Linear Cryptanalysis and Countermeasures in Persistent Fault Model

22 June 2023

by **Viet-Sang Nguyen**

joint work with **Vincent Grosso** and **Pierre-Louis Cayrel**

in ANR PROPHY project

UNIVERSITÉ DE LYON

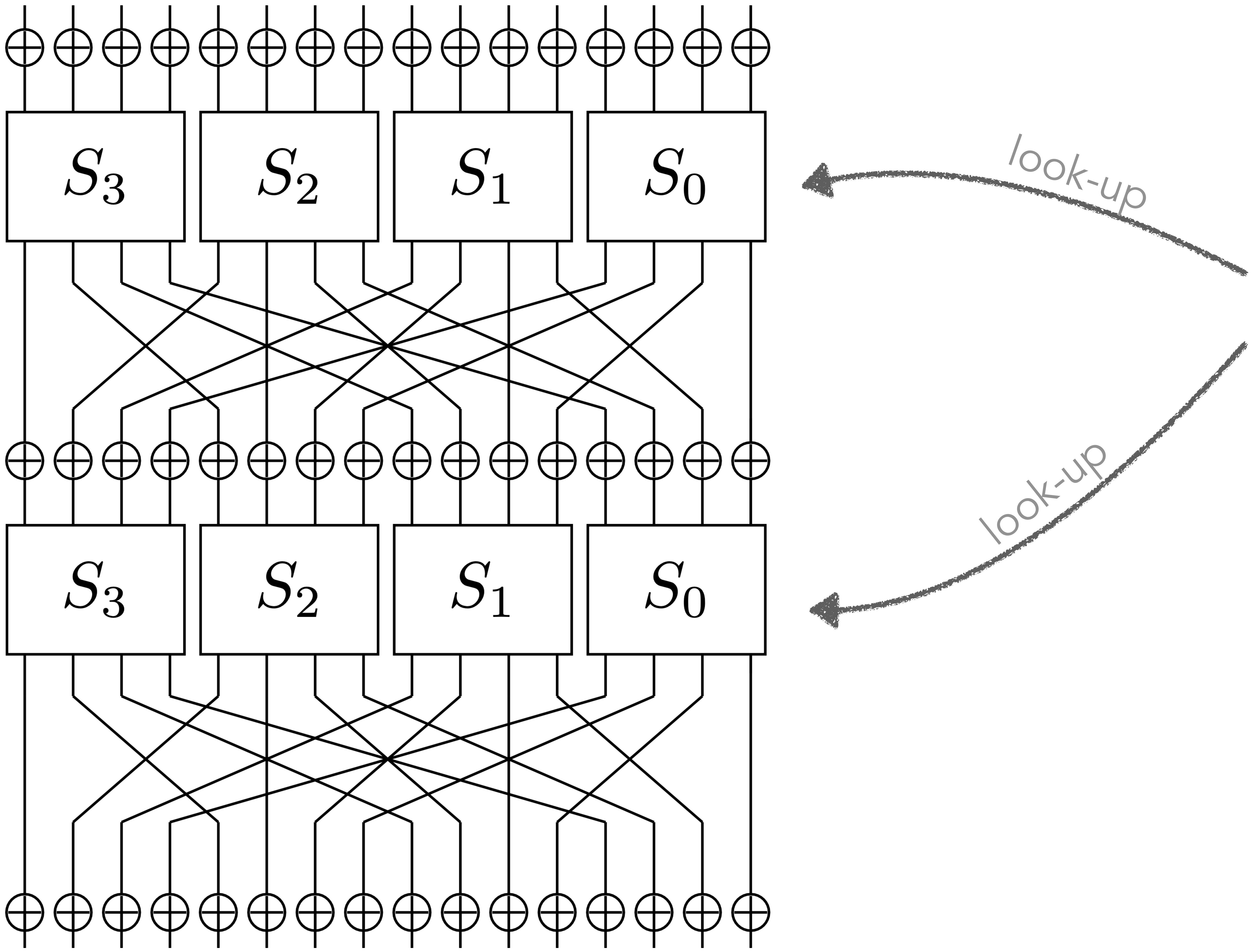
LABORATOIRE HUBERT CURIEN
UMR • CNRS • 5516 • SAINT-ETIENNE

Outlines

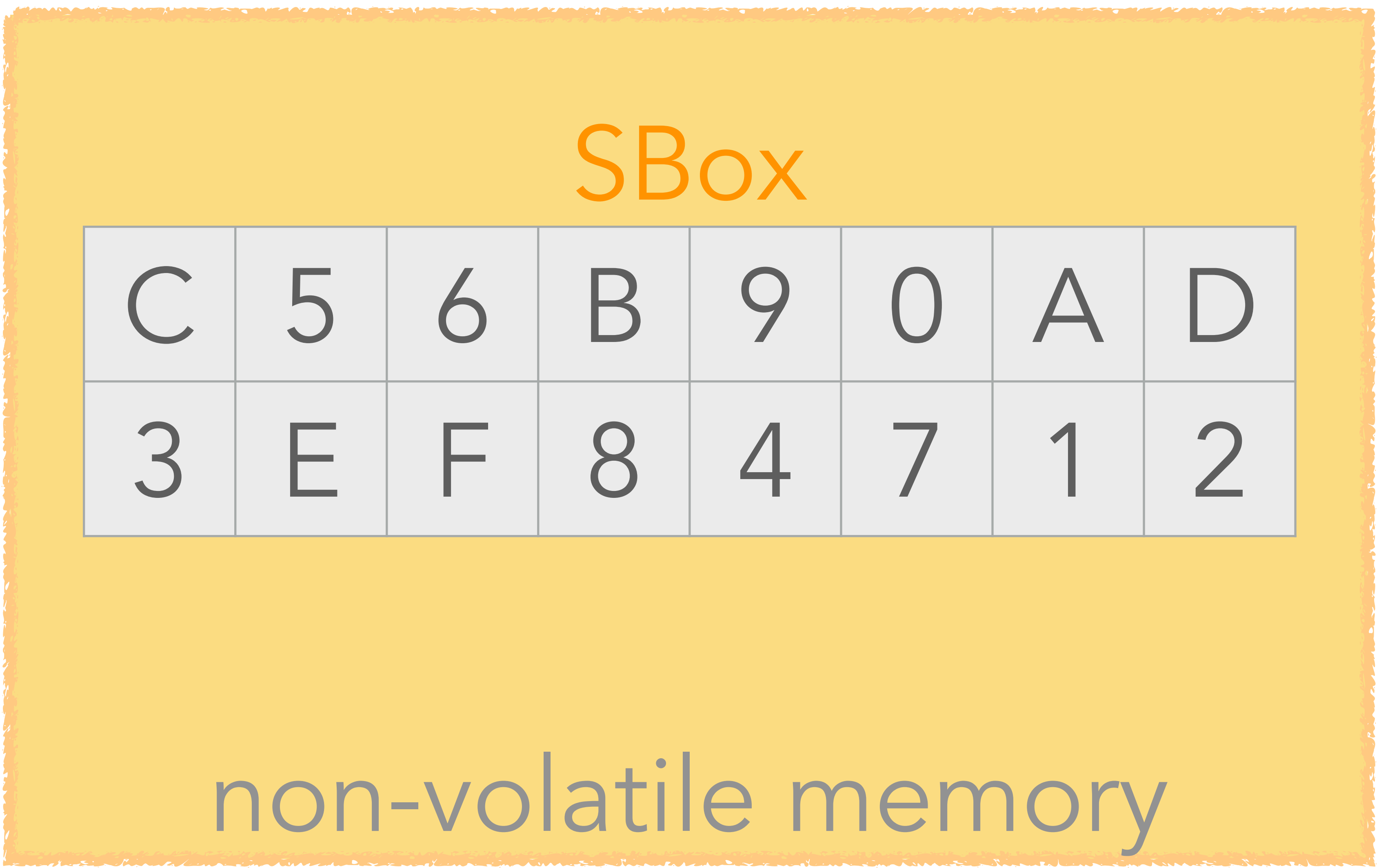
Outlines

1. Context
   - Previous PFA
   - Our research questions
2. Countermeasures against biased faulty SBoxes
   - BALoo
   - Frequency Checking
3. Linear Cryptanalysis: PRESENT with non-biased faulty SBox
4. Stronger Countermeasures
   - Permutation Network
   - Cyclic Redundancy Code
5. Summary

Previous Persistent Fault Attacks (PFA)

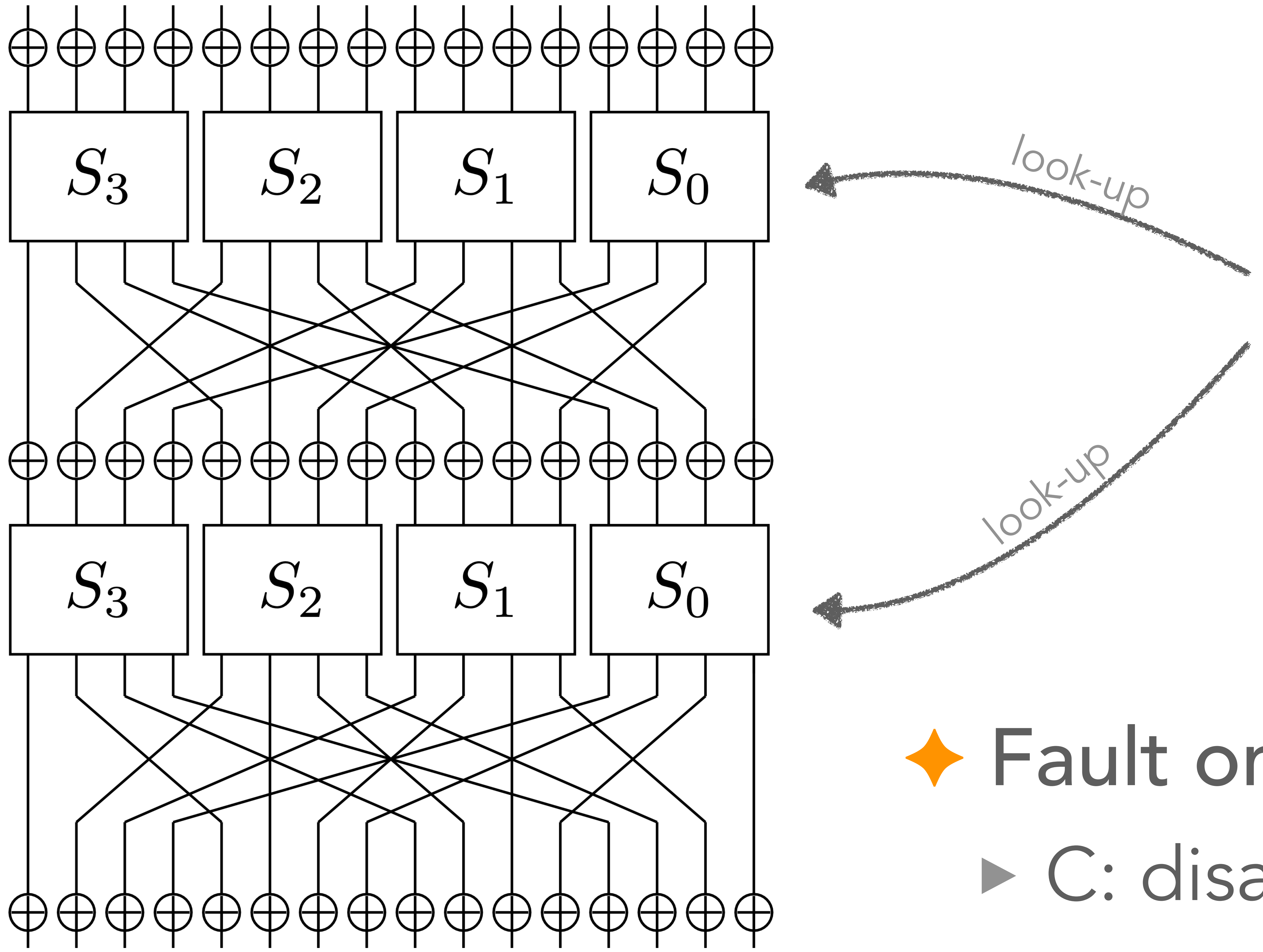


ciphertexts: uniform distribution



C	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

Previous Persistent Fault Attacks (PFA)



ciphertexts: non-uniform distribution

biased faulty SBox

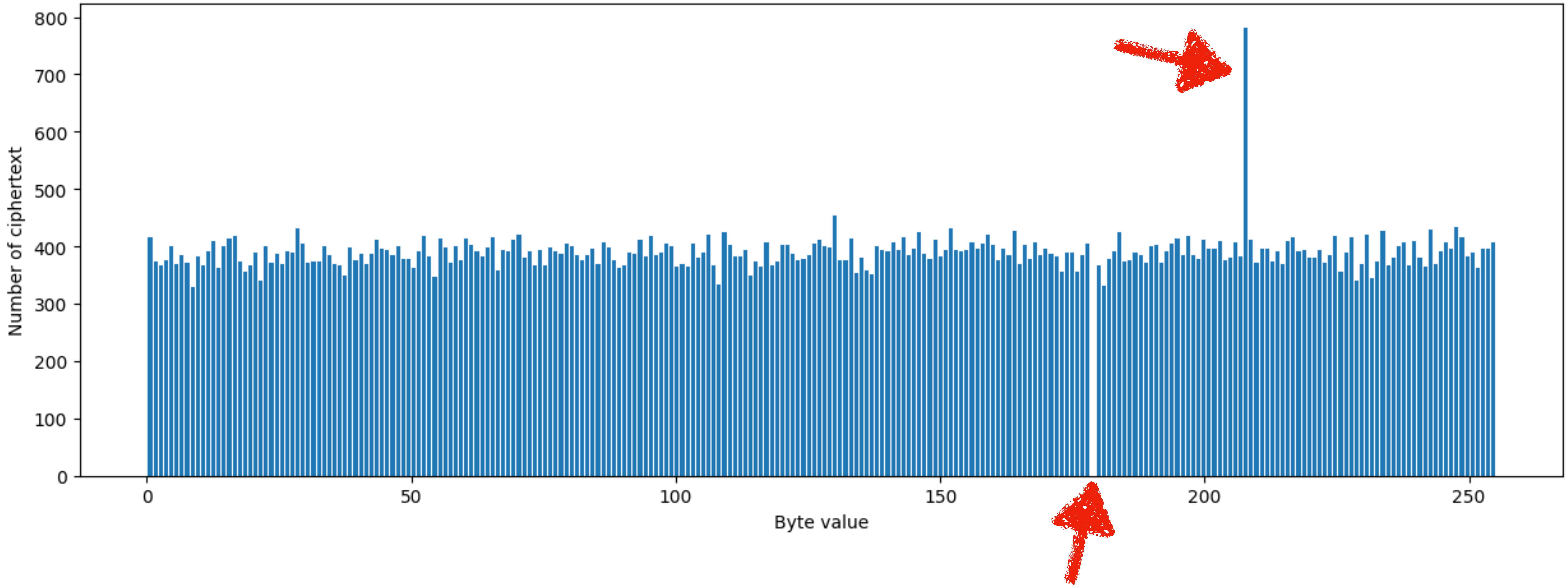
5	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

non-volatile memory

- **Fault on first element: C → 5**
  - C: disappears
  - 5: appears twice

Non-uniform Distribution of Ciphertexts

✦ Attacks: [Zhang et al., CHES18,20], [Pan et al., DATE19], [Gruber et al., FDTC19], [Engels et al., FDTC20], [Soleimany et al., CHES22]



Research Questions

- Countermeasures ?
  - *biased* faulty SBox
- What if swap 2 elements ?
  - *non-biased* faulty SBox
  - possible to recover key ?
- (Stronger) Countermeasures ?
  - *both* biased and non-biased faulty SBoxes

5	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

biased faulty SBox

C	6	5	B	9	0	A	D
3	E	F	8	4	7	1	2

non-biased faulty SBox

Principle of Countermeasures

- Ensure the integrity of SBox 👮
  - Detect any (?) injected faults 🚨
  - Make it impractical for attacker to successfully inject faults

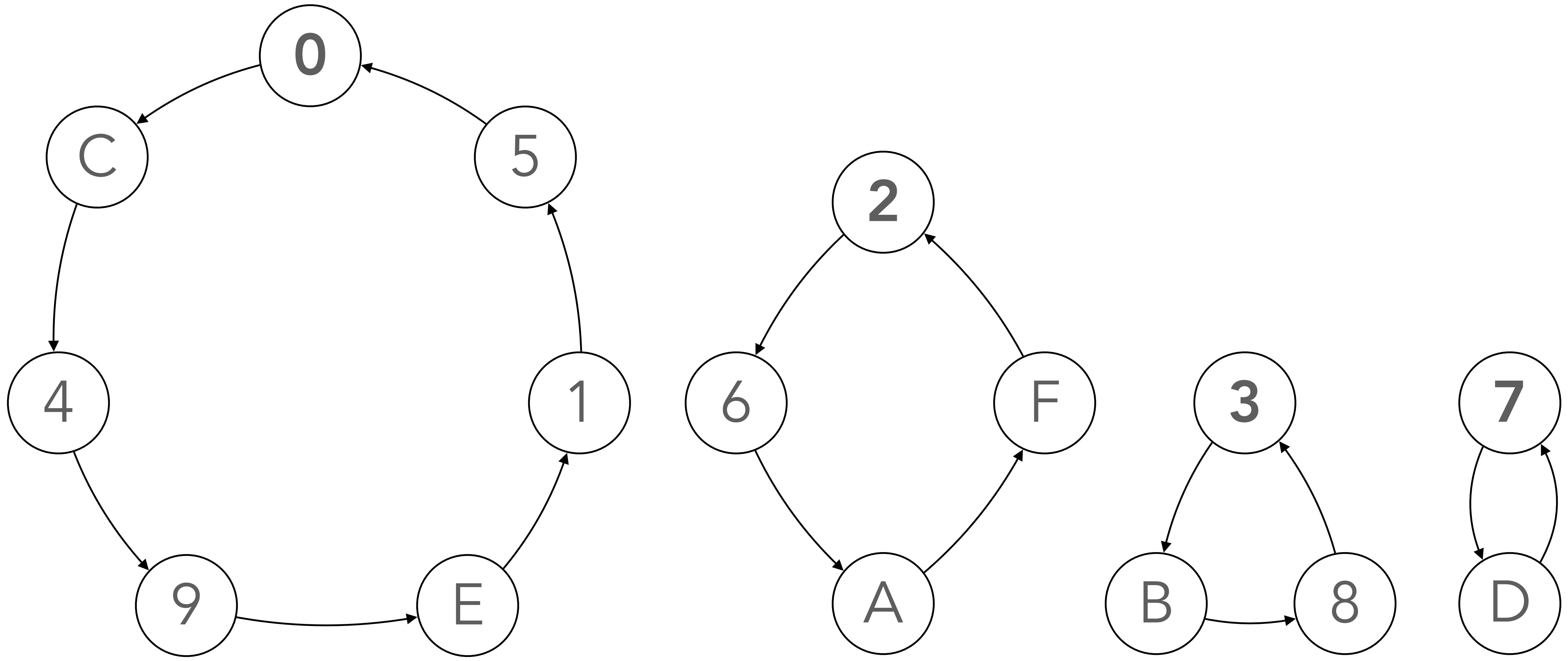
Outlines

1. Context
   - Previous PFA
   - Our research questions
2. Countermeasures against biased faulty SBoxes
   - BALoo
   - Frequency Checking
3. Linear Cryptanalysis: PRESENT with non-biased faulty SBox
4. Stronger Countermeasures
   - Permutation Network
   - Cyclic Redundancy Code
5. Summary

Countermeasure: BALoo [Tissot et al., 2023]

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

- ✦ Redundancy info:
  (stored in non-volatile memory)
  - ▸ Number of cycles
  - ▸ Starting indices
  - ▸ Their lengths
- ✦ Verify before encryption



S(0) = C, **S**[C] = 4, …

Countermeasure: Frequency Check

- Each element should appear ONCE
- Example:
  - Freq(6) = 1 → OK
  - Freq(5) = 2 → fault detected
- Not require redundancy info

5	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

biased faulty SBox

BALoo and Frequency Check

⚡

5	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

biased faulty SBox

- **Efficiency:**
  - detect any *biased* faulty SBoxes ✅
  - prevent attacks in prior works ✅

But…can we bypass them ? 🤔

- ✦ YES !!! 😈

1110 ↔ 0110
2 bitflips

C	5	E	B	9	0	A	D
3	6	F	8	4	7	1	2

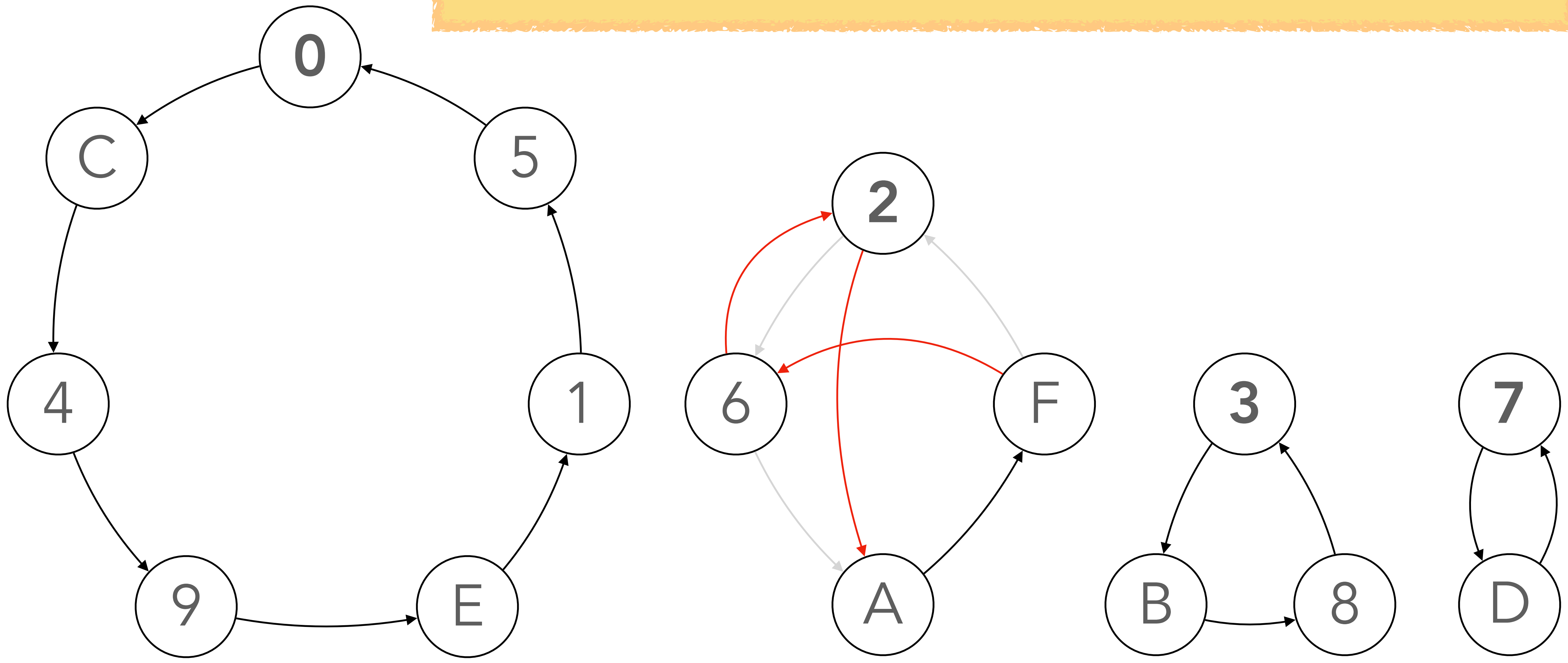
non-biased faulty SBox

4 bitflips

C	5	A	B	9	0	2	D
3	E	F	8	4	7	1	6

non-biased faulty SBox

Freq(E) = 1 → OK
Freq(6) = 1 → OK



OK! Bypass…then what next ? 🤔

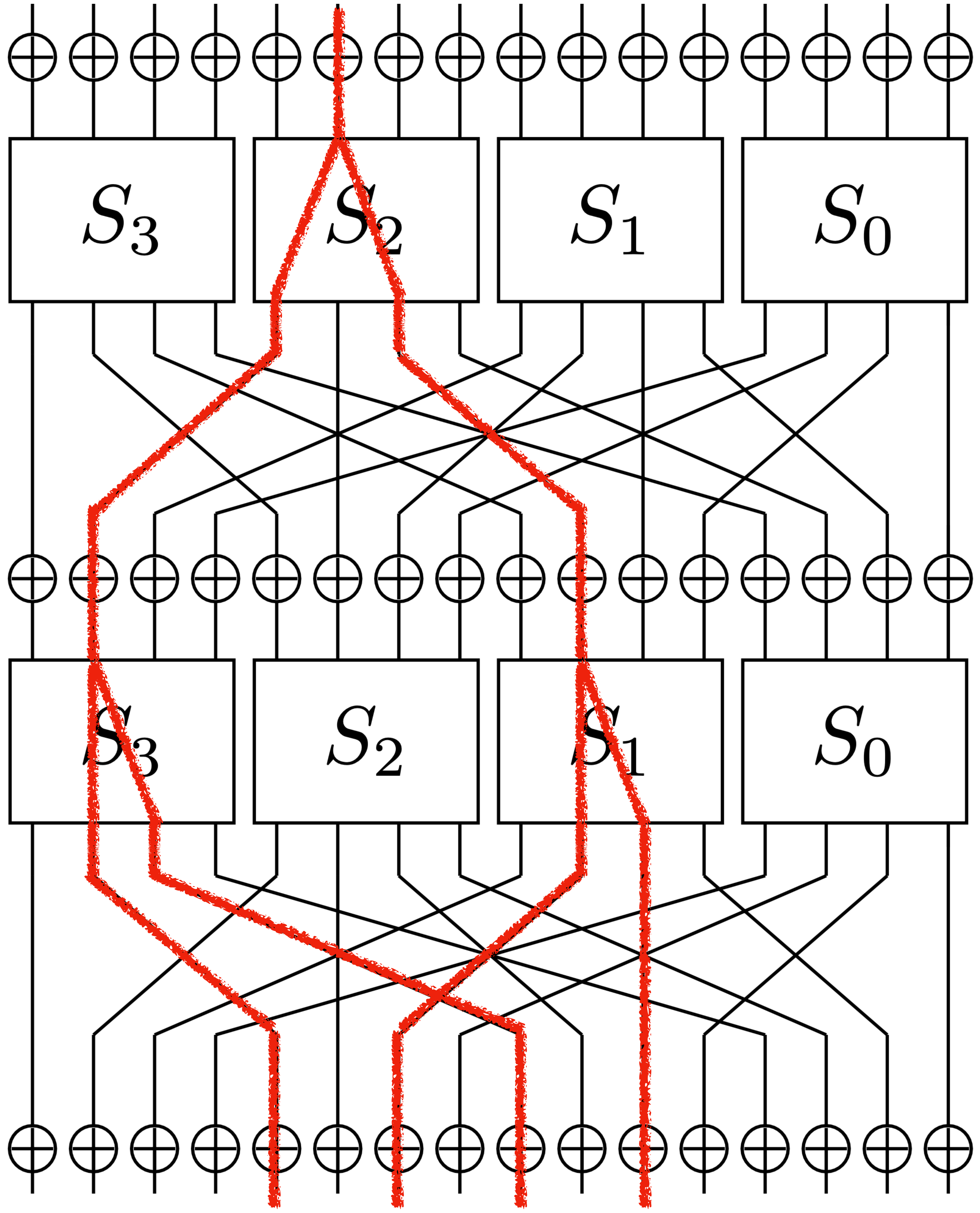
- Prior attacks are still applicable ?
  - NO 👿
  - Non-biased faulty SBox → still uniform ciphertexts

- New attack ? 🤔
  - YES (but very classical, not new) 😈
  - Linear Cryptanalysis

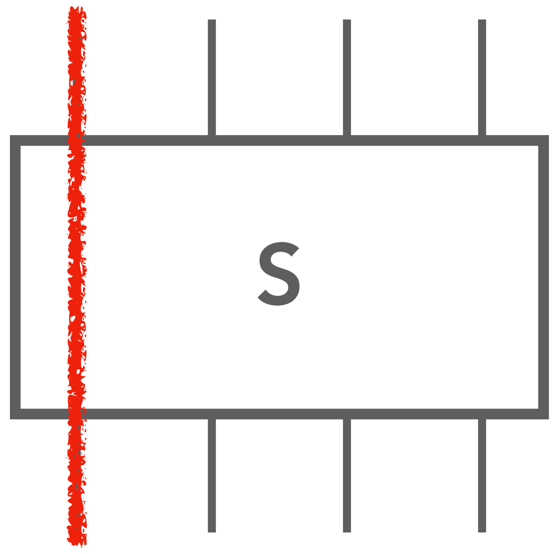
Outlines

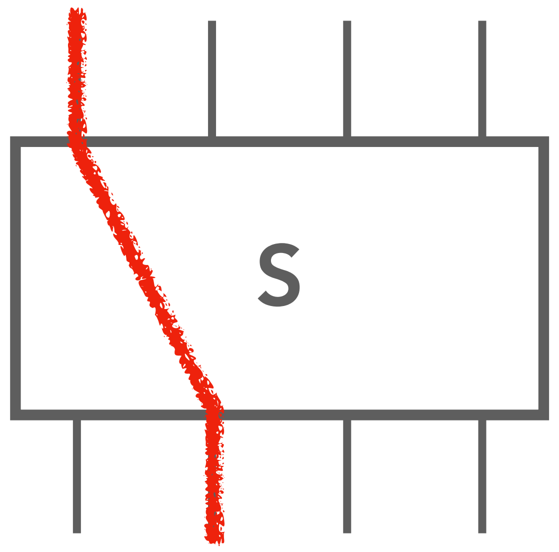
Classical Linear Cryptanalysis [Matsui, CRYPTO94]

- Find a good linear approximation
- Use statistical analysis
  - many plaintext-ciphertext pairs
- Recover part of key



Linear Approximation Table: 1-bit LAT





LAT (biases): $\#\{x \in \mathbb{F}_2^4 : u \cdot x = v \cdot S[x]\} - 8$

u\v	1	2	4	8
1	0	0	0	0
2	0	2	-2	2
4	0	-2	-2	-2
8	0	2	0	-2

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

Linear Approximation Table: 1-bit LAT

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2

original

u\v	1	2	4	8
1	0	0	0	0
2	0	2	-2	2
4	0	-2	-2	-2
8	0	2	0	-2

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	3	D	E	F	8	4	7	1	2

2 swaps

u\v	1	2	4	8
1	0	2	-2	-2
2	0	4	-4	0
4	0	0	-4	-4
8	0	0	2	0

x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	F	B	9	0	A	D	3	E	2	8	4	7	1	6

3 swaps

u\v	1	2	4	8
1	0	0	2	0
2	0	2	-2	2
4	0	-2	0	-2
8	-2	2	0	-4

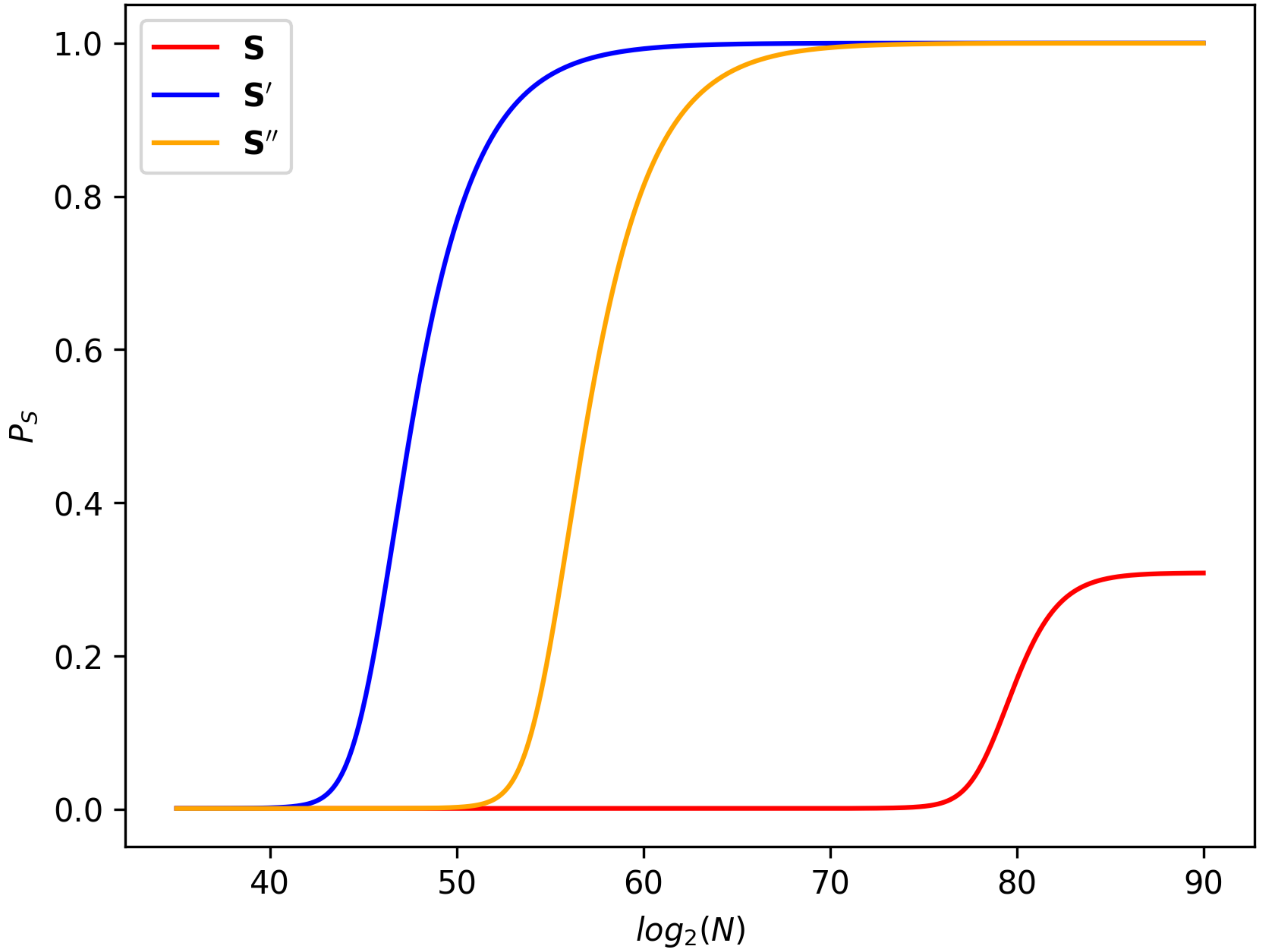
seems very vulnerable !!! 😈

Complexity Estimation [Nyberg et al., FSE17]

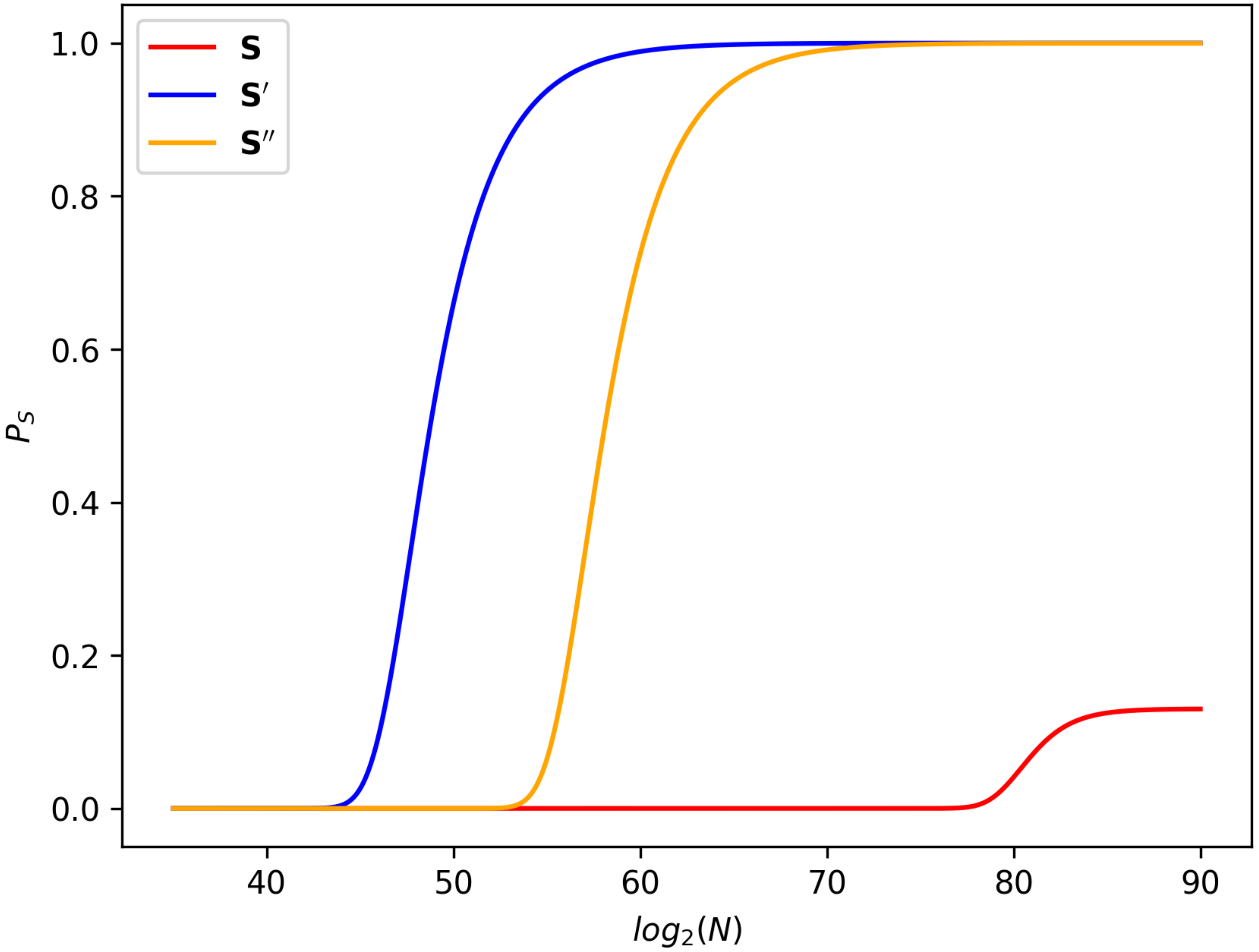
- Success probability: $P_S = 2 - 2\Phi\left(\sqrt{\frac{1/N + \text{ELP}}{1/N + 2^{-b}}}\Phi^{-1}(1 - 2^{-a-b})\right)$
- Data complexity: $N = \frac{\Phi^{-1}(1 - 2^{-a-1})^2 - \Phi^{-1}(1 - P_S/2)^2}{\text{ELP} \cdot \Phi^{-1}(1 - P_S/2)^2 - 2^{-|K|}\Phi^{-1}(1 - 2^{-a-1})^2}$
- For PRESENT:
  - $b = 64$: block size
  - $|K| = 80$: key size
- Estimated Linear Potential (ELP):
  - derived from 1-bit LAT
  - computed over 28 rounds (out of 31)
- a: number of advantage bits
  - recover a bits by linear attack
  - only need to brute-force $|K| - a$ bits

relations between a, P_S, N ? 🤔

Attack on full-round PRESENT

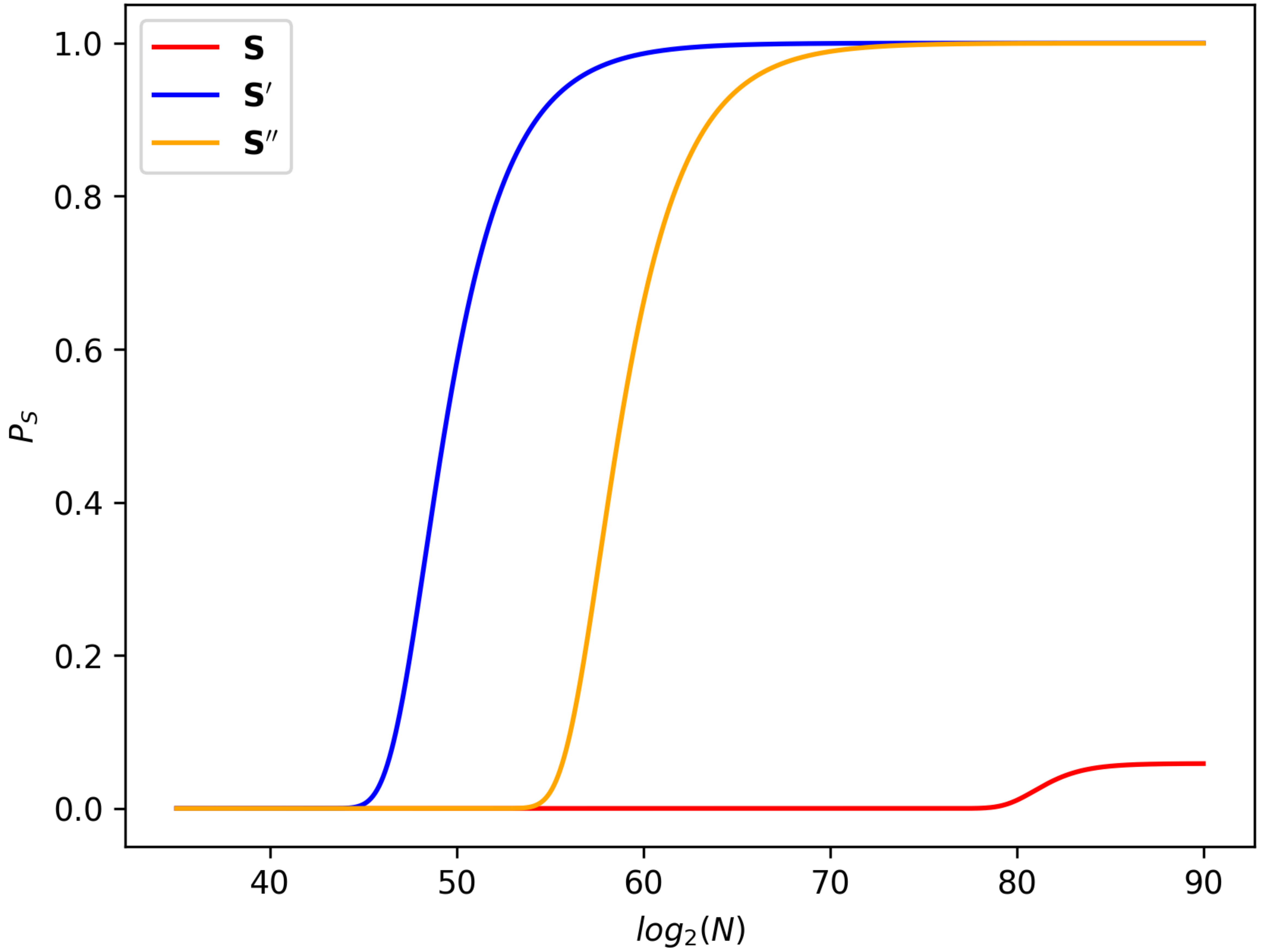


$a = 10$

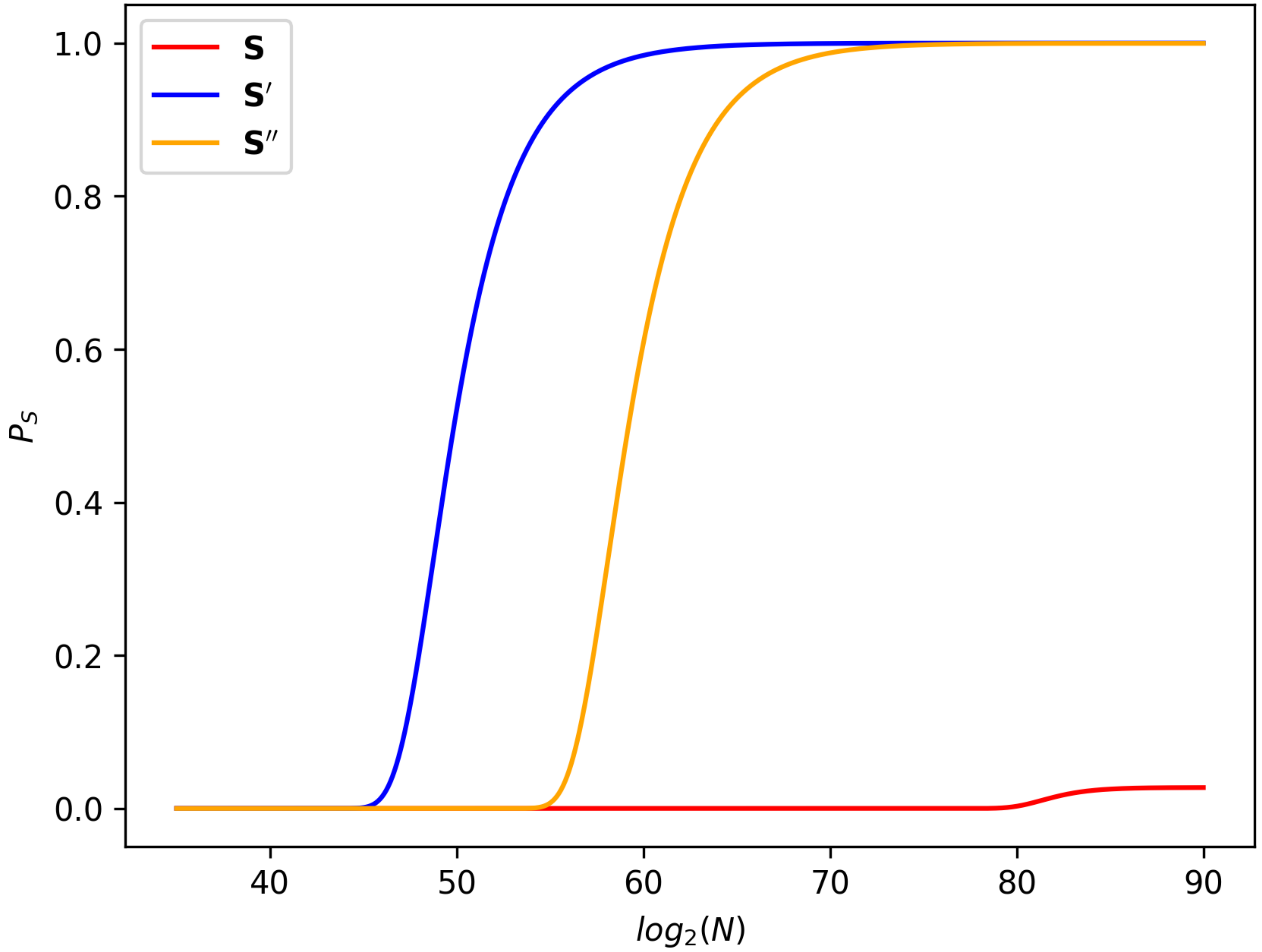


$a = 20$

Attack on full-round PRESENT



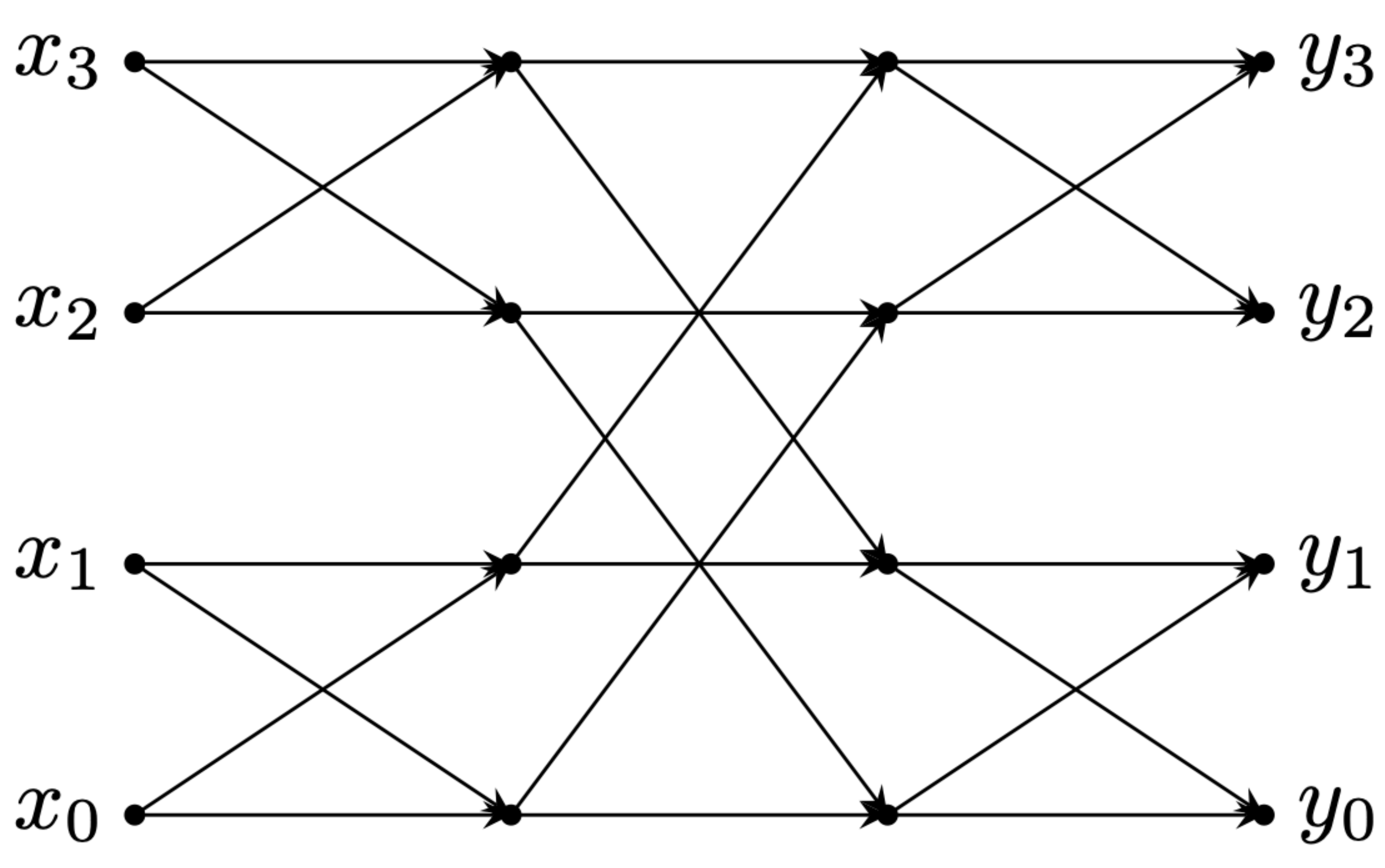
$a = 30$



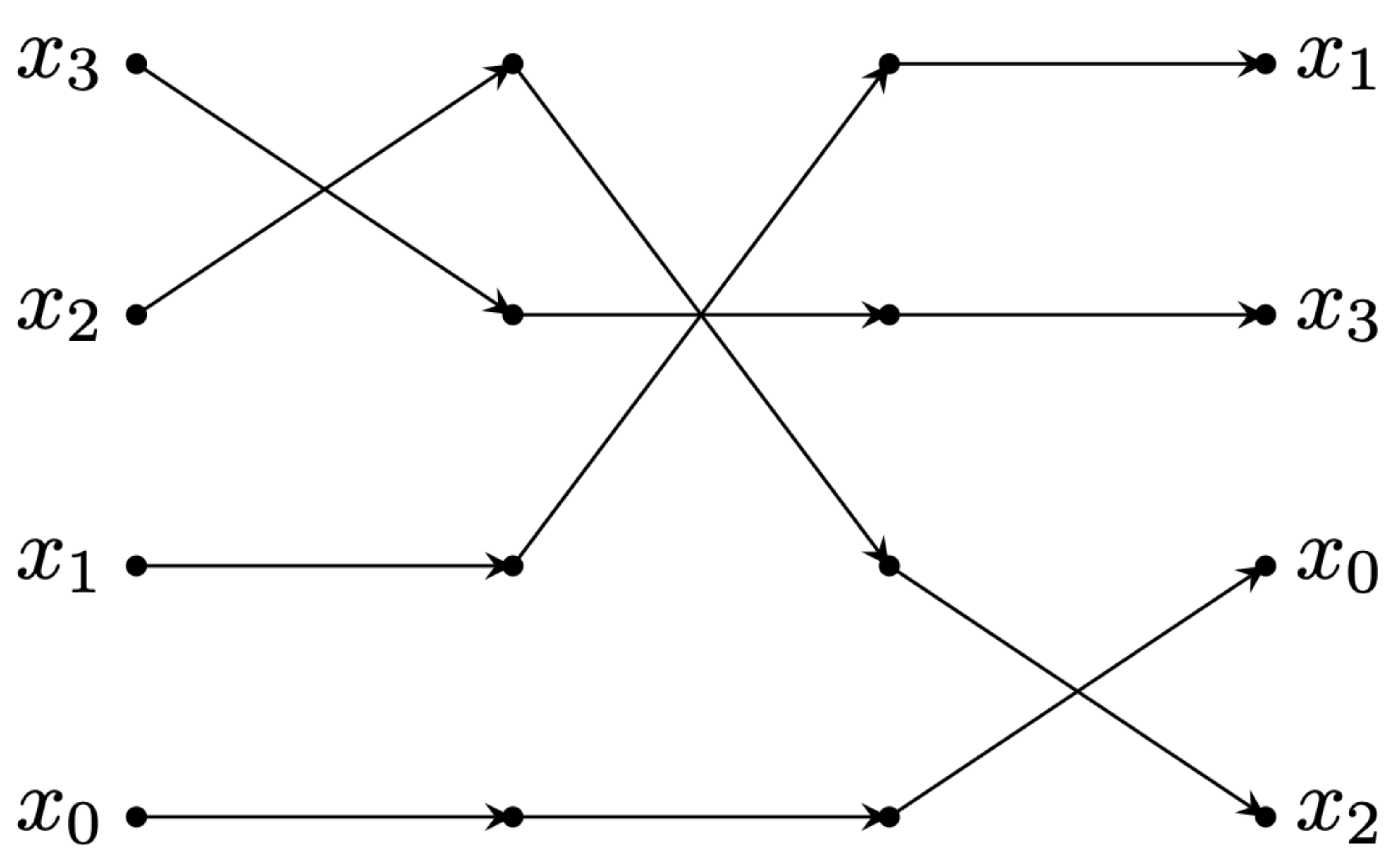
$a = 40$

Outlines

Permutation Network [Beneš, 1964]



data flow



control bits: 010110

- **PN computations [Bernstein, 2020]:**
  - PN → Control bits
  - Control bits → PN

PN-based Countermeasure

- ✦ Control bits as redundancy
- ✦ Before encryption:
  - ▸ control bits → PN: SBox'
  - ▸ compare SBox' with SBox
- ✦ Seems good, but… 🤔

SBox

C	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

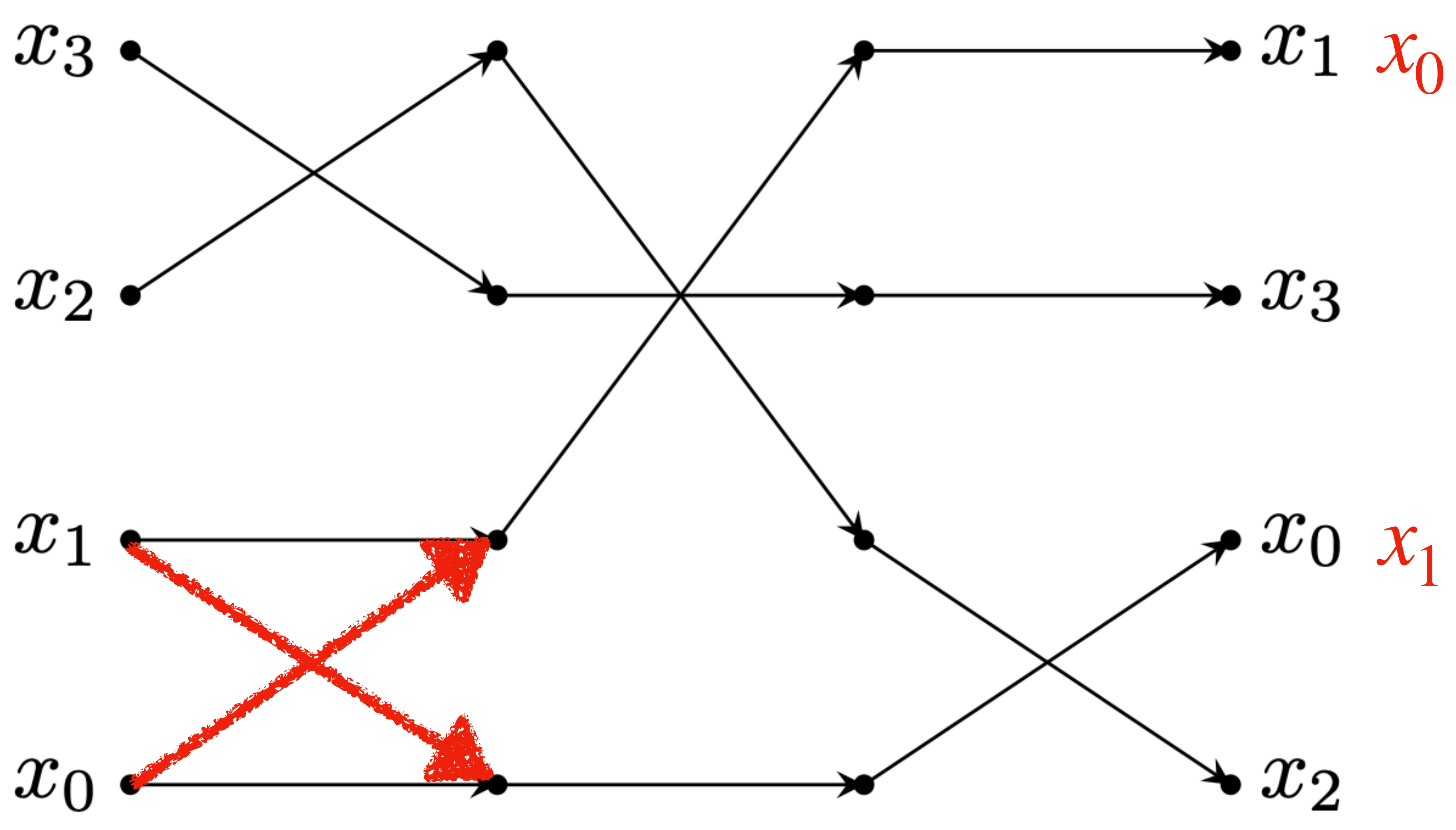
control bits

00000000…00101101

non-volatile memory

PN-based Countermeasure

- What if **both** SBox and control bits are faulted ? 🤔



control bits: 110110

⚡ ⚡ SBox

C	6	5	B	9	0	A	D
3	E	F	8	4	7	1	2

⚡ control bits

00100000…00101101

non-volatile memory

≥ 3 bitflips at precise locations to bypass !!! 😈

But still (always) able to detect biased faulty SBox ✅

Improved PN-based Countermeasure

SBox	#bits	#controlbits	#bit1 (orig. SBox)	#bit1 (faulty SBox with 2 elements swapped)
AES	8	1920	846	{803, 805, 813, 829, 831, 833, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859, 861, 863, 865, 867, 869, 871, 873, 877, 879, 881, 891}
PRESENT/LED	4	56	18	{15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37}
GIFT	4	56	26	{19, 21, 23, 25, 27, 29 }
PRINCE	4	56	22	{19, 21, 23, 25, 27}

#bit1 (orig. SBox) $\notin$ {#bit1 (all faulty SBoxes with 2 elememts swapped)}

- Indices of bit 1 (in control bits) as redundancy
  - Attacker cannot change #bit1

SBox

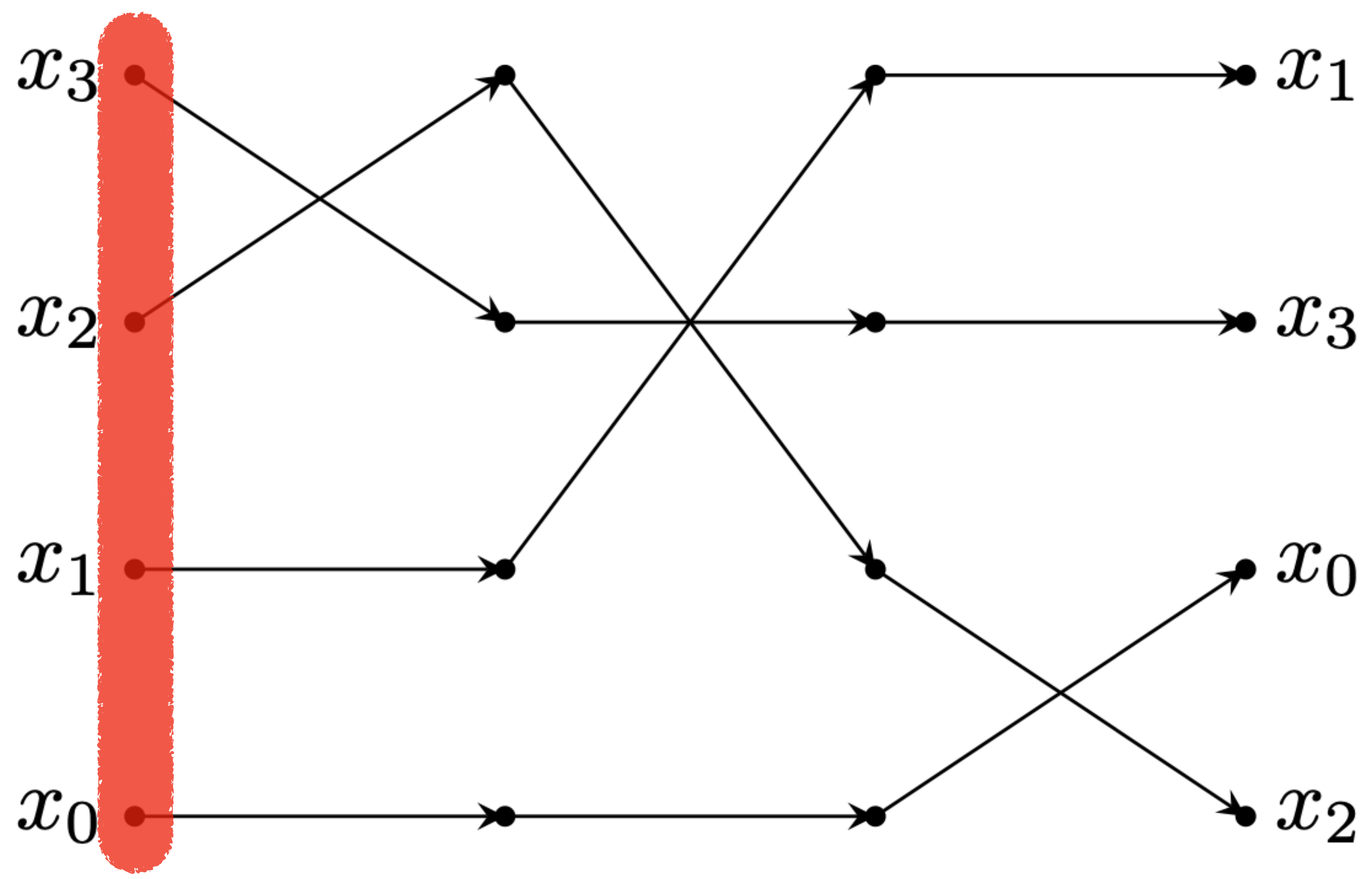
C	5	6	B	9	0	A	D
3	E	F	8	4	7	1	2

indices of bit1 in control bits

15, 25, 26, ..., 52, 53, 55

Improved PN-based Countermeasure

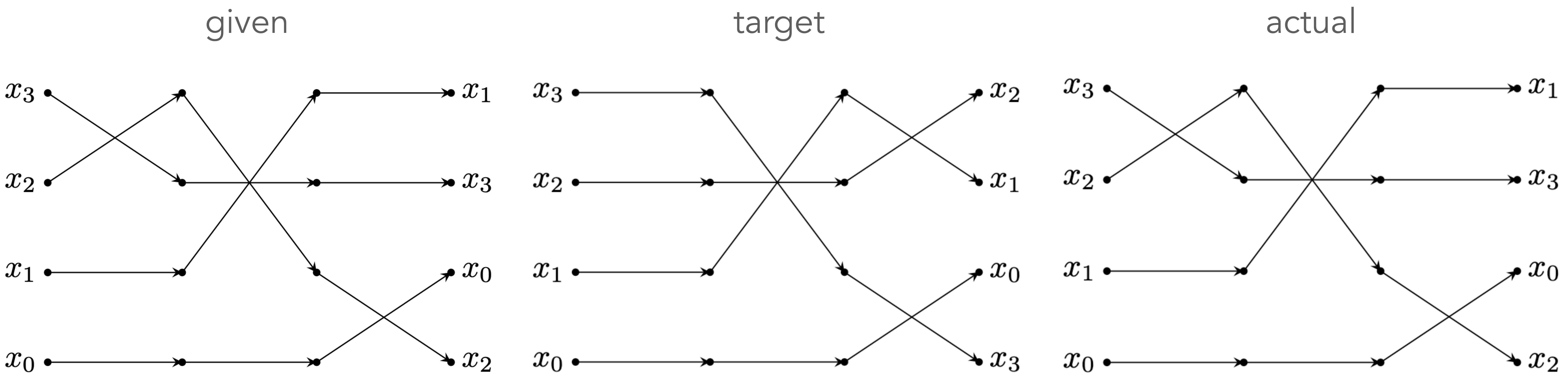
- ✦ <u>Always</u> able to detect
  - ▸ biased faulty SBoxes ✅
  - ▸ faulty SBoxes with 2 elements swapped ✅
  - ▸ Simplify algorithm: control bits → PN 👍
    (no need to traverse all swap gates)
- ✦ "In-place" property:
  - ▸ Maintain an array for different layers
  - ▸ Control bits must be processed in order



control bits: 010110
indices of bit1: [1, 3, 4]

Improved PN-based Countermeasure

✦ If inject faults on indices…



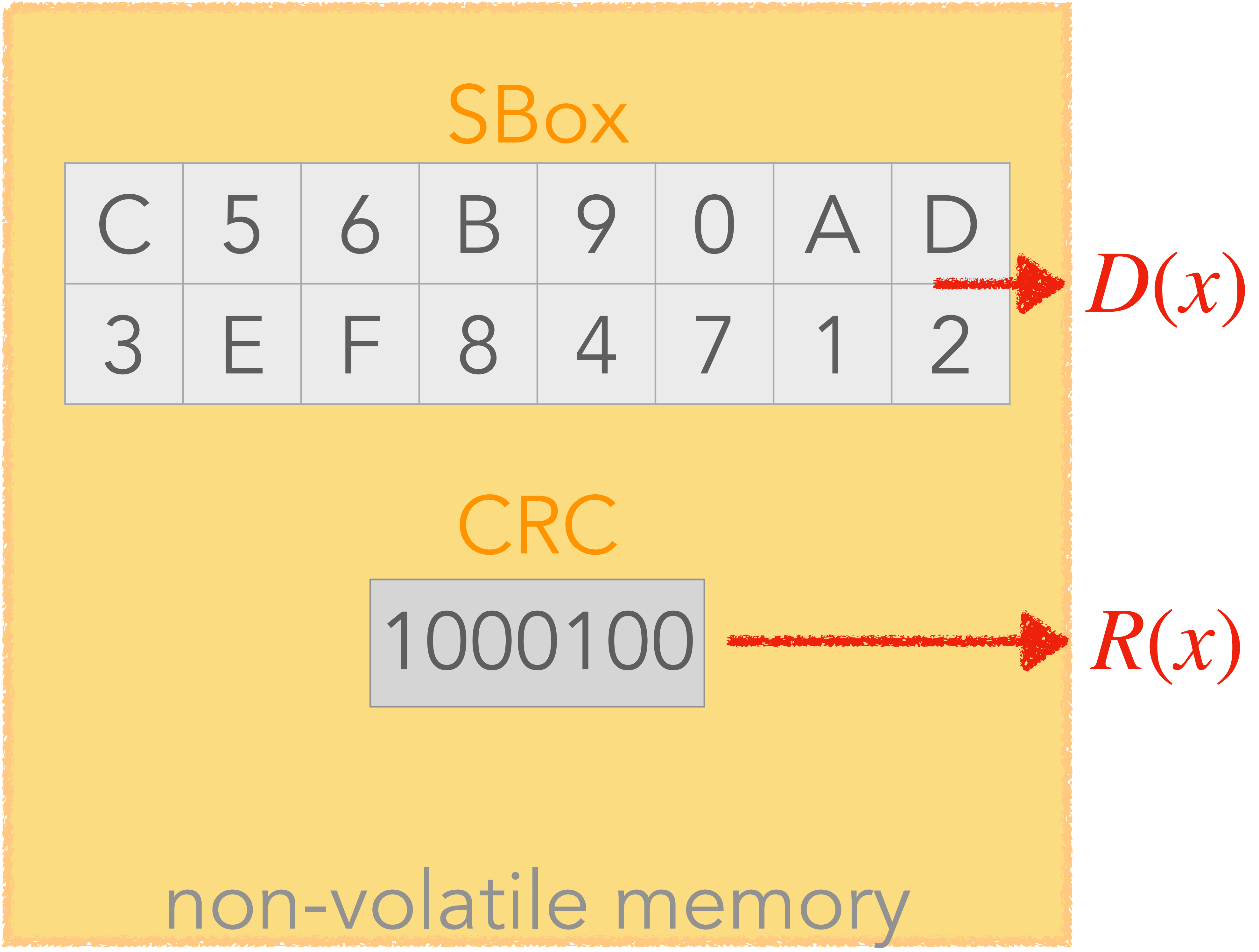
given	target	actual
control bits: 010110	control bits: 000111	control bits: 00011<u>1</u>
indices of bit1: [1, 3, 4]	indices of bit1: [3, 4, 5]	indices of bit1: [<u>5</u>, 3, 4]

→ very challenging for attacker !!! 😈

rough est.: $\geq$ 4 bitflips <u>at precise locations</u> to bypass !!! 😈

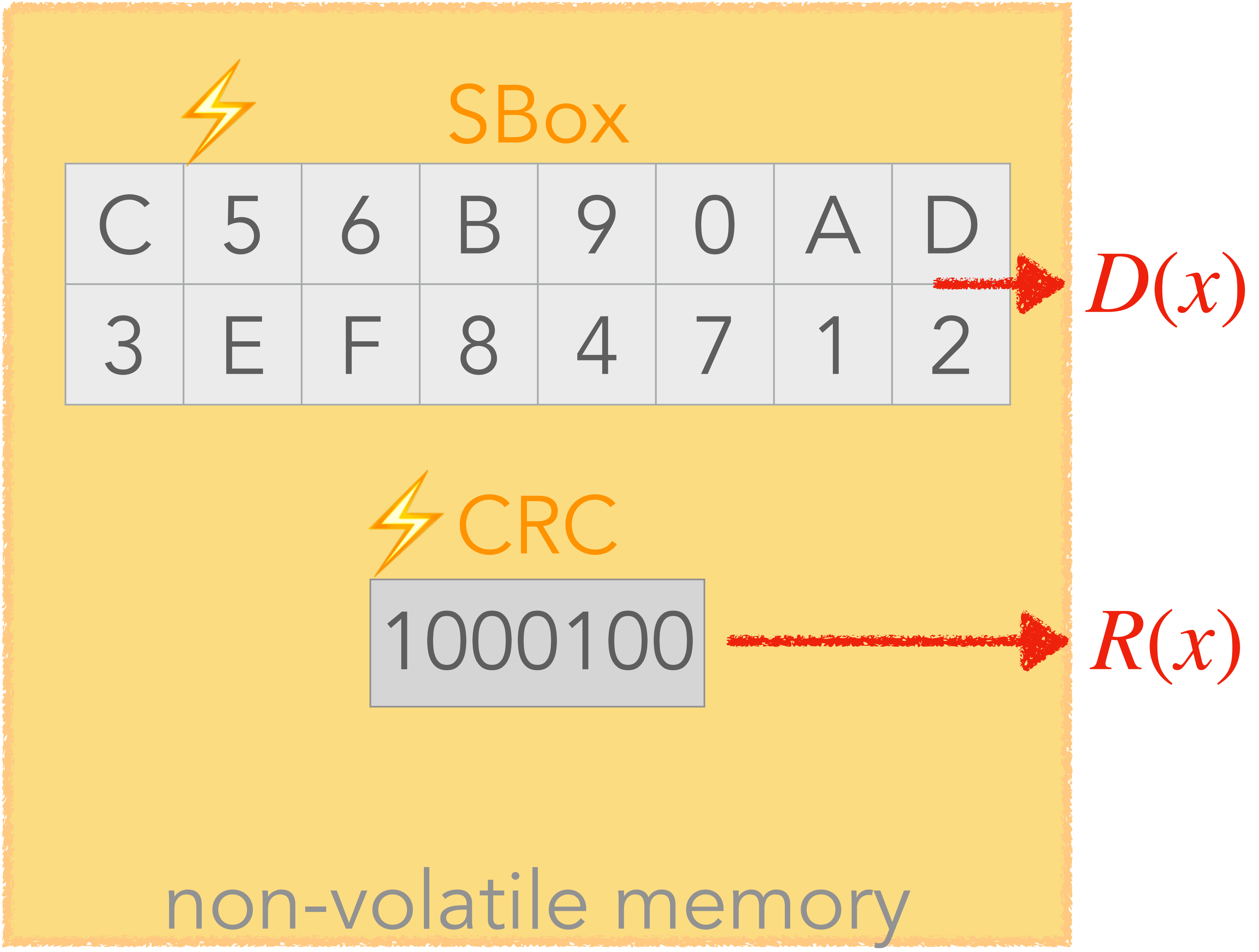
CRC: Cyclic Redundancy Code

- Common method to protect data integrity
  - k-bit data: $D(x)$
  - generator polynomial (of degree $n-k+1$): $P(x)$
  - $(n-k)$-bit redundancy:
    $R(x) = x^{n-k}D(x) \mod P(x)$
  - to transmit/store: $T(x) = x^{n-k}D(x) + R(x)$
- Verification
  - $T(x) \mod P(x) \stackrel{?}{=} 0$
- Efficient soft/hardware implementations



Choice of $P(x)$ [Koopman et al., 2004]

- 4-bit SBox: 0x97 - 8 bits $R(x)$
- 8-bit SBox: 0xC07 - 12 bits $R(x)$
- Advantage:
  - Detect any 1-, 2-, 3-bit errors



- What if faults on **both SBox and CRC**?

rough est.: $\geq$ 2 bitflips at precise locations to bypass !!! 😈

Outlines

Summary

- Detecting biased faulty SBoxes is not enough
- Linear Cryptanalysis with non-biased faulty SBoxes
- Stronger countermeasures:
  - PN-based
  - CRC-based

Summary

Countermeasure	Biased Sbox	Non-biased SBox (2 elements swapped)	Non-biased SBox (3 elements swapped)
Frequency Check	Yes / --	No	No
BALoo	Yes / --	Yes / --	Yes / 3
PN-based	Yes / --	Yes / 3*	Yes / 4*
Improved PN-based	Yes / --	Yes / --	Yes / 4*
CRC-based	Yes / 2*	Yes / 3*	Yes / 4*

Thank you!

Any questions? 🤔

Appendix

Why not duplication ?

- High chance to be bypassed
  - Inject a fault in first SBox
  - Try to inject the same fault in second SBox at different locations until bypass

- Essential to have stronger countermeasures ! 💪

Algo: BALoo

Algorithm 1 BALoo countermeasure

Require: SBox $\mathbf{S}$, number of cycles s, starting indices $I[0], \ldots, I[s-1]$ and lengths $L[0], \ldots, L[s-1]$ corresponding to each cycle

Ensure: `False` in case of faulty SBox, `True` otherwise

```
1: t ← False
2: for i from 0 to s − 1 do
3:     ℓ ← 1                                  ▷ Initialize length of the i-th cycle
4:     j ← i
5:     while S[j] ≠ I[i] and ℓ < L[i] + 1 do  ▷ Traverse the i-th cycle
6:         j ← S[j]
7:         ℓ ← ℓ + 1                          ▷ Increment the length by 1
8:     t ← t ∨ ¬(ℓ ≟ L[i])                    ▷ Check if the length is correct
   return ¬t
```

Algo: Frequency Check

Algorithm 2 Frequency checking countermeasure

Require: SBox $\mathbf{S}$ and its number of elements n

Ensure: `False` in case of faulty SBox, `True` otherwise

$D = (0, \ldots, 0)$ ▷ Initialization of frequency list by n zeros

for i from 0 to $n-1$ **do**

  $D[\mathbf{S}[i]] \leftarrow D[\mathbf{S}[i]] + 1$ ▷ Increment the frequency by 1

$t \leftarrow$ `False`

for i from 0 to $n-1$ **do**

  $t \leftarrow t \vee \neg(D[i] \stackrel{?}{=} 1)$ ▷ Check if the frequency is 1

return $\neg t$

Algo: PN-based Countermeasure

Algorithm 4 First version of PN-based countermeasure

Require: SBox $\mathbf{S}$, its bit length m, control bits $c[0], \ldots, c[2^m(m-1/2)]$

Ensure: `False` in case of faulty SBox, `True` otherwise

$n \leftarrow 1 \ll m$ ▷ Number of elements

$g \leftarrow 1 \ll (m-1)$ ▷ Number of swap gates in each layer

$\pi \leftarrow (0, \ldots, n-1)$

for i from 0 to $2m-2$ **do** ▷ i-th layer

  $\Delta \leftarrow 1 \ll \min(i, 2m-i-2)$ ▷ Gap between two elements of a gate in i-th layer

  for j from 0 to $g-1$ **do**

    if $c[ig+j] = 1$ **then**

      $l \leftarrow (j \mod \Delta) + 2\Delta\lfloor j/\Delta \rfloor$ ▷ Smaller index in two elements

      swap $\pi[l]$ and $\pi[l+\Delta]$

$t \leftarrow$ `False`

for i from 0 to $n-1$ **do**

  $t \leftarrow t \vee \neg(\mathbf{S}[i] \stackrel{?}{=} \pi[i])$ ▷ Compare π and S

return $\neg t$

Algo: Improved PN-based Countermeasure

Algorithm 5 Improved version of PN-based countermeasure

Require: SBox $\mathbf{S}$, its bit length m, list $\mathcal{D} = \{v_0, v_1, \ldots, v_{|\mathcal{D}|}\}$, where $v_0 < v_1 < \ldots < v_{|\mathcal{D}|}$, of indices corresponding to control bits 1

Ensure: `False` in case of faulty SBox, `True` otherwise

$n \leftarrow 1 \ll m$ ▷ Number of elements
$g \leftarrow 1 \ll (m-1)$ ▷ Number of swap gates in each layer
$\pi \leftarrow (0, \ldots, n-1)$
for each v in $\mathcal{D}$ **do**
  $i \leftarrow \lfloor v/g \rfloor$ ▷ i-th layer
  $j \leftarrow v \mod g$ ▷ j-th swap gate
  $\Delta \leftarrow 1 \ll \min(i, 2m-i-2)$ ▷ Gap between two elements of j-th gate
  $l \leftarrow (j \mod \Delta) + 2\Delta\lfloor j/\Delta \rfloor$ ▷ Smaller index in two elements
  swap $\pi[l]$ and $\pi[l+\Delta]$
$t \leftarrow$ `False`
for i from 0 to $n-1$ **do**
  $t \leftarrow t \vee \neg(\mathbf{S}[i] \stackrel{?}{=} \pi[i])$ ▷ Compare π and S
return $\neg t$

Algo: CRC-based Countermeasure

Algorithm 6 CRC-based countermeasure

Require: Bitstring of data $T[0..(L-1)]$ (SBox elements and redundancy) and coefficients of generator polynomial $G[0..p]$ ($G[0]$ is the coefficient of x^p)

Ensure: `False` in case of faulty SBox, `True` otherwise

1: $r[0..(p-1)] \leftarrow (0..0)$ ▷ Initialize reminder
2: **for** i from 0 to $L-1$ **do**
3: $\quad r \leftarrow r \oplus (T[i] \ll (p-1))$
4: $\quad r \leftarrow r \ll 1$
5: $\quad$ **if** $r[0] = 1$ **then**
6: $\quad\quad r \leftarrow r \oplus G[1..p]$
return $(r \stackrel{?}{=} 0)$ ▷ Verify if reminder is 0