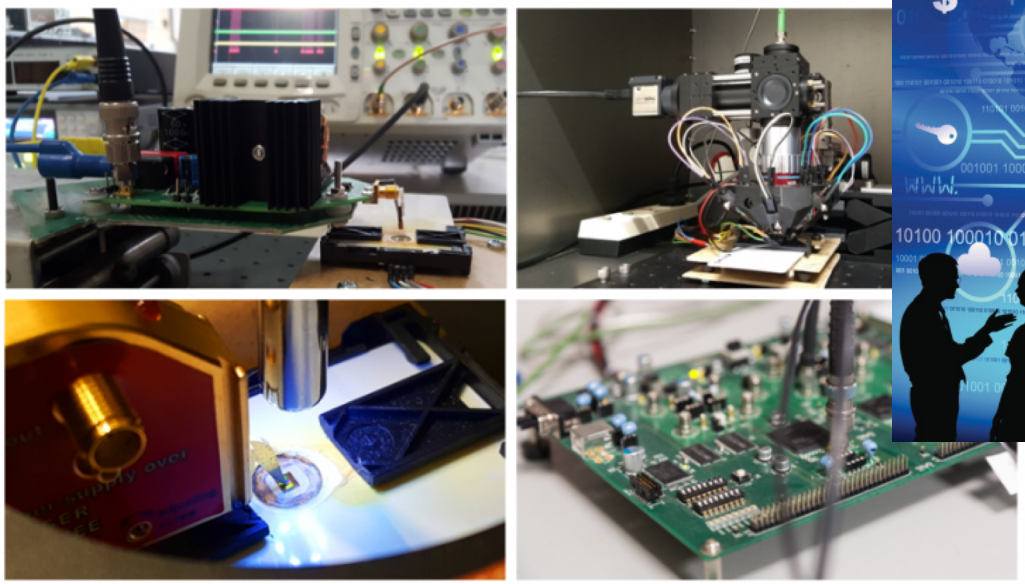
VerMI & VerFI
Verification Tools for Masked Implementations

Svetla Nikova, Victor Arribas
COSIC, KULeuven
Joint work with Vincent Rijmen (KU Leuven),
Felix Wegener, Amir Moradi (RU Bochum)

KU LEUVEN

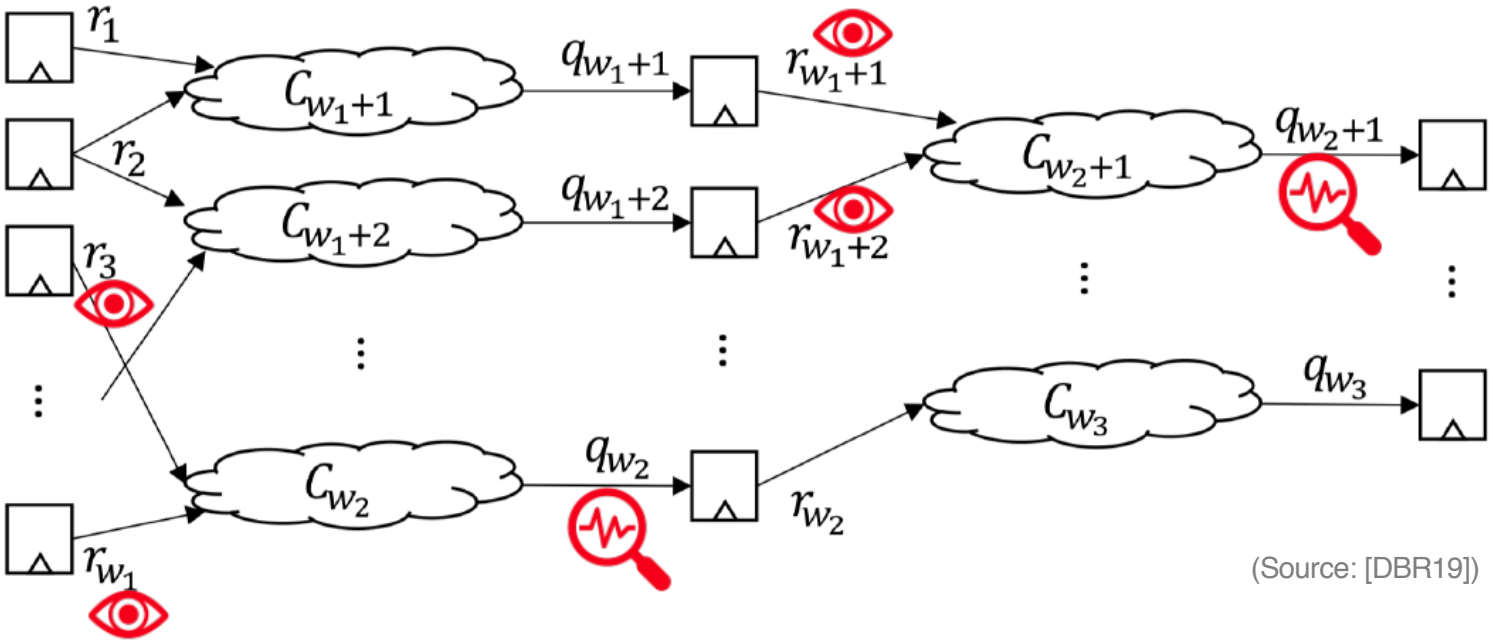
Verification Tools

- Why do we need verification tools?
- When should we test implementations?
- What kind of tools we need?



KU LEUVEN

Side-channel attacks: what to verify?



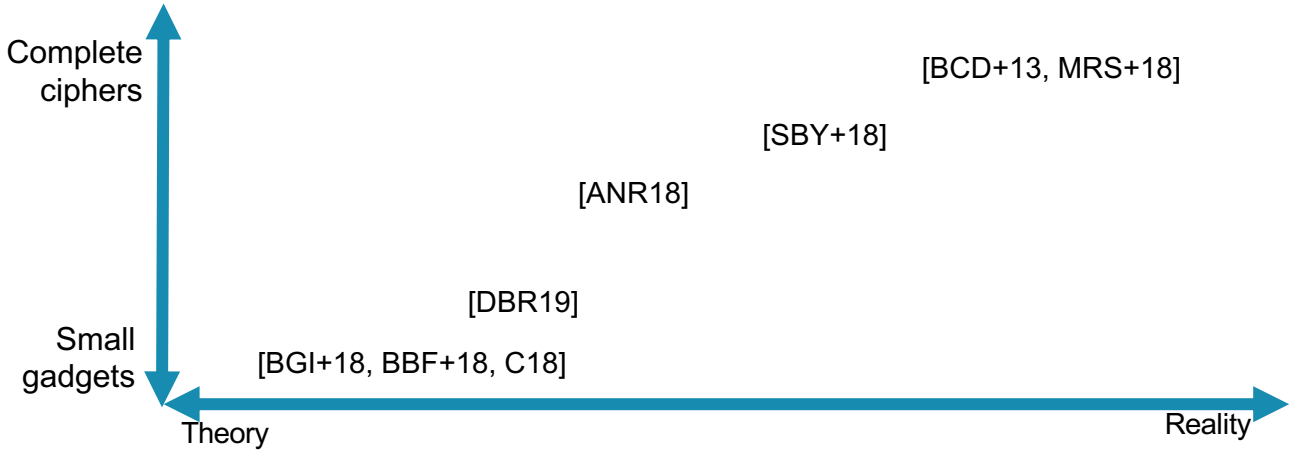
(Source: [DBR19])

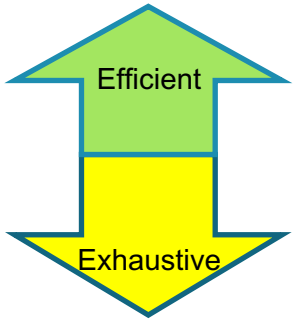
Necessary conditions [ANR18] vs. Sufficient conditions [DBR19]
e.g.: Non-completeness e.g.: MI(glitch-extended probes, secret) = 0

$$\#position\ of\ probes \approx \binom{\#wires}{security\ order\ d}$$

KU LEUVEN

SCA verification tools

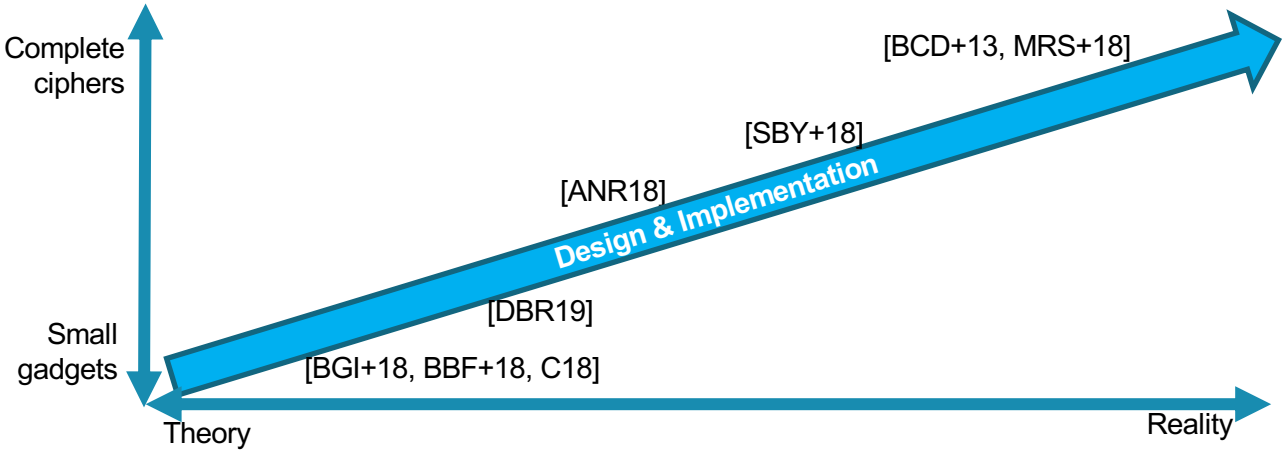




KU LEUVEN

SCA verification: how-to

Verification mechanisms – the more the better!
One does NOT suffice



References

[ANR18] V. Arribas, S. Nikova, V. Rijmen: **VerMI: Verification Tool for Masked Implementations.** ICECS 2018

[BCD+13] G. Becker, J. Cooper, E. De Mulder, et al: **Test vector leakage assessment (TVLA) methodology in practice.** ICMC 2013

[BBF+18] G. Barthe, S. Belaid, P.-A. Fouque, B. Gregoire: **maskVerif: a formal tool for analyzing software and hardware masked implementations.** ESORICS 2019

[BGI+18] R. Bloem, H. Gros, R. Iusupov, B. Konighofer, S. Mangard, J. Winter: **Formal Verification of Masked Hardware Implementations in the Presence of Glitches.** EUROCRYPT 2018

[C18] J.-S. Coron: **Formal Verication of Side-channel Countermeasures via Elementary Circuit Transformations.** ACNS 2018

[DBR19] L. De Meyer, B. Bilgin, O. Reparaz: **Consolidating Security Notions in Hardware Masking.** CHES 2019

[MRS+18] A. Moradi, B. Richter, T. Schneider, F.-X. Standaert: **Leakage Detection with the chi-squared-Test.** CHES 2018

[SBY+18] D. Sijacic, J. Balasch, B. Yang, S. Ghosh, I. Verbauwhede: **Towards Efficient and Automated Side Channel Evaluations at Design Time.** PROOFS@CHES 2018

VerMI

Verification Tool for Masked Implementations

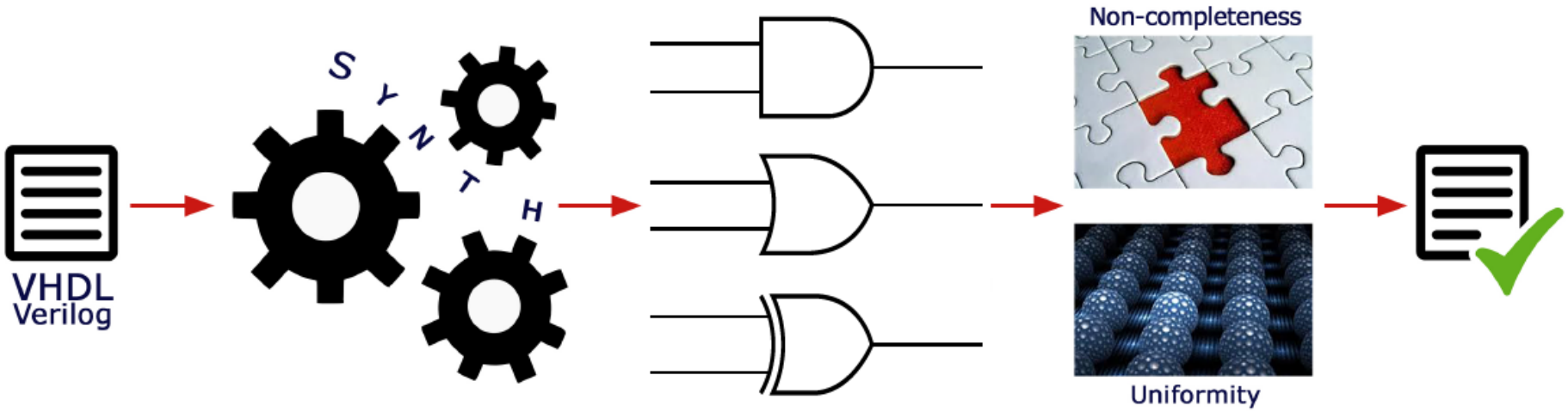
KU LEUVEN

VerMI - outline

- VerMI
- Threshold Implementations
- Non-Completeness
- Sequential Logic
- Uniformity

KU LEUVEN

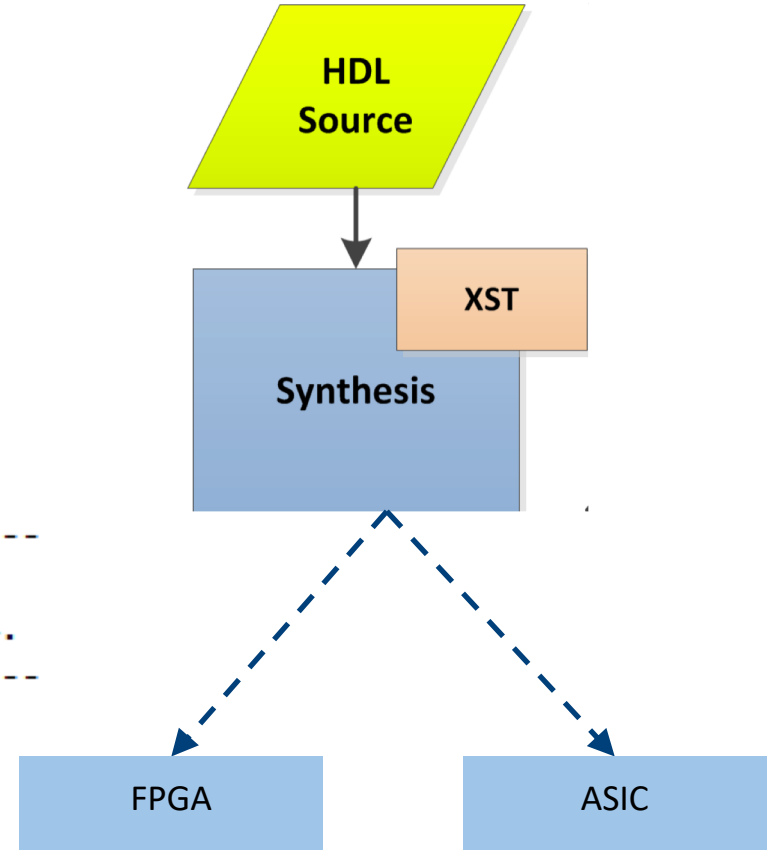
VerMI



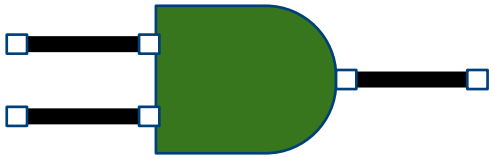
Verification Tool

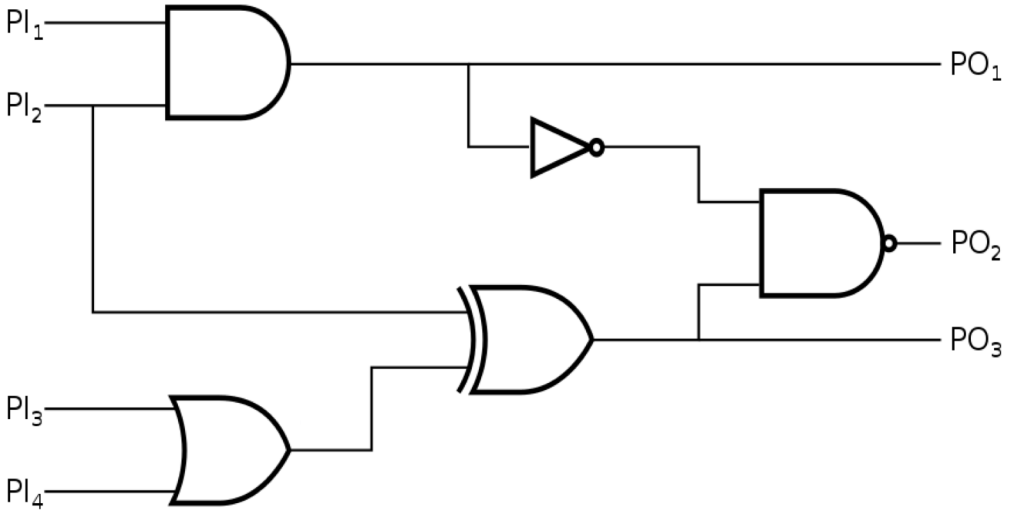
- C++
- Synopsys DC Compiler

```
-----------------------------------------
--Input_shares: x1, x2, x3, x4.
--Random_vars: b_i_3, b_i_4, c_i_3, c_i_4.
-----------------------------------------
```

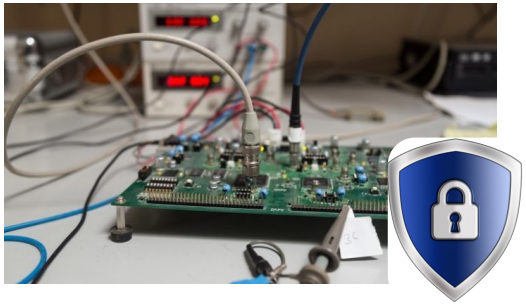


Structural Model





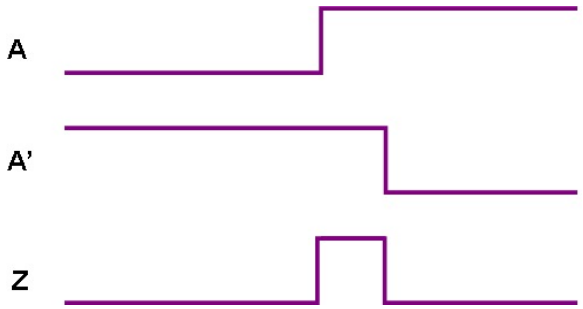
Threshold Implementations



Side-Channel Analysis (SCA) countermeasure

Provable security with minimal assumptions on the HW

Security in the presence of glitches



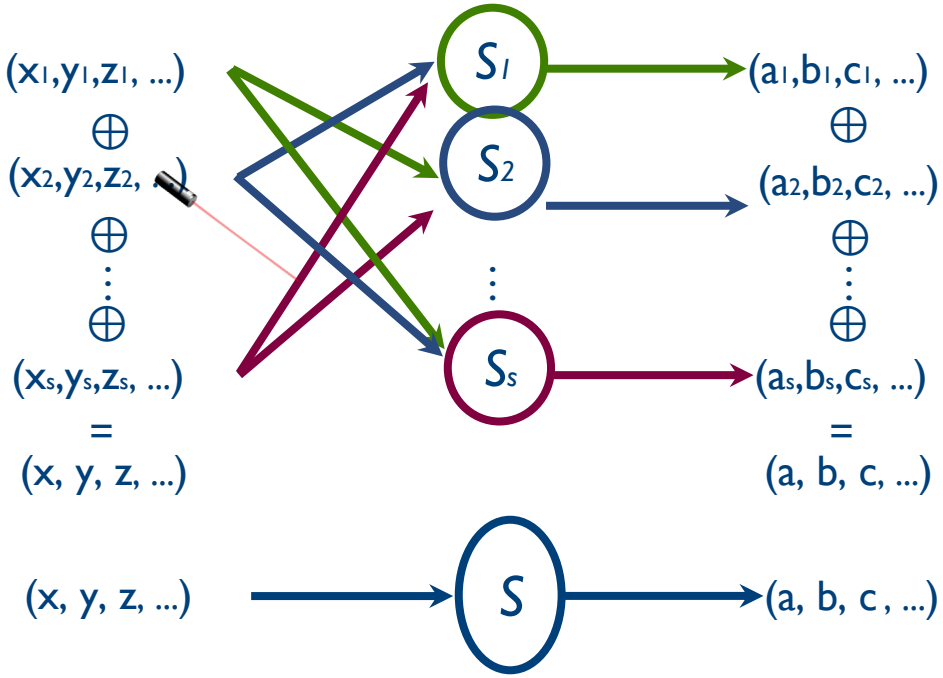
KU LEUVEN

Threshold Implementations (1st order)

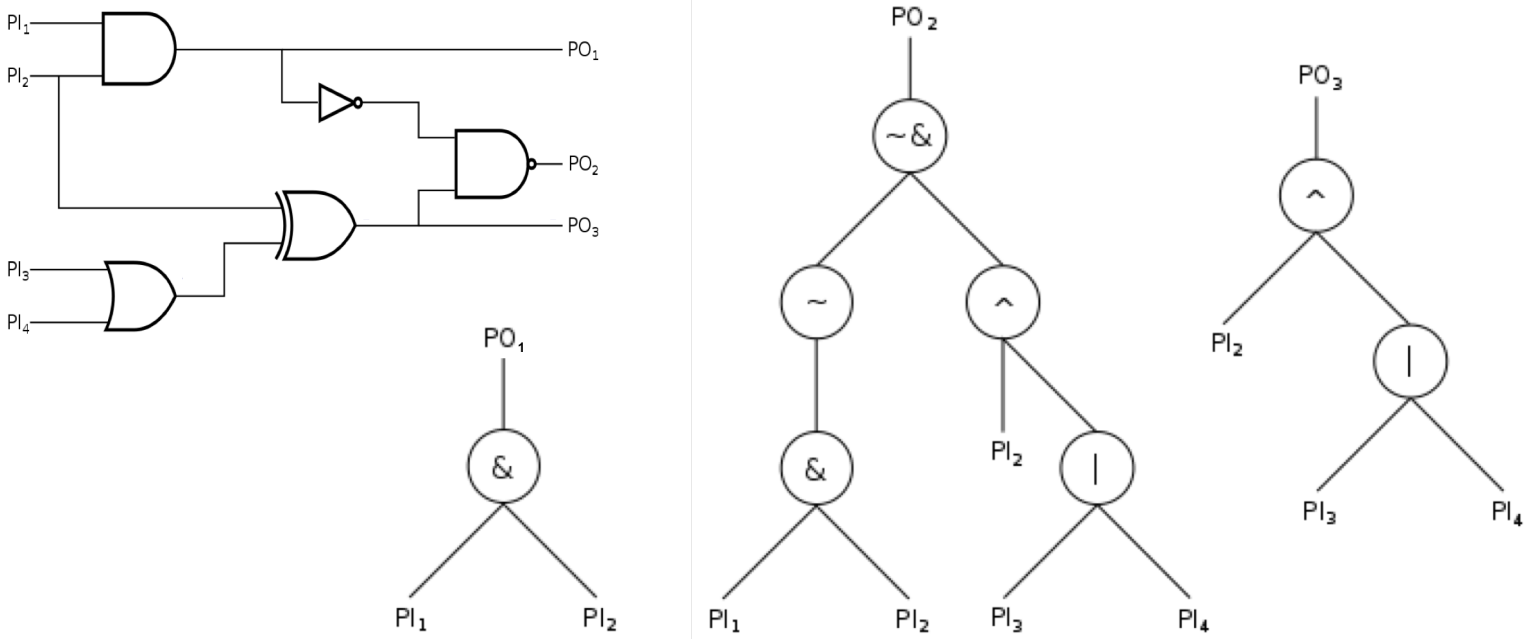
Boolean masking scheme

Secret sharing and multi-party computation techniques

- Correctness
- Non-completeness
- Uniformity

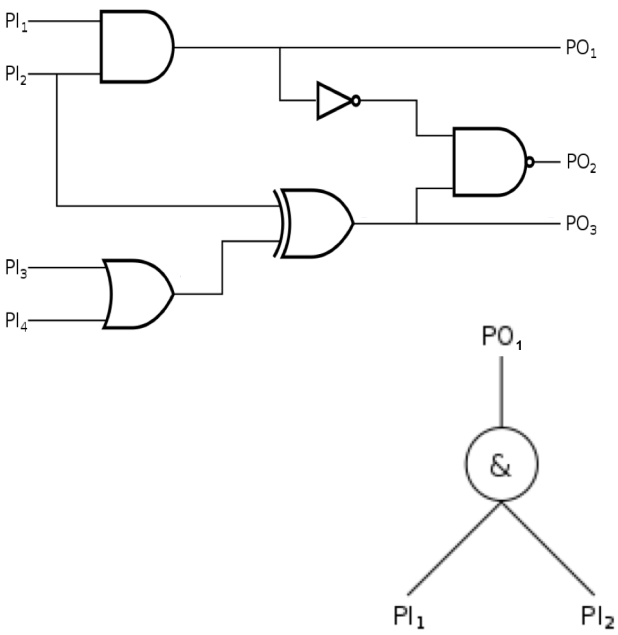


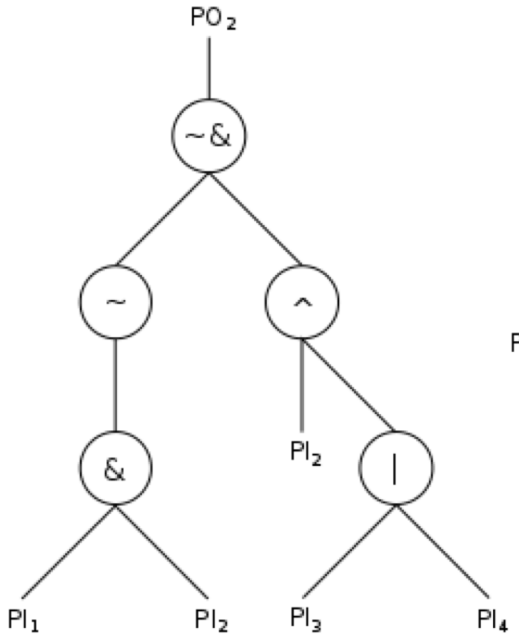
Tree Search

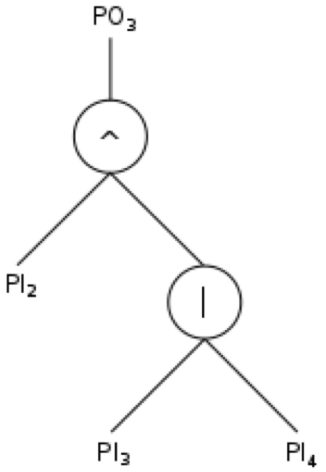


$$PO_1 = f_1(P_1, P_2) \qquad PO_2 = f_2(P_1, P_2, P_3, P_4) \qquad PO_3 = f_3(P_2, P_3, P_4)$$

Non-completeness







$PO_1 = f_1(P_1, P_2)$ ✓ $PO_2 = f_2(P_1, P_2, P_3, P_4)$ ✗ $PO_3 = f_3(P_2, P_3, P_4)$ ✓

Non-completeness

E.g.: Multiplier

$$z_1 = F_1(x_1, x_2, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1$$

$$z_2 = F_2(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2$$

$$z_3 = F_3(x_1, x_3, y_1, y_3) = x_3y_3 \oplus x_1y_3 \oplus x_3y_1$$

Sensitive data

Vars.	Shares		
	x_1	x_2	x_3
	y_1	y_2	y_3

Dependencies

z_1		
x_1	x_2	
y_1	y_2	

z_2		
	x_2	x_3
	y_2	y_3

z_3		
x_1		x_3
y_1		y_3

Non-completeness

E.g.: Multiplier

$$z_1 = F_1(x_1, x_2, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_3y_1$$

$$z_2 = F_2(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2$$

$$z_3 = F_3(x_1, x_3, y_1, y_3) = x_3y_3 \oplus x_1y_3 \oplus x_2y_1$$

Sensitive data

Vars.	Shares		
	x_1	x_2	x_3
	y_1	y_2	y_3

Dependencies

z_1		
x_1		x_3
y_1	y_2	

z_2		
	x_2	x_3
	y_2	y_3

z_3		
x_1	x_2	x_3
y_1		y_3

HO Non-completeness

E.g.: Multiplier (1st order)

$$z_1 = F_1(x_1, x_2, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1$$

$$z_2 = F_2(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2$$

$$z_3 = F_3(x_1, x_3, y_1, y_3) = x_3y_3 \oplus x_1y_3 \oplus x_3y_1$$

Sensitive data

Dependencies

Vars.	Shares		
	x_1	x_2	x_3
	y_1	y_2	y_3

(z_1, z_2)		
x_1	x_2	x_3
y_1	y_2	y_3

(z_1, z_3)		
x_1	x_2	x_3
y_1	y_2	y_3

(z_2, z_3)		
x_1	x_2	x_3
y_1	y_2	y_3

HO Non-completeness

E.g.: Multiplier (2nd order)

$$z_1 = x_2y_2 \oplus x_1y_2 \oplus x_2y_1 \oplus x_1y_3 \oplus x_3y_1 \oplus x_2y_3 \oplus x_3y_2$$

$$z_2 = x_3y_3 \oplus x_3y_4 \oplus x_4y_3 \oplus x_3y_5 \oplus x_5y_3$$

$$z_3 = x_4y_4 \oplus x_2y_4 \oplus x_4y_2 \oplus x_2y_6 \oplus x_6y_2$$

$$z_4 = x_5y_5 \oplus x_1y_4 \oplus x_4y_1 \oplus x_1y_5 \oplus x_5y_1$$

$$z_5 = x_2y_5 \oplus x_5y_2 \oplus x_4y_5 \oplus x_5y_4$$

$$z_6 = x_6y_6 \oplus x_3y_6 \oplus x_6y_3 \oplus x_4y_6 \oplus x_6y_4$$

$$z_7 = x_1y_1 \oplus x_1y_6 \oplus x_6y_1 \oplus x_5y_6 \oplus x_6y_5$$

Sensitive data

	Shares					
Vars.	x_1	x_2	x_3	x_4	x_5	x_6
	y_1	y_2	y_3	y_4	y_5	y_6

KU LEUVEN

HO Non-completeness

E.g.: Multiplier (2nd order) [BGN+]

$$z_1 = x_2y_2 \oplus x_1y_2 \oplus x_2y_1 \oplus x_1y_3 \oplus x_3y_1 \oplus x_2y_3 \oplus x_3y_2$$

$$z_2 = x_3y_3 \oplus x_3y_4 \oplus x_4y_3 \oplus x_3y_5 \oplus x_5y_3$$

$$z_3 = x_4y_4 \oplus x_2y_4 \oplus x_4y_2 \oplus x_2y_6 \oplus x_6y_2$$

$$z_4 = x_5y_5 \oplus x_1y_4 \oplus x_4y_1 \oplus x_1y_5 \oplus x_5y_1$$

$$z_5 = x_2y_5 \oplus x_5y_2 \oplus x_4y_5 \oplus x_5y_4$$

$$z_6 = x_6y_6 \oplus x_3y_6 \oplus x_6y_3 \oplus x_4y_6 \oplus x_6y_4$$

$$z_7 = x_1y_1 \oplus x_1y_6 \oplus x_6y_1 \oplus x_5y_6 \oplus x_6y_5$$

Dependencies

	(z_1, z_2)					
	x_1	x_2	x_3	x_4	x_5	
	y_1	y_2	y_3	y_4	y_5	

✓

ALL possible combinations must be checked

	(z_2, z_5)					
		x_2	x_3	x_4	x_5	
		y_2	y_3	y_4	y_5	

✓

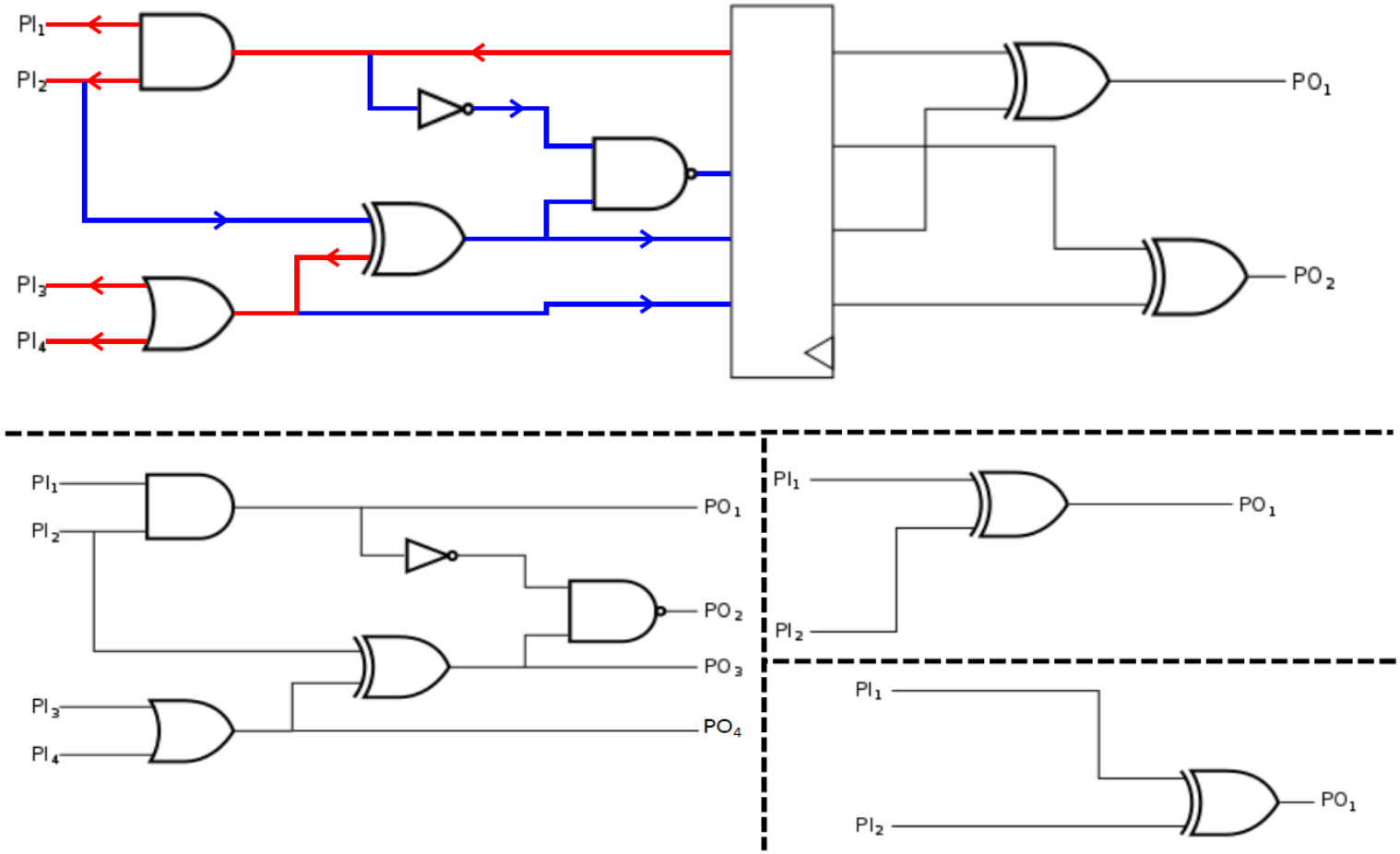
	(z_4, z_7)					
	x_1			x_4	x_5	x_6
	y_1			y_4	y_5	y_6

✓

KU LEUVEN

[BGN+] B. Bilgin, B. Gierlichs, S. Nikova, V. Nikov, V. Rijmen, Higher-Order Threshold Implementations. In Asiacryopt 2014

Subcircuit


PI1
PI2
PI3
PI4
PO1
PO2
PI1
PI2
PI3
PI4
PO1
PO2
PO3
PO4
PI1
PI2
PO1
PI1
PI2
PO1

AES Sbox

```
--------------------------------------------------------------------------------
--Input_shares: Is1, Is2.
--Random_vars: Randomness.
--Regs_layer: St2_Is1, St2_Is2.
--Regs_layer: St3_Is1, St3_Is2, St3_Is3, St3_Is4; St3_Is_Top1, St3_Is_Top2; St3_Is_Bottom1, St3_Is_Bottom2.
--Regs_layer: St4_Is1, St4_Is2, St4_Is3, St4_Is4; St4_Is_Top1, St4_Is_Top2; St4_Is_Bottom1, St4_Is_Bottom2;
--Regs_layer: St5_Is1, St5_Is2, St5_Is3, St5_Is4; St5_Is_Top1, St5_Is_Top2; St5_Is_Bottom1, St5_Is_Bottom2.
--Regs_layer: St6_Is1, St6_Is2, St6_Is3, St6_Is4.
--------------------------------------------------------------------------------
```

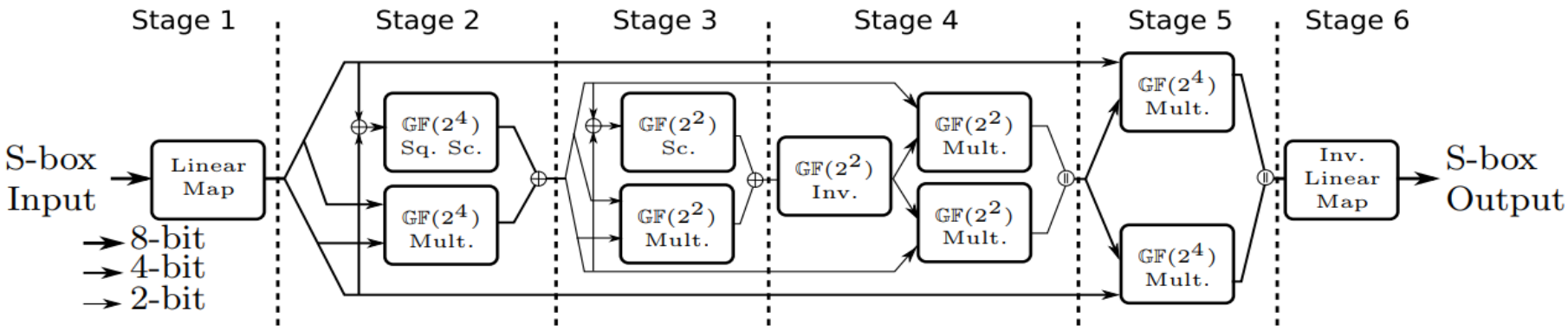


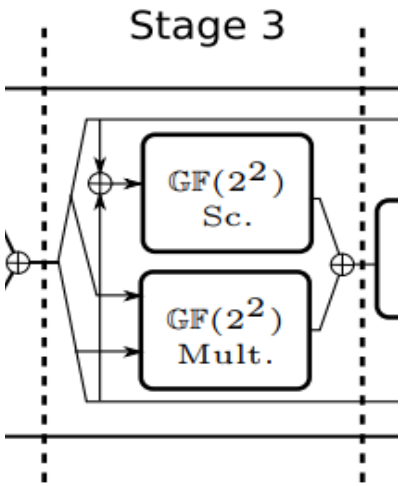
Fig. 2: Operations in the unmasked AES Sbox [CRB+]

[CRB+] *T. D. Cnudde, O. Reparaz, B. Bilgin, S. Nikova, V. Nikov, and V. Rijmen. Masking aes with d+1 shares in hardware. In CHES 2016.*

KU LEUVEN

AES Sbox

Variables	Shares			
	$St3\_1[0]$	$St3\_2[0]$	$St3\_3[0]$	$St3\_4[0]$
	$St3\_1[1]$	$St3\_2[1]$	$St3\_3[1]$	$St3\_4[1]$
	$St3\_1[2]$	$St3\_2[2]$	$St3\_3[2]$	$St3\_4[2]$
	$St3\_1[3]$	$St3\_2[3]$	$St3\_3[3]$	$St3\_4[3]$
	$St3\_Top\_1[0]$	$St3\_Top\_2[0]$		
	$St3\_Top\_1[1]$	$St3\_Top\_2[1]$		
	$St3\_Top\_1[2]$	$St3\_Top\_2[2]$		
	$St3\_Top\_1[3]$	$St3\_Top\_2[3]$		
	$St3\_Bot\_1[0]$	$St3\_Bot\_2[0]$		
	$St3\_Bot\_1[1]$	$St3\_Bot\_2[1]$		
	$St3\_Bot\_1[2]$	$St3\_Bot\_2[2]$		
	$St3\_Top\_1[3]$	$St3\_Top\_2[3]$		



KU LEUVEN

Uniformity (1st order)

E.g.: Multiplier

$$z_1 = F_1(x_1, x_2, y_1, y_2) = x_1y_1 \oplus x_1y_2 \oplus x_2y_1$$

$$z_2 = F_2(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2$$

$$z_3 = F_3(x_1, x_3, y_1, y_3) = x_3y_3 \oplus x_1y_3 \oplus x_3y_1$$

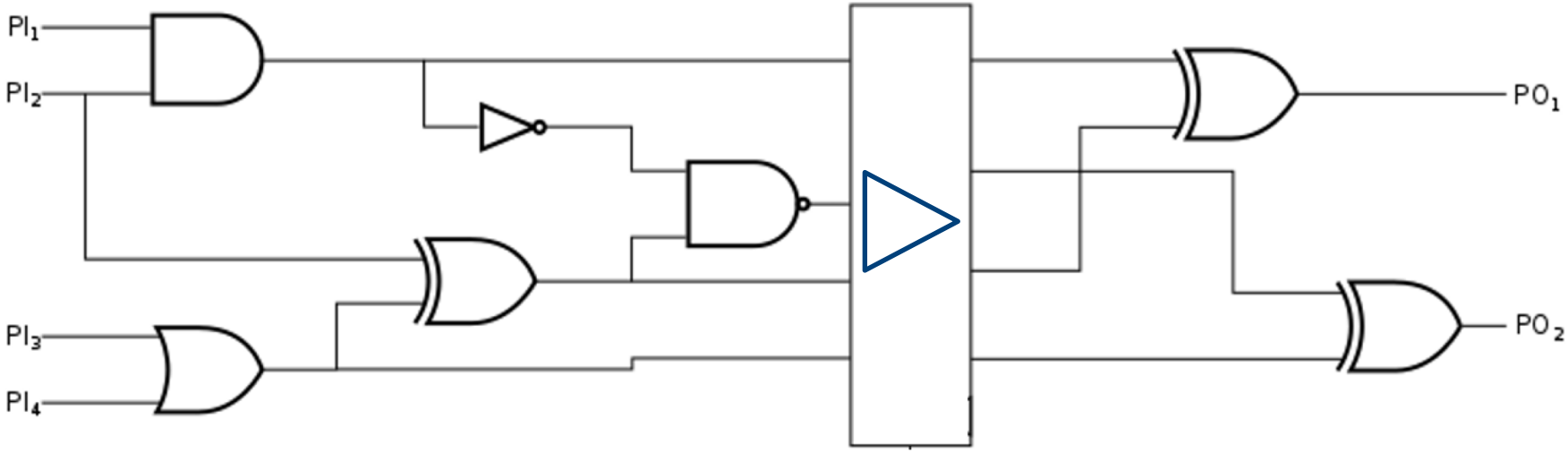
x	y	$z_1z_2z_3$							
		000	011	110	101	001	010	100	111
0	0	7	3	3	3	0	0	0	0
0	1	7	3	3	3	0	0	0	0
1	0	7	3	3	3	0	0	0	0
1	1	0	0	0	0	5	5	5	1

KU LEUVEN

Simulation

Event-Driven simulation

Flip-Flops treated as buffers



KU LEUVEN

Uniformity

Three shares implementation by G. Bertoni et. al. [BDP+]

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

Four shares implementation by B. Bilgin et. al. [BDN+]

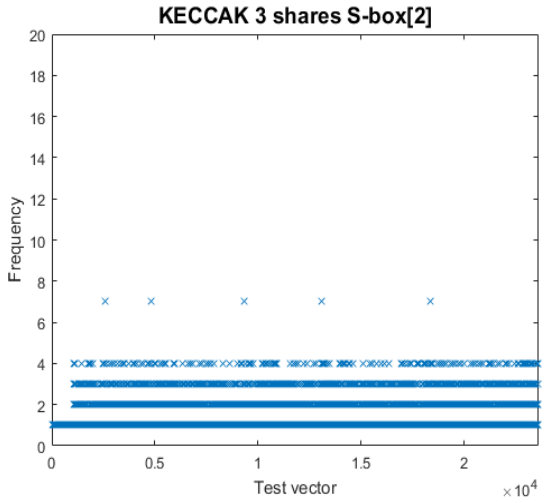
$$A'_i \leftarrow B_i + B_{i+2} + ((B_{i+1} + C_{i+1} + D_{i+1})(B_{i+2} + C_{i+2} + D_{i+2})),$$
$$B'_i \leftarrow C_i + C_{i+2} + (A_{i+1}(C_{i+2} + D_{i+2}) + A_{i+2}(C_{i+1} + D_{i+1}) + A_{i+1}A_{i+2}),$$
$$C'_i \leftarrow D_i + D_{i+2} + (A_{i+1}B_{i+2} + A_{i+2}B_{i+1}),$$
$$D'_i \leftarrow A_i + A_{i+2},$$

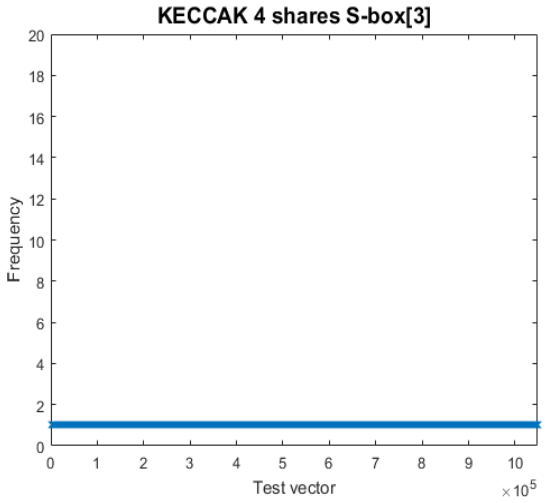
Changing of the Guards by J. Daemen [Daemen]

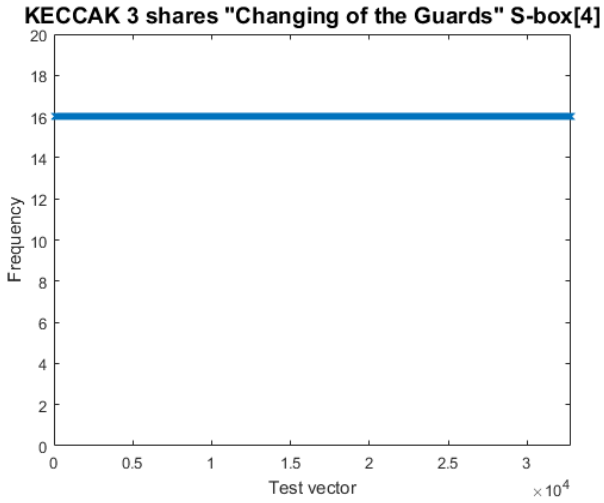
$$A_i = S_a(b_i, c_i) + b_{i-1} + c_{i-1}$$
$$B_i = S_b(a_i, c_i) + c_{i-1}$$
$$C_i = S_c(a_i, b_i) + b_{i-1}$$
$$B_0 = c_m$$
$$C_0 = b_m$$

[BDP+] *G. Bertoni, J. Daemen, M. Peeters, and G. V. Assche. Building power analysis resistant implementations of Keccak. Second SHA-3 candidate conference, August 2010.*
[BDN+] *B. Bilgin, J. Daemen, V. Nikov, S. Nikova, V. Rijmen, and G. V. Assche. Efficient and First-order DPA resistant implementations of Keccak. In CARDIS, volume 8419 of LNCS. June 2014.*
[Daemen] *J. Daemen. Changing of the guards: A simple and efficient method for achieving uniformity in threshold sharing. In CHES, volume 10529 of LNCS. September 2017.*

Uniformity


KECCAK 3 shares S-box[2]
Frequency
Test vector
×10^4


KECCAK 4 shares S-box[3]
Frequency
Test vector
×10^5


KECCAK 3 shares "Changing of the Guards" S-box[4]
Frequency
Test vector
×10^4

VerFI

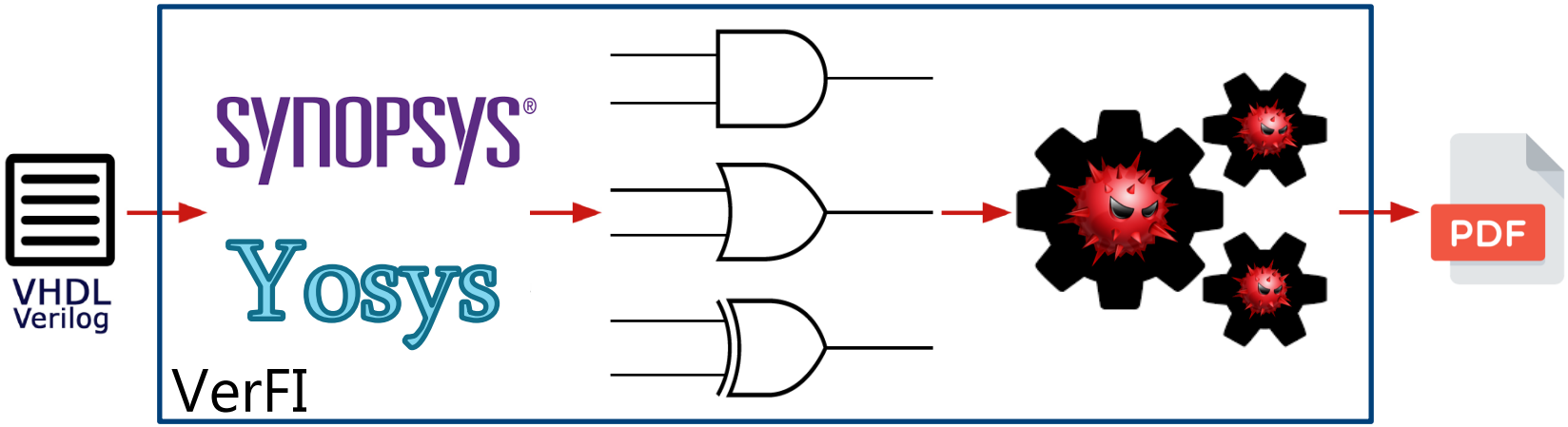
Verification Tool for Fault Injection

KU LEUVEN

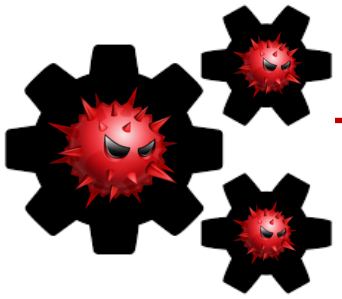
Evaluation

SIDE-CHANNEL EVALUATION	FAULT EVALUATION
maskVerif: automated analysis of software and hardware higher-order masked implementations [Barthe et al. ePrint 2018/562]	Framework for the analysis and evaluation of algebraic fault attacks [Zhang et al. IEEE Trans. on Information Forensics And Security 2016]
Formal Verification of Masked Hardware Implementations in the Presence of Glitches [Bloem et al. EUROCRYPT2018]	XFC: A Framework for eXploitable Fault Characterization in Block Ciphers. [Khanna et al. DAC 2017]
VerMI: Verification Tool for Masked Implementations [Arribas et al. ICECS 2018]	ExpFault: An Automated Framework for Exploitable Fault Characterization in Block Ciphers [Saha et al. CHES 2018]
Towards Efficient and Automated Side Channel Evaluations at Design Time CASCADE [Šijačić et al. PROOFS 2018]	
TVLA [Cooper et al. International Cryptographic Module Conference 2013]	

Framework



Faults Machine

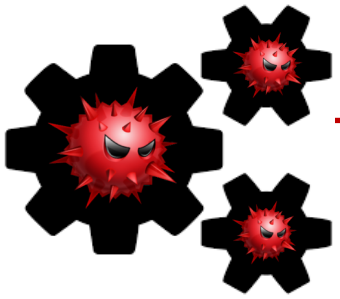


Create faults

- Fault free simulation
- Fault injection
- Fault simulation
- Fault classification

KU LEUVEN

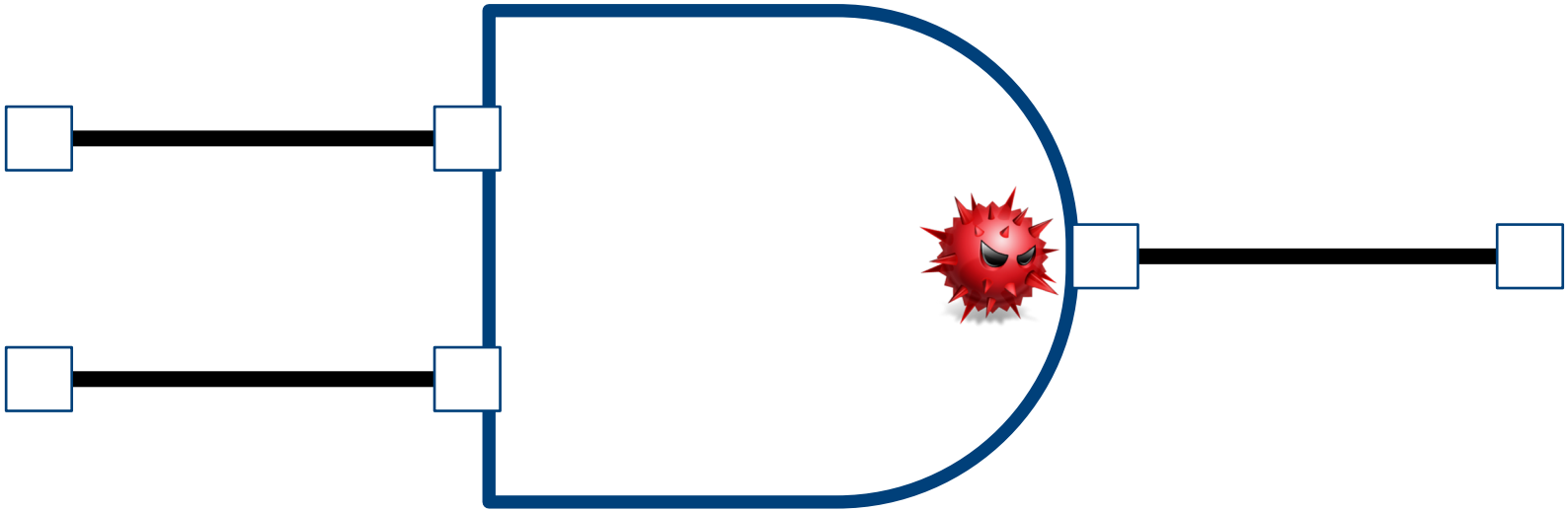
Faults Machine



- Create faults
- Fault free simulation
- Fault injection
- Fault simulation
- Fault classification

KU LEUVEN

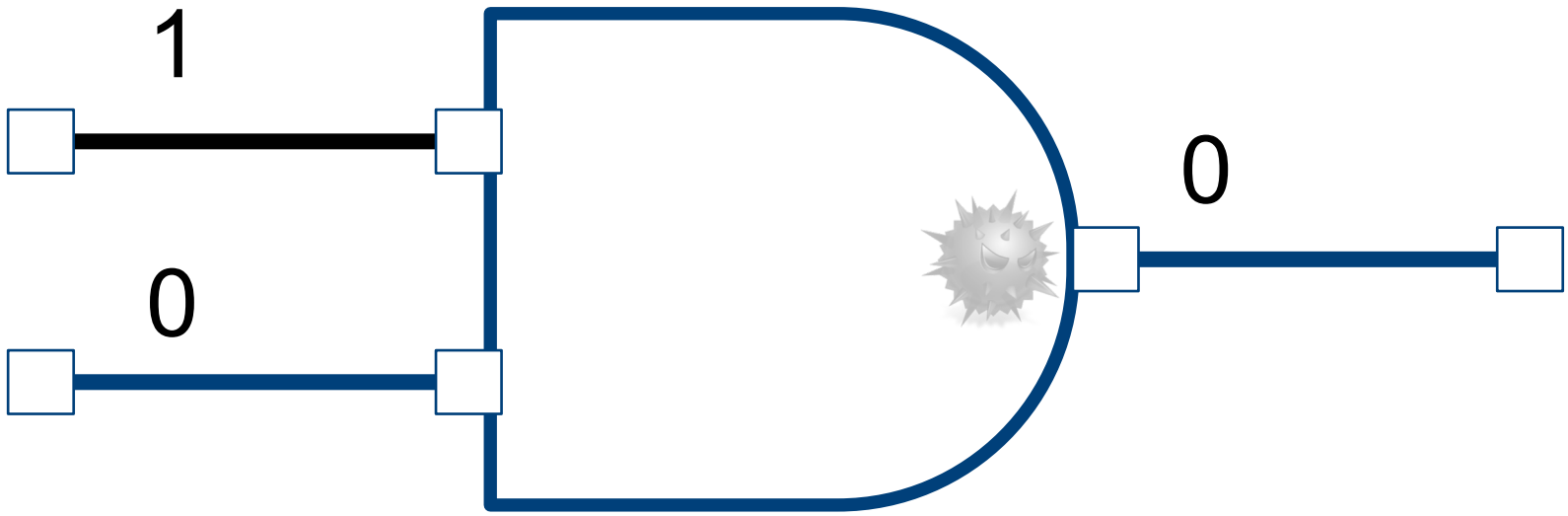
Fault Simulator



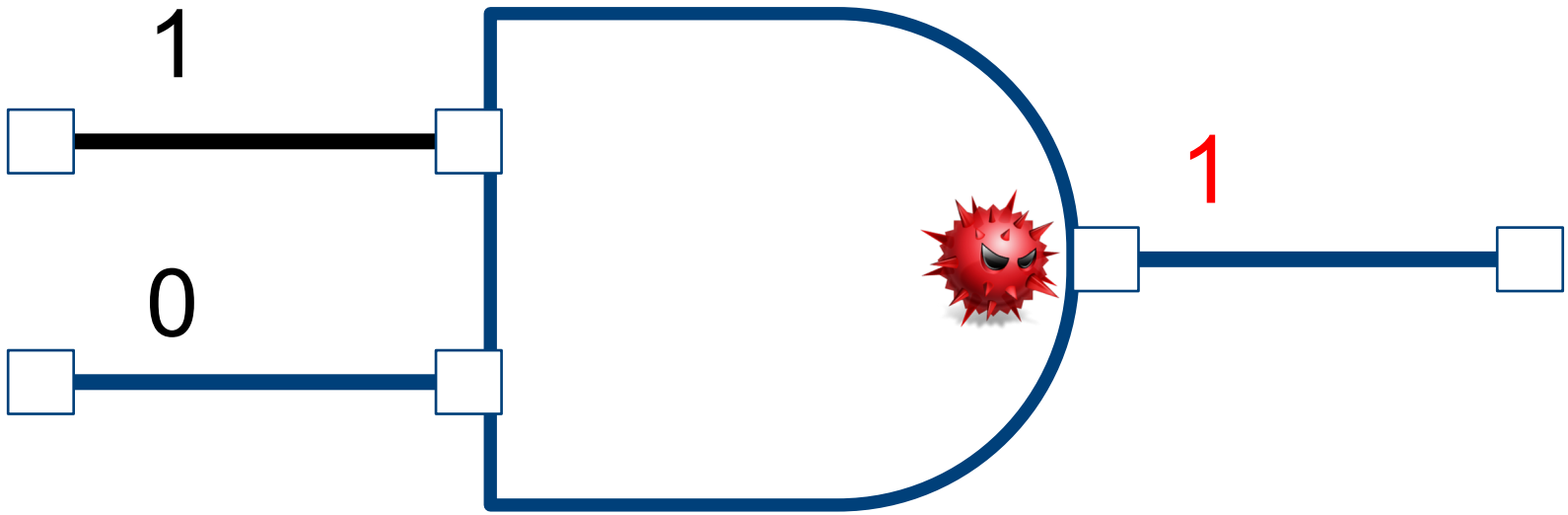
Fault

- Type
- Active
- Cycle

Fault Simulator

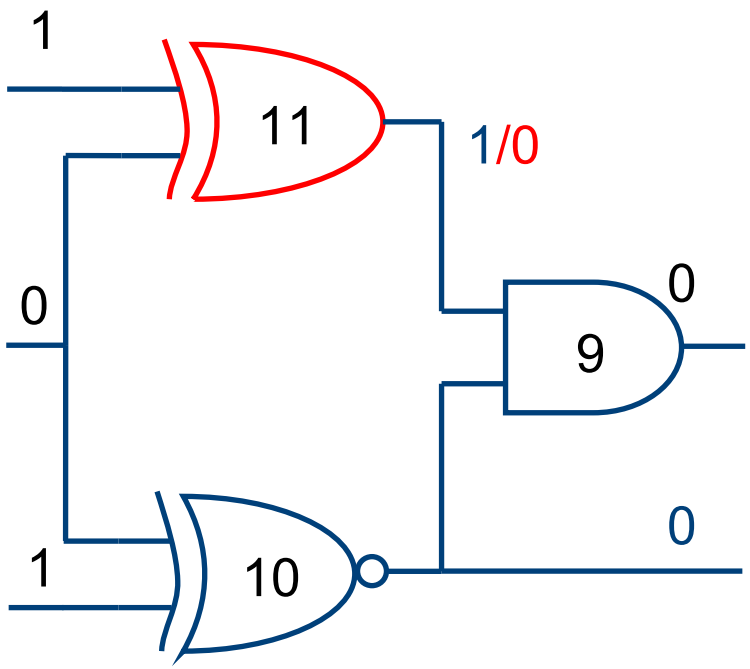


Fault Simulator



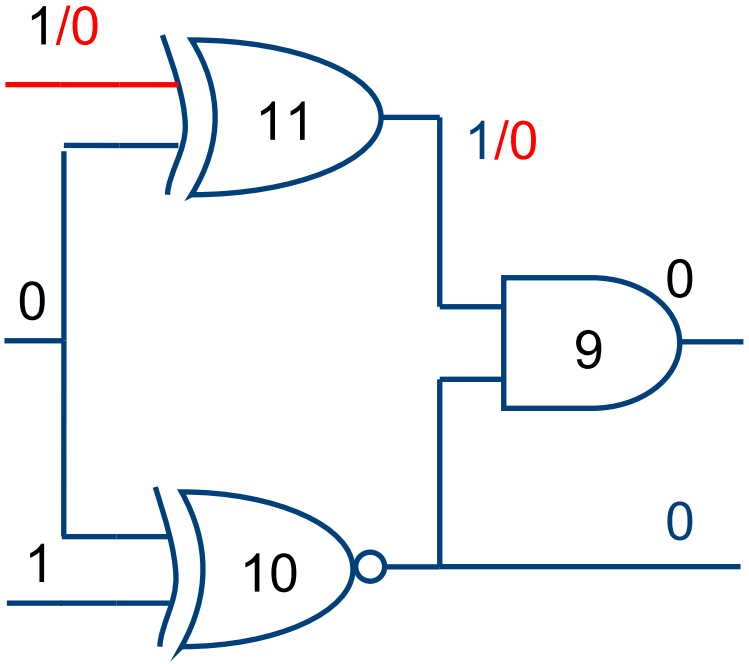
Fault Models

Output Gate faults



Gates to fault

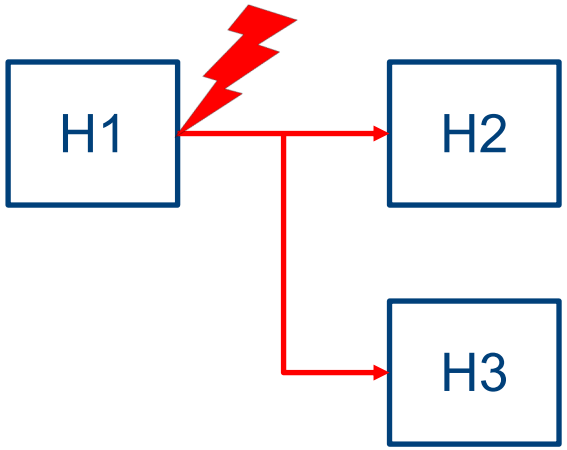
Input Gate faults



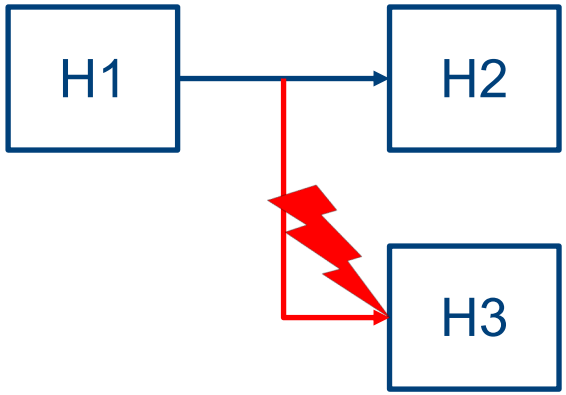
Type of fault

Fault Models

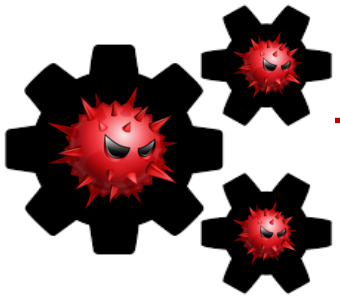
Gate faults



Wire faults



Faults Machine



Create faults

- Fault free simulation
- **Fault injection**
- Fault simulation
- Fault classification

KU LEUVEN

Fault Injection

- Fault: gate, cycle and type tuple

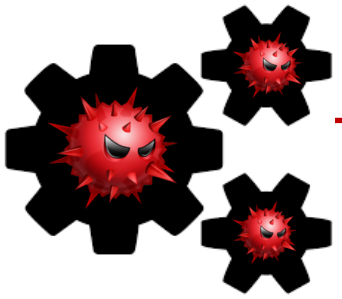
$$F = \{gate, \#cycle, \#\text{type}\}$$

- Allowed faults vector

$$\boldsymbol{F} = [F_0, \ldots, F_{N-1}]$$

KU LEUVEN

Faults Machine



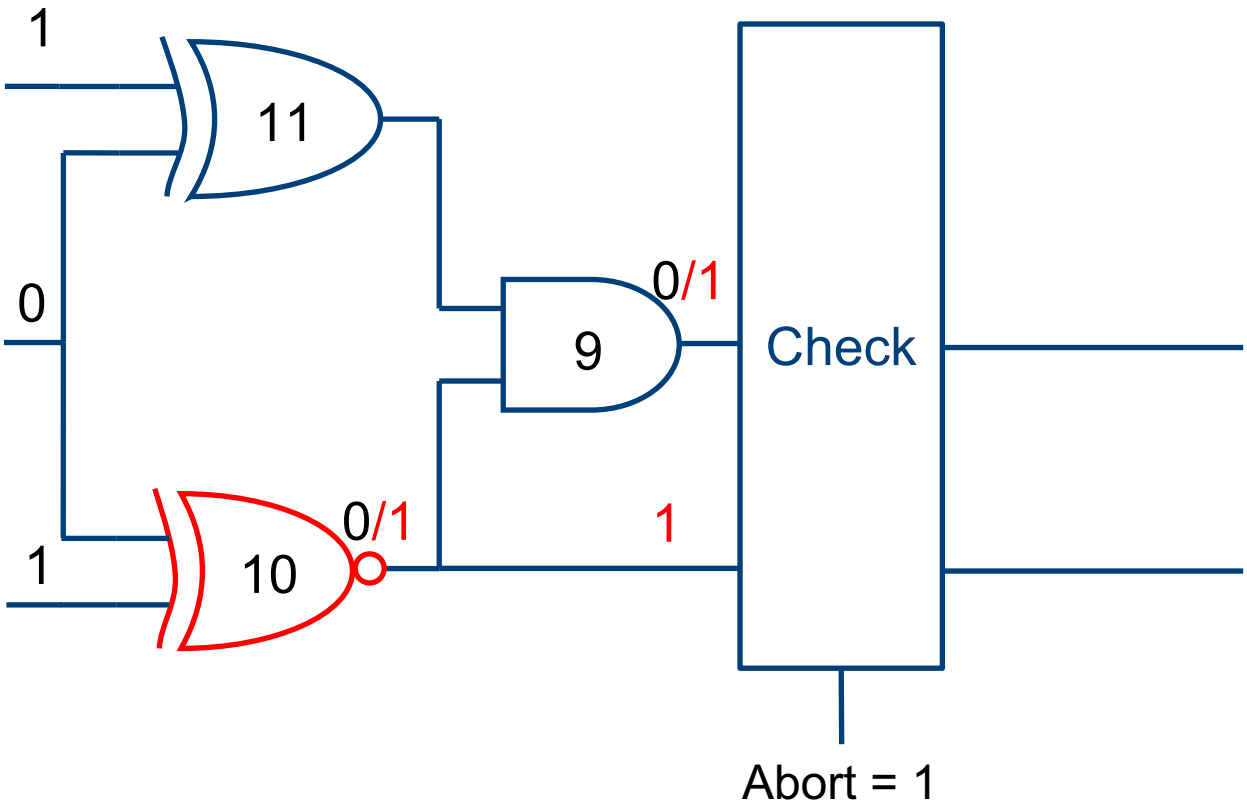
- Create faults
- Fault free simulation
- Fault injection
- Fault simulation
- Fault classification

KU LEUVEN

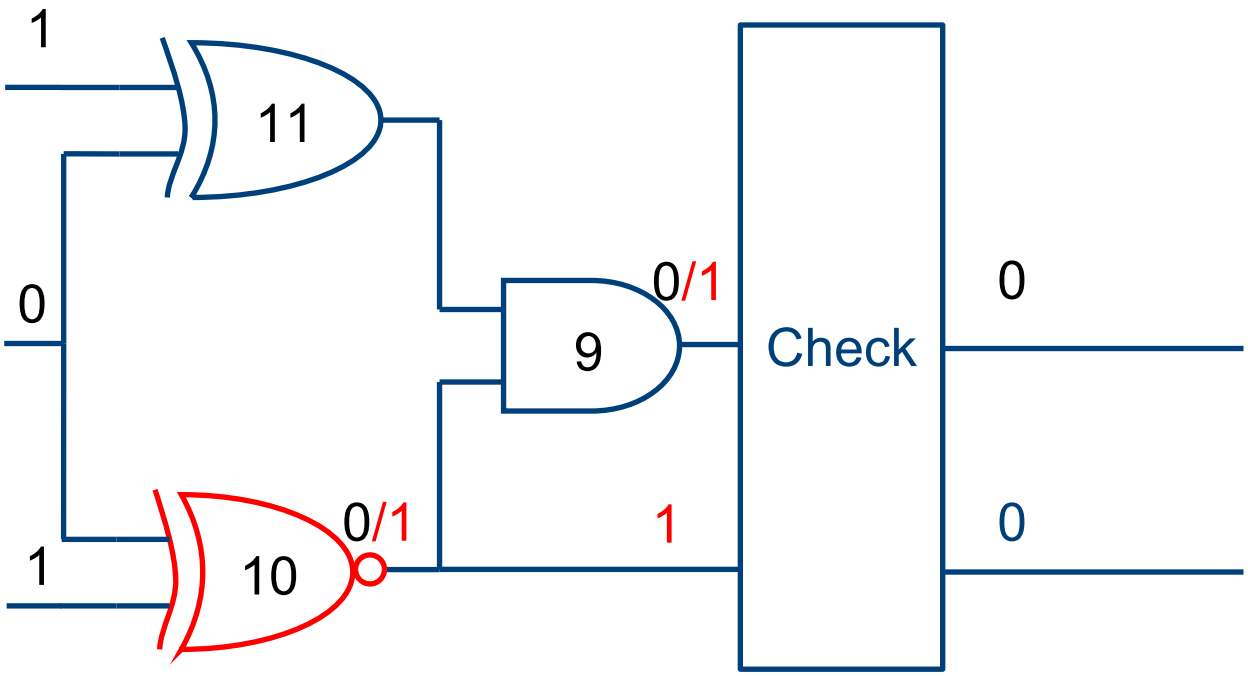
Inputs

- User defines the inputs
- Same fault evaluation per input
- Inputs dependent countermeasure

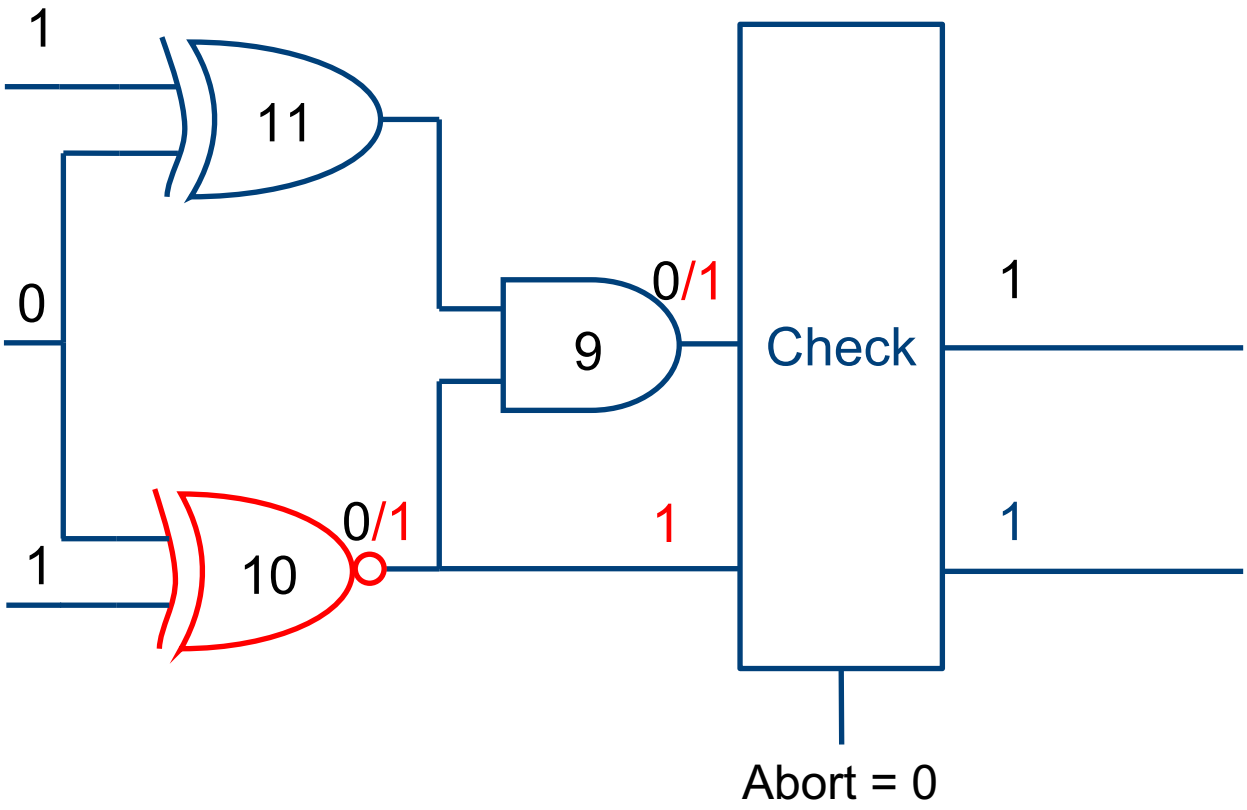
Detected



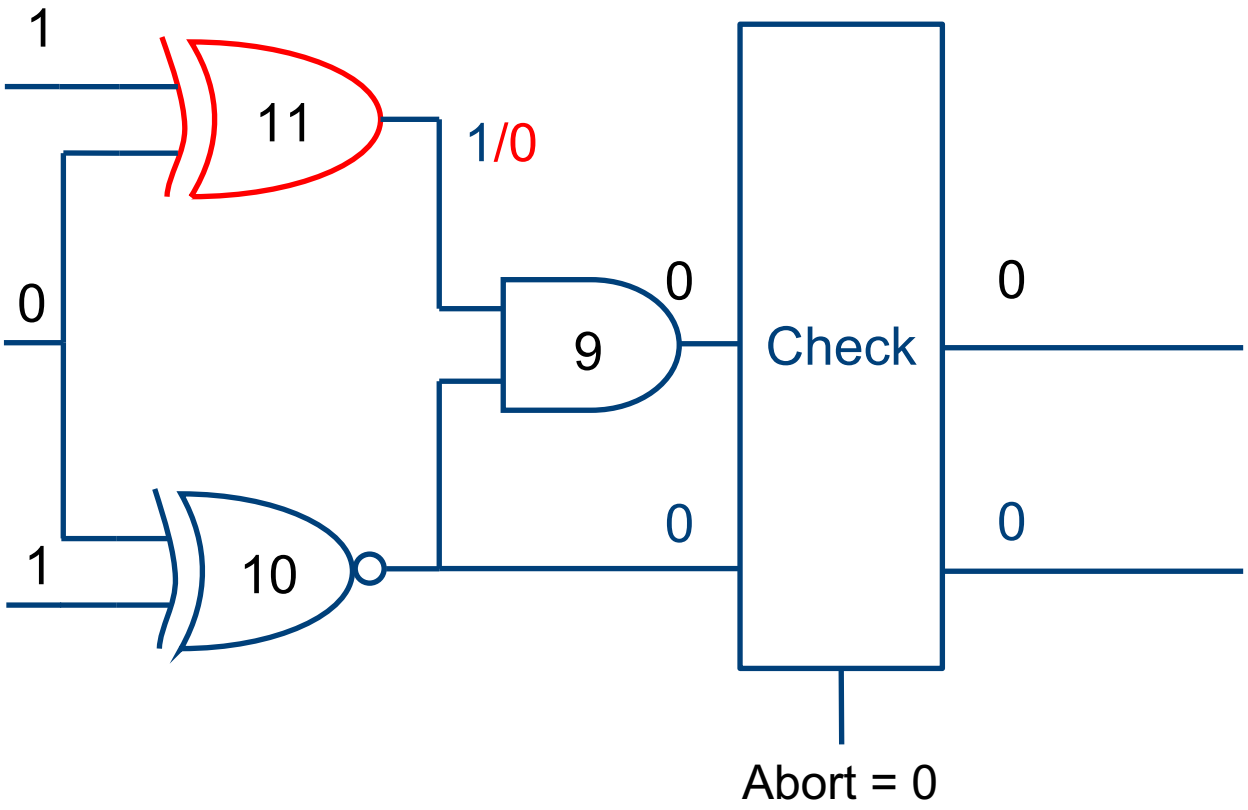
Detected



Not Detected



Ineffective



Coverage

- $Cov = \frac{Detected}{NonDetected+Detected}$

- $\pi = \frac{Ineffective}{NonDetected+Detected}$

π: Fault ineffective rate

Conclusions

- Practical SCA Verification Tool at synthesis level
- Necessary condition
- Univariate assessment
- Fault evaluation tool VerFI (in submission)
- Successful performance in practice

KU LEUVEN

Future Work

- Multivariate analysis
- Combined evaluations
- Improved performance

KU LEUVEN

Thank you!

Tools to be released by 31st October:

https://github.com/vmarribas