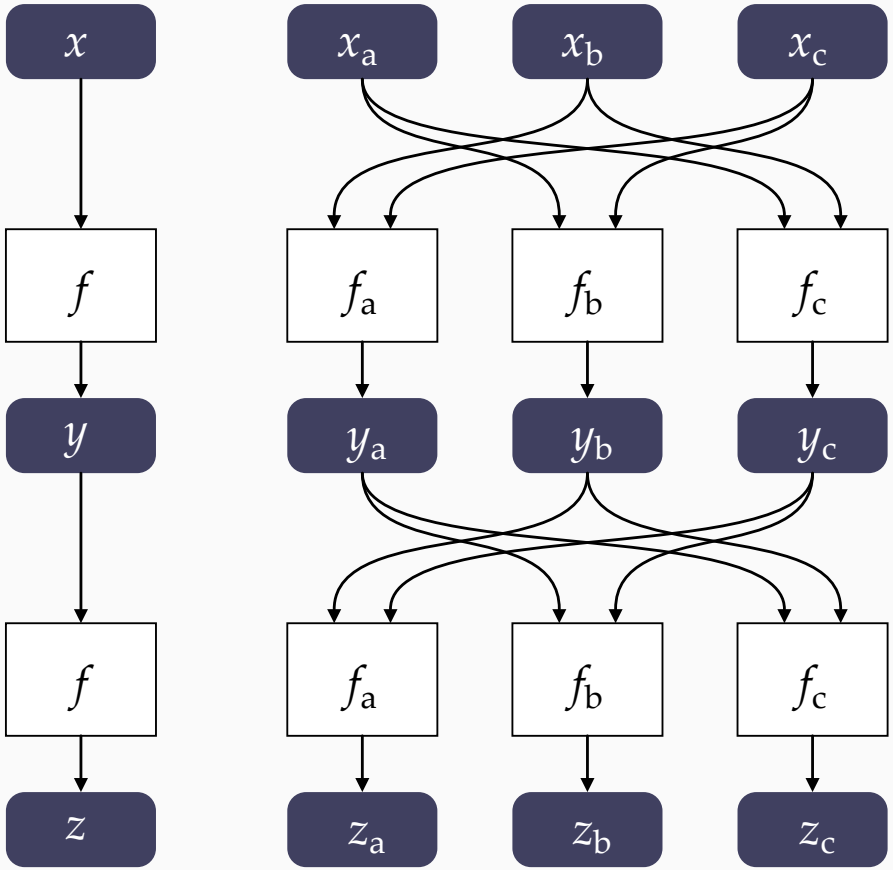
- (f_a, f_b, f_c) is a correct sharing of f
- (f_a, f_b, f_c) is incomplete: requires # shares $\geq d + 1$
- (x_a, x_b, x_c) is a uniform sharing of x:
  - all values (x_a, x_b, x_c) with $x_a + x_b + x_c = x$ equiprobable
  - $x = 0 : (x_a, x_b, x_c) \in \{(0,0,0)(1,1,0)(1,0,1)(0,1,1)\}$
  - $x = 1 : (x_a, x_b, x_c) \in \{(1,1,1)(0,0,1)(0,1,0)(1,0,0)\}$

Uniformity of a threshold masking scheme



- Sharing (f_a, f_b, f_c) of f is called *uniform* if it preserves uniformity

Uniformity of a threshold masking scheme



- Sharing (f_a, f_b, f_c) of f is called *uniform* if it preserves uniformity
- If f is invertible, for (f_a, f_b, f_c) uniformity $=$ invertibility

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct?

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete?

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!
- Uniform?

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!
- Uniform? Yes!

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!
- Uniform? Yes! ...but wait

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!
- Uniform? Yes! ...but wait ...we may have a problem here

Back to χ'

$$
\begin{aligned}
a_0 &\leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1 \\
b_0 &\leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1 \\
c_0 &\leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1
\end{aligned}
$$

Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!
- Uniform? Yes! ...but wait ...we may have a problem here it isn't

Back to χ'

$$a_0 \leftarrow b_0 + (b_1 + 1)b_2 + b_1c_2 + b_2c_1$$
$$b_0 \leftarrow c_0 + (c_1 + 1)c_2 + c_1a_2 + c_2a_1$$
$$c_0 \leftarrow a_0 + (a_1 + 1)a_2 + a_1b_2 + a_2b_1$$

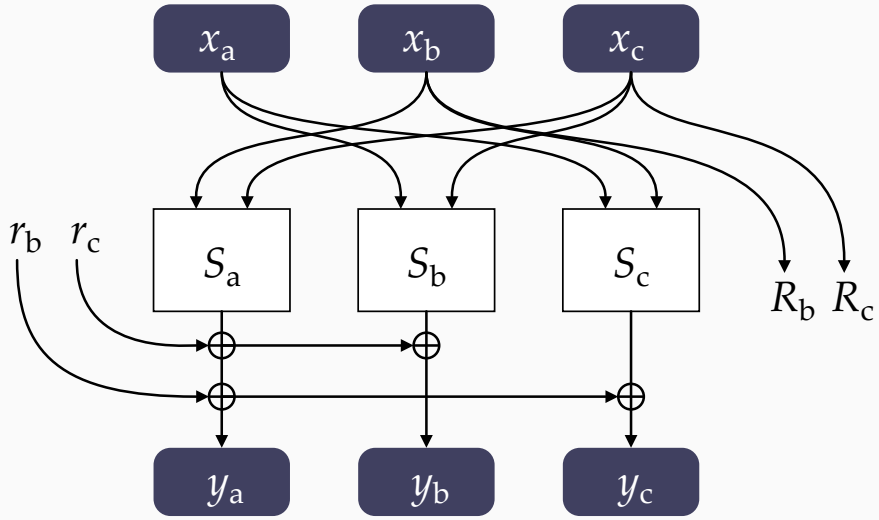
Is this a secure threshold masking scheme of χ?

- Correct? Yes!
- Incomplete? Yes!
- Uniform? Yes! ...but wait ...we may have a problem here it isn't

In general, for many S-boxes:

- no uniform $d + 1$-share threshold schemes are known
- it is an active research area to find the best compromise

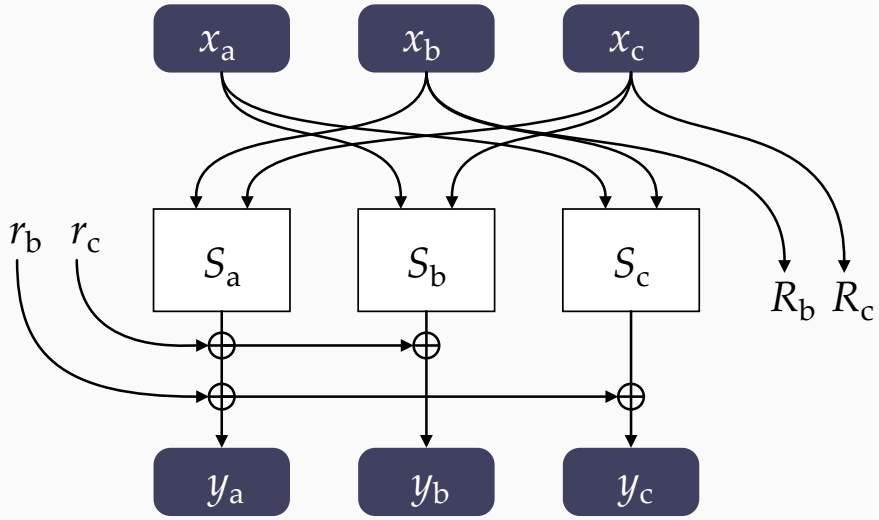
An out-of-the-box approach to achieving uniformity



(1) Build (S_a, S_b, S_c): a correct and incomplete sharing of S

- straightforward and generalizes to $d+1$ shares for $d > 2$

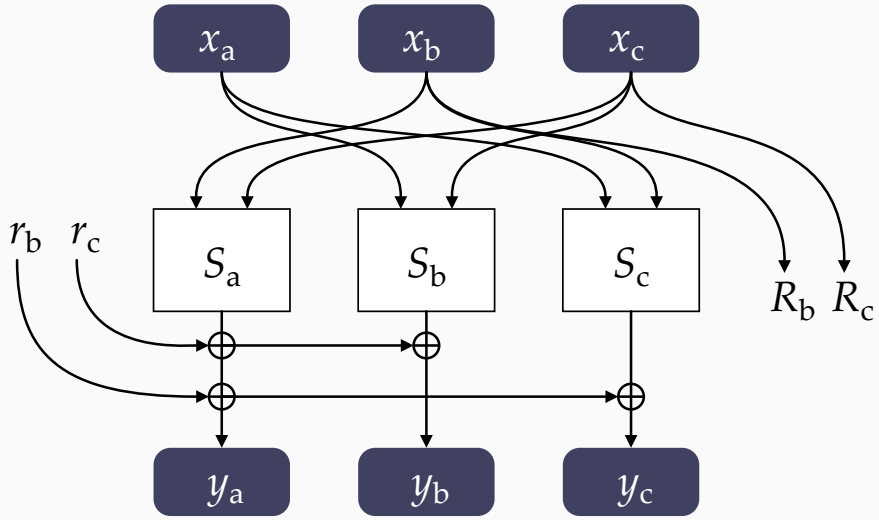
An out-of-the-box approach to achieving uniformity



(1) Build (S_a, S_b, S_c): a correct and incomplete sharing of S
   - straightforward and generalizes to $d+1$ shares for $d > 2$

(2) Mask output with (r_b, r_c) that has uniform distribution

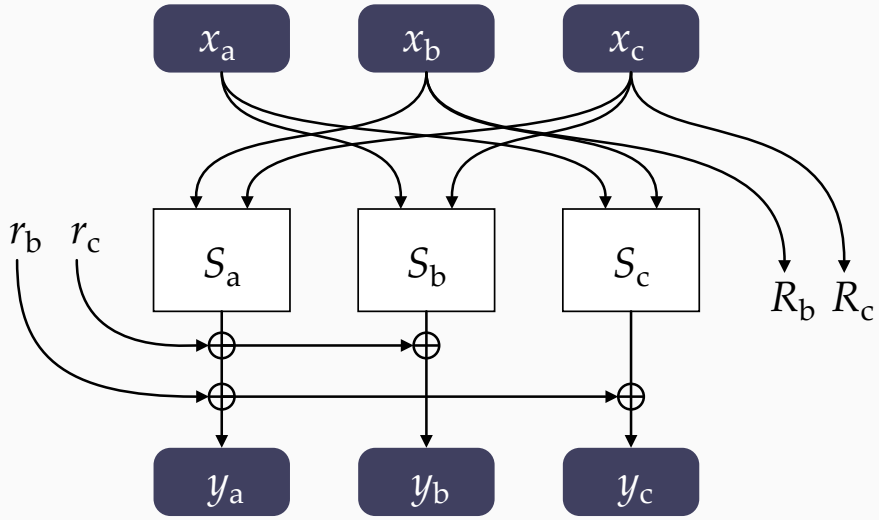
An out-of-the-box approach to achieving uniformity



(1) Build (S_a, S_b, S_c): a correct and incomplete sharing of S
- straightforward and generalizes to $d+1$ shares for $d > 2$

(2) Mask output with (r_b, r_c) that has uniform distribution
- correctness and incompleteness are preserved

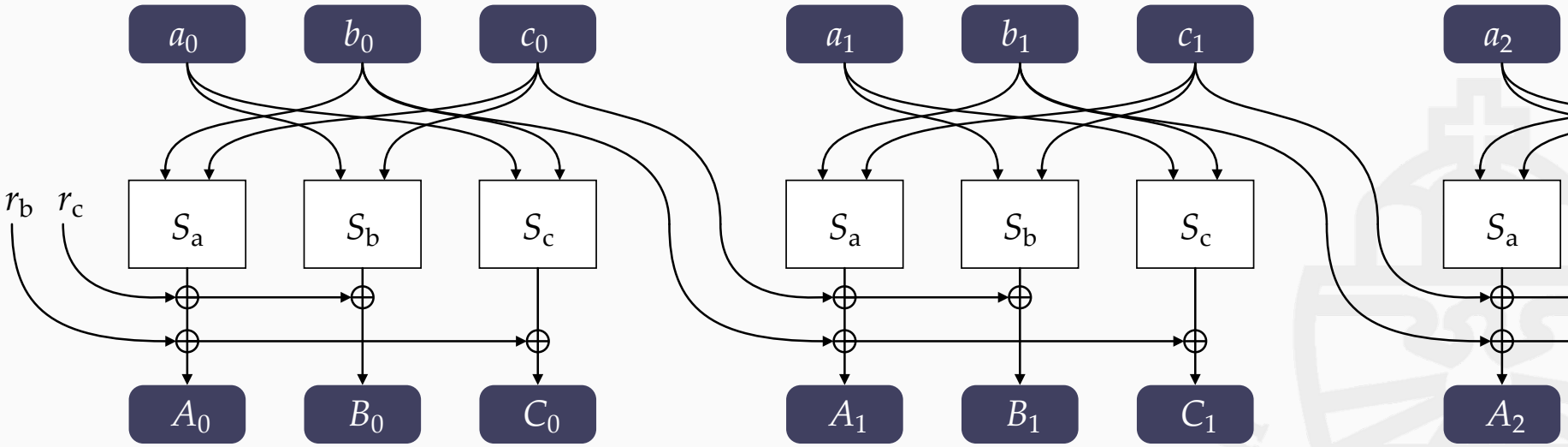
An out-of-the-box approach to achieving uniformity



(1) Build (S_a, S_b, S_c): a correct and incomplete sharing of S
- straightforward and generalizes to $d+1$ shares for $d > 2$

(2) Mask output with (r_b, r_c) that has uniform distribution
- correctness and incompleteness are preserved
- output (y_a, y_b, y_c) becomes uniform sharing of y

An out-of-the-box approach to achieving uniformity

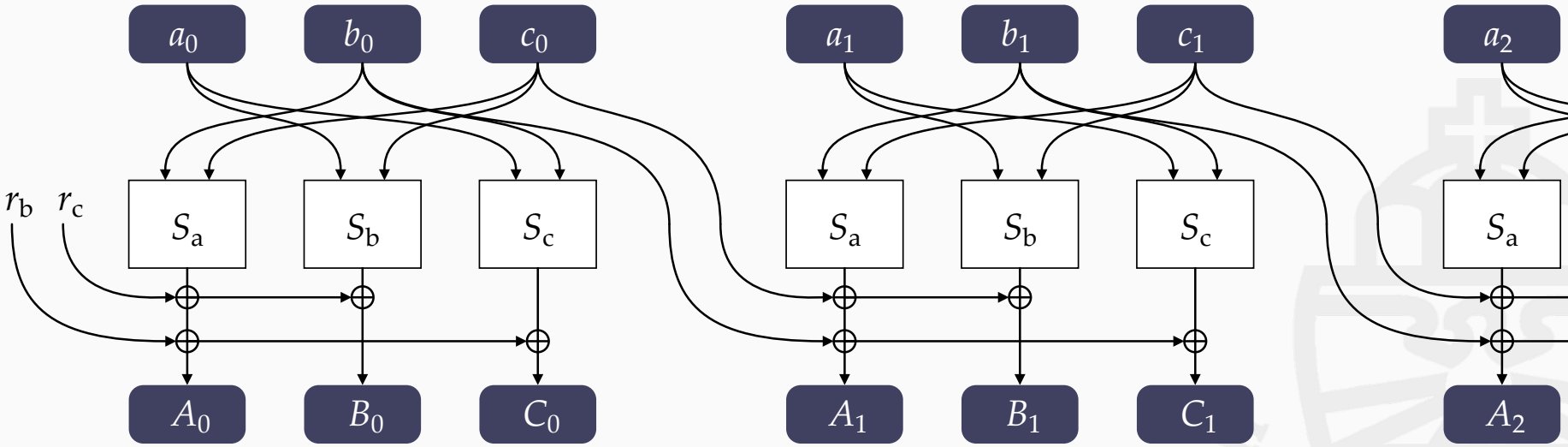


(1) Build (S_a, S_b, S_c): a correct and incomplete sharing of S
- straightforward and generalizes to $d+1$ shares for $d > 2$

(2) Mask output with (r_b, r_c) that has uniform distribution
- correctness and incompleteness are preserved
- output (y_a, y_b, y_c) becomes uniform sharing of y

(3) Chain this

An out-of-the-box approach to achieving uniformity



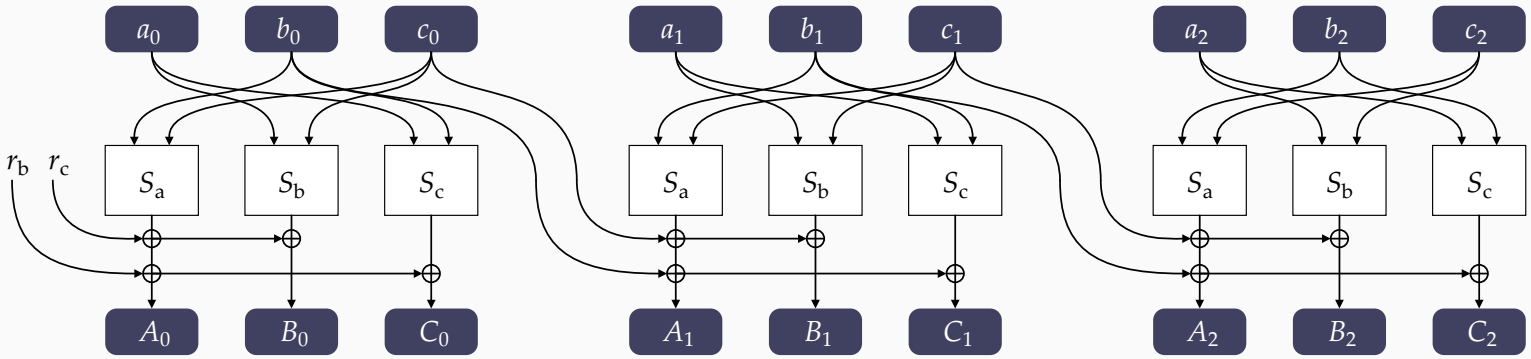
(1) Build (S_a, S_b, S_c): a correct and incomplete sharing of S
- straightforward and generalizes to $d+1$ shares for $d > 2$

(2) Mask output with (r_b, r_c) that has uniform distribution
- correctness and incompleteness are preserved
- output (y_a, y_b, y_c) becomes uniform sharing of y

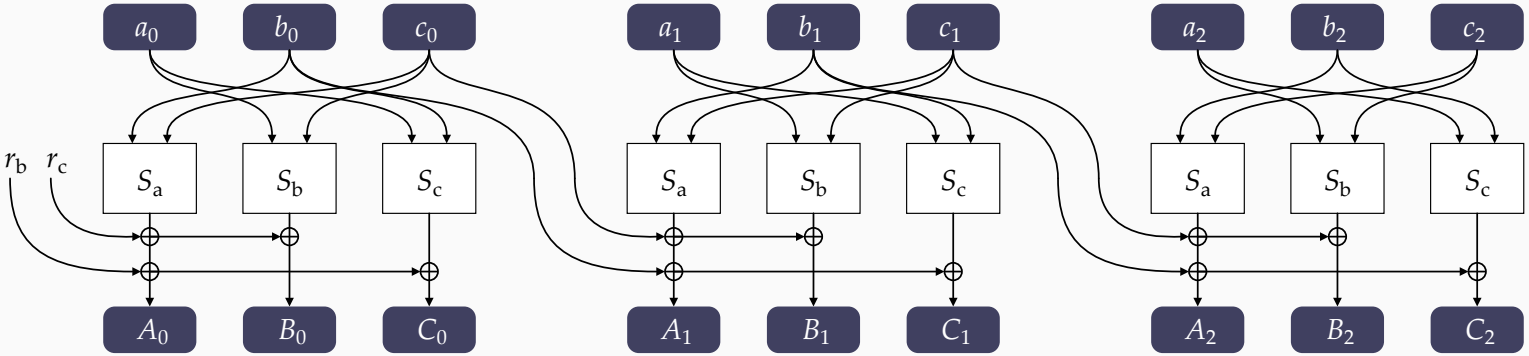
(3) Chain this

But where does leftmost (r_b, r_c) come from?

Attempt 1: injecting fresh randomness

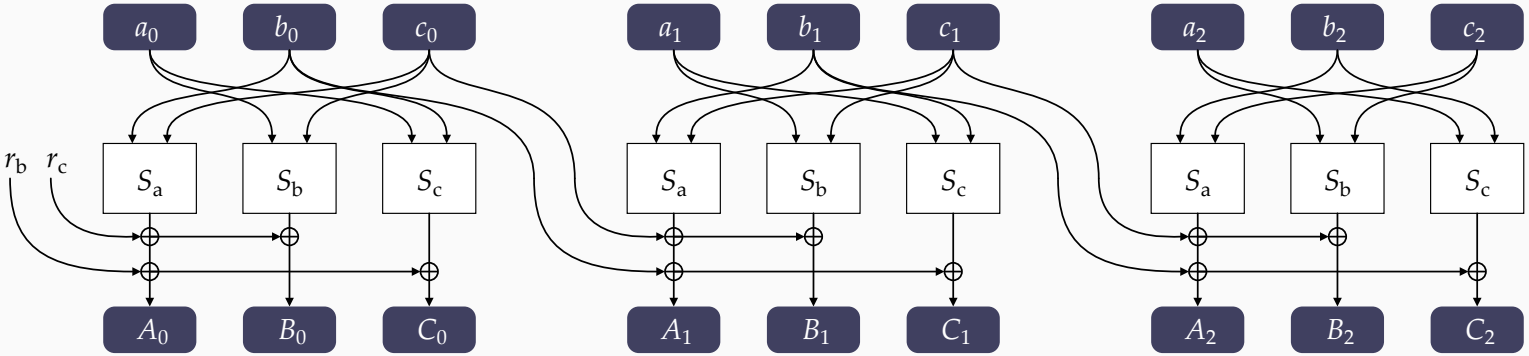


Attempt 1: injecting fresh randomness



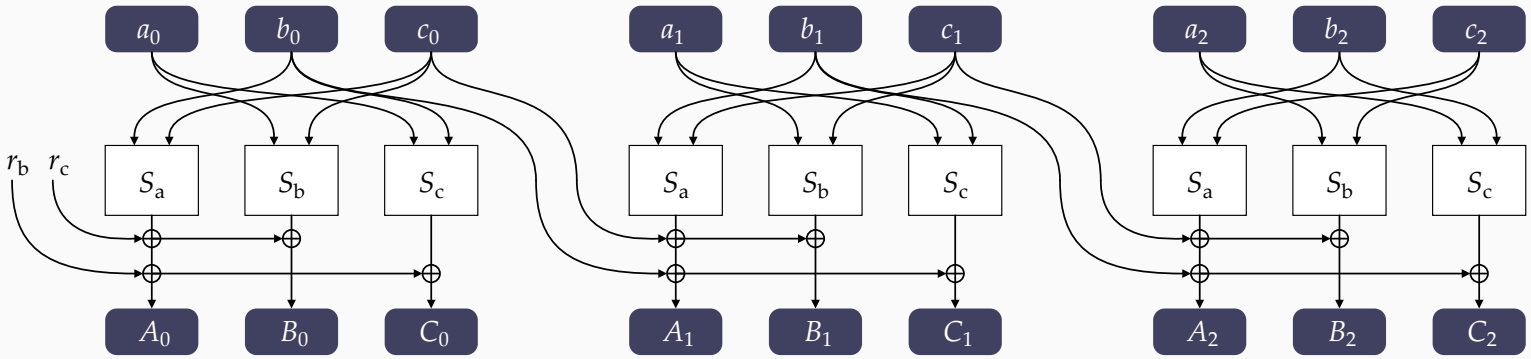
- (r_b, r_c) are generated freshly every round

Attempt 1: injecting fresh randomness



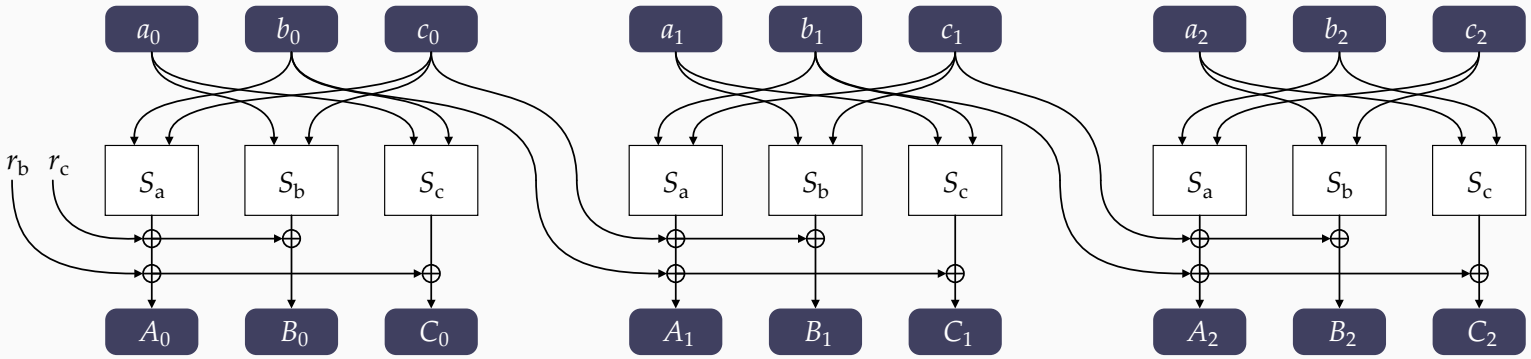
- (r_b, r_c) are generated freshly every round
- For n-bit S-box, this requires $2n$ random bits per round

Attempt 1: injecting fresh randomness



- (r_b, r_c) are generated freshly every round
- For n-bit S-box, this requires $2n$ random bits per round
- Downsides:
  - real-world: requires random generation during operation
  - academic: no uniform sharing is obtained

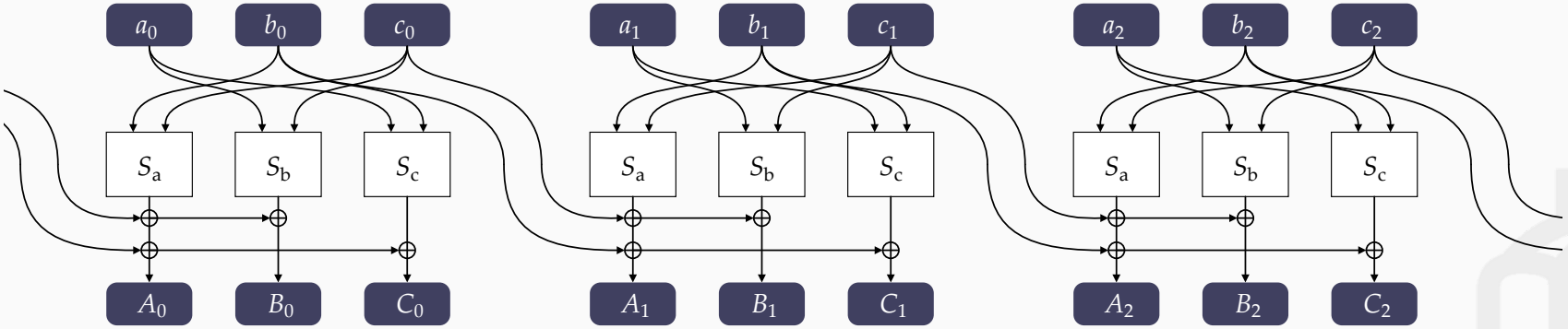
Attempt 1: injecting fresh randomness



- (r_b, r_c) are generated freshly every round
- For n-bit S-box, this requires $2n$ random bits per round
- Downsides:
  - real-world: requires random generation during operation
  - academic: no uniform sharing is obtained

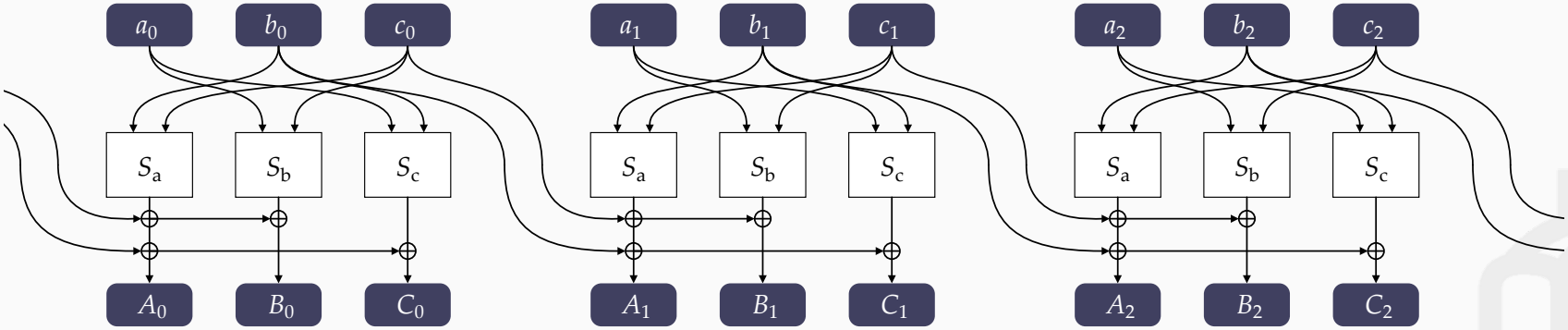
[Bilgin, Daemen, Nikova, Nikov, Rijmen, Van Assche, Cardis '13]

Attempt 2: cycling randomness



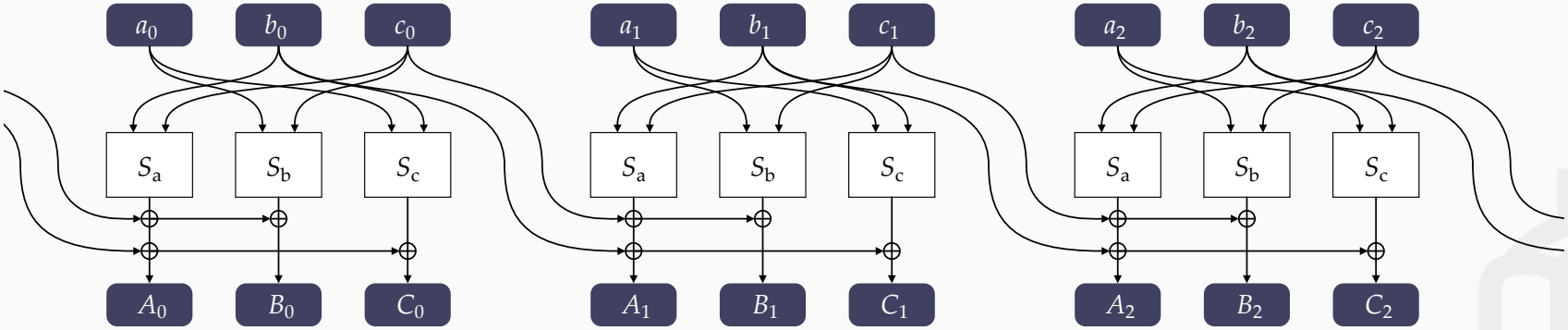
- S-boxes are arranged in circle: $(r_b, r_c) = (R_b, R_c)$

Attempt 2: cycling randomness



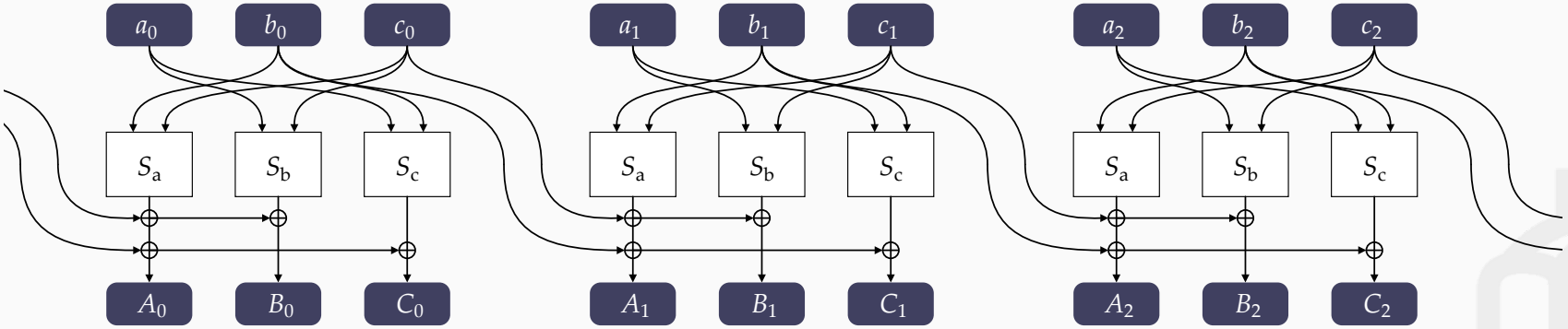
- S-boxes are arranged in circle: $(r_b, r_c) = (R_b, R_c)$
- No more need for generating randomness during operation

Attempt 2: cycling randomness



- S-boxes are arranged in circle: $(r_b, r_c) = (R_b, R_c)$
- No more need for generating randomness during operation
- The remaining amount of non-uniformity is negligible
- Downsides:
  - real-world: hard to explain why that is the case ...
  - academic: it is simply not uniform!

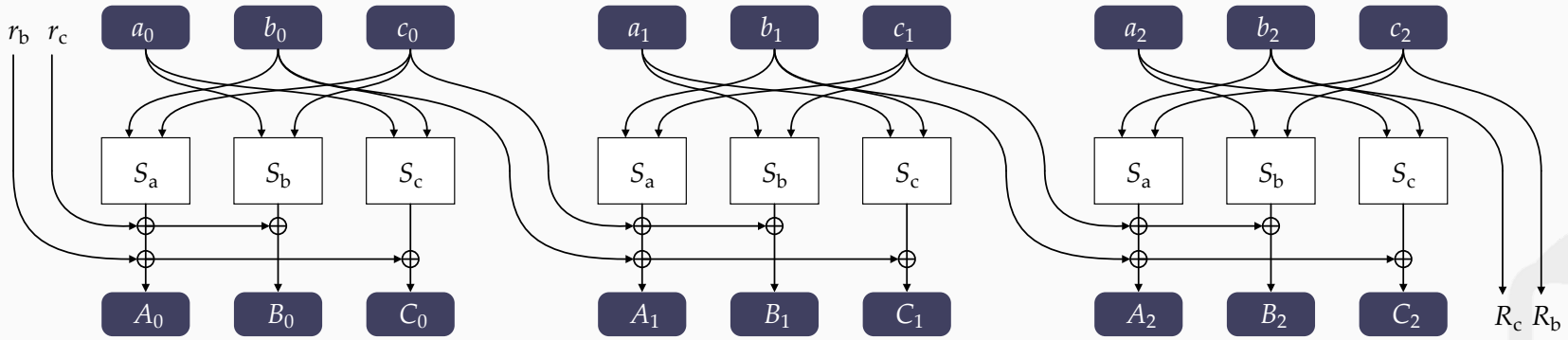
Attempt 2: cycling randomness



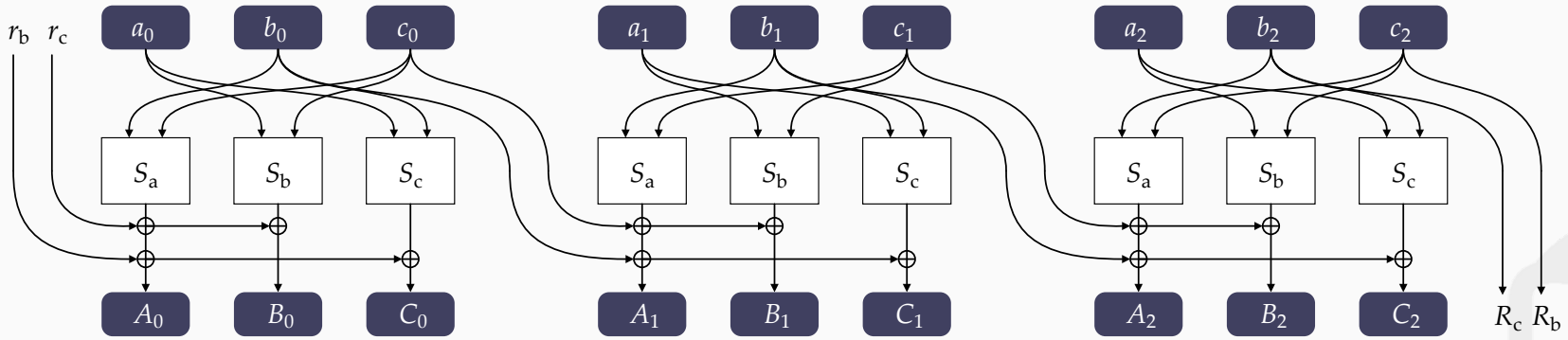
- S-boxes are arranged in circle: $(r_b, r_c) = (R_b, R_c)$
- No more need for generating randomness during operation
- The remaining amount of non-uniformity is negligible
- Downsides:
  - real-world: hard to explain why that is the case ...
  - academic: it is simply not uniform!

I presented this at [Shonan, Sep.'14] [ESC, Jan.'15] [TI Day, May'15]

Attempt 3: recycling randomness

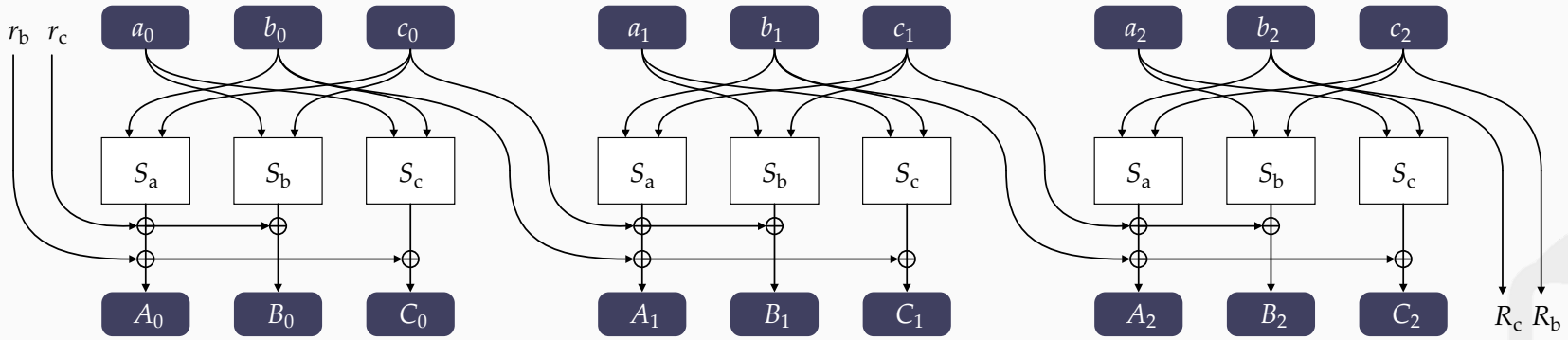


Attempt 3: recycling randomness



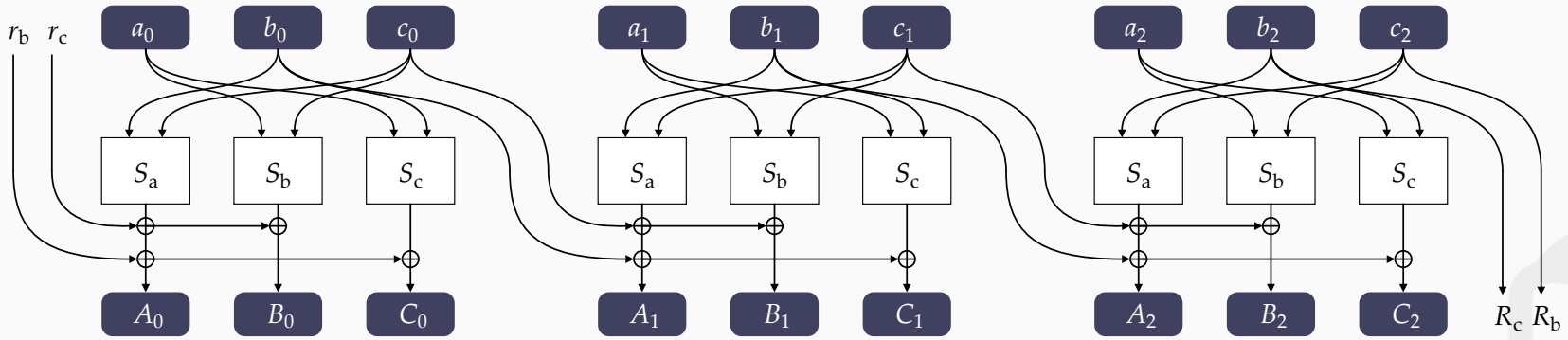
- We make (R_b, R_c) part of the shared state: the Guards
- input Guards (r_b, r_c) are previous-round output Guards (R_b, R_c)

Attempt 3: recycling randomness



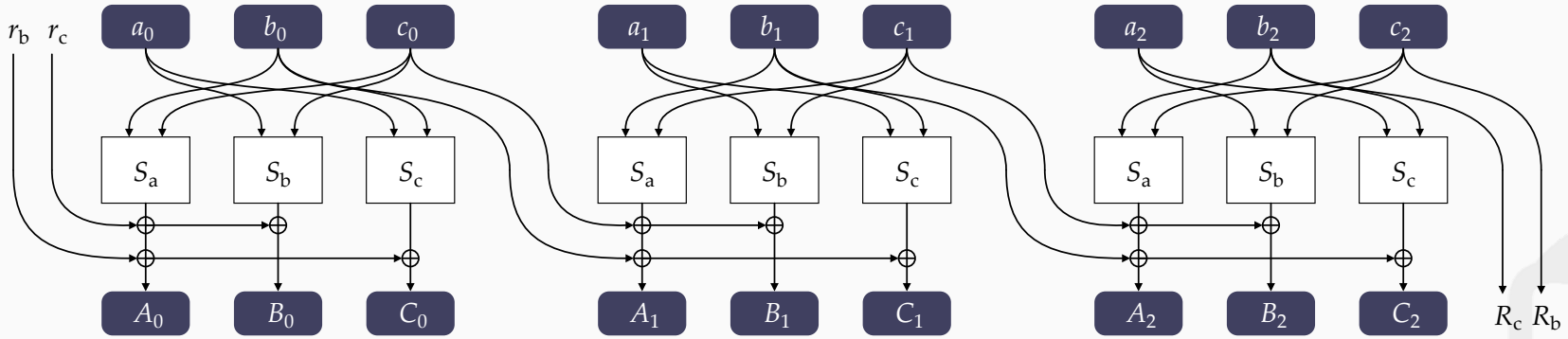
- We make (R_b, R_c) part of the shared state: the Guards
- input Guards (r_b, r_c) are previous-round output Guards (R_b, R_c)
- Achieves uniformity if S-box is invertible

Attempt 3: recycling randomness



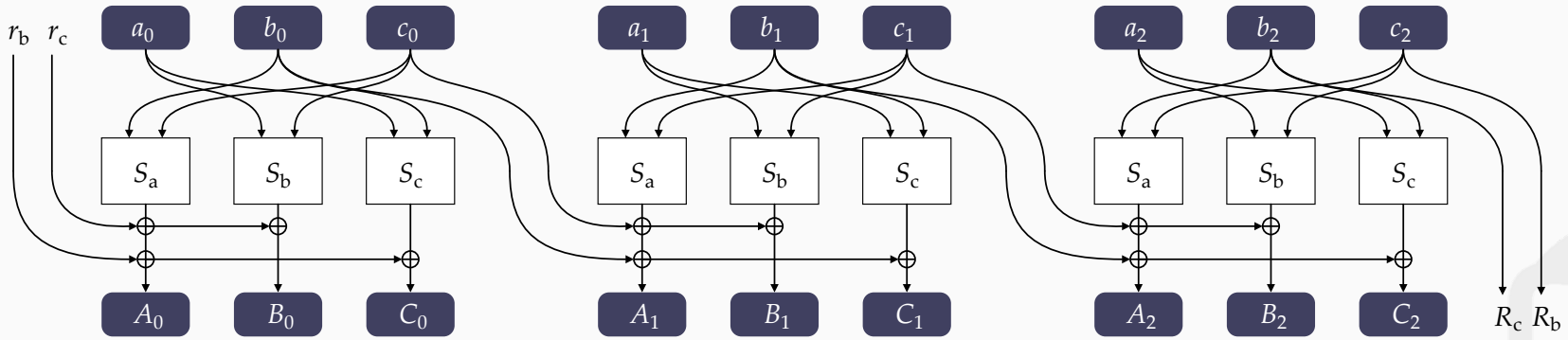
- We make (R_b, R_c) part of the shared state: the Guards
- input Guards (r_b, r_c) are previous-round output Guards (R_b, R_c)
- Achieves uniformity if S-box is invertible
- Cost:
  - 4 additional XORs per native bit
  - shared state extended by $2n$ additional bits (for n-bit S-box)

Proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

Proof of uniformity



Computing (a, b, c) and (r_b, r_c) from (A, B, C) and (R_b, R_c)

- Initial step: $b_2 \leftarrow R_c$ and $c_2 \leftarrow R_b$
- Iteration: compute (a_i, b_{i-1}, c_{i-1}) from (A_i, B_i, C_i) and (b_i, c_i)
  - $a_i = S^{-1}(A_i + B_i + C_i) + b_i + c_i$
  - $b_{i-1} = S_c(a_i, b_i) + C_i$
  - $c_{i-1} = S_b(a_i, b_i) + B_i$
- Final step: $r_b \leftarrow b_{-1}$ and $r_c \leftarrow c_{-1}$
- Invertibility implies uniformity: QED

Optimization for χ'

Multi-transformation property of χ':

Optimization for χ'

Multi-transformation property of χ':

- Assume we know
  - bits of (a, b, c) with indices 0 and 1
  - bits of (A, B, C) with indices $2, 3, \ldots m$

Optimization for χ'

Multi-transformation property of χ':

- Assume we know
  - bits of (a, b, c) with indices 0 and 1
  - bits of (A, B, C) with indices $2, 3, \ldots m$
- then we can compute bits of (a, b, c) with index m, $m-1$, ...2

$$
\begin{aligned}
A_m &= b_m + (b_0 + 1)b_1 + b_0 c_1 + b_1 c_0 \\
B_m &= c_m + (c_0 + 1)c_1 + c_0 a_1 + c_1 a_0 \\
C_m &= a_m + (a_0 + 1)a_1 + a_0 b_1 + a_1 b_0
\end{aligned}
$$

Optimization for χ'

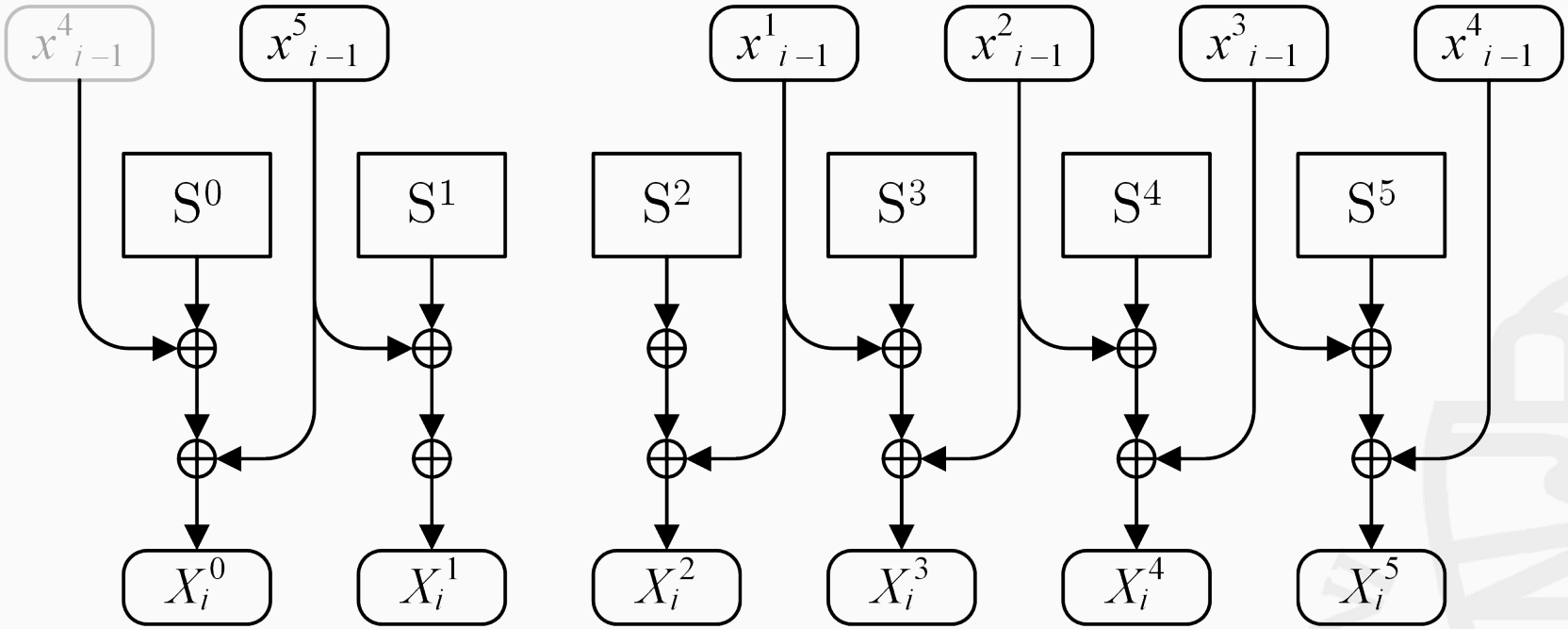
Multi-transformation property of χ':

- Assume we know
  - bits of (a, b, c) with indices 0 and 1
  - bits of (A, B, C) with indices $2, 3, \ldots m$
- then we can compute bits of (a, b, c) with index m, $m-1$, ...2

$$
\begin{aligned}
A_m &= b_m + (b_0 + 1)b_1 + b_0 c_1 + b_1 c_0 \\
B_m &= c_m + (c_0 + 1)c_1 + c_0 a_1 + c_1 a_0 \\
C_m &= a_m + (a_0 + 1)a_1 + a_0 b_1 + a_1 b_0
\end{aligned}
$$

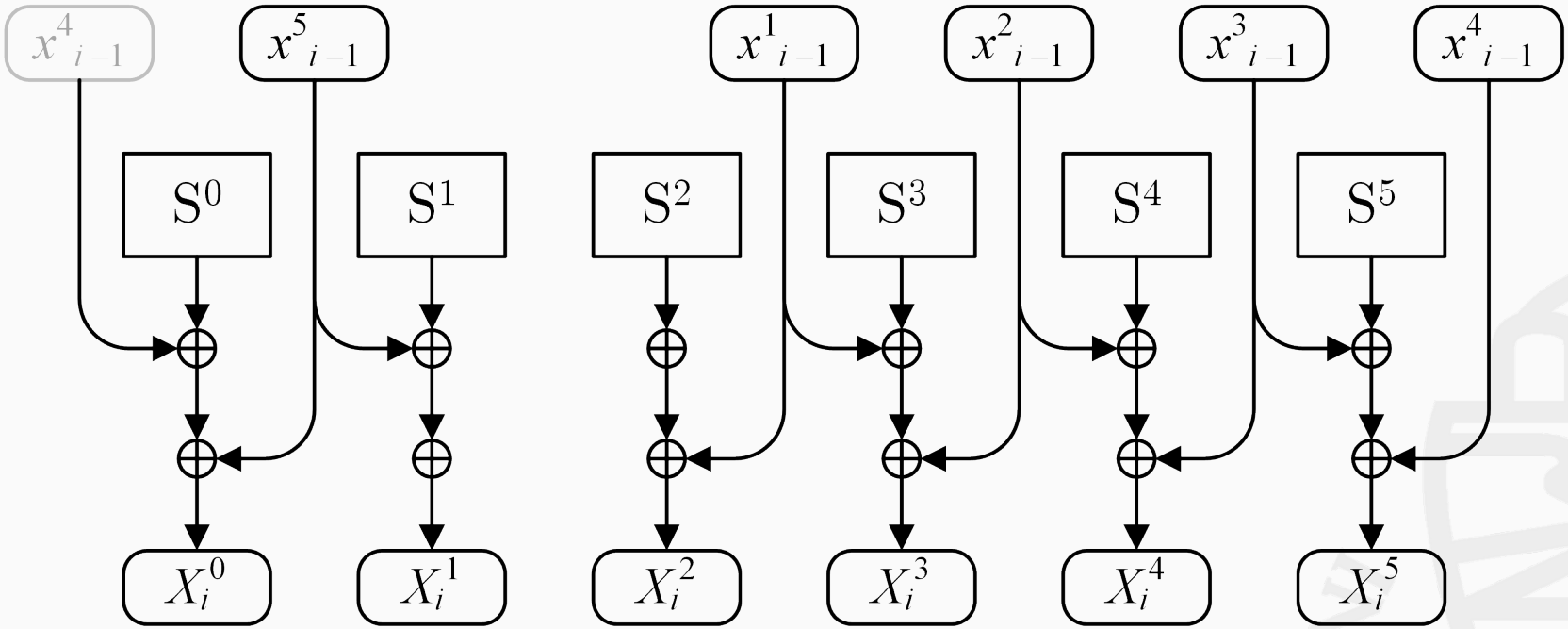
- This allows us to
  - reduce output masking to bits with indices in 0 and 1
  - shrink r_b and r_c to two bits each

Generalization for invertible n-bit S-box of degree d



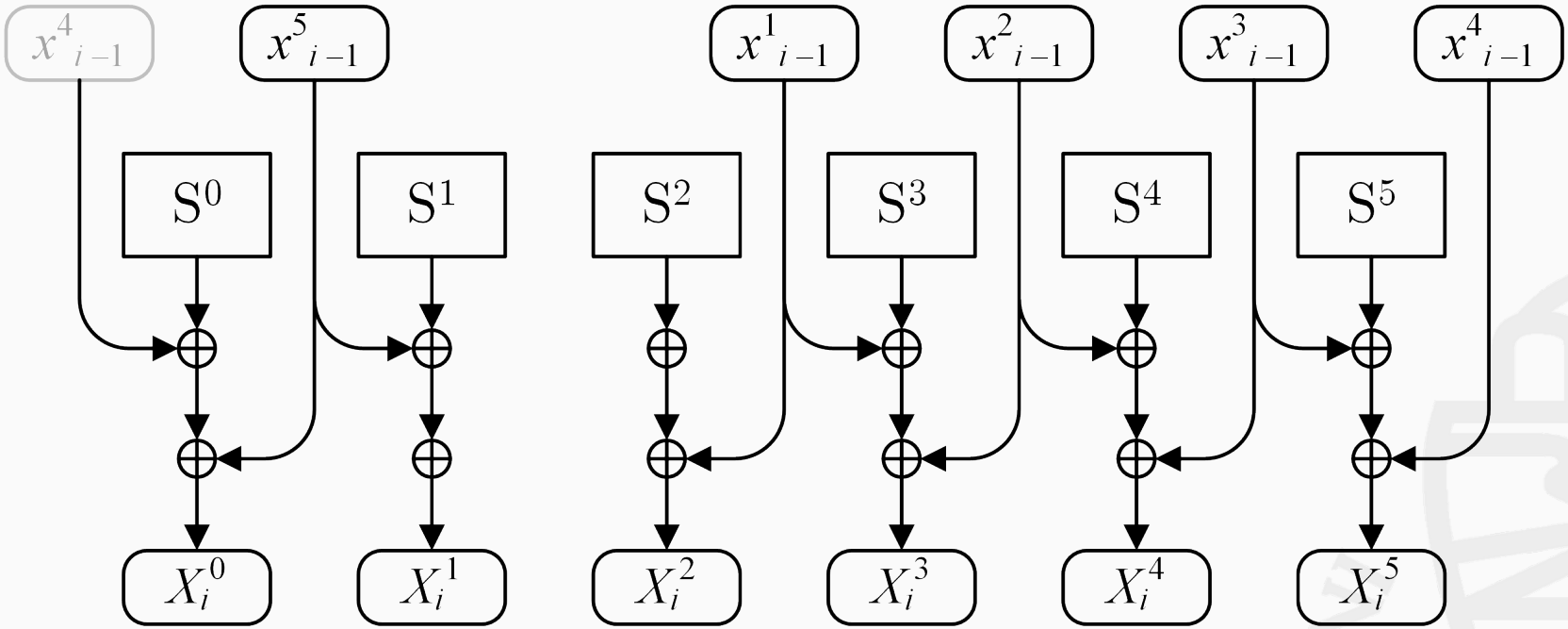
- Guards: d shares of n bits
- each *guard* share of S-box $i-1$ is added to 2 shares of S-box i
- Total cost (worst case)

Generalization for invertible n-bit S-box of degree d



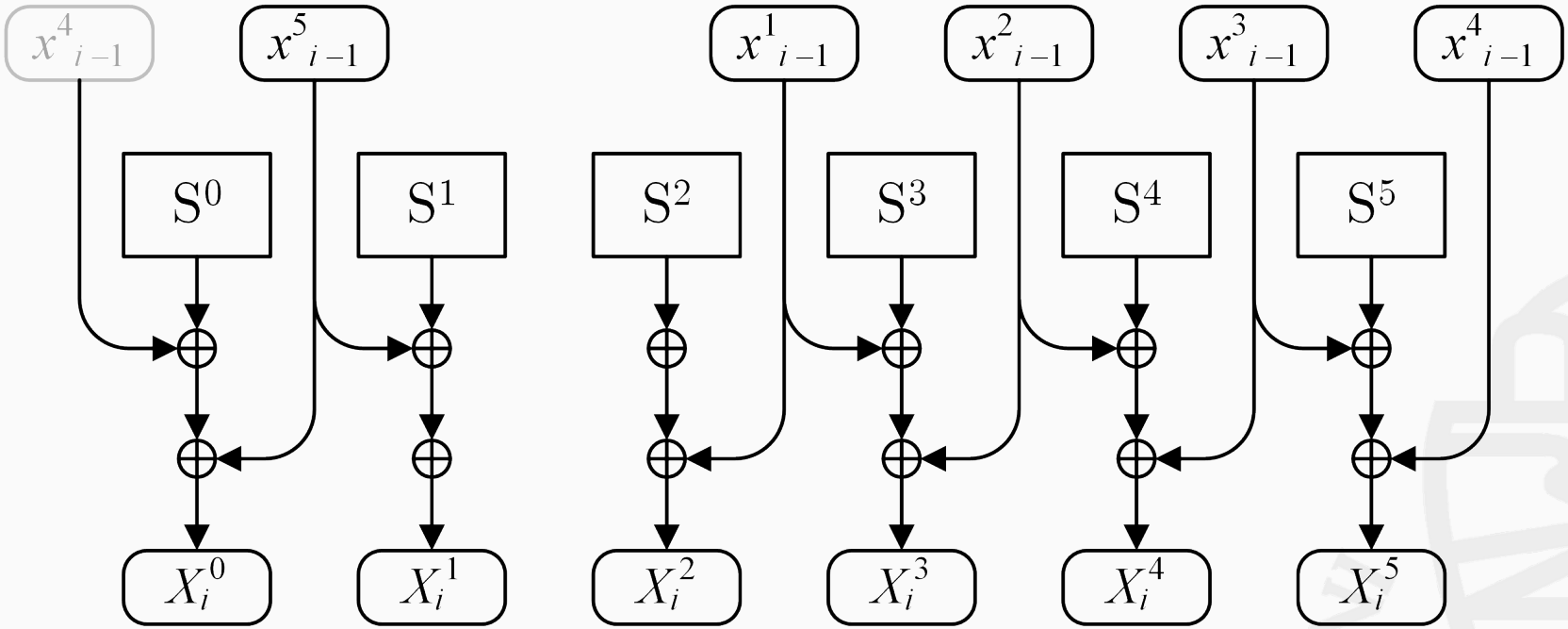
- Guards: d shares of n bits
- each *guard* share of S-box $i-1$ is added to 2 shares of S-box i
- Total cost (worst case)
  - feedforward: $2d$ XORs per native bit

Generalization for invertible n-bit S-box of degree d



- Guards: d shares of n bits
- each *guard* share of S-box $i-1$ is added to 2 shares of S-box i
- Total cost (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits

Generalization for invertible n-bit S-box of degree d



- Guards: d shares of n bits
- each *guard* share of S-box $i-1$ is added to 2 shares of S-box i
- Total cost (worst case)
  - feedforward: $2d$ XORs per native bit
  - state expansion by $d \times n$ bits
- Cost is reduced if shared S-box has multi-transformation property

Conclusions

- Solution for achieving uniformity for invertible S-box layers
  - only $d + 1$ shares for S-boxes of degree d
  - uniformity achieved outside the S-box

Conclusions

- Solution for achieving uniformity for invertible S-box layers
  - only $d + 1$ shares for S-boxes of degree d
  - uniformity achieved outside the S-box
- Real-world relevance:
  - sharing χ' (KECCAK) made uniform at little overhead

Conclusions

- Solution for achieving uniformity for invertible S-box layers
  - only $d + 1$ shares for S-boxes of degree d
  - uniformity achieved outside the S-box
- Real-world relevance:
  - sharing χ' (KECCAK) made uniform at little overhead
- Academic relevance:

Conclusions

- Solution for achieving uniformity for invertible S-box layers
  - only $d + 1$ shares for S-boxes of degree d
  - uniformity achieved outside the S-box
- Real-world relevance:
  - sharing χ' (KECCAK) made uniform at little overhead
- Academic relevance:
  - non-uniformity problem essentially solved

Conclusions

- Solution for achieving uniformity for invertible S-box layers
  - only $d + 1$ shares for S-boxes of degree d
  - uniformity achieved outside the S-box
- Real-world relevance:
  - sharing χ' (KECCAK) made uniform at little overhead
- Academic relevance:
  - non-uniformity problem essentially solved
  - search multi-transformation sharing of low-degree S-boxes

Conclusions

- Solution for achieving uniformity for invertible S-box layers
  - only $d + 1$ shares for S-boxes of degree d
  - uniformity achieved outside the S-box
- Real-world relevance:
  - sharing χ' (KECCAK) made uniform at little overhead
- Academic relevance:
  - non-uniformity problem essentially solved
  - search multi-transformation sharing of low-degree S-boxes

Thanks for your attention!

Q?