



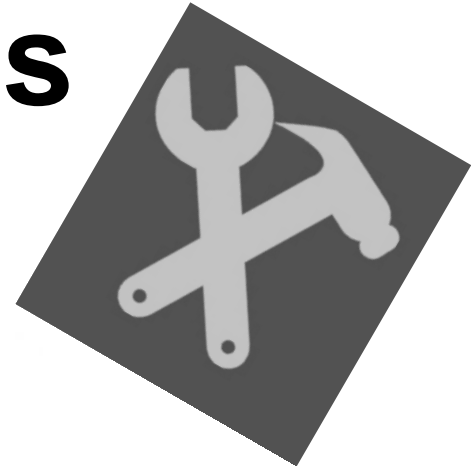
VerMI: Verification Tool for Masked Implementations

Victor Arribas

Svetla Nikova, Vincent Rijmen

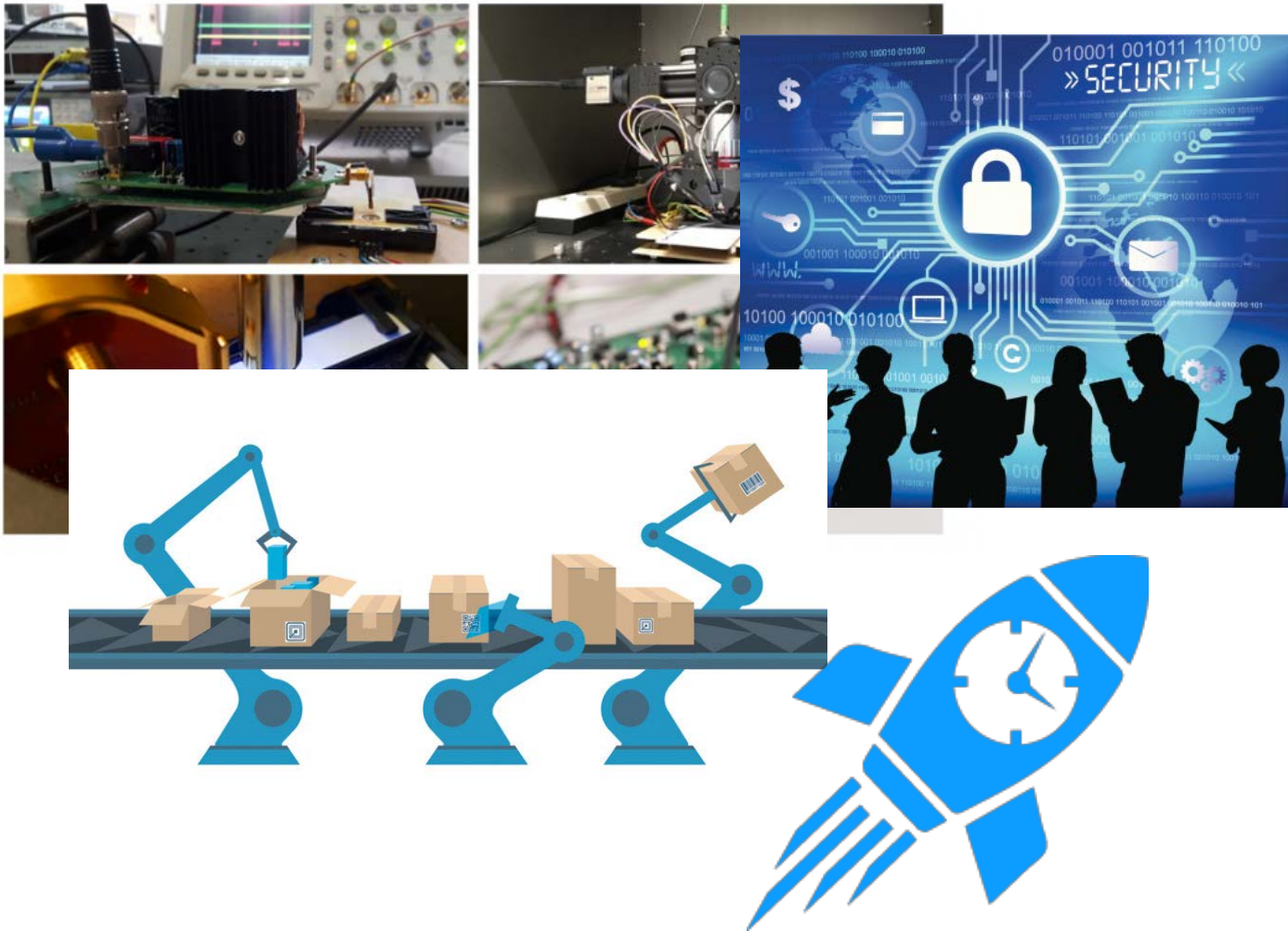


Verification Tools



WHY?

Verification Tools



VerMI

VerMI

Threshold Implementations

Non-Completeness

Sequential Logic

Uniformity

VerMI

VerMI

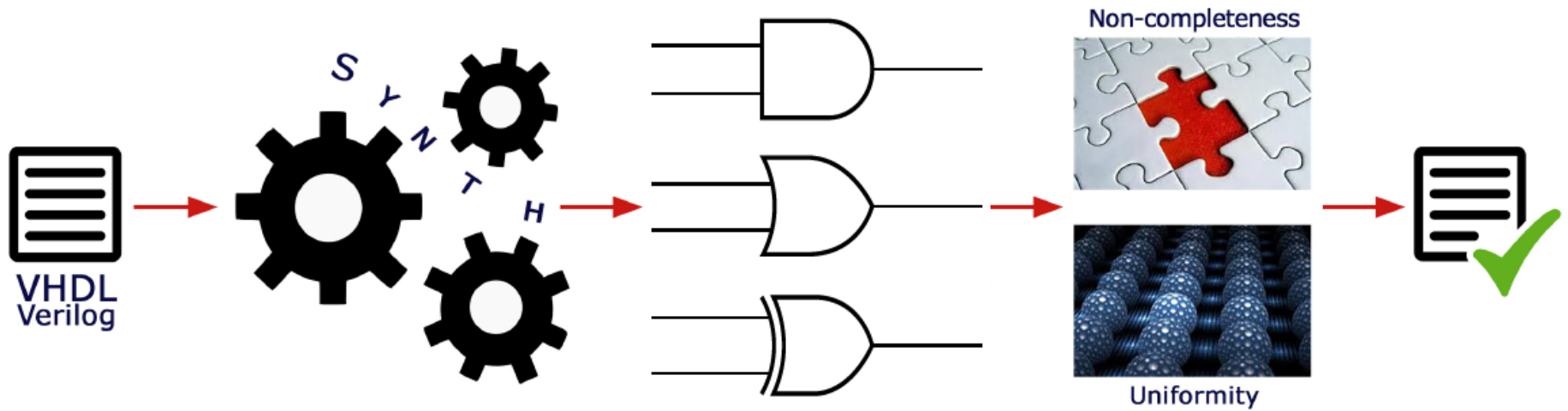
Threshold Implementations

Non-Completeness

Sequential Logic

Uniformity

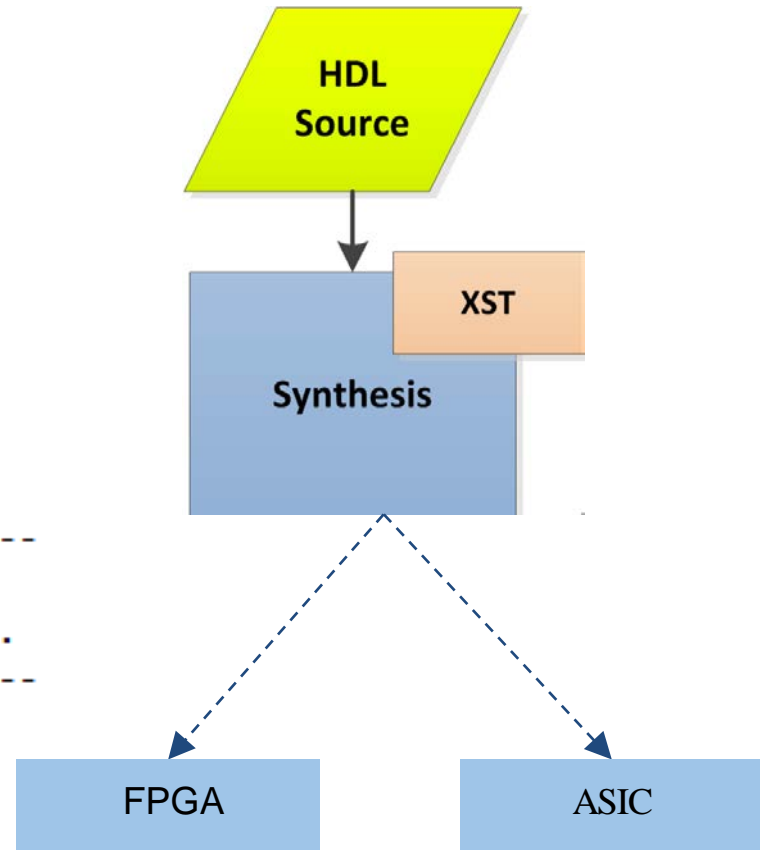
VerMI



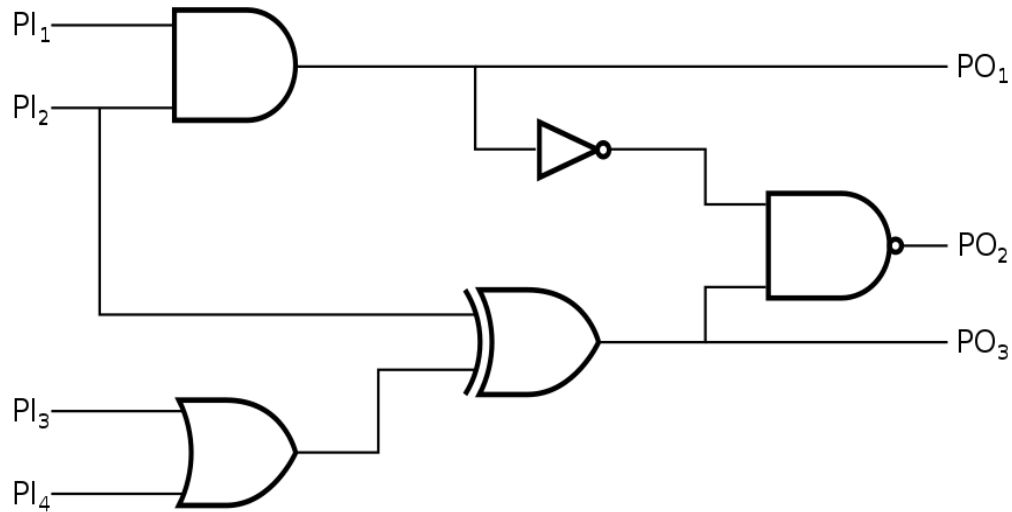
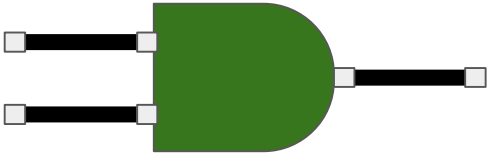
Verification Tool

- 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



VerMI

VerMI

Threshold Implementations

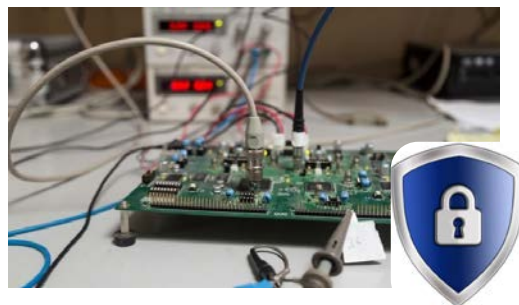
Non-Completeness

Sequential Logic

Uniformity

Threshold Implementations

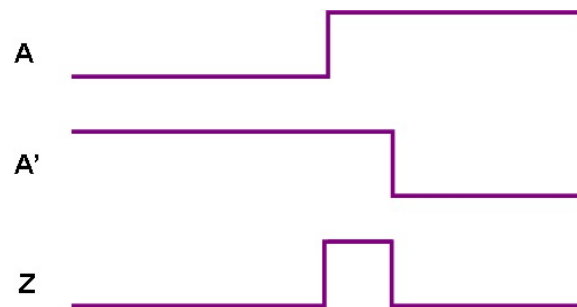
Side-Channel Analysis (SCA) countermeasure



Provable security with minimal assumptions on the HW

Q.E.D.

Security in the presence of glitches



Threshold Implementations

Boolean masking scheme

Secret sharing and multi party computation techniques

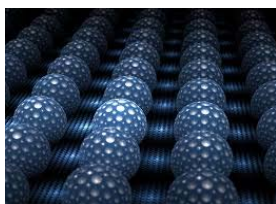
- Correctness



- Non-completeness



- Uniformity



VerMI

VerMI

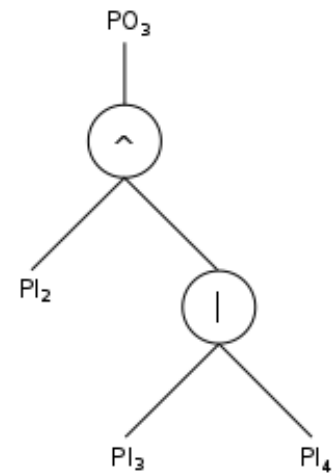
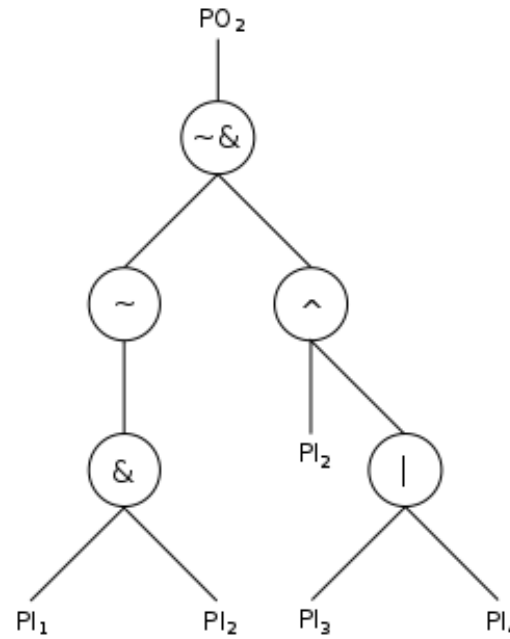
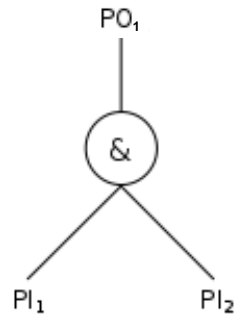
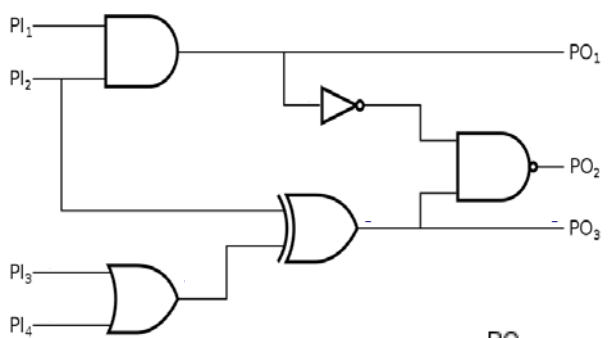
Threshold Implementations

Non-Completeness

Sequential Logic

Uniformity

Tree Search

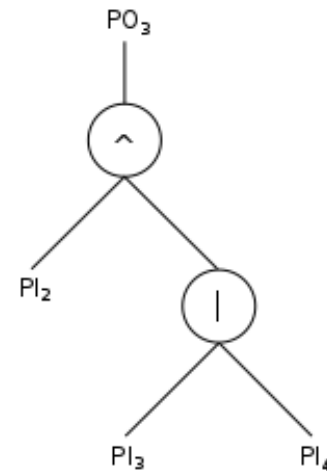
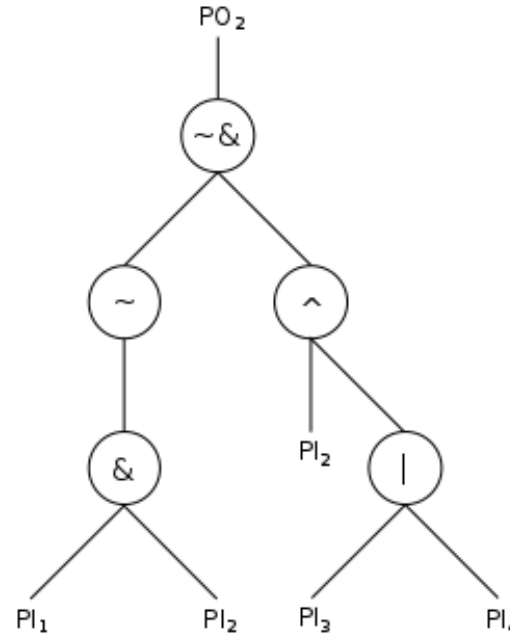
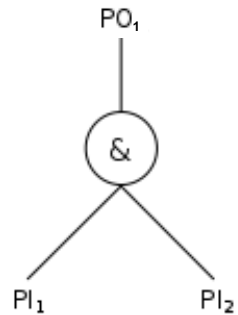
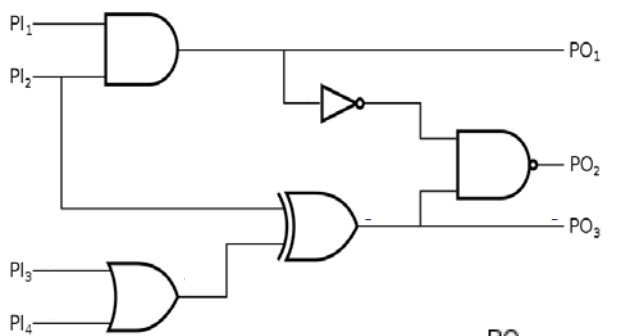


$$PO_1 = f_1(P_1, P_2)$$

$$PO_2 = f_2(P_1, P_2, P_3, P_4)$$

$$PO_3 = f_3(P_2, P_3, P_4)$$

Non-completeness



$$PO_1 = f_1(P_1, P_2) \quad \checkmark$$

$$PO_2 = f_2(P_1, P_2, P_3, P_4) \quad \times$$

$$PO_3 = f_3(P_2, P_3, P_4) \quad \checkmark$$

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


Dependencies

		Shares		
Vars.	x_1	x_2	x_3	
	y_1	y_2	y_3	


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

		Shares		
Vars.	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

		Shares		
Vars.	x_1	x_2	x_3	
	y_1	y_2	y_3	

z_1z_2			
x_1	x_2	x_3	
y_1	y_2	y_3	

z_1z_3			
x_1	x_2	x_3	
y_1	y_2	y_3	

z_2z_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	

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



VerMI

VerMI

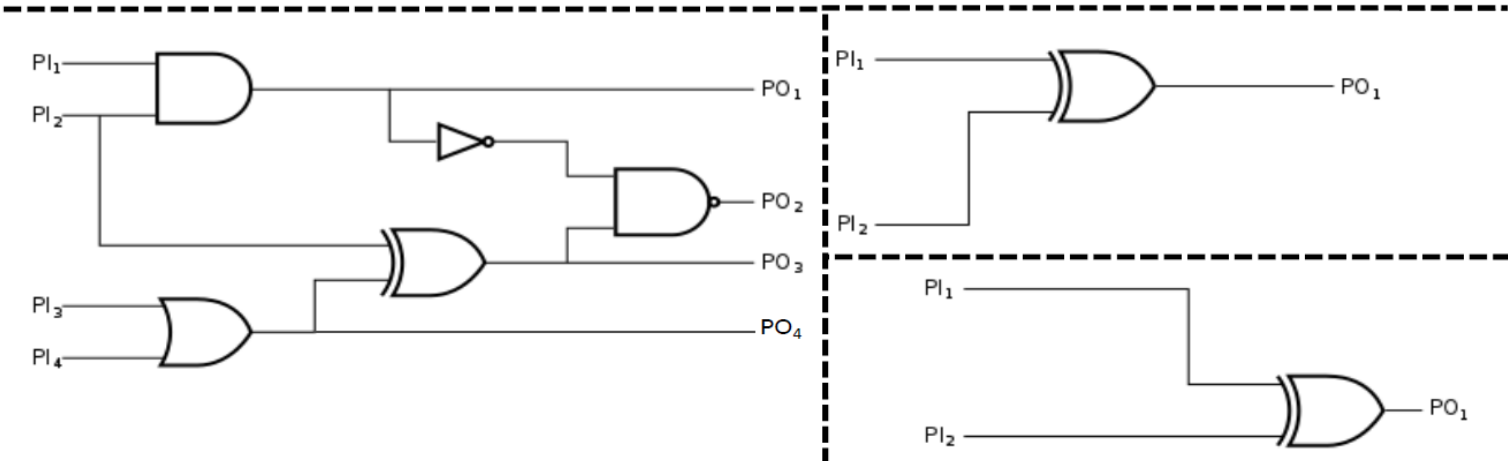
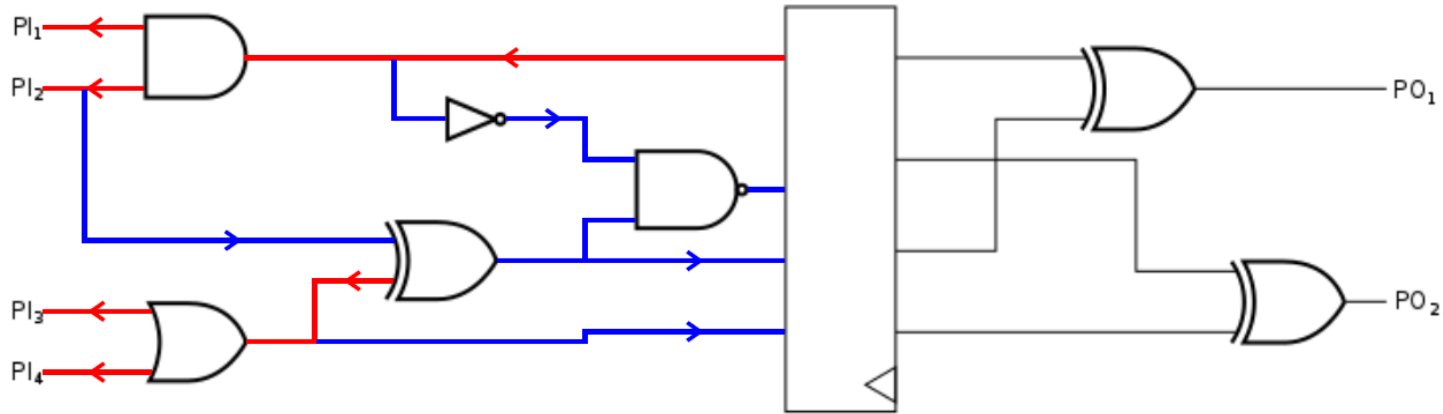
Threshold Implementations

Non-Completeness

Sequential Logic

Uniformity

Subcircuit



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.

```

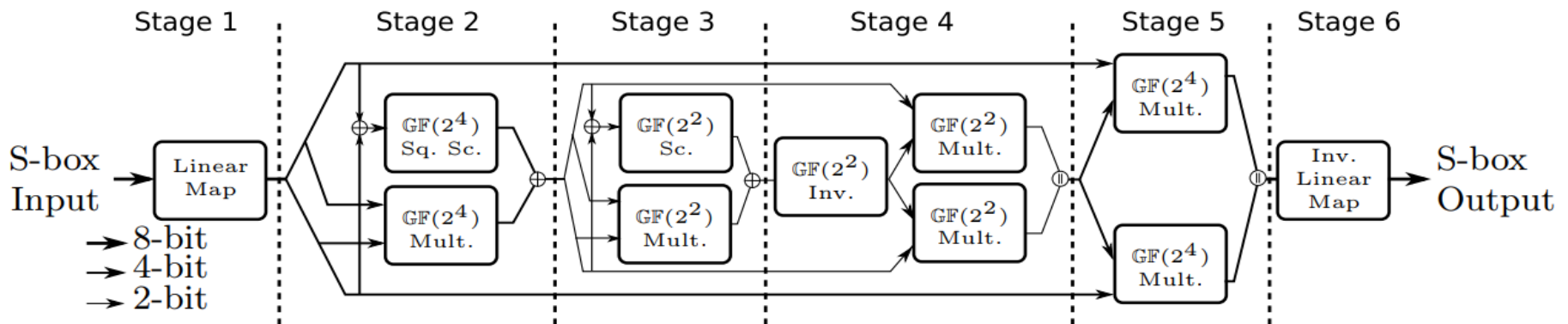
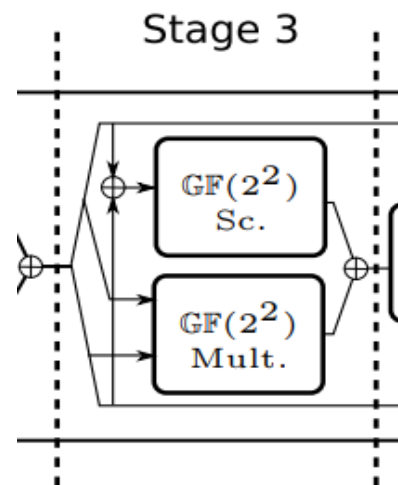


Fig. 2: Operations in the unmasked AES Sbox [CRB+]

AES Sbox

		Shares			
Variables		$St3_1[0]$	$St3_2[0]$	$St3_3[0]$	$St3_4[0]$
		$St3_1[1]$	$St3_2[1]$	$St3_3[1]$	$St3_4[1]$
		$St3_1[2]$	$St3_2[2]$	$St3_3[2]$	$St3_4[2]$
		$St3_1[3]$	$St3_2[3]$	$St3_3[3]$	$St3_4[3]$
		$St3_Top_1[0]$	$St3_Top_2[0]$		
		$St3_Top_1[1]$	$St3_Top_2[1]$		
		$St3_Top_1[2]$	$St3_Top_2[2]$		
		$St3_Top_1[3]$	$St3_Top_2[3]$		
		$St3_Bot_1[0]$	$St3_Bot_2[0]$		
		$St3_Bot_1[1]$	$St3_Bot_2[1]$		
		$St3_Bot_1[2]$	$St3_Bot_2[2]$		
		$St3_Top_1[3]$	$St3_Top_2[3]$		



Performing in Practice

NC failure in the parallel implementation protected Keccak [GSMa]

Acknowledged by the authors [GSMb]

Published in CHES 2018 [ABP+]

[ABP+] V. Arribas, B. Bilgin, G. Petrides, S. Nikova, and V. Rijmen.
Rhythmic Keccak: SCA Security and Low Latency in HW. In TCHES 2018

[GSMa] H. Gross, D. Schaffenrath, and S. Mangard.
Higher-Order Side-Channel Protected Implementations of KECCAK. In Euromicro DSD, 2017

[GSMb] H. Gross, D. Schaffenrath, and S. Mangard.
Higher-Order Side-Channel Protected Implementations of KECCAK. Cryptology ePrint Archive, 2017/395.

VerMI

VerMI

Threshold Implementations

Non-Completeness

Sequential Logic

Uniformity

Uniformity

E.g.: Multiplier

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

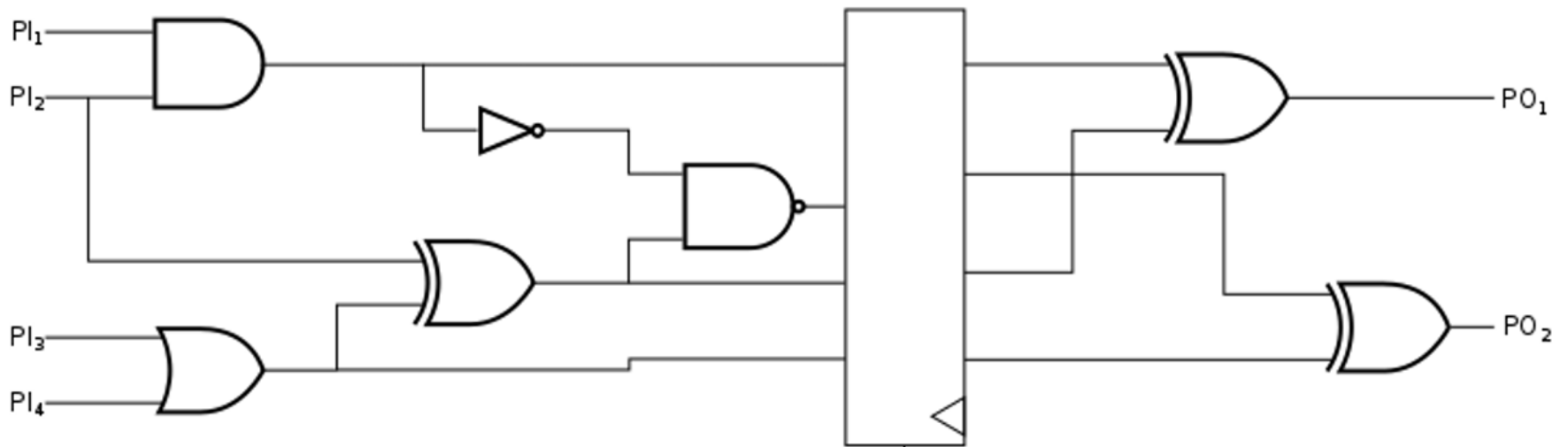
$$z_2 = F_2(x_2, x_3, y_2, y_3) = x_2y_2 \oplus x_2y_3 \oplus x_3y_2$$

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

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

Simulation

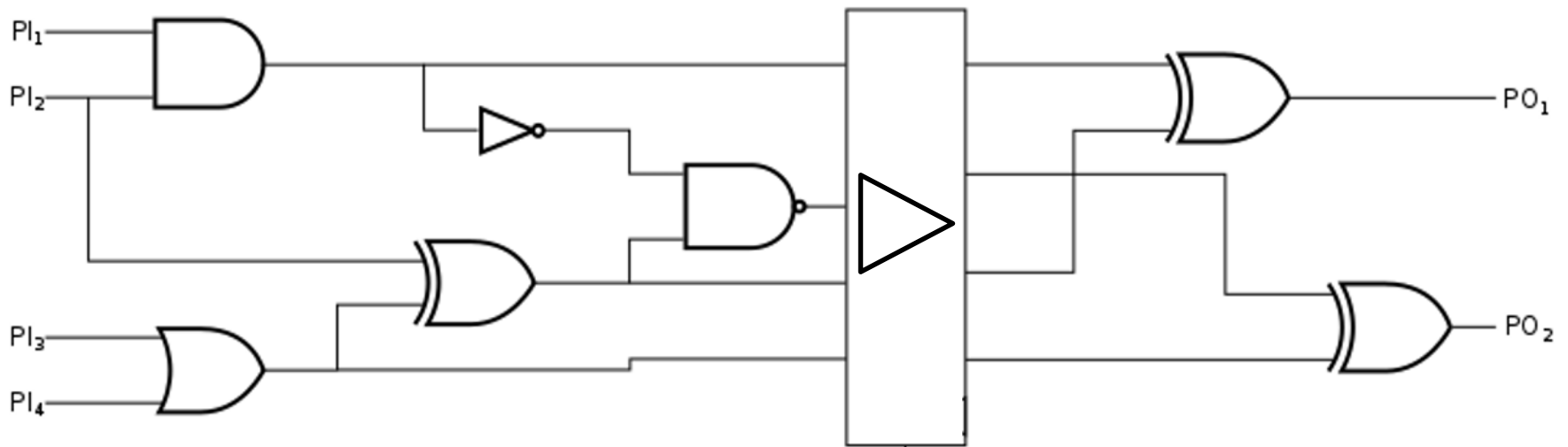
Event-Driven simulation



Simulation

Event-Driven simulation

Flip-Flops treated as buffers



Uniformity

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

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

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

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

“Changing of the Guards”
by J. Daemen [Daemen]

$$A_i = S_a(b_i, c_i) + b_{i-1} + c_{i-1}$$

$$B_i = S_b(a_i, c_i) + c_{i-1}$$

$$C_i = S_c(a_i, b_i) + b_{i-1}$$

$$B_0 = c_m$$

$$C_0 = b_m$$

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

$$A'_i \leftarrow B_i + B_{i+2} + ((B_{i+1} + C_{i+1} + D_{i+1})(B_{i+2} + C_{i+2} + D_{i+2})),$$

$$B'_i \leftarrow C_i + C_{i+2} + (A_{i+1}(C_{i+2} + D_{i+2}) + A_{i+2}(C_{i+1} + D_{i+1}) + A_{i+1}A_{i+2}),$$

$$C'_i \leftarrow D_i + D_{i+2} + (A_{i+1}B_{i+2} + A_{i+2}B_{i+1}),$$

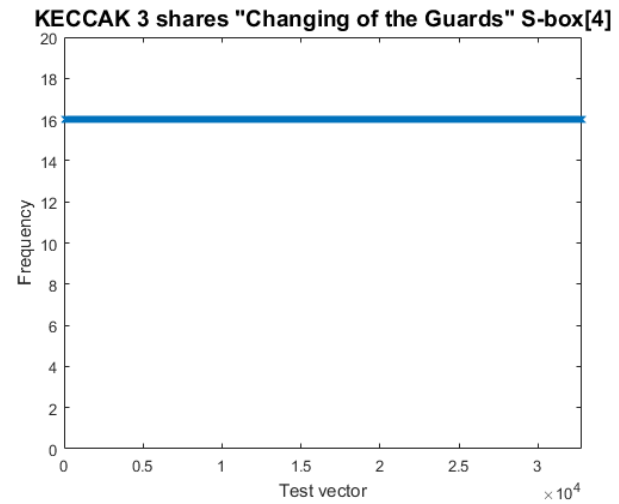
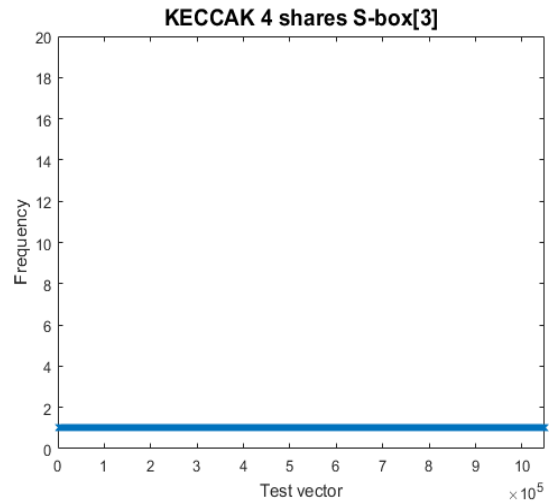
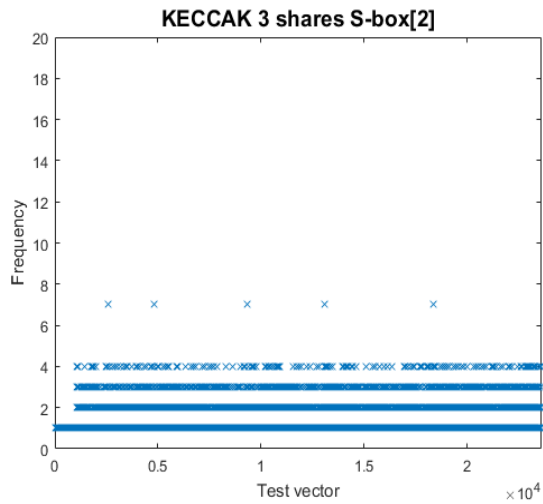
$$D'_i \leftarrow A_i + A_{i+2},$$

[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



Conclusion

- Practical Verification Tool at synthesis level
- Necessary condition
- Univariate assessment
- Successful performance in practice

Future (& Present) Work

- Fault detection capabilities already implemented
VerFI (in submission)
- Multivariate analysis
- Combined evaluations

Thank you!

