More Efficient Private Circuits II Through Threshold Implementations

Thomas De Cnudde

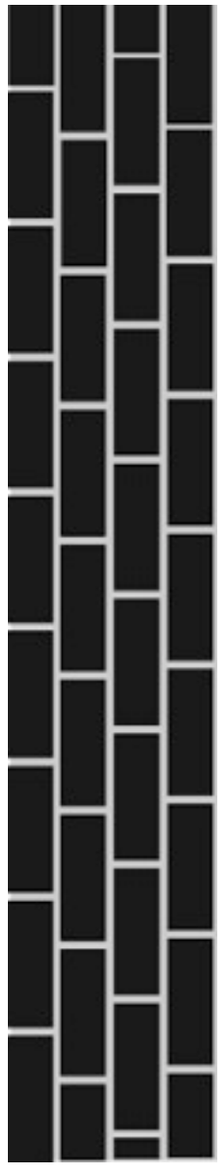
Svetla Nikova

08/16 – FDTC 2016 – Santa Barbara

Investigating a novel approach towards a hardware implementation resisting combined SCA and FAs

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

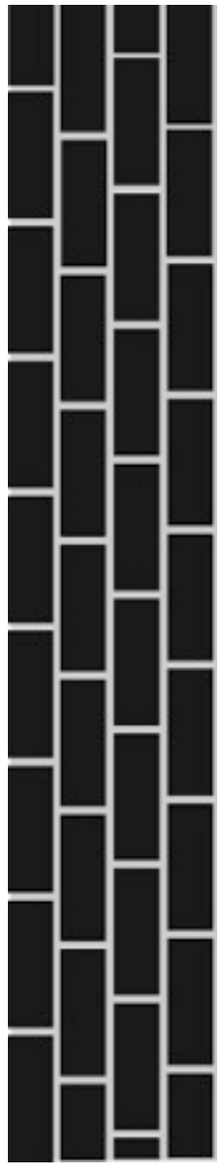


FA Countermeasures

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

Masking



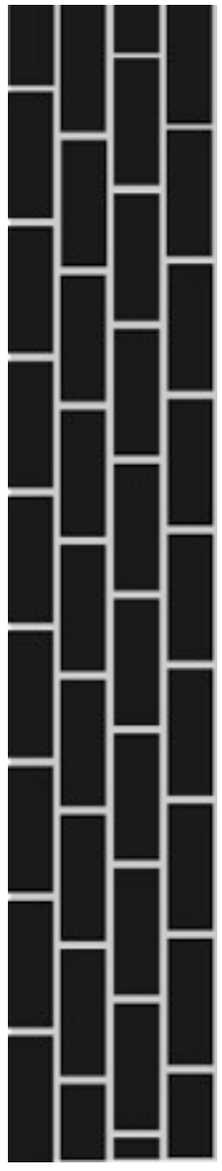
FA Countermeasures

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

Masking

Hiding



FA Countermeasures

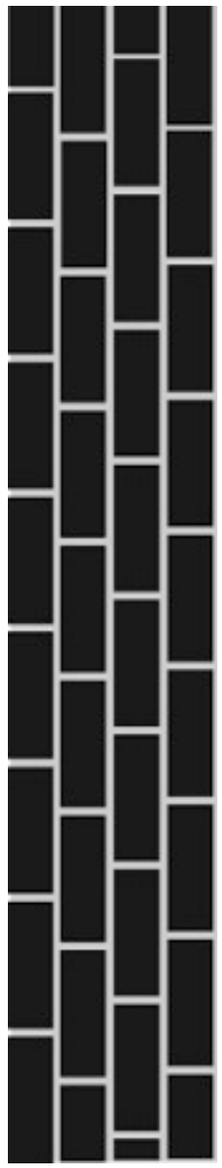
Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

Masking

Private Circuits

Hiding



FA Countermeasures

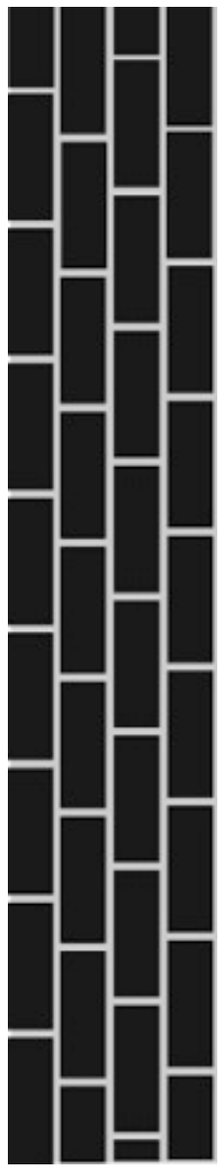
Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

Masking

- Private Circuits
- Threshold Implementations

Hiding

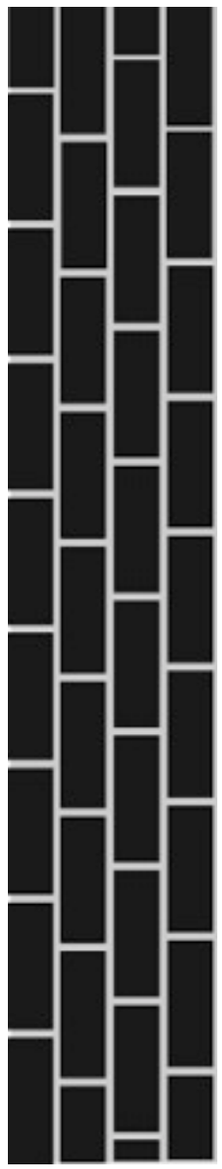


FA Countermeasures

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

- Masking
  - Private Circuits
  - Threshold Implementations
  - ...
- Hiding

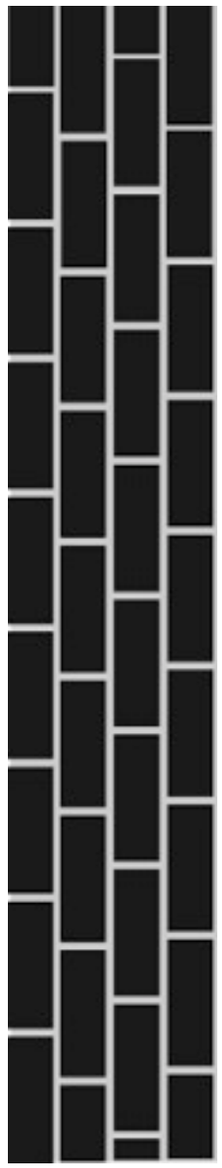


FA Countermeasures

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

- Masking
  - Private Circuits
  - Threshold Implementations
  - ...
- Hiding



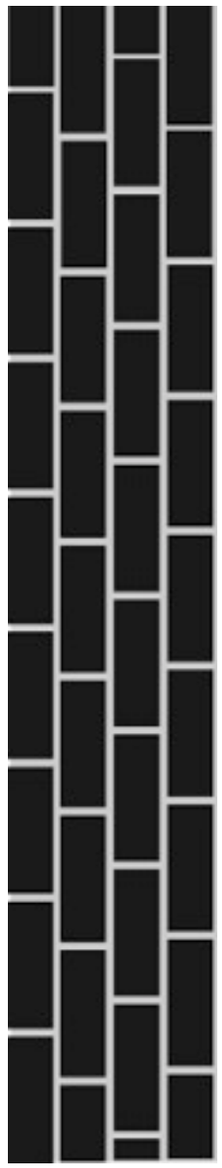
FA Countermeasures

- Temporal or Spatial redundancy

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

- Masking
  - Private Circuits
  - Threshold Implementations
  - ...
- Hiding



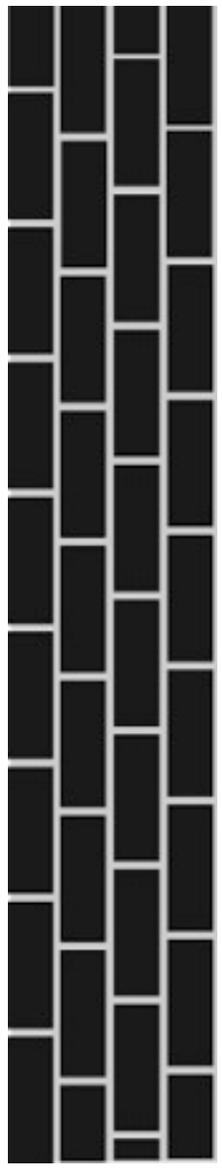
FA Countermeasures

- Temporal or Spatial redundancy
- Error correction/ detection

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

- Masking
  - Private Circuits
  - Threshold Implementations
  - ...
- Hiding



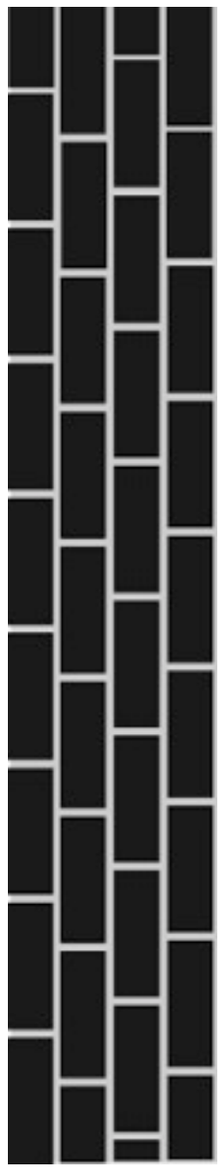
FA Countermeasures

- Temporal or Spatial redundancy
- Error correction/ detection
- Infective computing

Countermeasures for SCA and FA are generally researched separately

SCA Countermeasures

- Masking
  - Private Circuits
  - Threshold Implementations
  - ...
- Hiding



FA Countermeasures

- Temporal or Spatial redundancy
- Error correction/ detection
- Infective computing
- Sensors

Private Circuits II provides resistance against combined SCA and FA

SCA Countermeasures

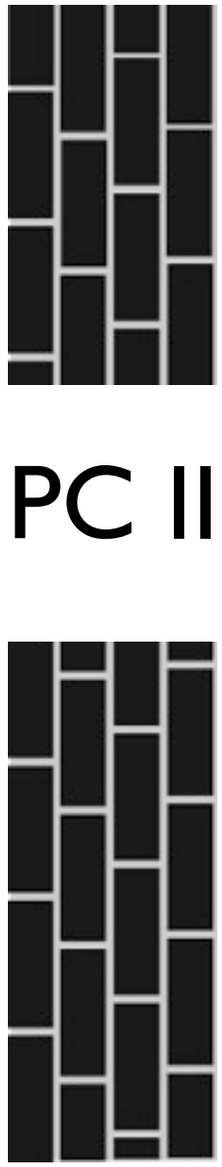
Masking

Private Circuits

Threshold Implementations

...

Hiding



PC II

FA Countermeasures

Temporal or Spatial redundancy

Error correction/ detection

Infective computing

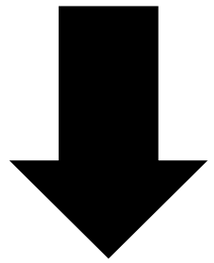
Sensors

Applying Private Circuits II requires a series of transformation

PRESENT

Applying Private Circuits II requires a series of transformation

PRESENT

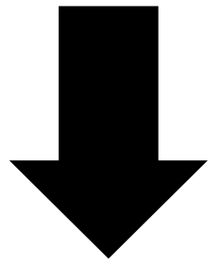


obtaining SCA resistance

Private Circuits

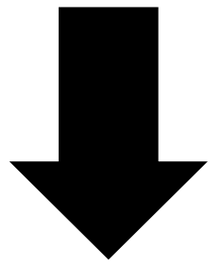
Applying Private Circuits II requires a series of transformation

PRESENT



obtaining SCA resistance

Private Circuits



obtaining combined SCA and FA resistance

Private Circuits II

Private Circuits and Threshold Implementations are closely related

Private Circuits ⟷ Threshold Implementations

Private Circuits and Threshold Implementations are closely related

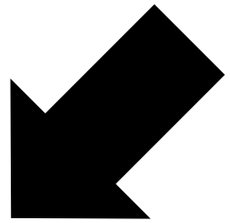
Private Circuits ⟷ Threshold Implementations

Glitches not allowed

Glitches allowed

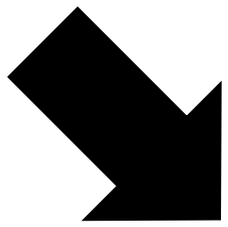
Combined SCA and FA resistance for the PRESENT block cipher

PRESENT



Private Circuits

Threshold Implementations



Private Circuits II

Combined SCA and FA resistance for the PRESENT block cipher

PRESENT

Private Circuits

Threshold Implementations

Private Circuits II

Combined SCA and FA resistance for the PRESENT block cipher

PRESENT

Private Circuits

Threshold Implementations

Which approach is more efficient in HW ?

PC and TI are both boolean masking schemes

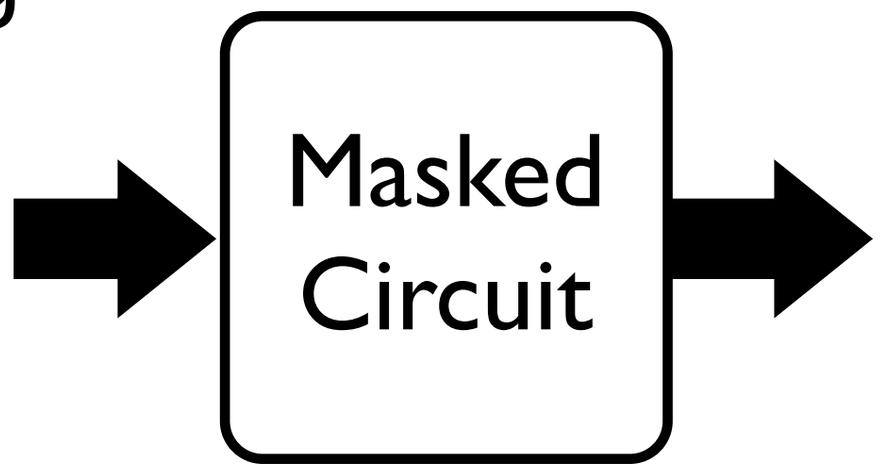
Input Encoding

$I_1 = \text{Input} + R_1 + R_2$
$I_2 = R_1$
$I_3 = R_2$

PC and TI are both boolean masking schemes

Input Encoding

I_1 = Input + R_1 + R_2
I_2 = R_1
I_3 = R_2



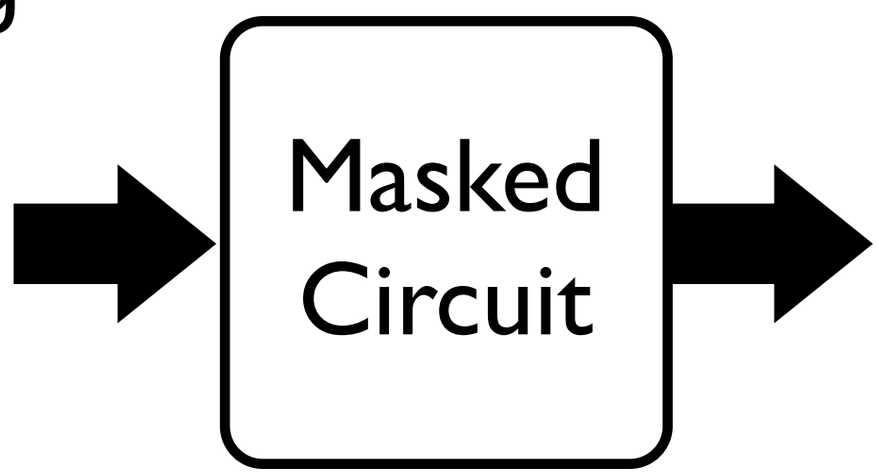
PC and TI are both boolean masking schemes

Input Encoding

$I_1 = Input + R_1 + R_2$

$I_2 = R_1$

$I_3 = R_2$



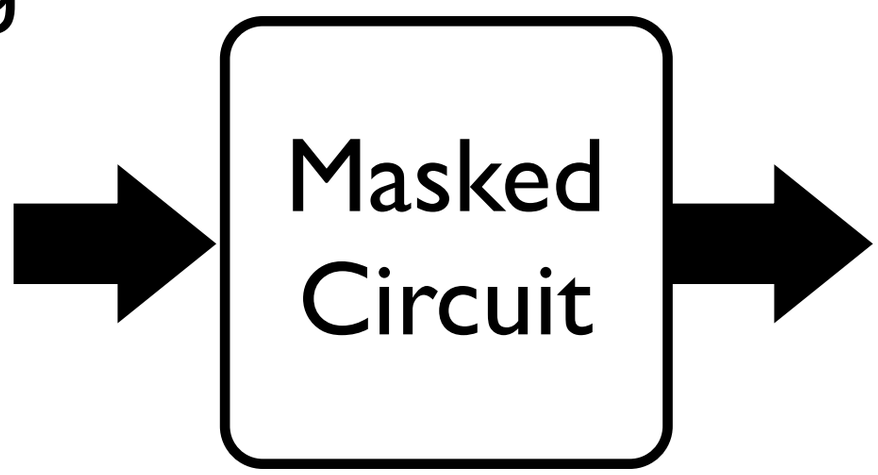
Output Decoding

$O_1 + O_2 + O_3 = Output$

PC and TI are both boolean masking schemes

Input Encoding

$I_1 = Input + R_1 + R_2$
$I_2 = R_1$
$I_3 = R_2$

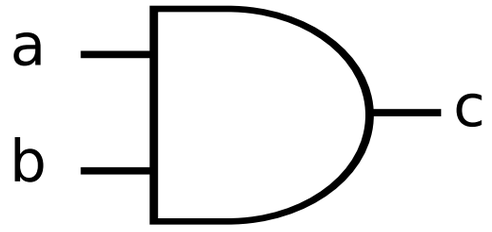


Output Decoding

$O_1 + O_2 + O_3 = Output$

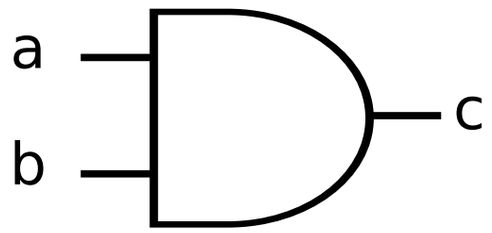
Linear operations are performed on individual shares

PC and TI differ in the nonlinear operations



PC and TI differ in the nonlinear operations

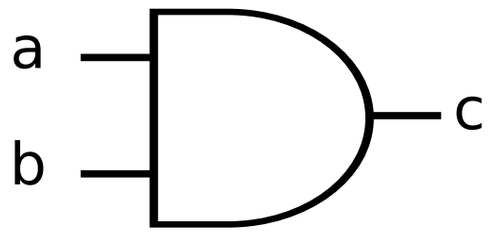
Private Circuits:



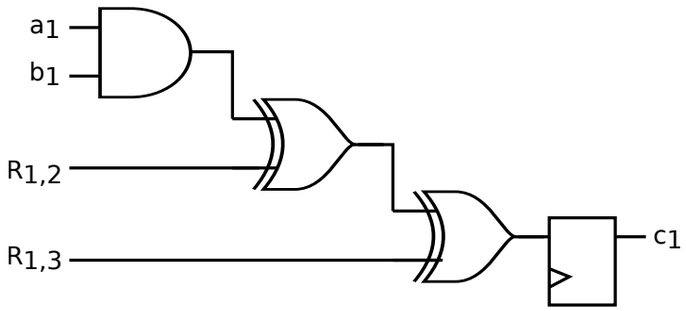
1) Generate $R_{1,2}$ $R_{1,3}$ $R_{2,3}$
2) Compute $R_{2,1} = R_{1,2}+a_1b_2$
   $R_{3,1} = R_{1,3}+a_1b_3$
3) Compute $c_1 = a_1b_1+R_{1,2}+R_{1,3}$
   $c_2 = a_2b_2+R_{2,1}+a_2b_1+R_{2,3}$
   $c_3 = a_3b_2+R_{3,1}+a_3b_1+R_{2,1}+a_2b_1$

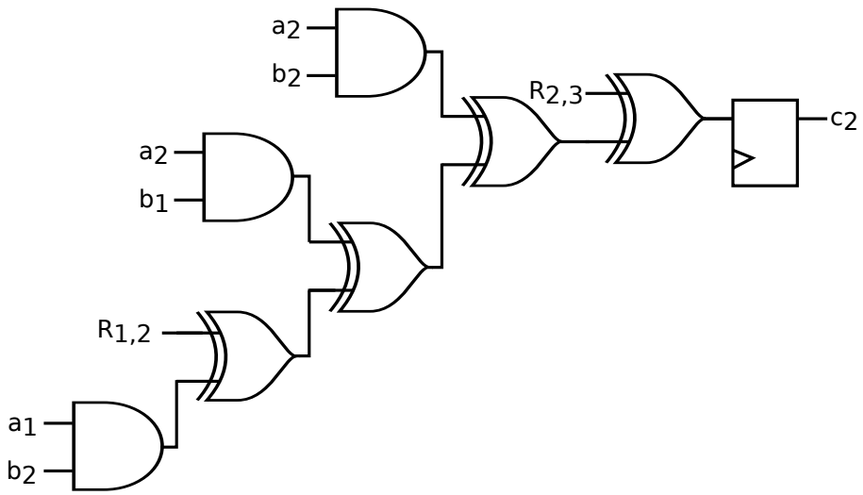
PC and TI differ in the nonlinear operations

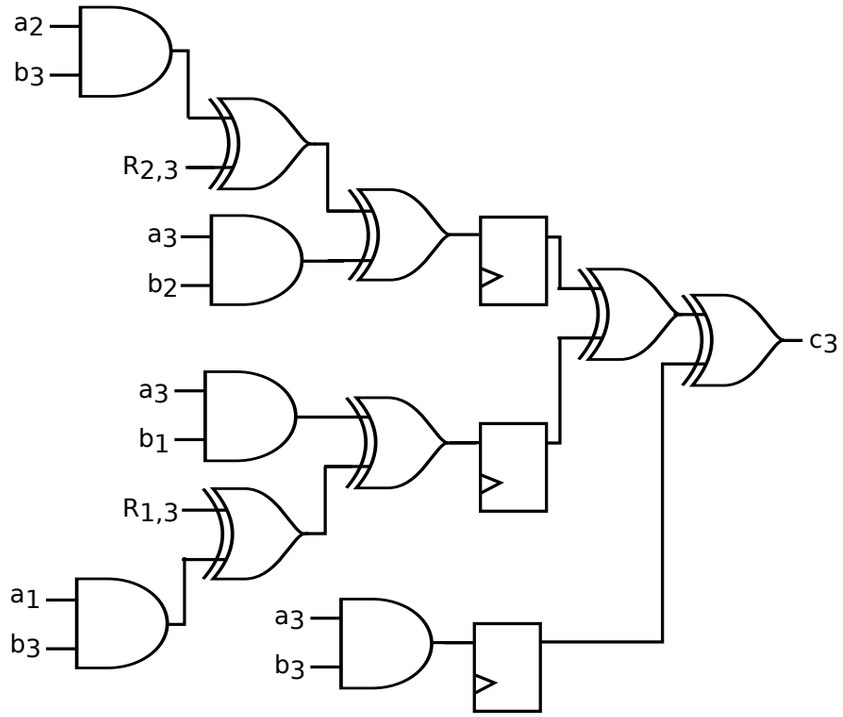
Private Circuits:



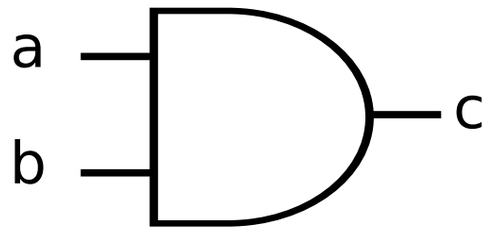
1) Generate $R_{1,2}$ $R_{1,3}$ $R_{2,3}$
2) Compute $R_{2,1} = R_{1,2}+a_1b_2$
   $R_{3,1} = R_{1,3}+a_1b_3$
3) Compute $c_1 = a_1b_1+R_{1,2}+R_{1,3}$
   $c_2 = a_2b_2+R_{2,1}+a_2b_1+R_{2,3}$
   $c_3 = a_3b_2+R_{3,1}+a_3b_1+R_{2,1}+a_2b_1$





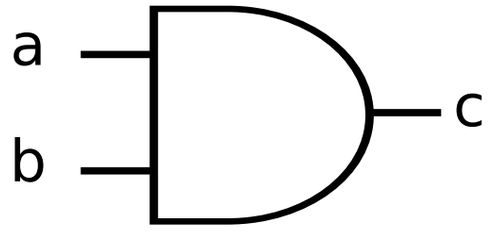


PC and TI differ in the nonlinear operations



PC and TI differ in the nonlinear operations

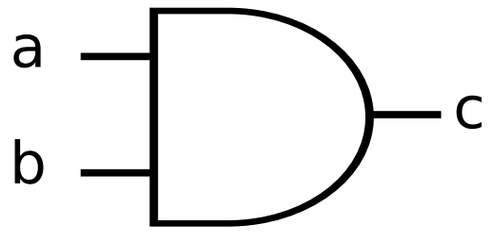
Threshold Implementations:



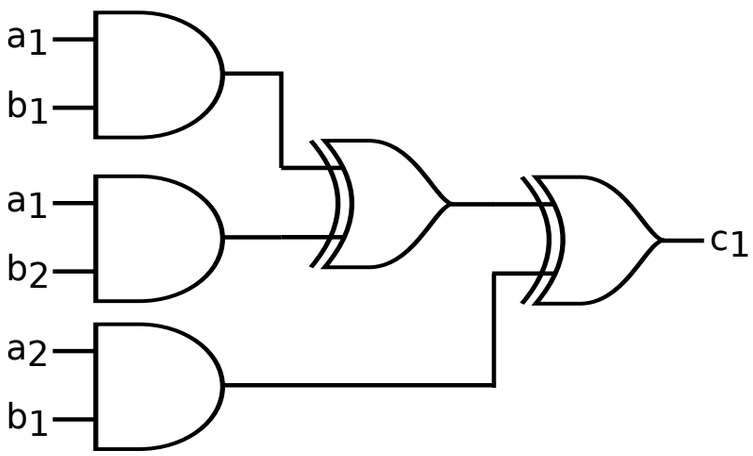
$c_1 = a_2b_2 + a_1b_2 + a_2b_1$
$c_2 = a_3b_3 + a_3b_2 + a_2b_3$
$c_3 = a_1b_1 + a_1b_3 + a_3b_1$

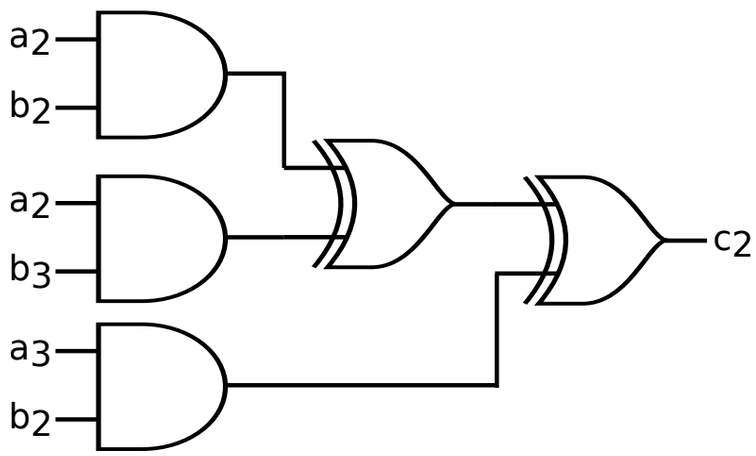
PC and TI differ in the nonlinear operations

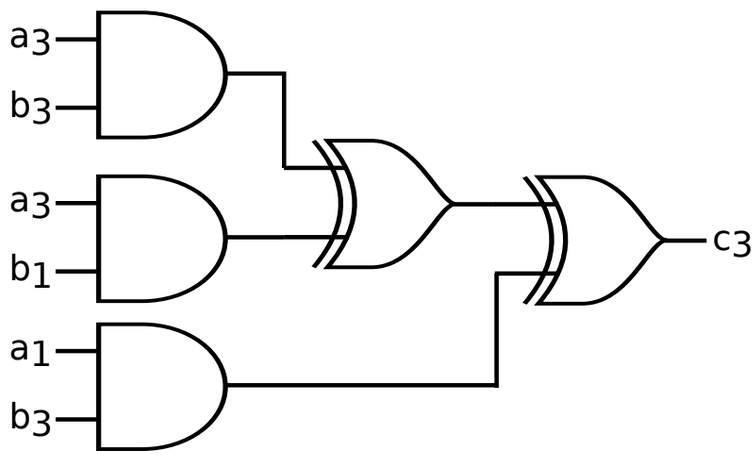
Threshold Implementations:



$c_1 = a_2b_2 + a_1b_2 + a_2b_1$
$c_2 = a_3b_3 + a_3b_2 + a_2b_3$
$c_3 = a_1b_1 + a_1b_3 + a_3b_1$

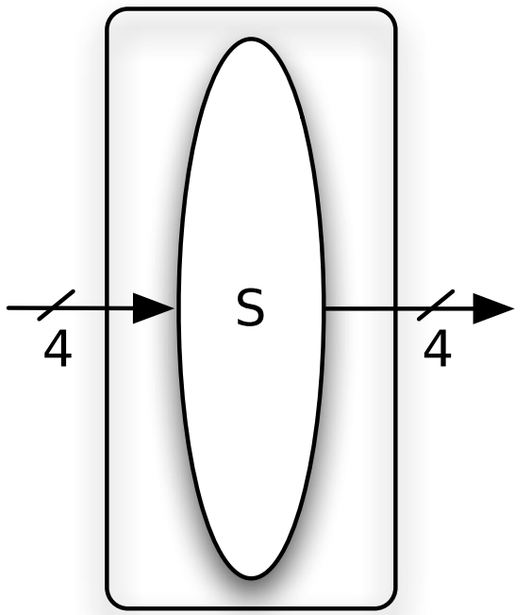






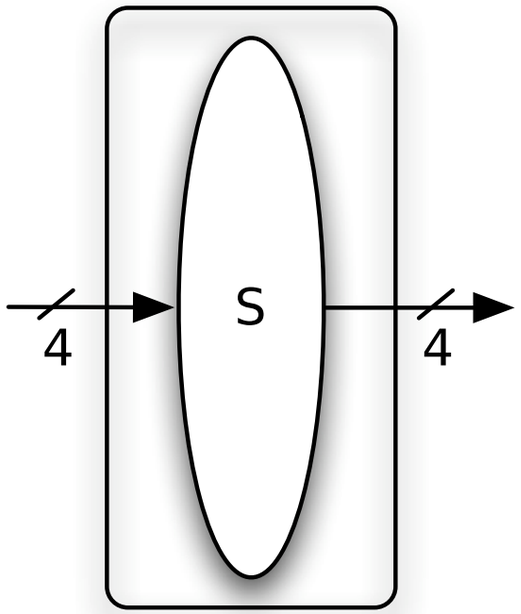
Implementing PRESENT S-box with Private Circuits

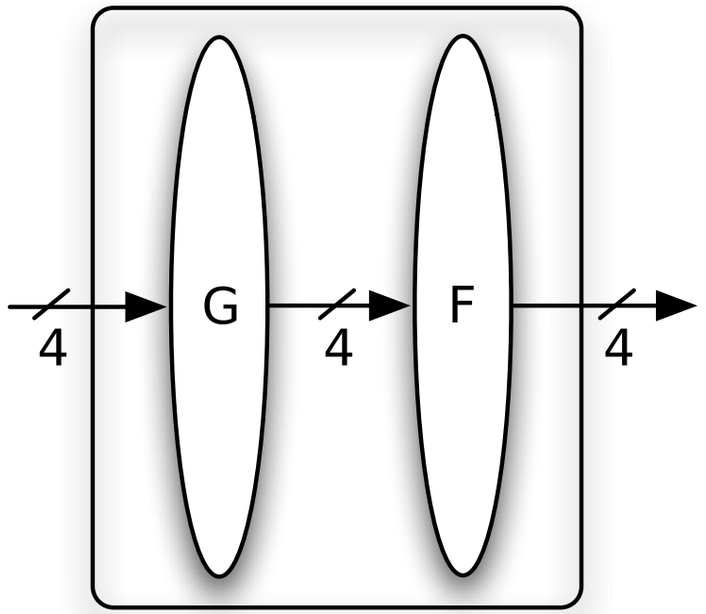
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2



Implementing PRESENT S-box with Private Circuits

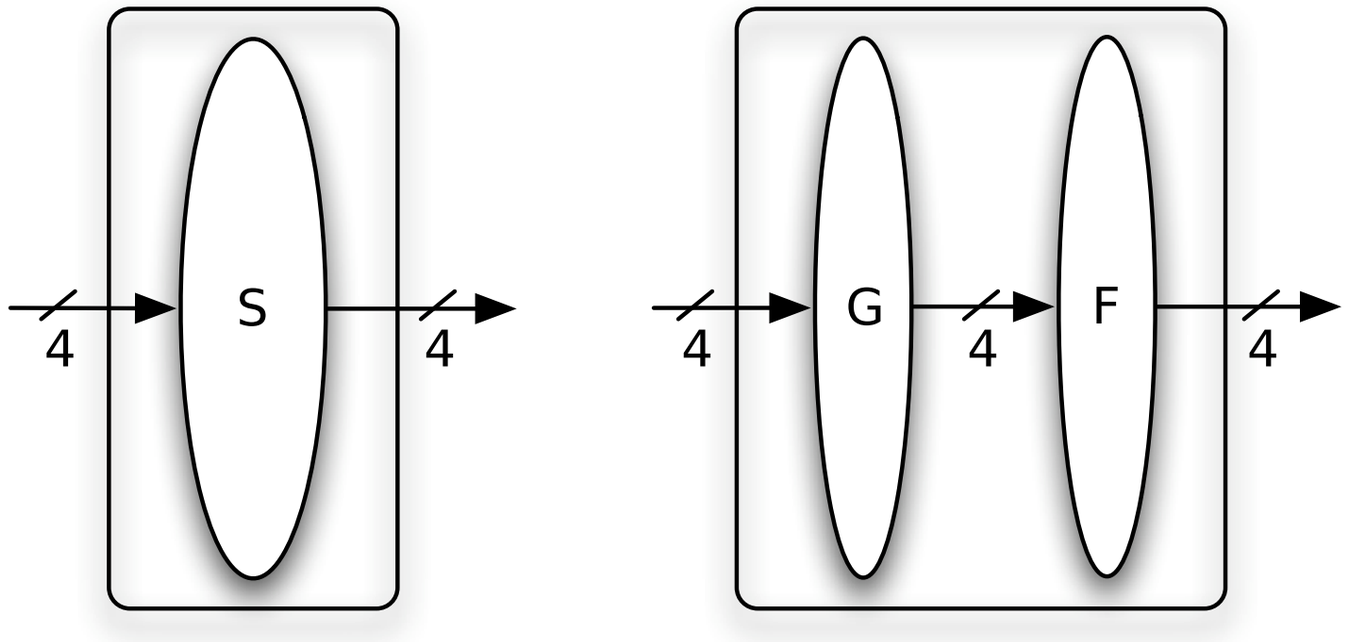
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2
G(x)	7	E	9	2	B	0	4	D	5	C	A	1	8	3	6	F
F (x)	0	8	B	7	A	3	1	C	4	6	F	9	E	D	5	2





Implementing PRESENT S-box with Private Circuits

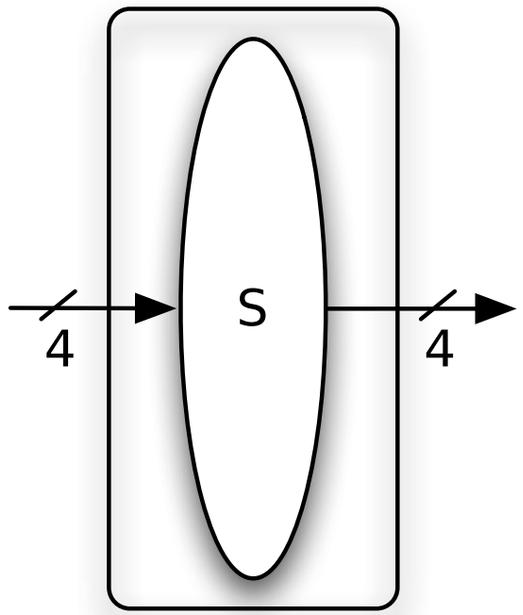
x	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S(x)	C	5	6	B	9	0	A	D	3	E	F	8	4	7	1	2
G(x)	7	E	9	2	B	0	4	D	5	C	A	1	8	3	6	F
F (x)	0	8	B	7	A	3	1	C	4	6	F	9	E	D	5	2

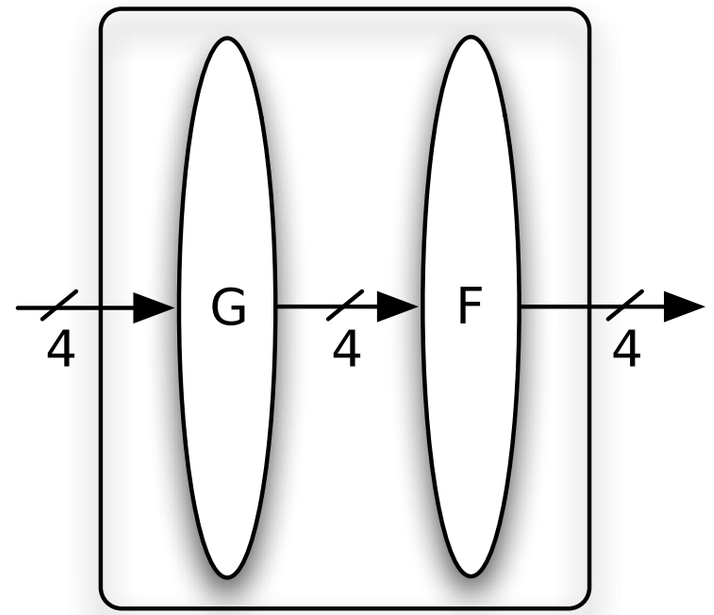


23 AND gadgets
23 XOR gadgets

Implementing PRESENT S-box with Private Circuits

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

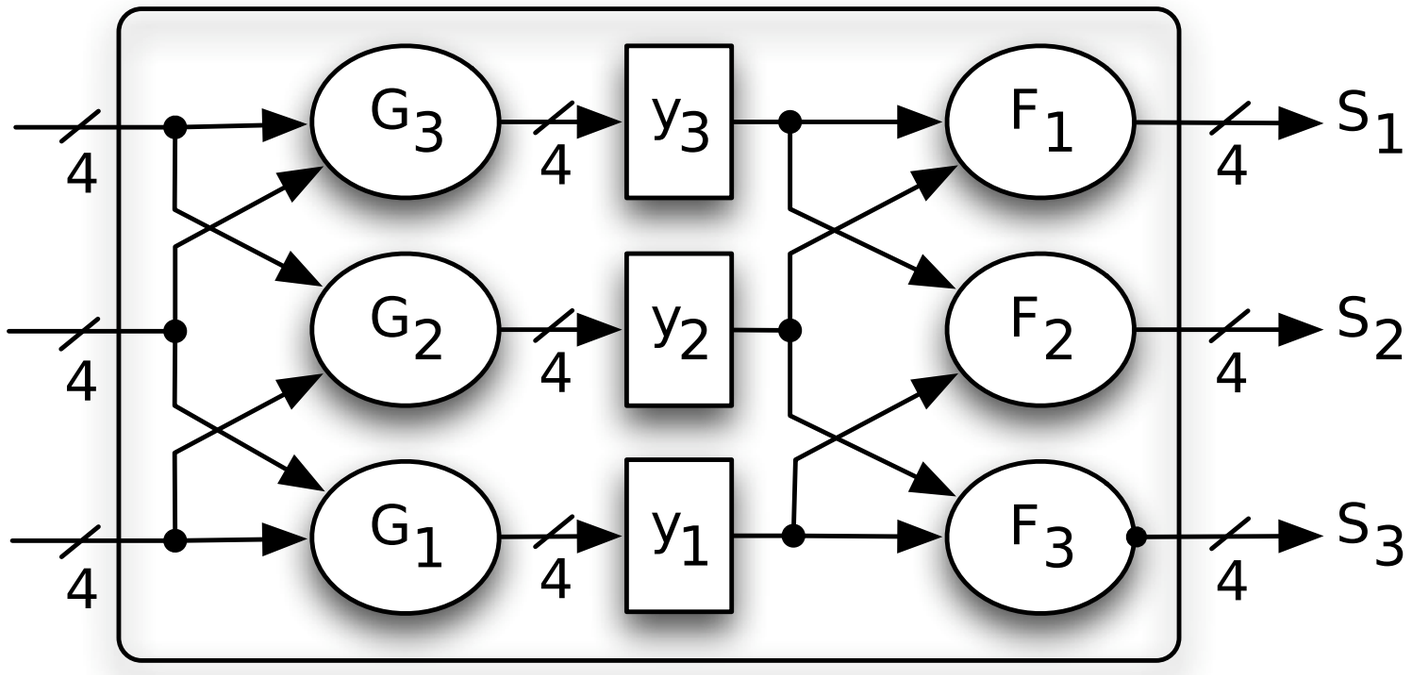




23 AND gadgets
23 XOR gadgets

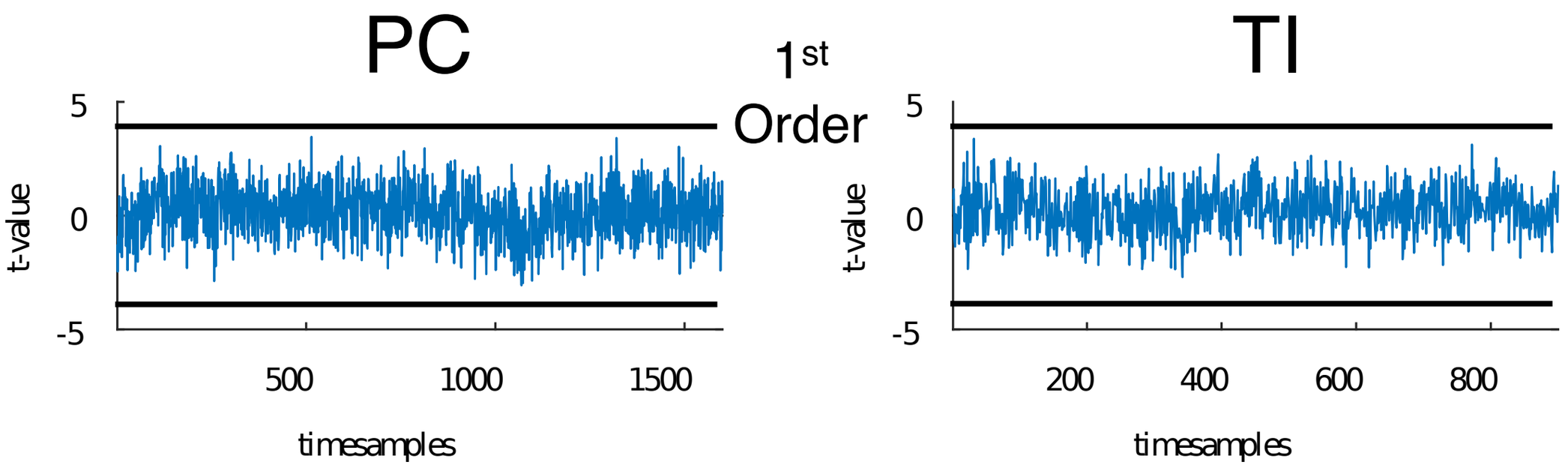
9 AND gadgets
19 XOR gadgets

Implementing PRESENT S-box with Threshold Implementations

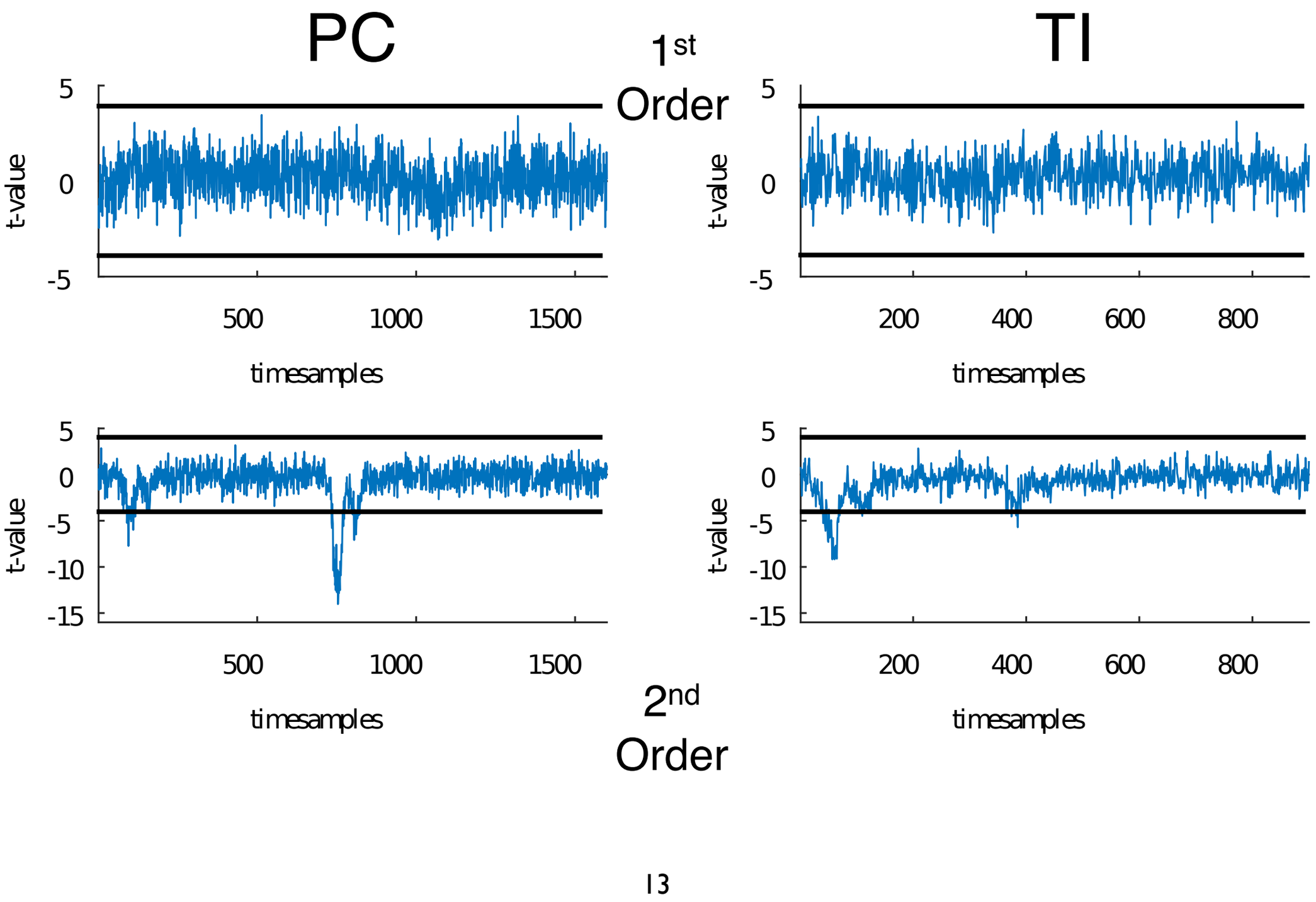


(Poschmann, 2011)

PC and TI achieve equivalent security with 25 Million traces



PC and TI achieve equivalent security with 25 Million traces

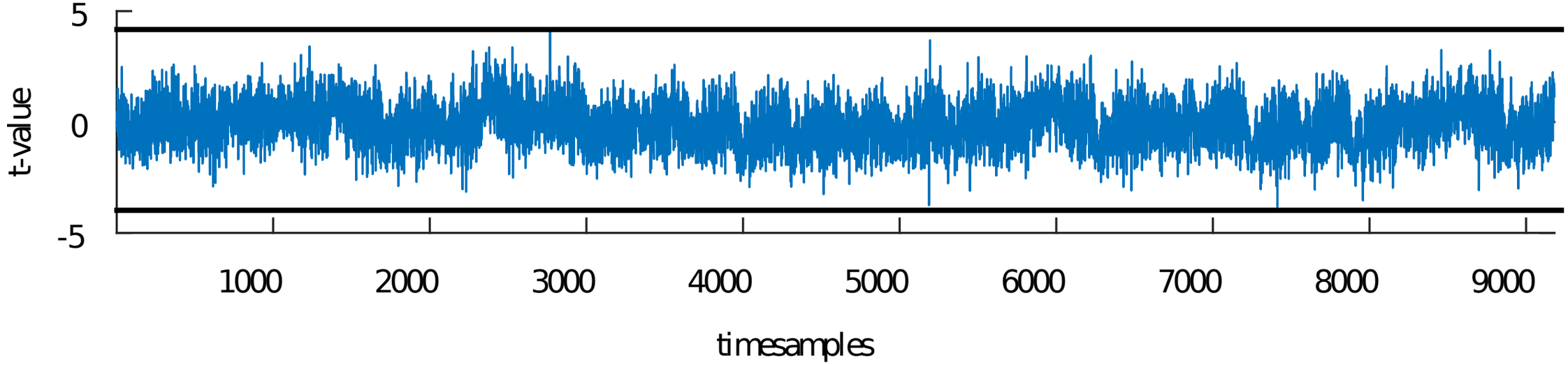


Threshold Implementations is less costly in all aspects

	PC-I	TI
Number of Slices	107	29
Number of Slice Flip Flops	166	48
Number of 4 input LUTs	96	57
Consumed Random Bits	28	0
Number of Clock Cycles	4	2

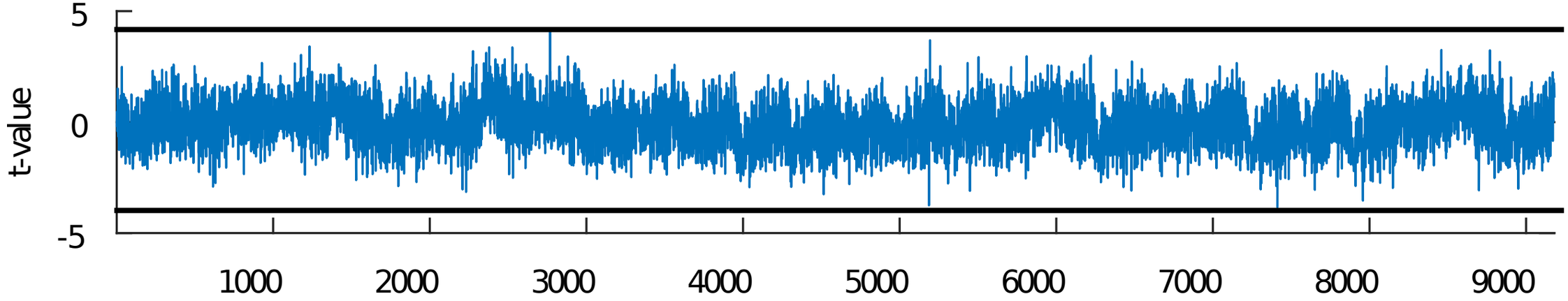
PRESENT-TI achieves its security with 100 Million traces

1st Order

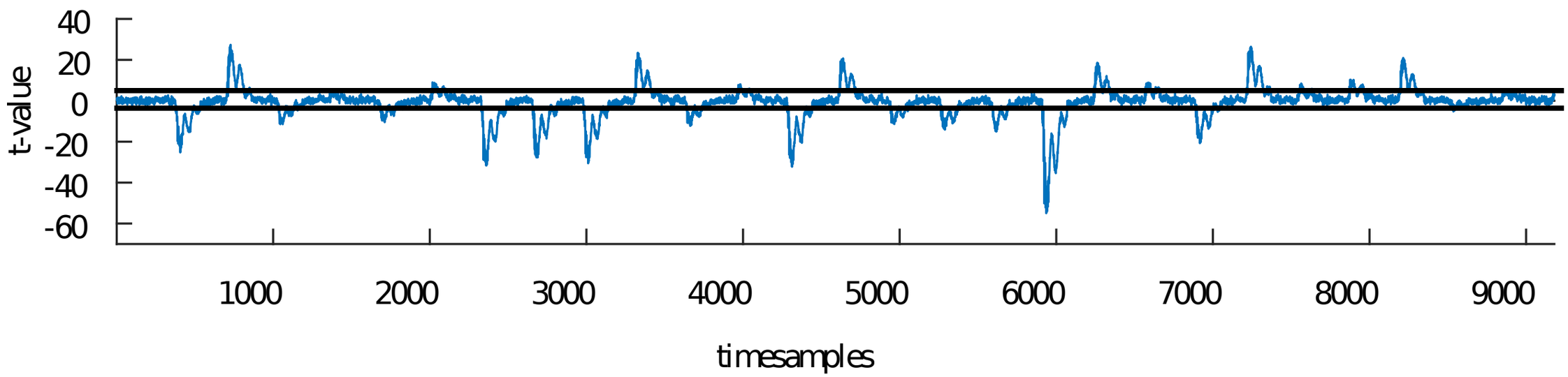


PRESENT-TI achieves its security with 100 Million traces

1st Order

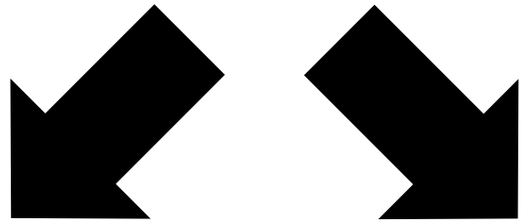


2nd Order



Combined SCA and FA resistance for the PRESENT block cipher

PRESENT

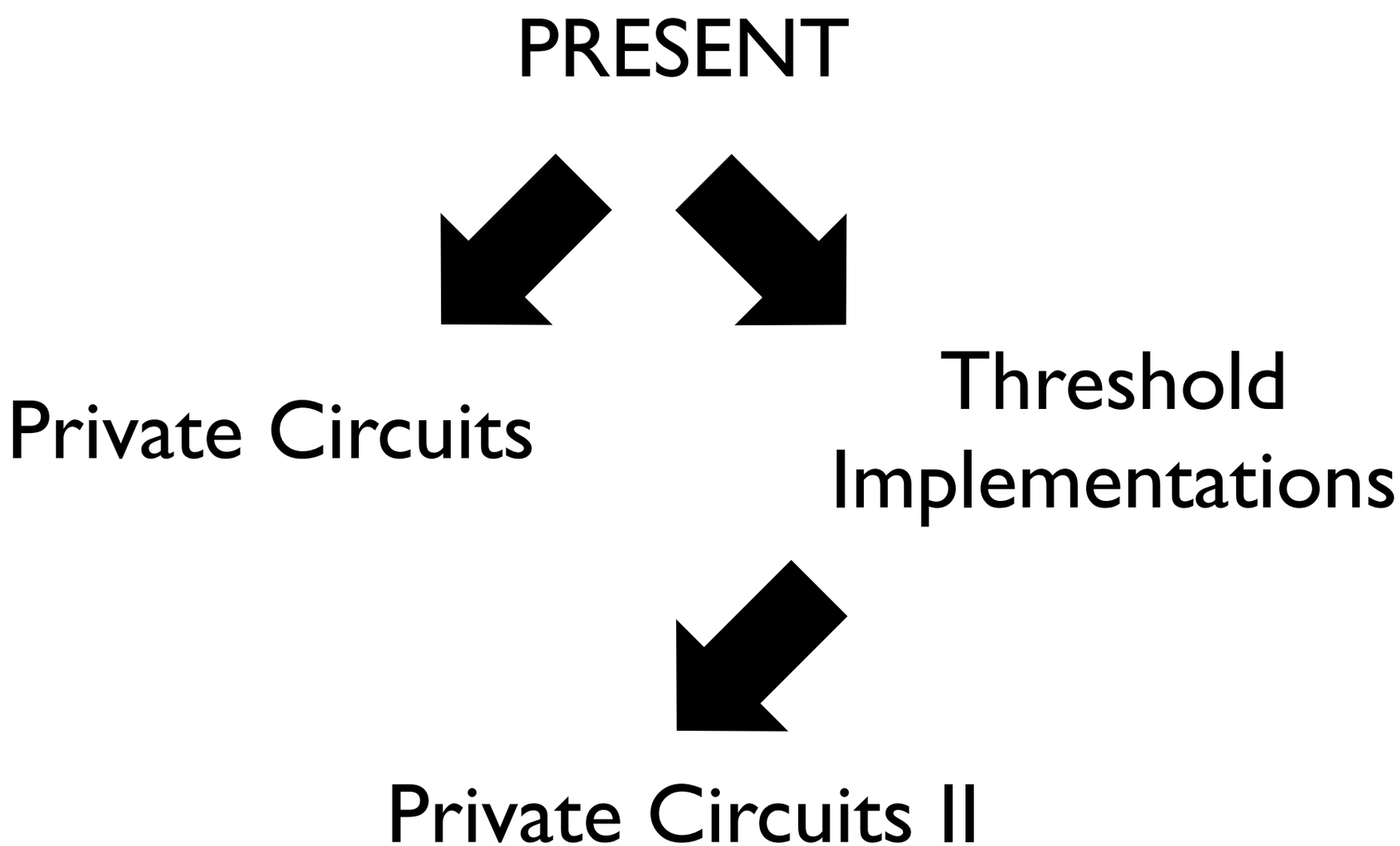


Private Circuits

Threshold Implementations

Private Circuits II

Combined SCA and FA resistance for the PRESENT block cipher



Combined SCA and FA resistance for the PRESENT block cipher

PRESENT

Private Circuits

Threshold Implementations

Private Circuits II

Private Circuits II comes in two variants of FA resistance

1) Resisting any number of reset-only wire faults

2) Resisting t arbitrary wire faults

Private Circuits II comes in two variants of FA resistance

1) Resisting any number of reset-only wire faults

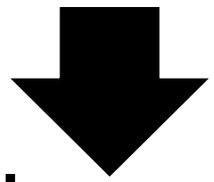
2) Resisting t arbitrary wire faults t = 1

Tamper resistance against any number of reset-only wire faults

SCA Resistant Circuit

Tamper resistance against any number of reset-only wire faults

SCA Resistant Circuit



Manchester Encoding

0 = (0,1)

1 = (1,0)

Tamper resistance against any number of reset-only wire faults

SCA Resistant Circuit

↓

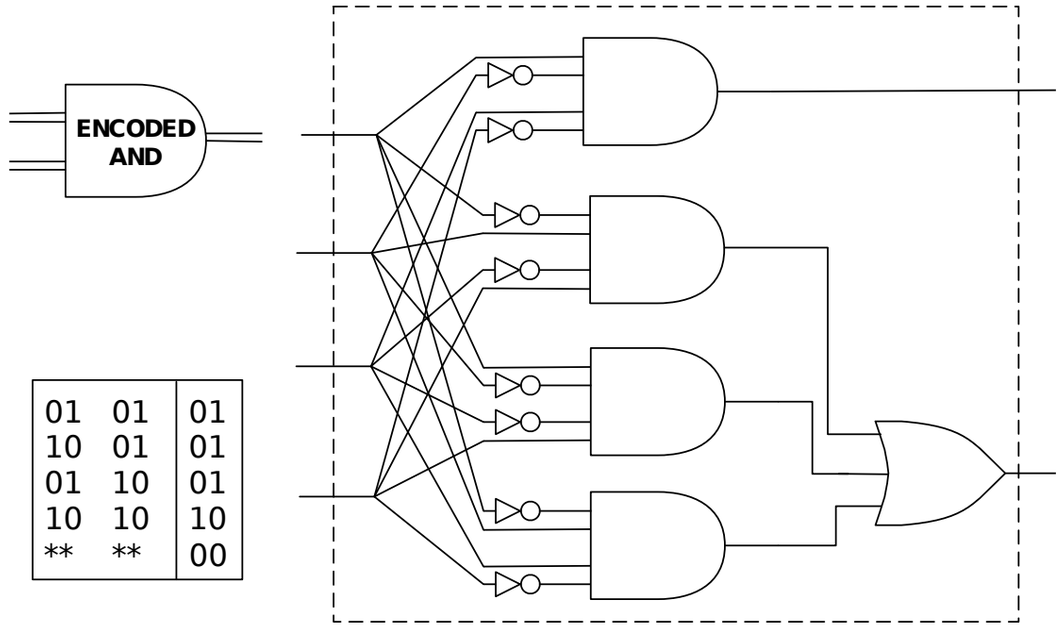
Manchester Encoding

0 = (0,1)
1 = (1,0)

↓

Gadgets Encoding and Error Cascading

propagates invalid 00



01	01	01
10	01	01
01	10	01
10	10	10
**	**	00

Tamper resistance against any number of reset-only wire faults

SCA Resistant Circuit

↓

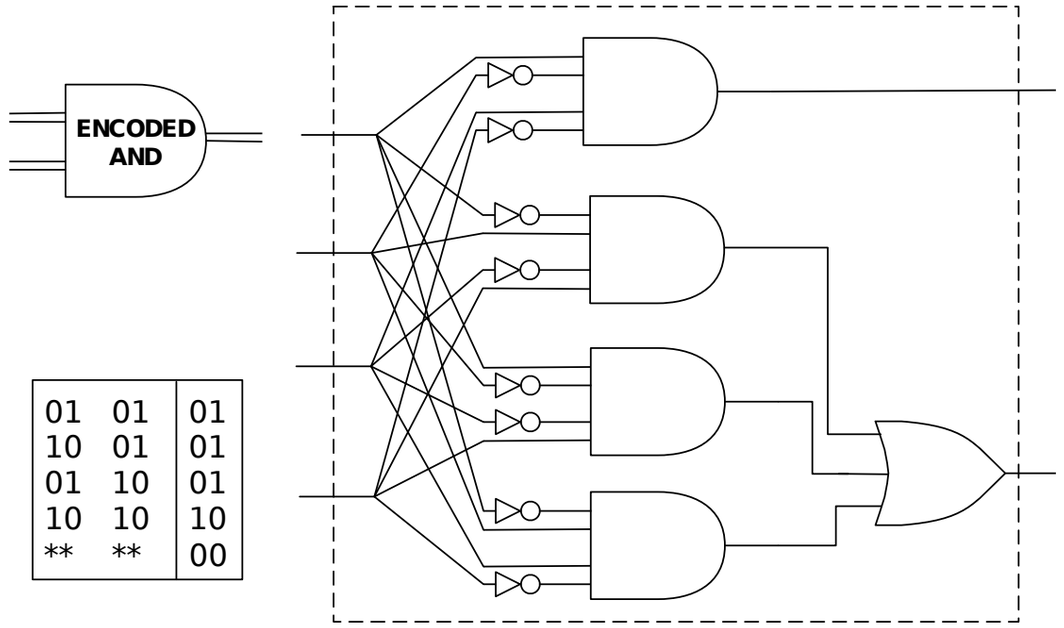
Manchester Encoding

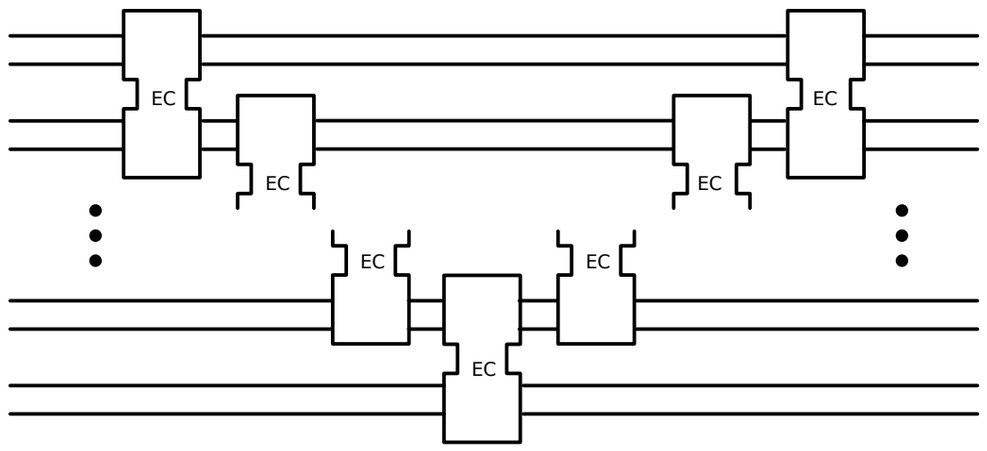
0 = (0,1)
1 = (1,0)

↓

Gadgets Encoding and Error Cascading

propagates invalid 00





Tamper resistance against any number of reset-only wire faults

SCA Resistant Circuit

↓

Manchester Encoding

0 = (0,1)
1 = (1,0)

↓

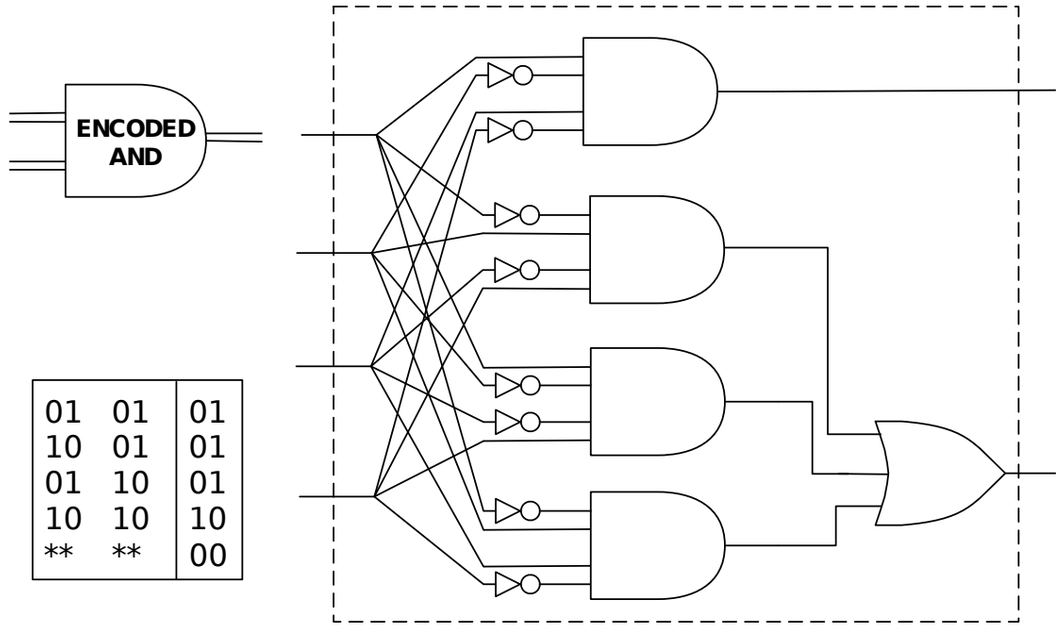
Gadgets Encoding and Error Cascading

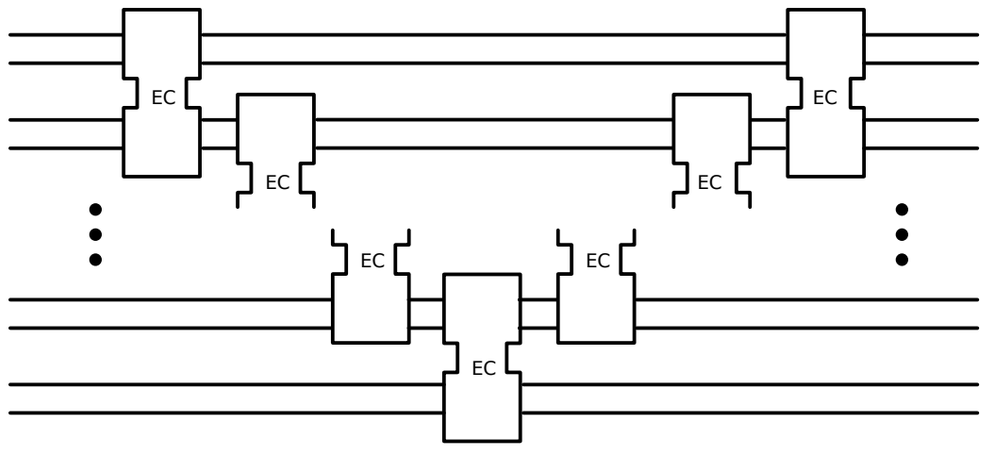
propagates invalid 00

↓

Output Decoding (0,1) = 0

(1,0) = 1





Tamper resistance against one arbitrary wire faults

SCA Resistant Circuit

Tamper resistance against
one arbitrary wire faults

SCA Resistant Circuit

Repetition Encoding

0 = (0,0)

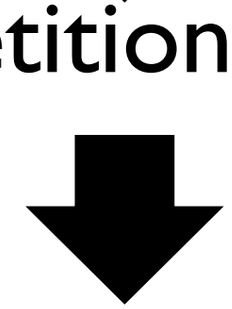
1 = (1,1)

Tamper resistance against one arbitrary wire faults

SCA Resistant Circuit

Repetition Encoding

0 = (0,0)
1 = (1,1)

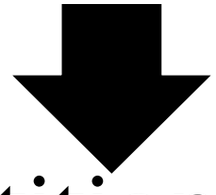


Gadgets Encoding and Error Cascading

propagates invalid 01

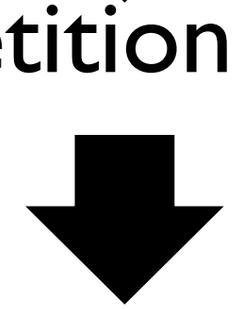
Tamper resistance against
one arbitrary wire faults

SCA Resistant Circuit

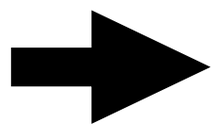


Repetition Encoding

0 = (0,0)
1 = (1,1)



Gadgets Encoding and Error Cascading



OR of ANDs form

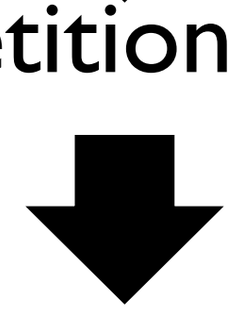
propagates invalid 01

Tamper resistance against
one arbitrary wire faults

SCA Resistant Circuit

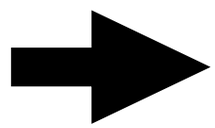
Repetition Encoding

0 = (0,0)
1 = (1,1)



Gadgets Encoding and Error Cascading

propagates invalid 01



OR of ANDs form

NOT gate is reversible

fault at output propagates to the input

Tamper resistance against
one arbitrary wire faults

SCA Resistant Circuit

↓

Repetition Encoding

0 = (0,0)
1 = (1,1)

↓

Gadgets Encoding and Error Cascading

→

OR of ANDs form

NOT gate is reversible

fault at output propagates to the input

propagates invalid 01

↓

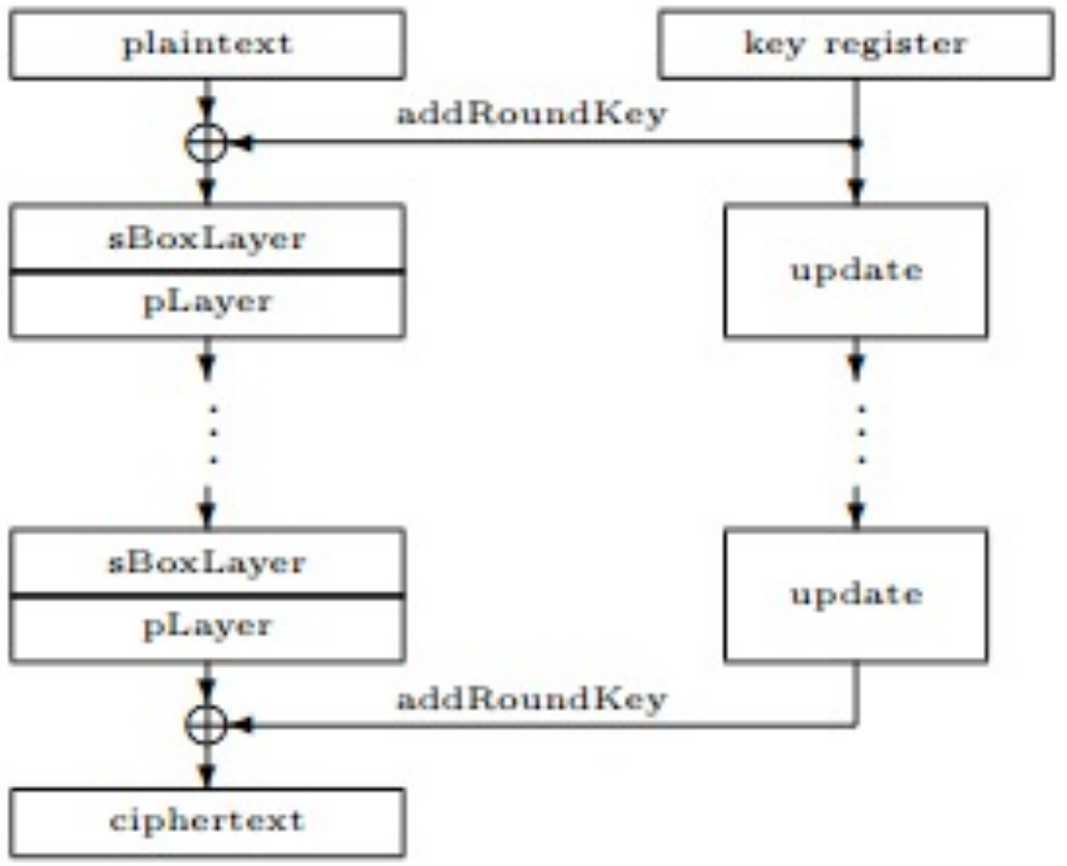
Output Decoding

(0,0) = 0

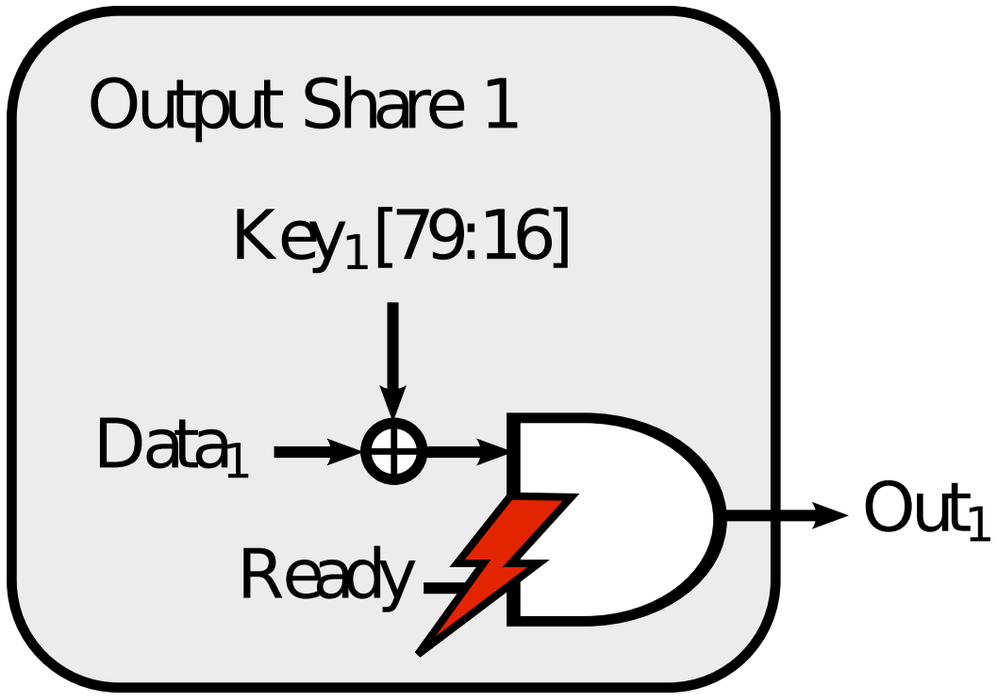
(1,1) = 1

For SCA resistance only the data dependent values need to be masked

generateRoundKeys()
for $i = 1$ to 31 **do**
  addRoundKey(STATE,K_i)
  sBoxLayer(STATE)
  pLayer(STATE)
end for
addRoundKey(STATE,K_{32})

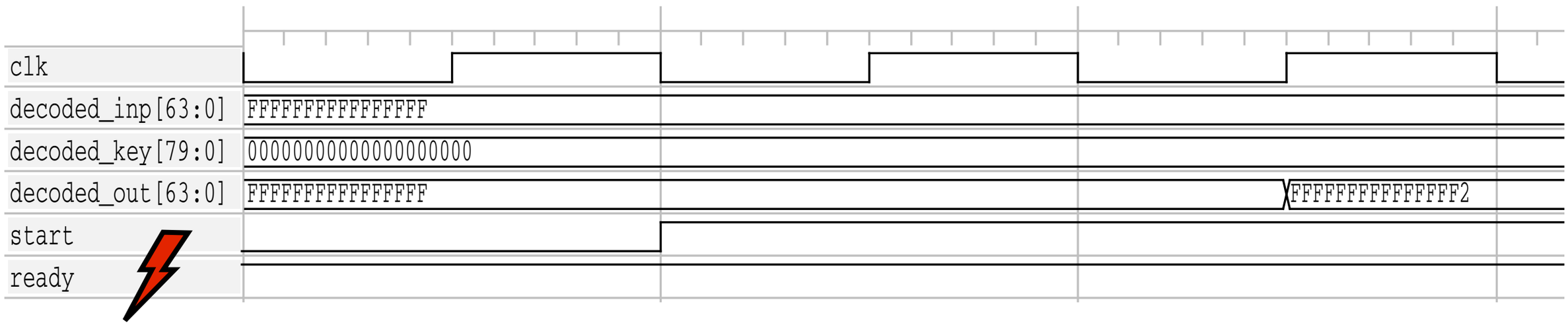


With FA, control signals can be the target of Fault Injection



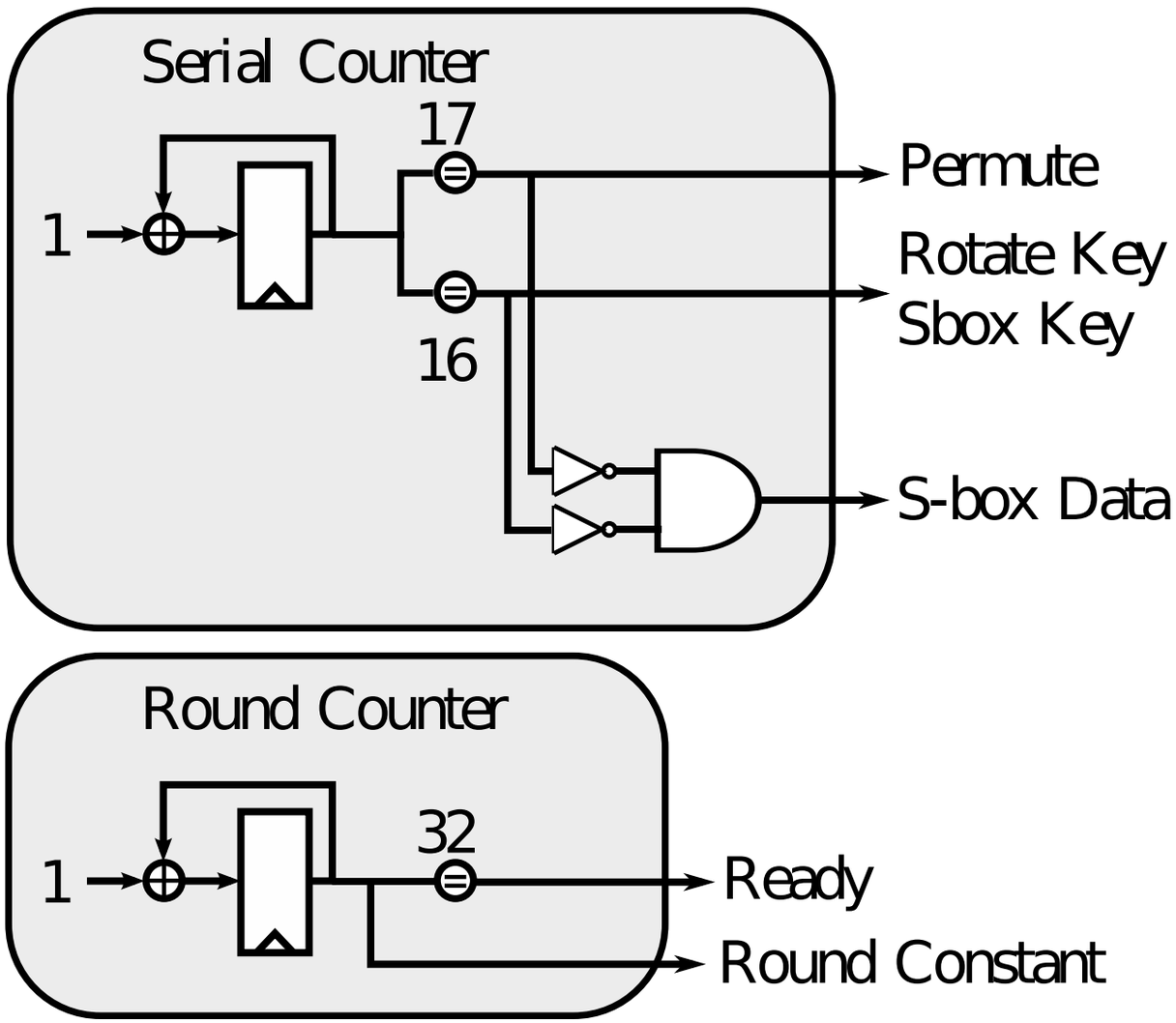
Fault on ready signal can reveal all intermediate results

With FA, control signals can be the target of Fault Injection

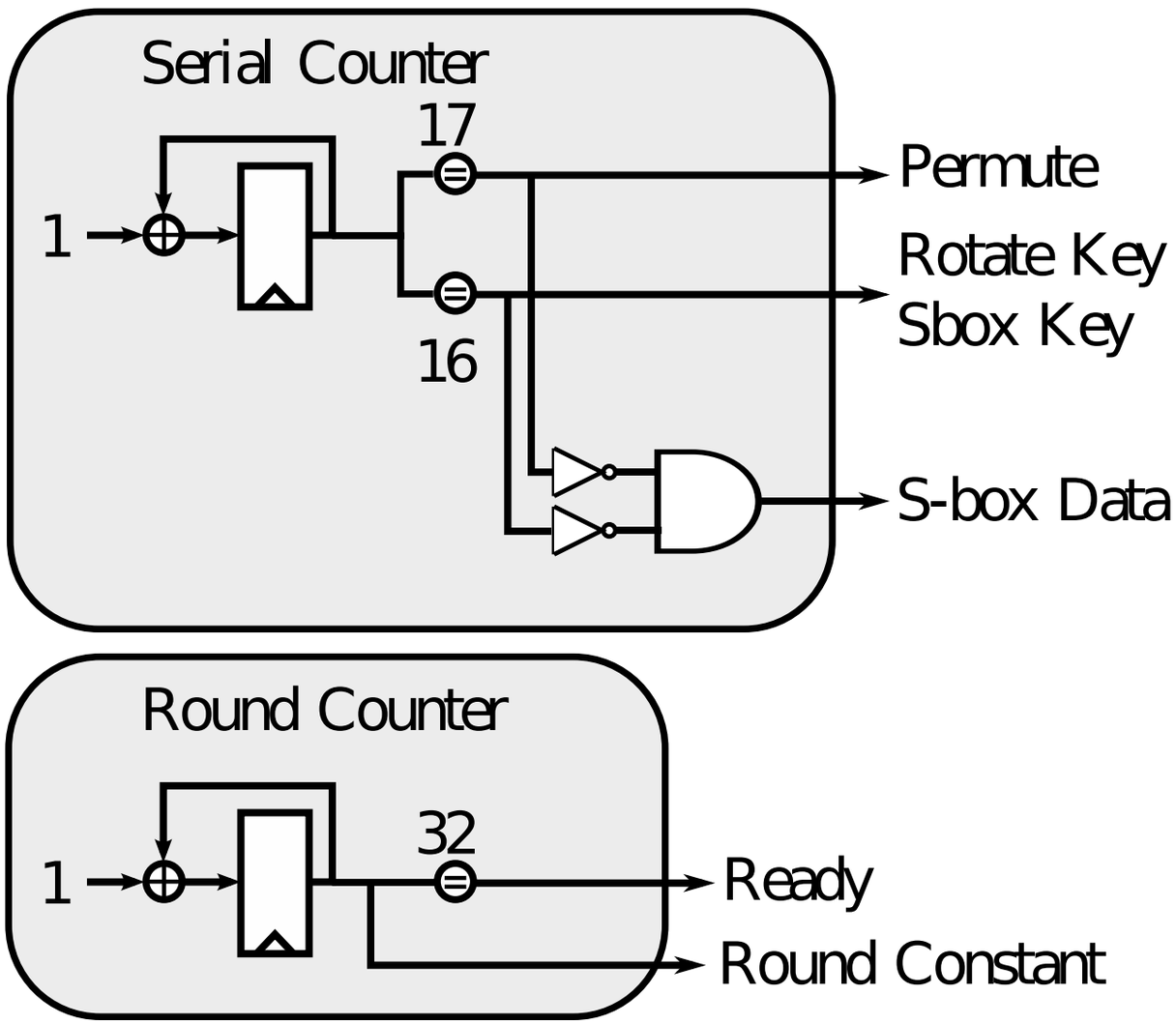


Fault on ready signal reveals all intermediate results

All possible signals need to be encoded with PC II

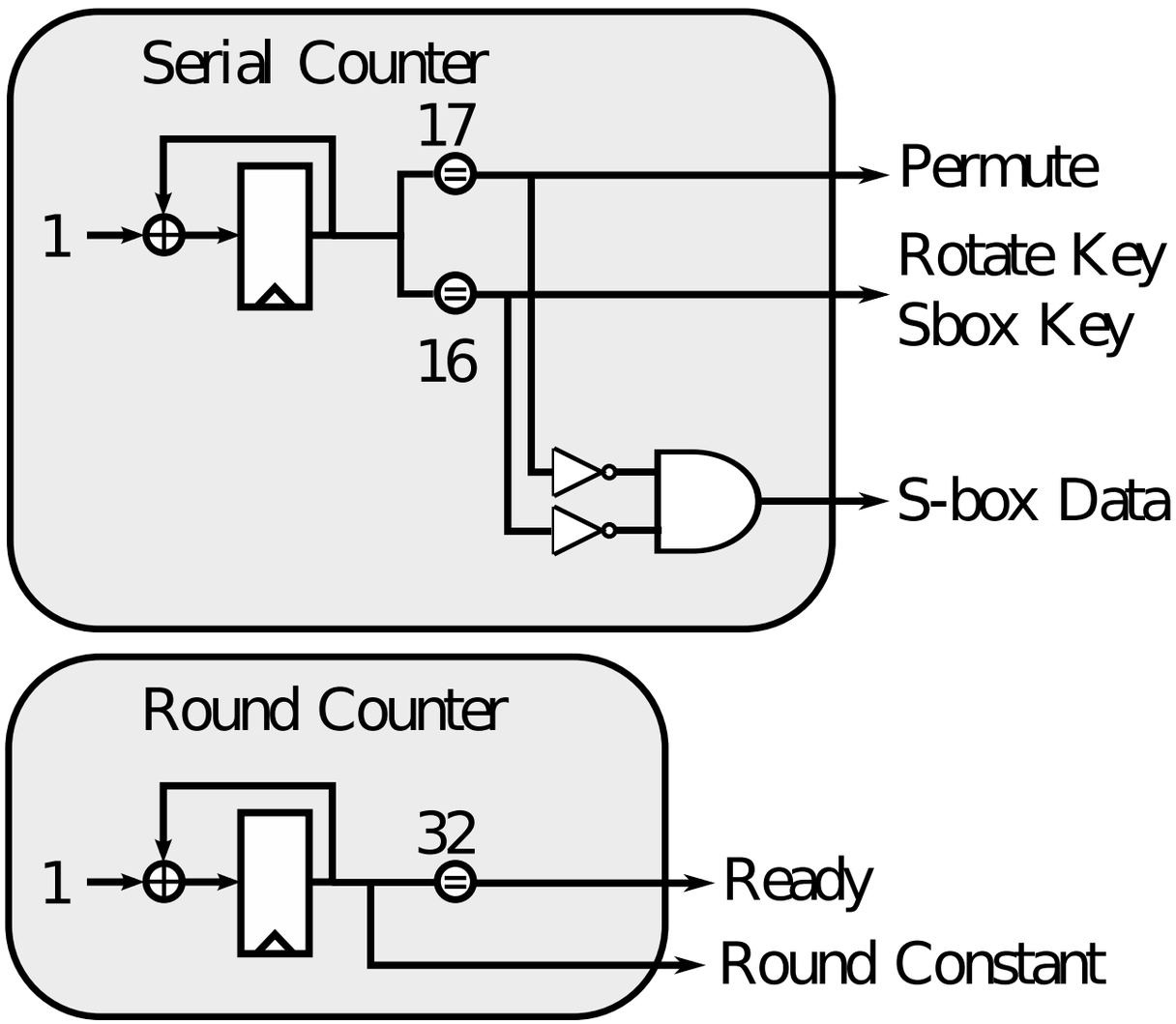


All possible signals need to be encoded with PC II



Adders

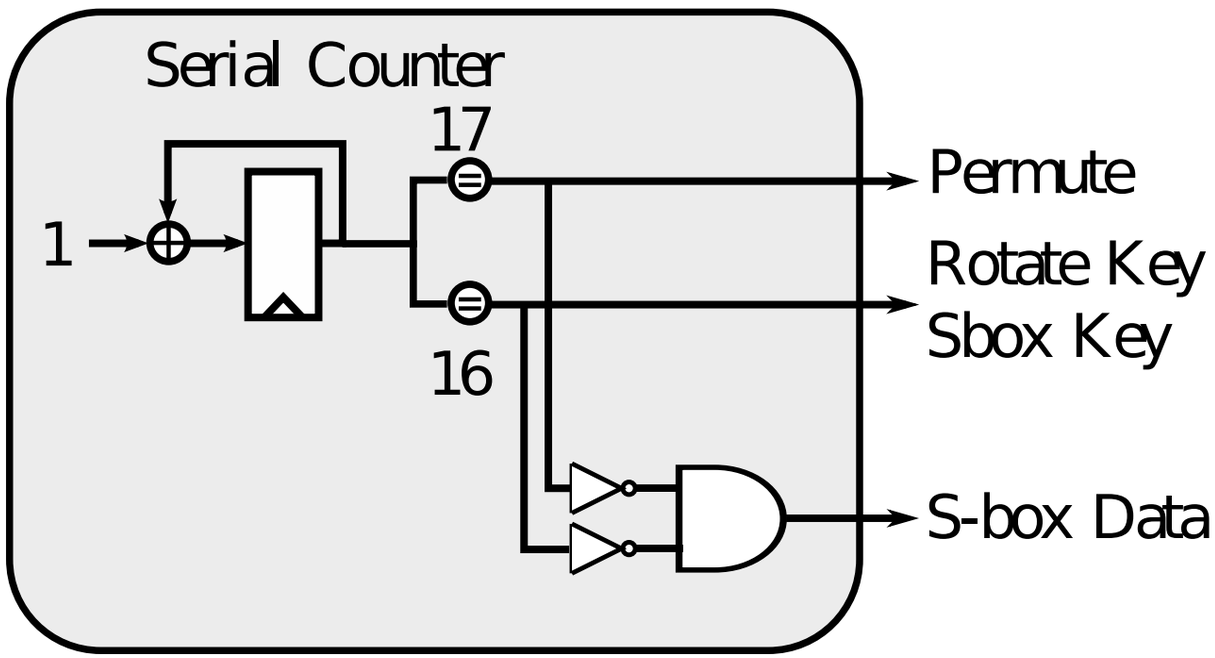
All possible signals need to be encoded with PC II

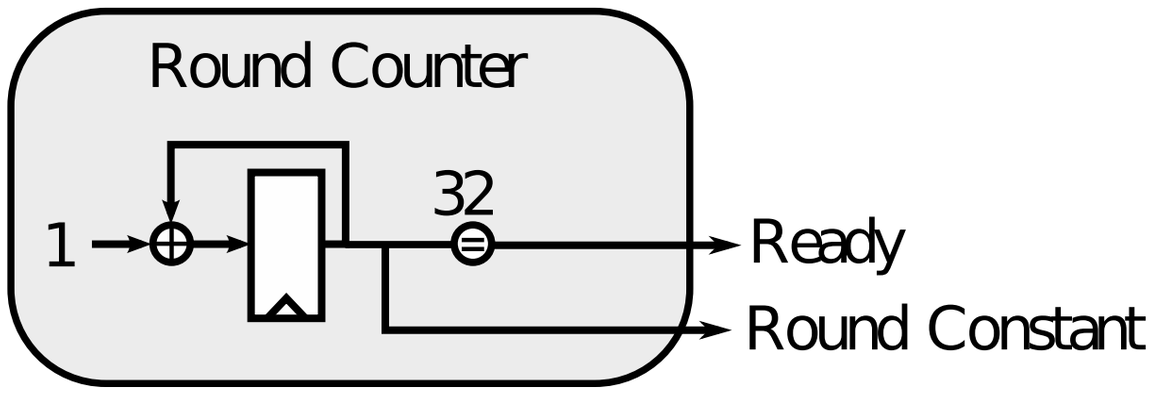


Adders

Comparators

All possible signals need to be encoded with PC II



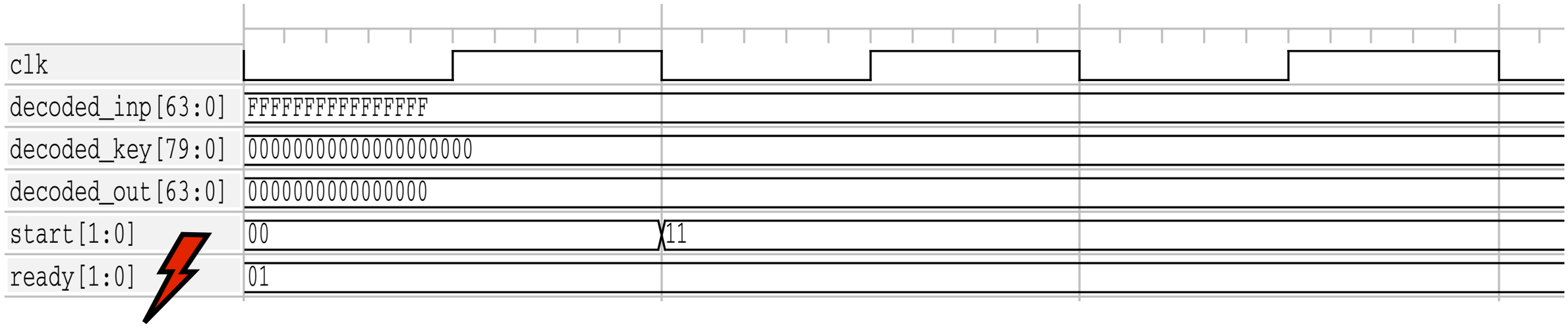


Adders

Comparators

Multiplexers

PC II effectively handles the injected fault on the ready signal



Fault on ready signal reveals no information on the intermediate results

Applying PC II results in a significant increase in area

	TI	TI + PC-II Reset-Only	TI + PC-II General Attack (t = 1)
Number of Slices	696	6125	6125
Number of Slice Flip Flops	647	1292	1292
Number of 4 input LUTs	1356	10341	10341
Consumed Random Bits	0		
Number of Clock Cycles	578		

Applying PC II results in a significant increase in area

	TI	TI + PC-II Reset-Only	TI + PC-II General Attack (t = 1)
Number of Slices	696	6125	6125
Number of Slice Flip Flops	647	1292	1292
Number of 4 input LUTs	1356	10341	10341
Consumed Random Bits	0		
Number of Clock Cycles	578		

Result of use of LUTs 4 input function vs atomic gates

Applying PC II results in a significant increase in area

	TI	TI + PC-II Reset-Only	TI + PC-II General Attack (t = 1)
Number of Slices	696	6125	6125
Number of Slice Flip Flops	647	1292	1292
Number of 4 input LUTs	1356	10341	10341
Consumed Random Bits	0		
Number of Clock Cycles	578		

Result of use of LUTs 4 input function vs atomic gates

Can be reduced when care is applied

Applying PC II results in a significant increase in area

Future work can improve the area cost for our FPGA implementations

1. Packing gates in LUT while satisfying the OR of AND structure

Applying PC II results in a significant increase in area

Future work can improve the area cost for our FPGA implementations

1. Packing gates in LUT while satisfying the OR of AND structure

2. Move implementations to larger FPGAs and launch combined attacks

Applying PC II results in a significant increase in area

Future work can improve the area cost for our FPGA implementations

1. Packing gates in LUT while satisfying the OR of AND structure

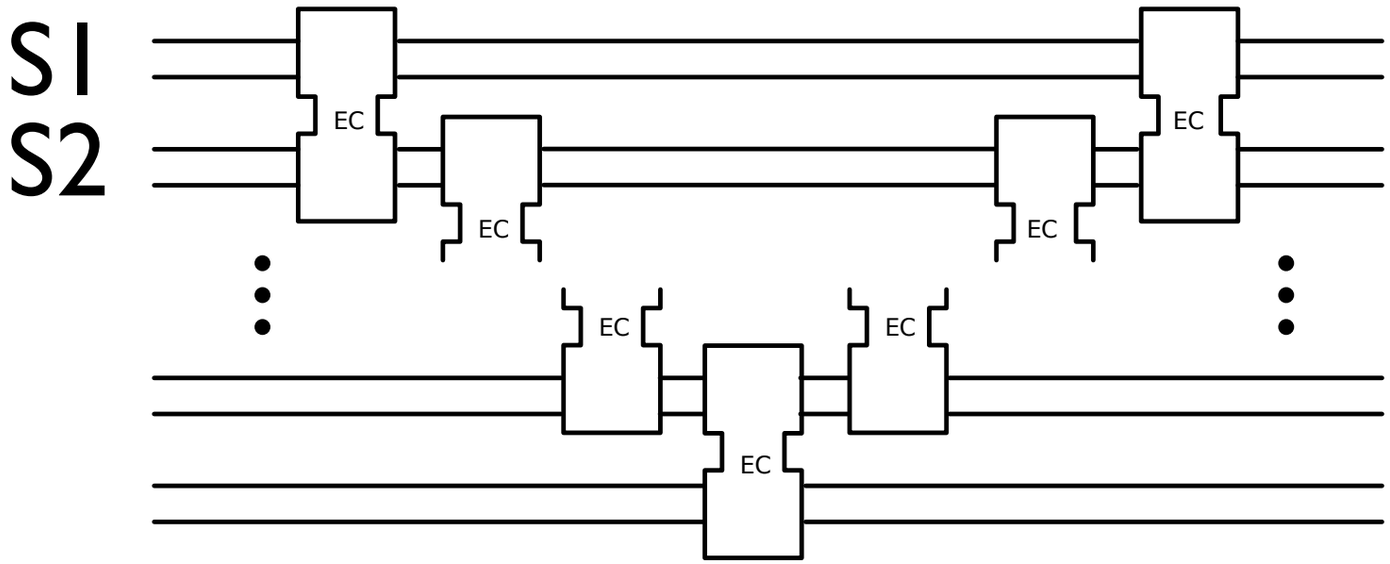
2. Move implementations to larger FPGAs and launch combined attacks

3. Circuits with randomness consumption

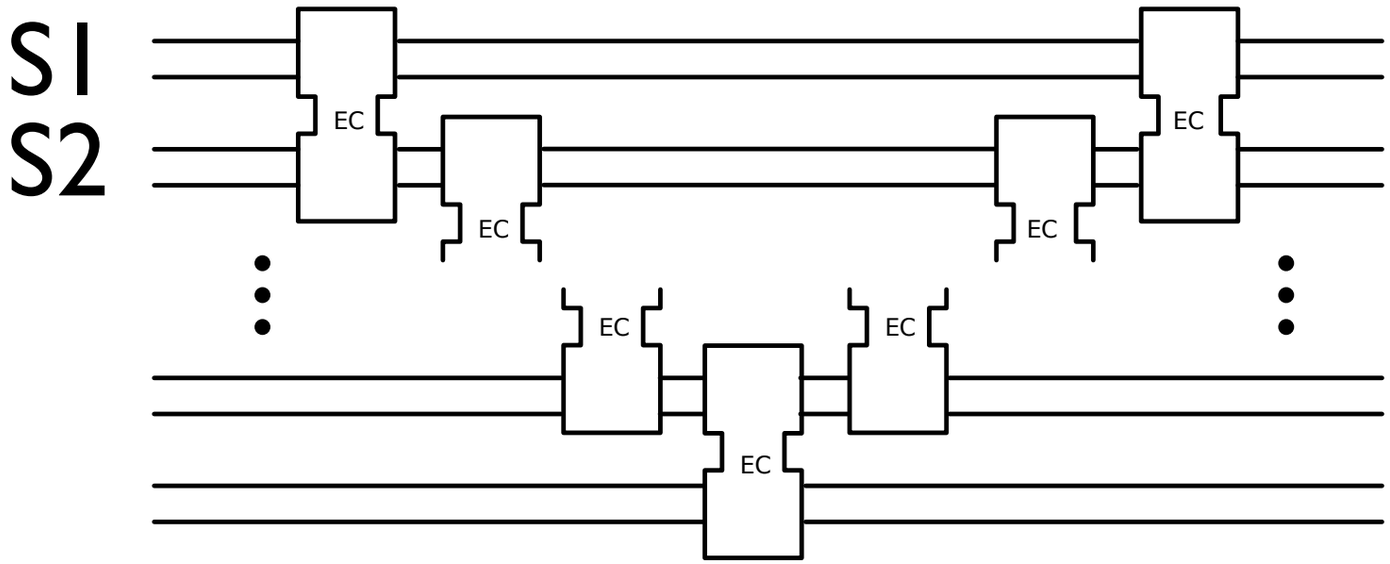
Thank you

Questions ?

Error Cascading Stage is nonlinear



Error Cascading Stage is nonlinear



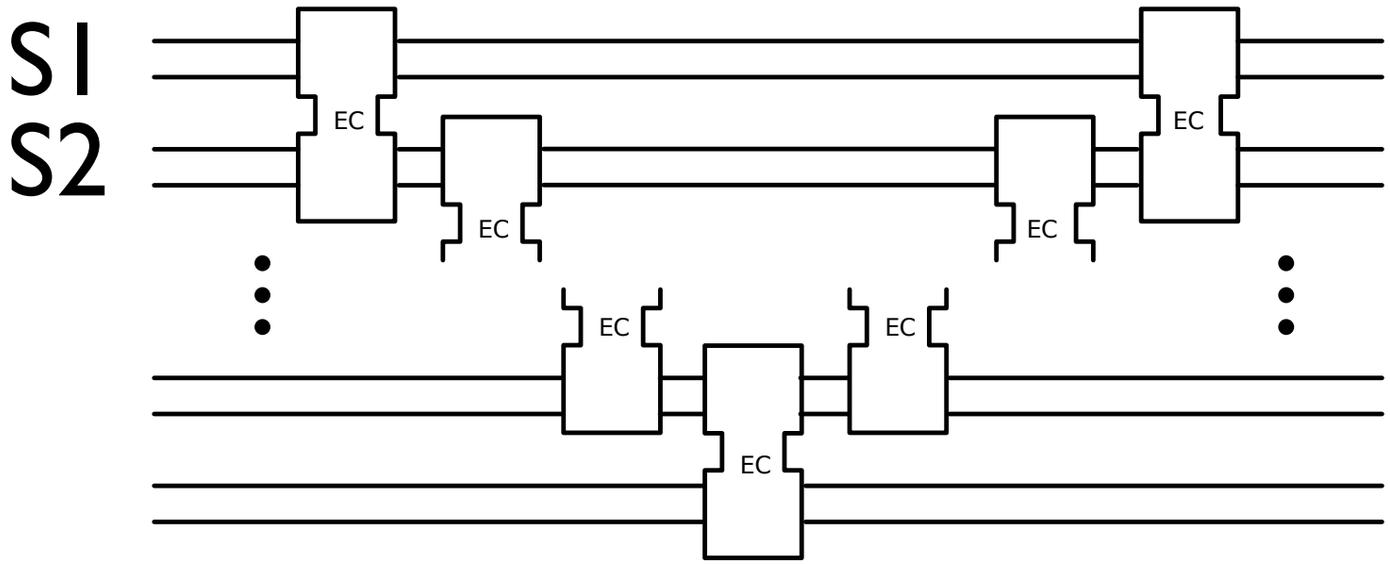
$$s_{1,EC,1} = (s_{1,1}\bar{s}_{1,0}\bar{s}_{2,1}s_{2,0}) \oplus (s_{1,1}\bar{s}_{1,0}s_{2,1}\bar{s}_{2,0})$$

$$s_{1,EC,0} = (\bar{s}_{1,1}s_{1,0}\bar{s}_{2,1}s_{2,0}) \oplus (\bar{s}_{1,1}s_{1,0}s_{2,1}\bar{s}_{2,0})$$

$$s_{2,EC,1} = (\bar{s}_{1,1}s_{1,0}s_{2,1}\bar{s}_{2,0}) \oplus (s_{1,1}\bar{s}_{1,0}s_{2,1}\bar{s}_{2,0})$$

$$s_{2,EC,0} = (\bar{s}_{1,1}s_{1,0}\bar{s}_{2,1}s_{2,0}) \oplus (s_{1,1}\bar{s}_{1,0}\bar{s}_{2,1}s_{2,0})$$

Error Cascading Stage is nonlinear



Non-completeness is broken !

$$s_{1,EC,1} = (s_{1,1}\neg s_{1,0}\neg s_{2,1}s_{2,0}) \oplus (s_{1,1}\neg s_{1,0}s_{2,1}\neg s_{2,0})$$
$$s_{1,EC,0} = (\neg s_{1,1}s_{1,0}\neg s_{2,1}s_{2,0}) \oplus (\neg s_{1,1}s_{1,0}s_{2,1}\neg s_{2,0})$$
$$s_{2,EC,1} = (\neg s_{1,1}s_{1,0}s_{2,1}\neg s_{2,0}) \oplus (s_{1,1}\neg s_{1,0}s_{2,1}\neg s_{2,0})$$
$$s_{2,EC,0} = (\neg s_{1,1}s_{1,0}\neg s_{2,1}s_{2,0}) \oplus (s_{1,1}\neg s_{1,0}\neg s_{2,1}s_{2,0})$$