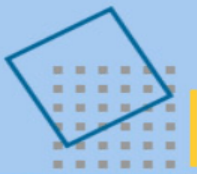


Test & Security



G. DiNatale, M-L. Flottes, B. Rouzeyre
K. Bouselam, J. Da Rolt, J. Di Battista
D. Hély, M. Doulcier



Test & Security : the dilemma

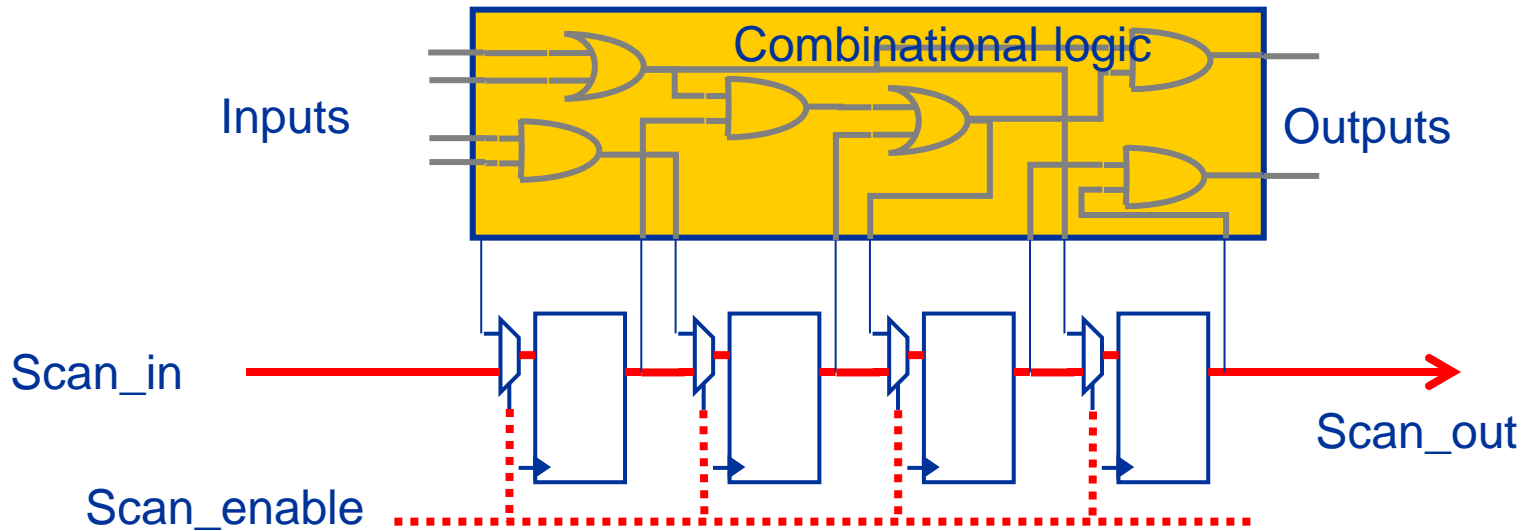
- Circuit testing is mandatory to guarantee a good security level

A hardware defect may induce some security vulnerability

- But

	Test	Security
Observability		
Controlability		

External Test + Scan path

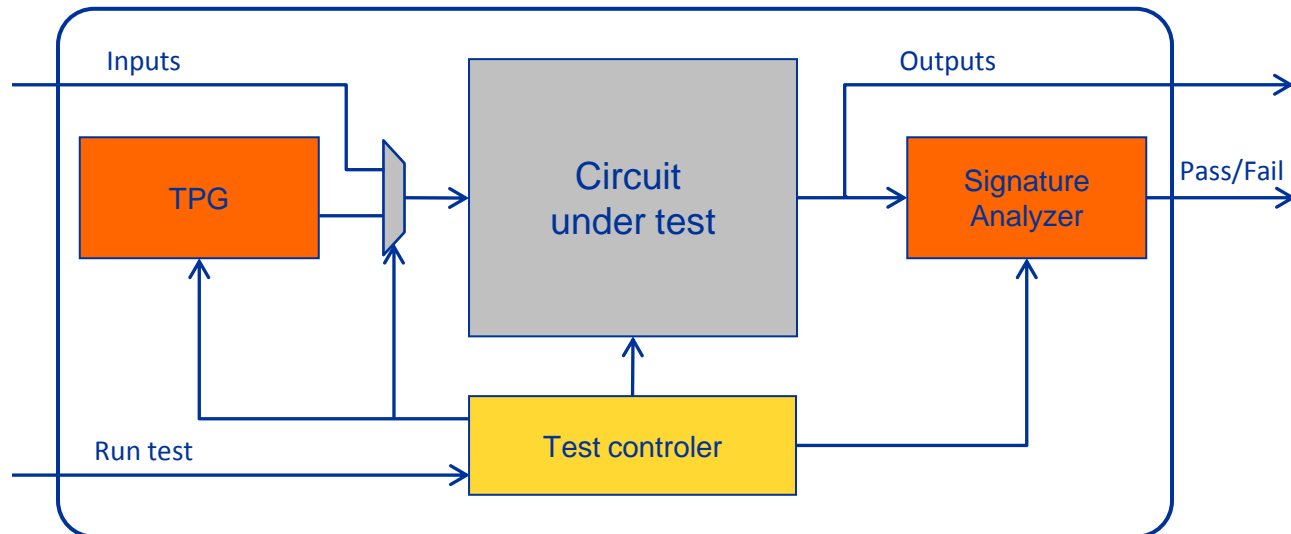


- High fault coverage
- Automatic generation of scan chains
- Easy test sequence generation

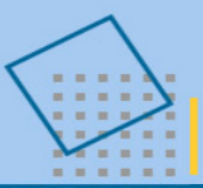
Vulnerability

- Control and observation of internal states of CUT
- => secret data retrieval

■ Built-in Self Test (BIST)



- No control/observation from the outside
- Area overhead
- Fault coverage (pseudo-random testing) ?



Scan-out based attack

LIRMM

■ Goal

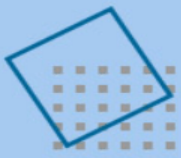
- Retrieve the User key (K)

■ Principle

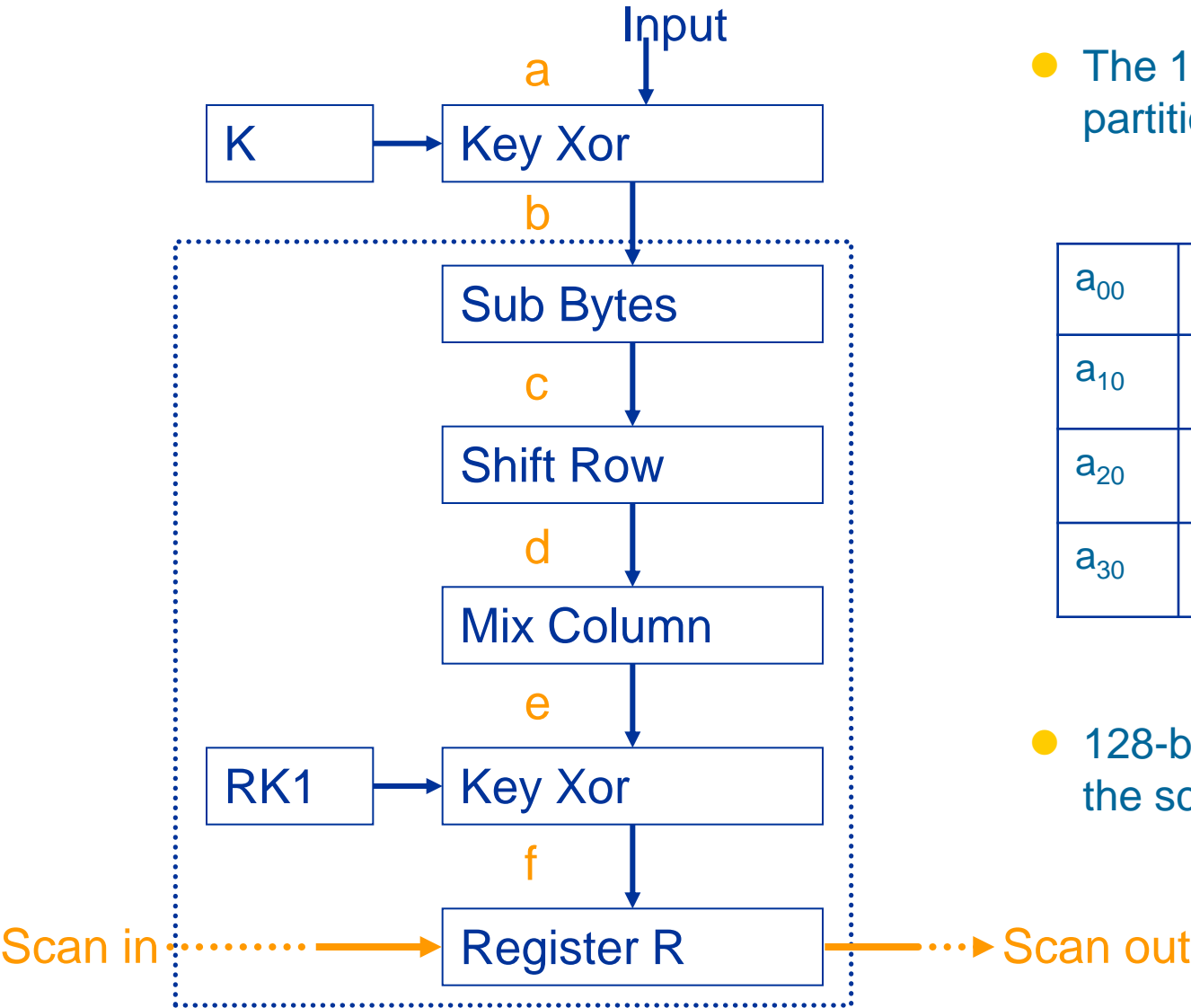
- Use the scan chain to observe the data processed by the circuit at various moments

■ Method

- 1- Retrieve the FFs storing the cipher text (among all FFs in the scan chain)
- 2- Knowing the plaintext and the cipher text, compute the user key



AES implementation



- The 128 bits input is partitioned into 16 bytes

a_{00}	a_{01}	a_{02}	a_{03}
a_{10}	a_{11}	a_{12}	a_{13}
a_{20}	a_{21}	a_{22}	a_{23}
a_{30}	a_{31}	a_{32}	a_{33}

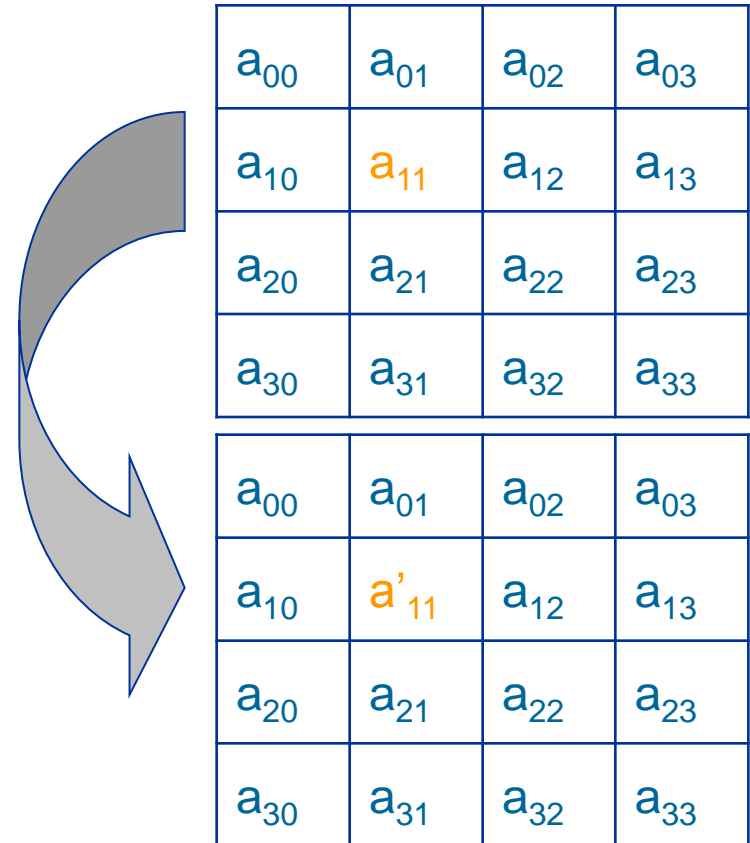
- 128-bits Round register R is in the scan chain

AES scan attack: retrieving the FFs of interest

LIRMM

- Note: change plaintext a to a' \Rightarrow change data in R

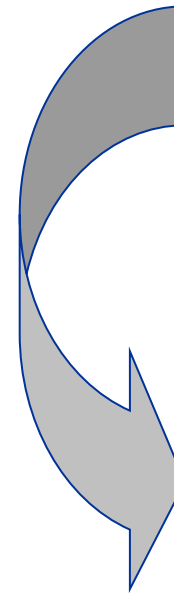
- a_{11} change to a'_{11} (1 byte in input)



AES scan attack: retrieving the FFs of interest

■ Note: change plaintext a to a' => change data in R

- a_{11} change to a'_{11} (1 byte in input)
- b_{11} change to b'_{11} (XOR Key)



b_{00}	b_{01}	b_{02}	b_{03}
b_{10}	b_{11}	b_{12}	b_{13}
b_{20}	b_{21}	b_{22}	b_{23}
b_{30}	b_{31}	b_{32}	b_{33}

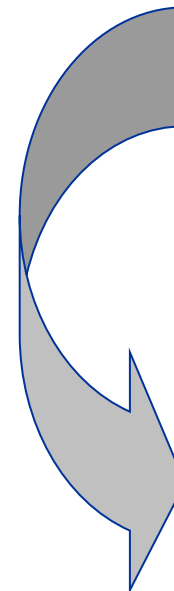
b_{00}	b_{01}	b_{02}	b_{03}
b_{10}	b'_{11}	b_{12}	b_{13}
b_{20}	b_{21}	b_{22}	b_{23}
b_{30}	b_{31}	b_{32}	b_{33}

AES scan attack: retrieving the FFs of interest

LIRMM

■ Note: change plaintext a to a' \Rightarrow change data in R

- a_{11} change to a'_{11}
- b_{11} change to b'_{11}
- c_{11} change to c'_{11} (SubByte)



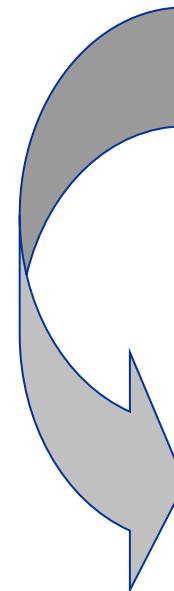
C_{00}	C_{01}	C_{02}	C_{03}
C_{10}	C_{11}	C_{12}	C_{13}
C_{20}	C_{21}	C_{22}	C_{23}
C_{30}	C_{31}	C_{32}	C_{33}

C_{00}	C_{01}	C_{02}	C_{03}
C_{10}	C'_{11}	C_{12}	C_{13}
C_{20}	C_{21}	C_{22}	C_{23}
C_{30}	C_{31}	C_{32}	C_{33}

AES scan attack: retrieving the FFs of interest

■ Note: change plaintext a to a' \Rightarrow change data in R

- a_{11} change to a'_{11}
- b_{11} change to b'_{11}
- c_{11} change to c'_{11}
- d_{10} change to d'_{10} (ShiftRow)



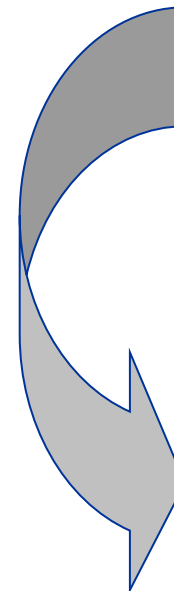
d_{00}	d_{01}	d_{02}	d_{03}
d_{10}	d_{11}	d_{12}	d_{13}
d_{20}	d_{21}	d_{22}	d_{23}
d_{30}	d_{31}	d_{32}	d_{33}

d_{00}	d_{01}	d_{02}	d_{03}
d'_{10}	d_{11}	d_{12}	d_{13}
d_{20}	d_{21}	d_{22}	d_{23}
d_{30}	d_{31}	d_{32}	d_{33}

AES scan attack: retrieving the FFs of interest

■ Note: change plaintext a to a' \Rightarrow change data in R

- a_{11} change to a'_{11} (1 byte in input)
- b_{11} change to b'_{11}
- c_{11} change to c'_{11}
- d_{10} change to d'_{10}
- e_{i0} change to e'_{i0} (MixColumn)



e_{00}	e_{01}	e_{02}	e_{03}
e_{10}	e_{11}	e_{12}	e_{13}
e_{20}	e_{21}	e_{22}	e_{23}
e_{30}	e_{31}	e_{32}	e_{33}
e'_{00}	e_{01}	e_{02}	e_{03}
e'_{10}	e_{11}	e_{12}	e_{13}
e'_{20}	e_{21}	e_{22}	e_{23}
e'_{30}	e_{31}	e_{32}	e_{33}

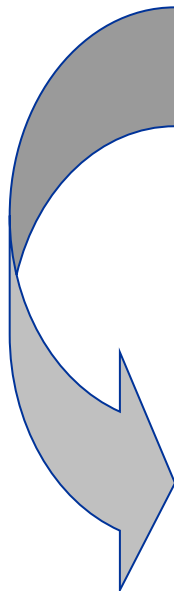
AES scan attack: retrieving the FFs of interest

LIRMM

■ Note: change plaintext a to a' \Rightarrow change data in R

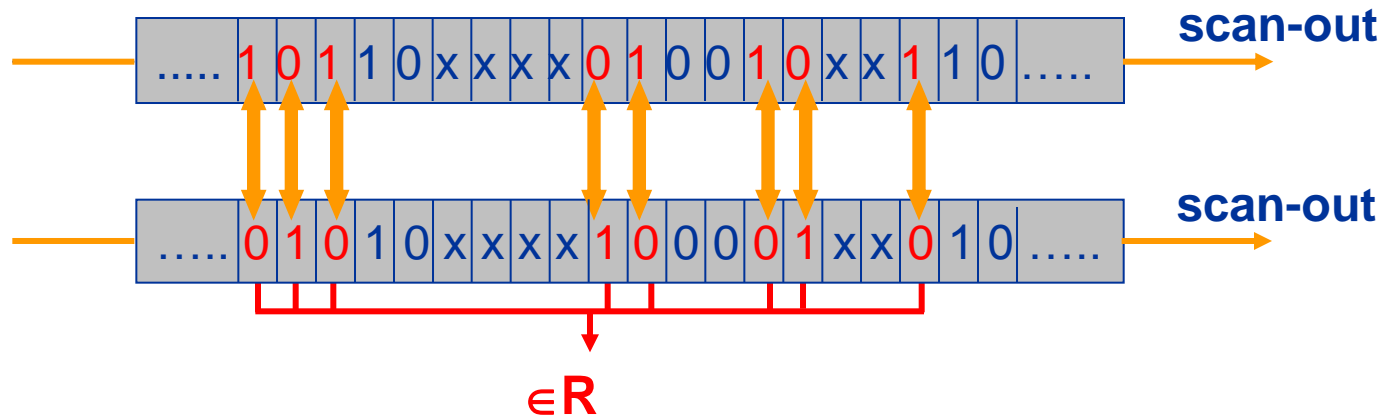
- a_{11} change to a'_{11}
- b_{11} change to b'_{11}
- c_{11} change to c'_{11}
- d_{10} change to d'_{10}
- e_{i0} change to e'_{i0}
- f_{i0} change to f'_{i0} (Add Round Key)

One input byte modification impacts 4 bytes after one round)



f_{00}	f_{01}	f_{02}	f_{03}
f_{10}	f_{11}	f_{12}	f_{13}
f_{20}	f_{21}	f_{22}	f_{23}
f_{30}	f_{31}	f_{32}	f_{33}
f'_{00}	f_{01}	f_{02}	f_{03}
f'_{10}	f_{11}	f_{12}	f_{13}
f'_{20}	f_{21}	f_{22}	f_{23}
f'_{30}	f_{31}	f_{32}	f_{33}

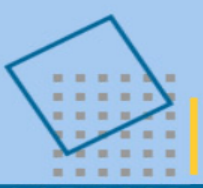
Apply $a = a_{00} \dots a_{11} \dots a_{33}$, Then $a' = a_{00} \dots a'_{11} \dots a_{33}$



Experiments : 6 plaintexts are sufficient for identifying 4 bytes in R (average)

- Apply any known message M
- Process one round (system mode)
 - Switch to test mode
 - Scan out C
 - Derive K

$$\text{Round}^{-1}(C) \oplus M = M \oplus K \oplus M = K$$



Securing the scan chain

LIRMM

■ Goal

- No observation or control of the functional data processed by the secure system

■ Principle

- Prevent illegal scan shift operations

■ Solutions

- Fuses on scan signals, but

- Test mode protection

- Scan protocol
- Test Patterns watermarking

protection against illegal usage of the test mode

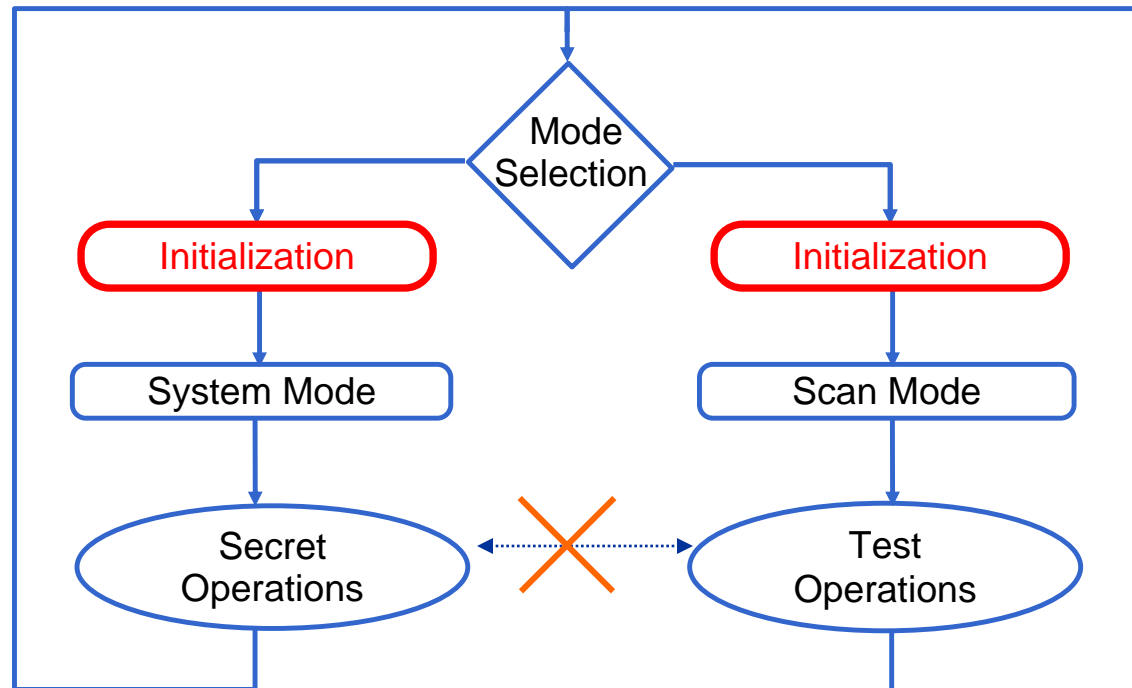
- System mode protection

- Scan chain scrambling
- Scan enable tree protection
- Spy FFs

protection against scan chain probing attacks

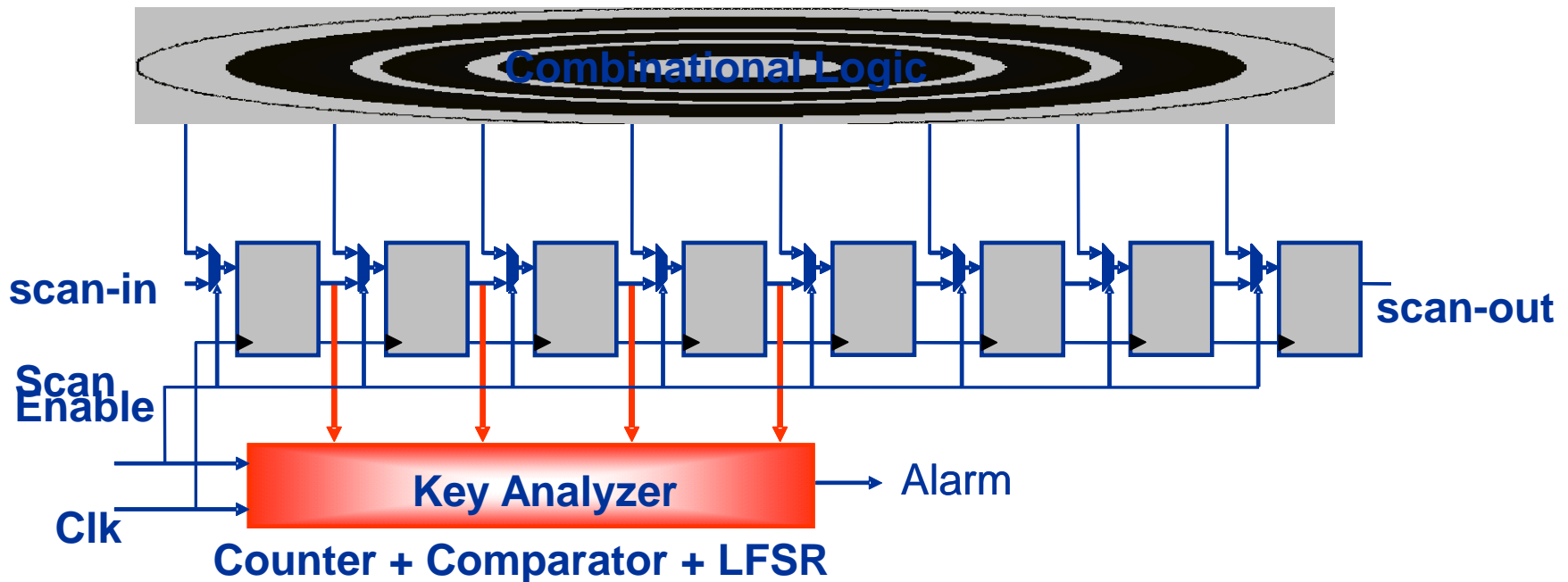
■ Scan protocol

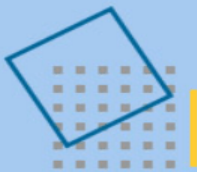
- The circuit is initialized before and after test mode
- Initialization is checked before switching to another mode
- Switch between the 2 modes, bypassing the initialization, is detected



■ Test pattern watermarking

- Test patterns embed authentication keys
- Keys are dynamically changed (e.g. LFSR-based)





System mode protection

■ Scrambling method

- Scan path with a prefixed segment organization during test mode

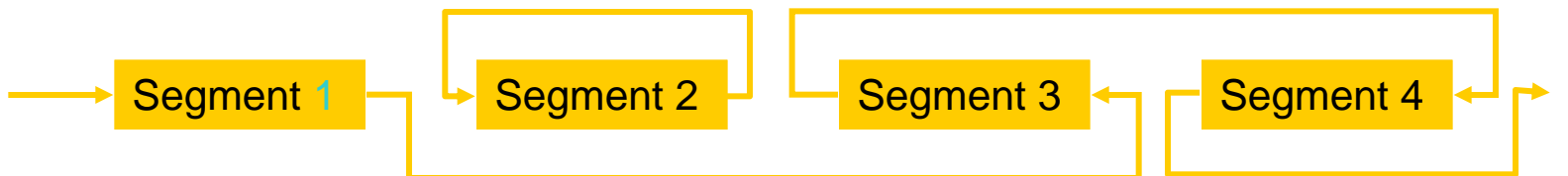


- Scan path with random segment organization if shift during system mode

- Time T1

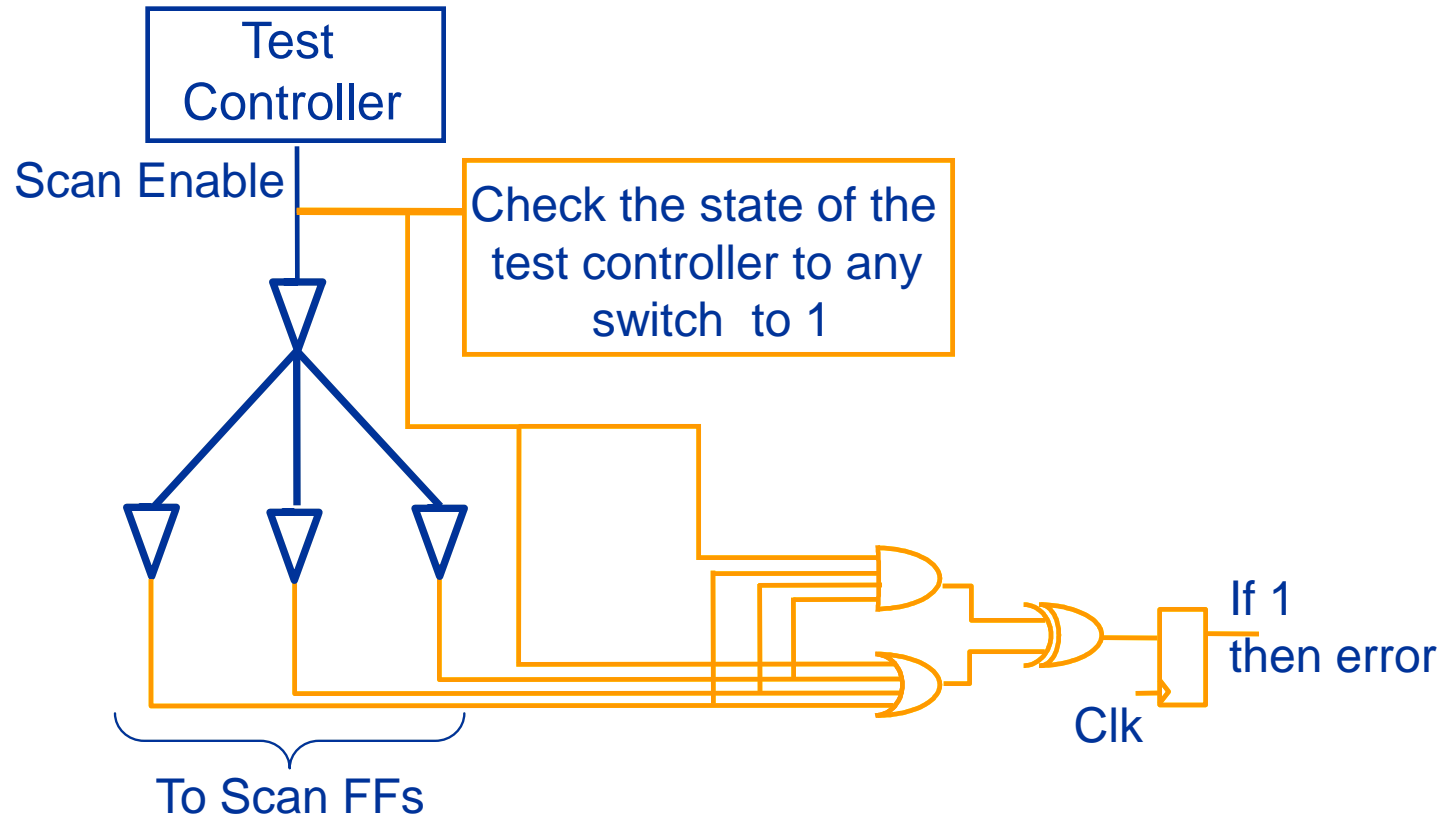


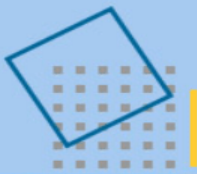
- Time T2



■ Scan-Enable Tree Protection

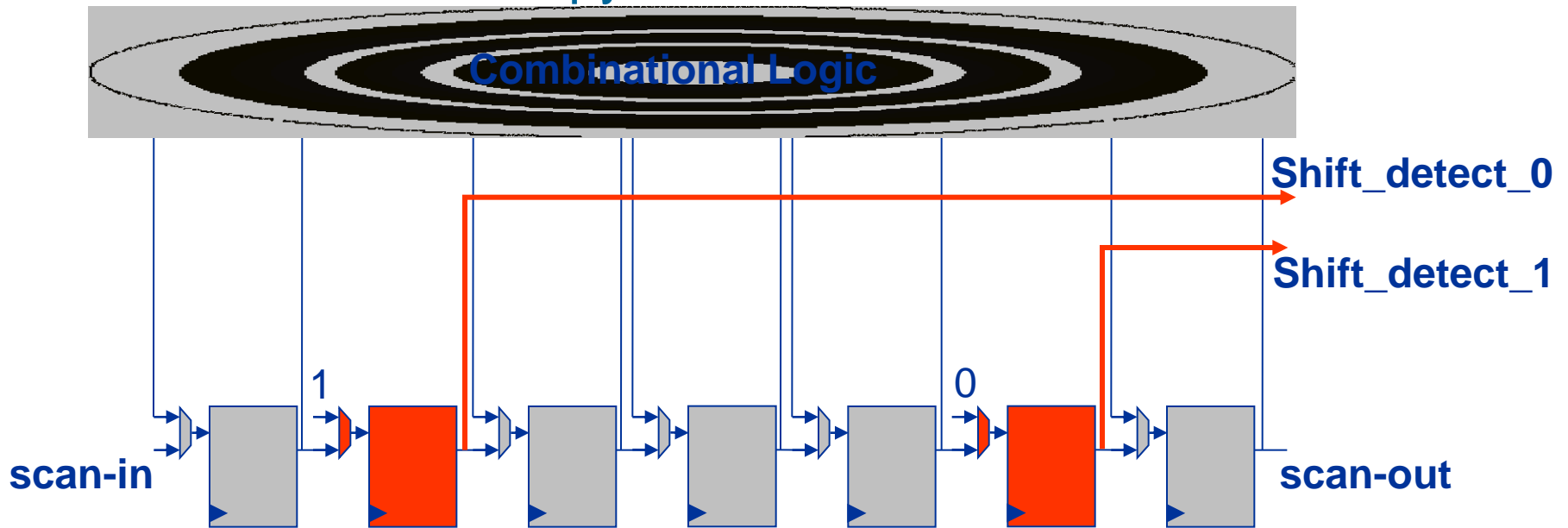
- Compare the scan enable signals at different locations

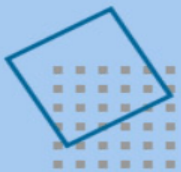




■ Spy Flip-Flops

- Include Spy cells in the scan chain
- Control the spy cells to a constant value
- Observe the spy cells states

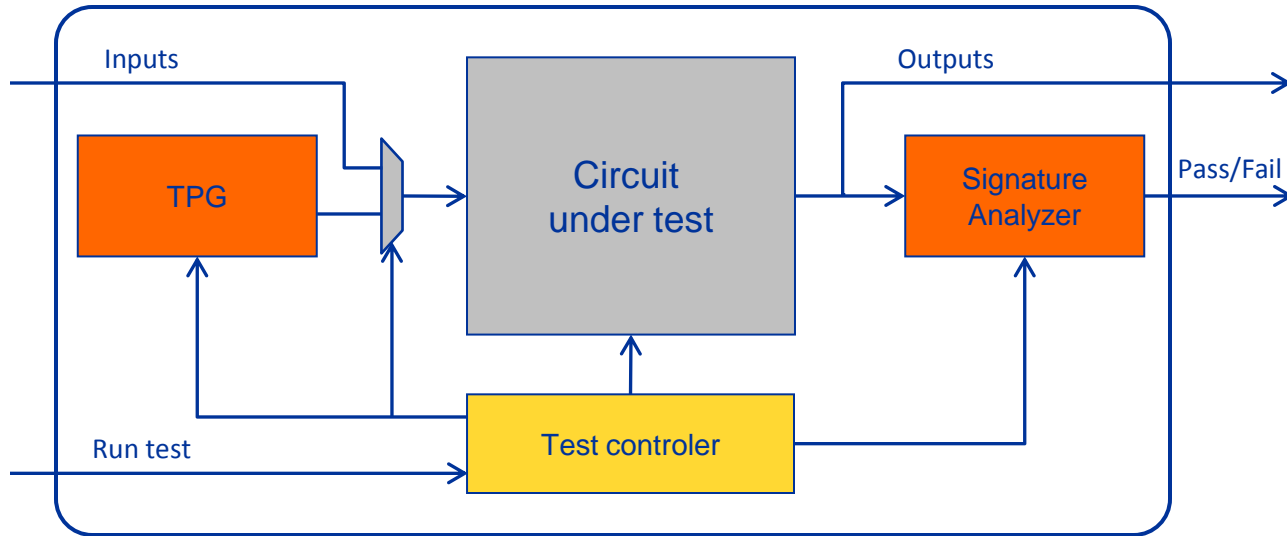




- Countermeasures address two kinds of attack
 - Legal activation of the test circuitry
 - corruption of the authentication scheme
 - malfunction of the security
 - insider attack
 - Physical access to the chip
 - high knowledge of the circuit
 - very expensive equipment

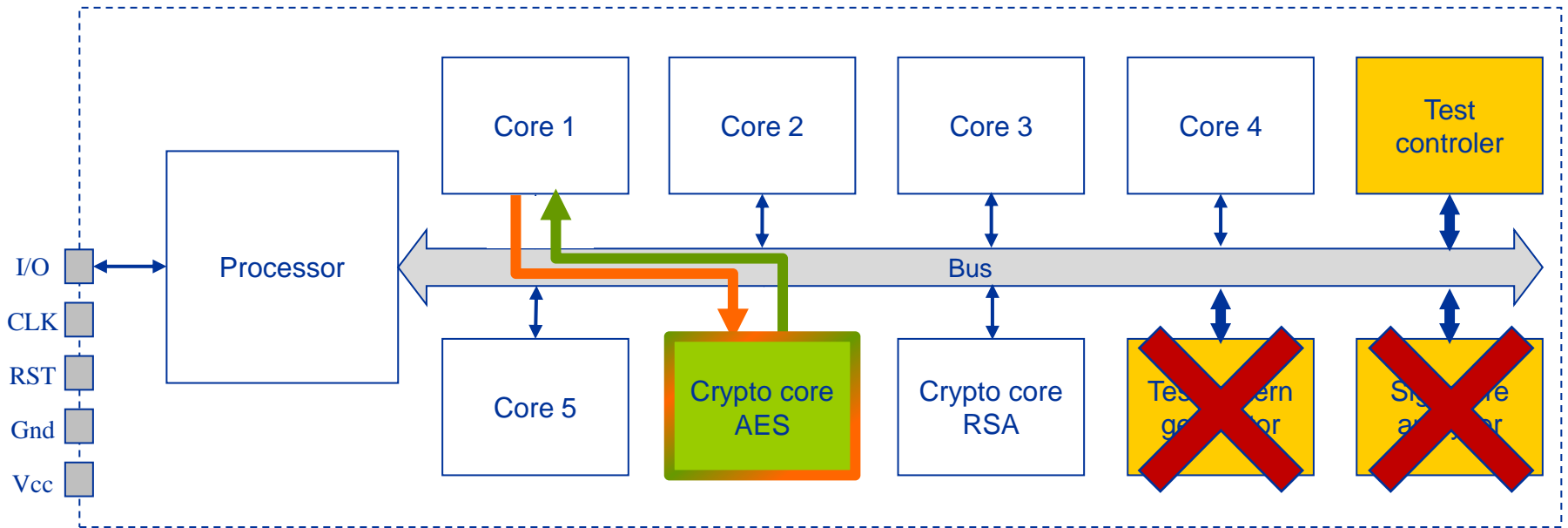
■ BIST

- Reduced ATE cost
- In-situ testing
- Reduced external access

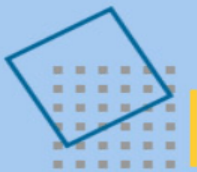


■ But

- Circuitry overhead



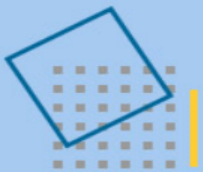
- Self-test of crypto-core
- Use the crypto-core as a test resource (TPG/SA)
- AES/DES



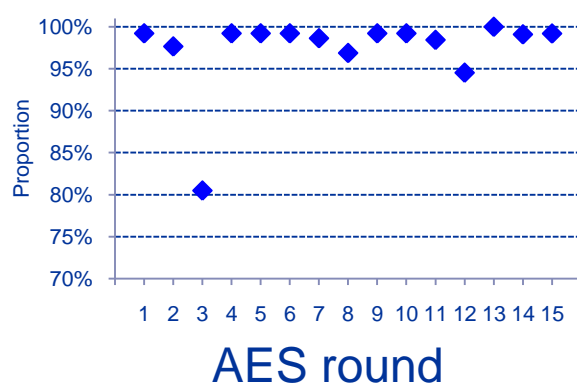
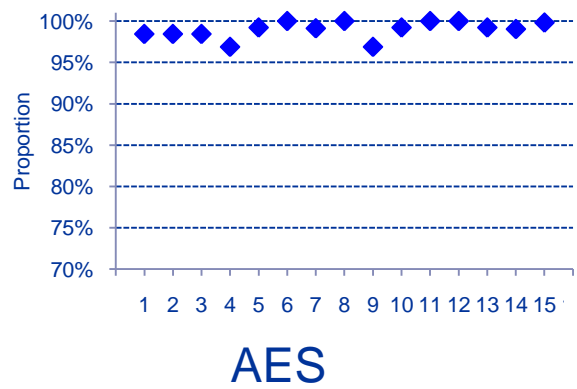
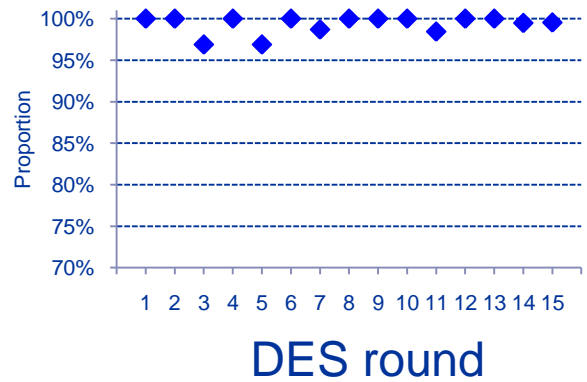
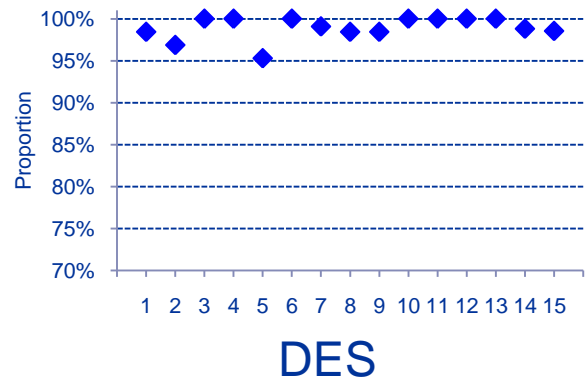
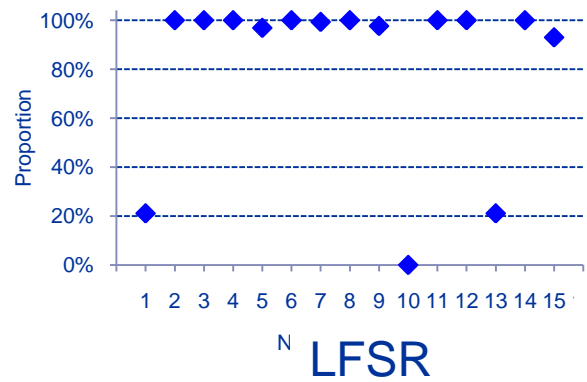
"Randomness" of cipher

LIRMM

- Self Test , BIST \cong random test
- Randomness of ciphering algorithms
 - already studied but 1 vector every encryption
- Randomness of AES/DES rounds ?



Randomness comparison

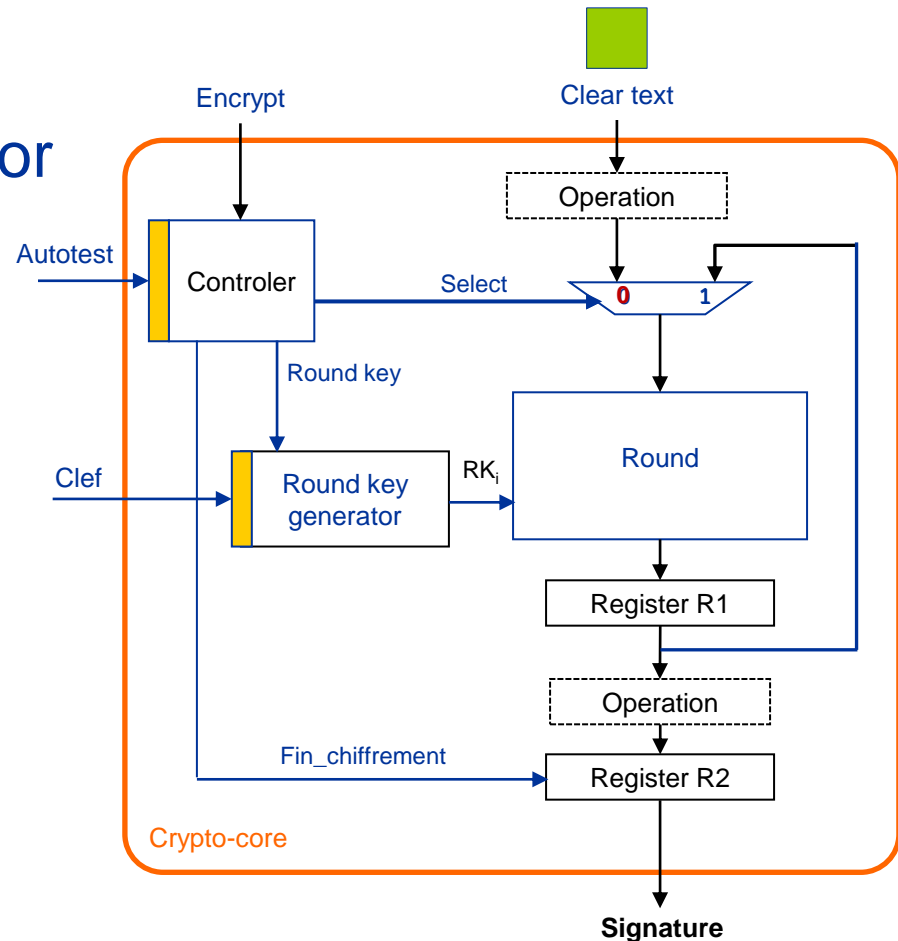


AES round / DES round : as good random pattern generators as LFSRs

Self-test Procedure

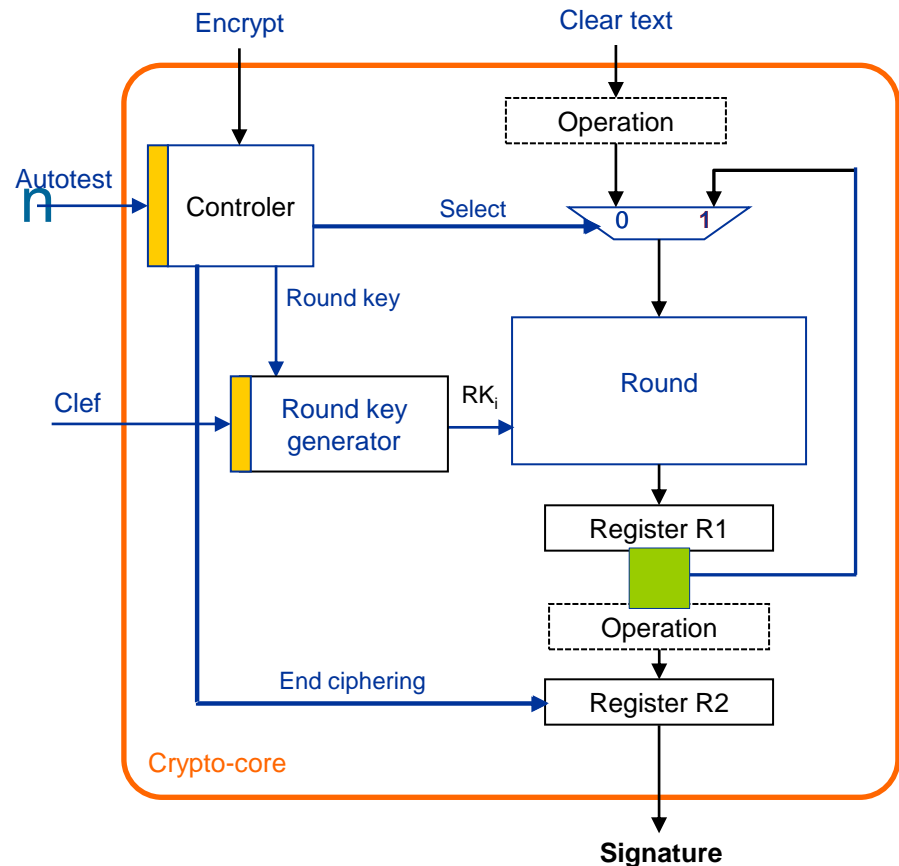
- Looped Crypto-core \Leftrightarrow random number generator

- First step
 - ✓ 1st cycle



■ Second step

- Cycles 2, 3, ..., n

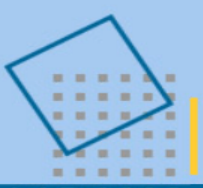


■ Theoretical result

- AES : Fault-coverage = 100% after $n \in \{2520, \dots, 2590\}$ clock cycles
- DES : Fault-coverage = 100% after $n \in \{620, \dots, 710\}$ clock cycles

■ In practice

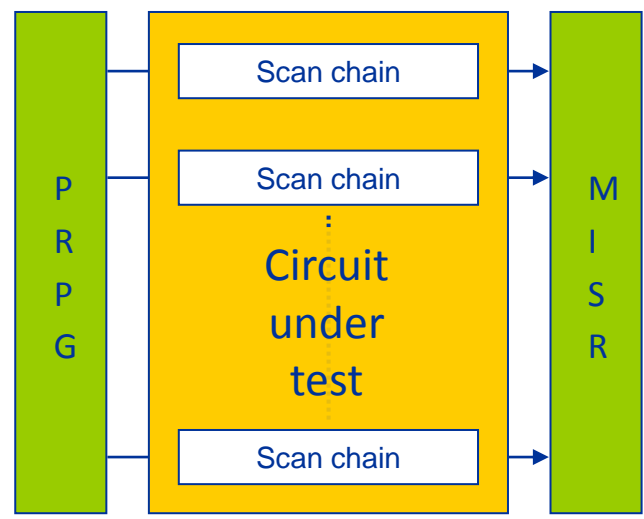
- AES
 - Fault-coverage = 100% after 2200 clock cycles (\forall key, \forall clear text)
- DES
 - Fault-coverage = 100% after 560 clock cycles (\forall key (not wk), \forall clear text)



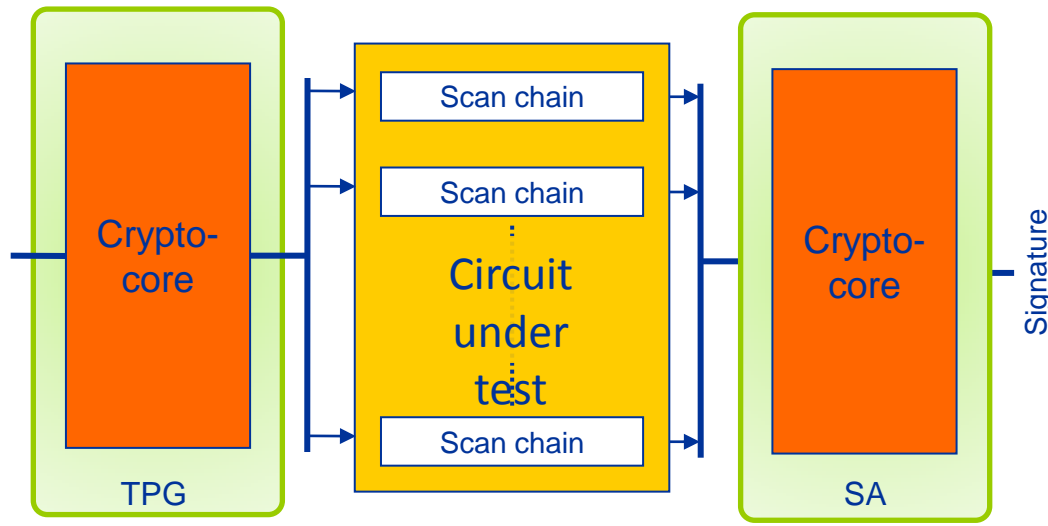
Crypto-core as TPG/SA

LIRMM

STUMPS Architecture



Proposed solution



TPG for other cores

Test response compactor for other cores

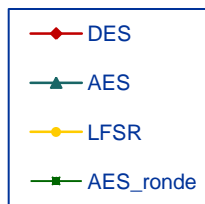
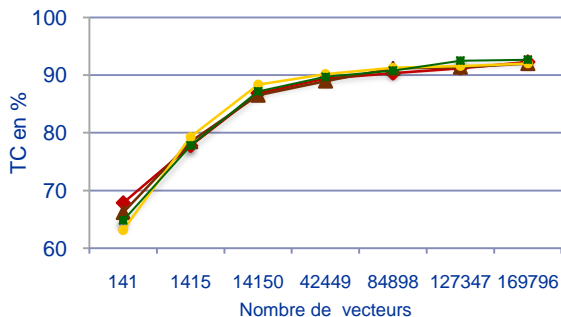


TPG : ISCAS'89 benchmarks

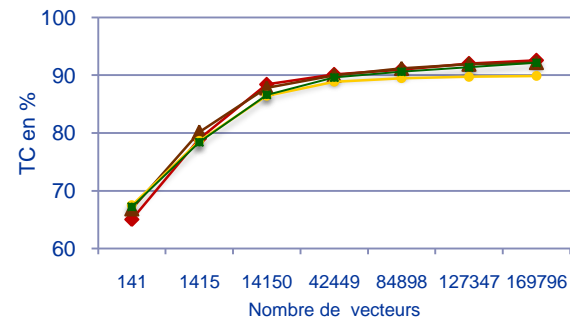
LIRMM

■ s9234

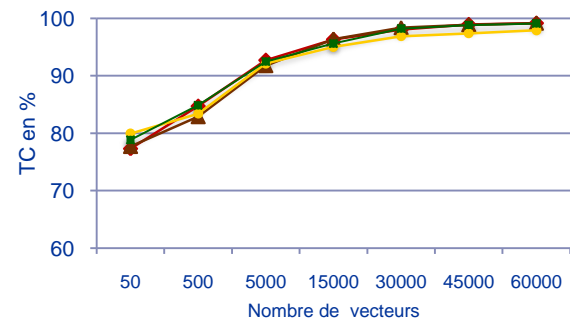
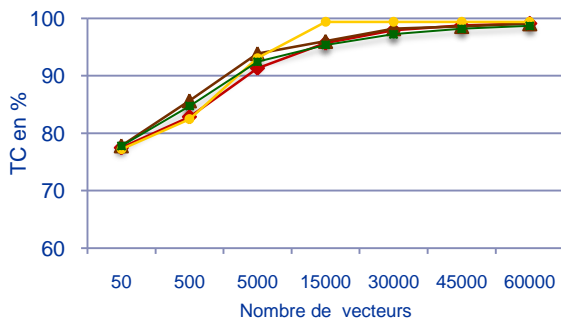
1 Scan chain



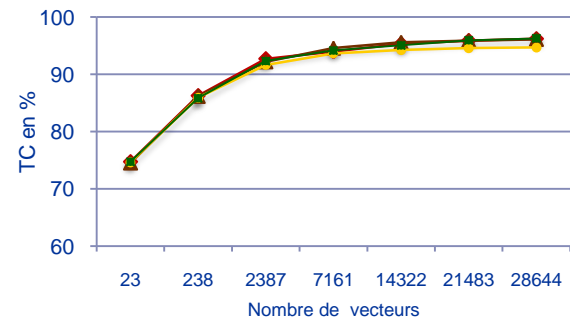
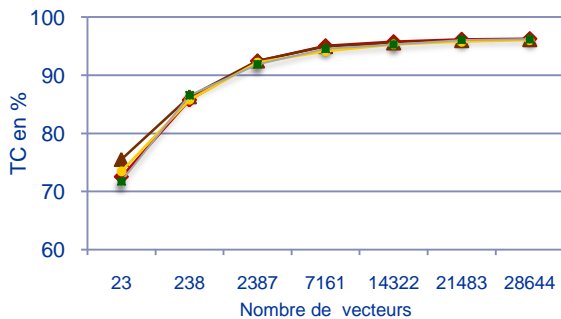
64 Scan chains



■ s13207



■ s38548

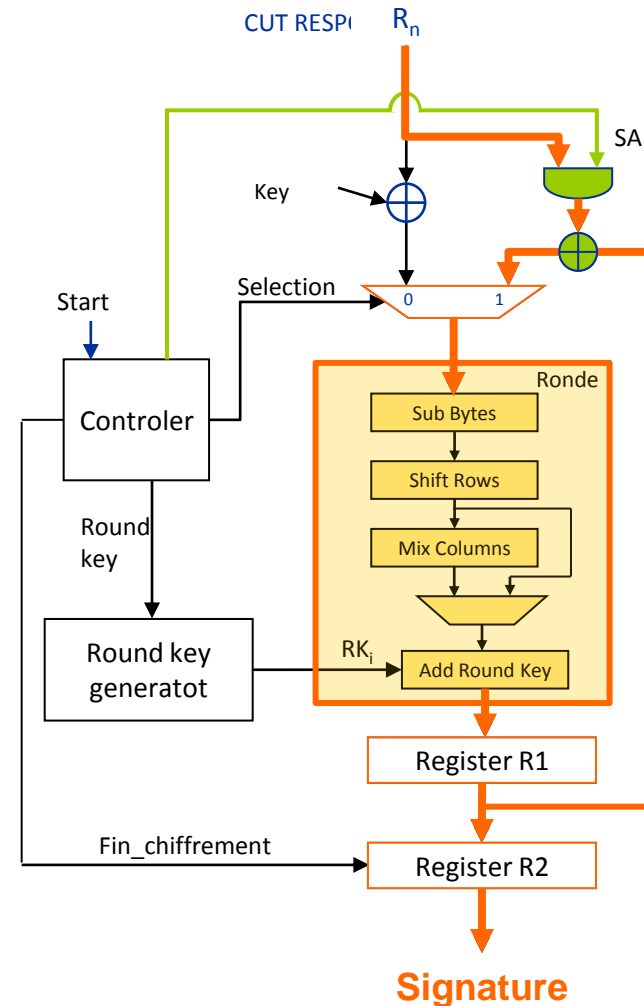


■ Response compaction mode :

- SA = Selection = 1

■ Functional mode

- SA=0



■ AES/DES

$$P(M_n) = \frac{1}{2^m} - \left(\frac{1}{2^m}\right)^n$$

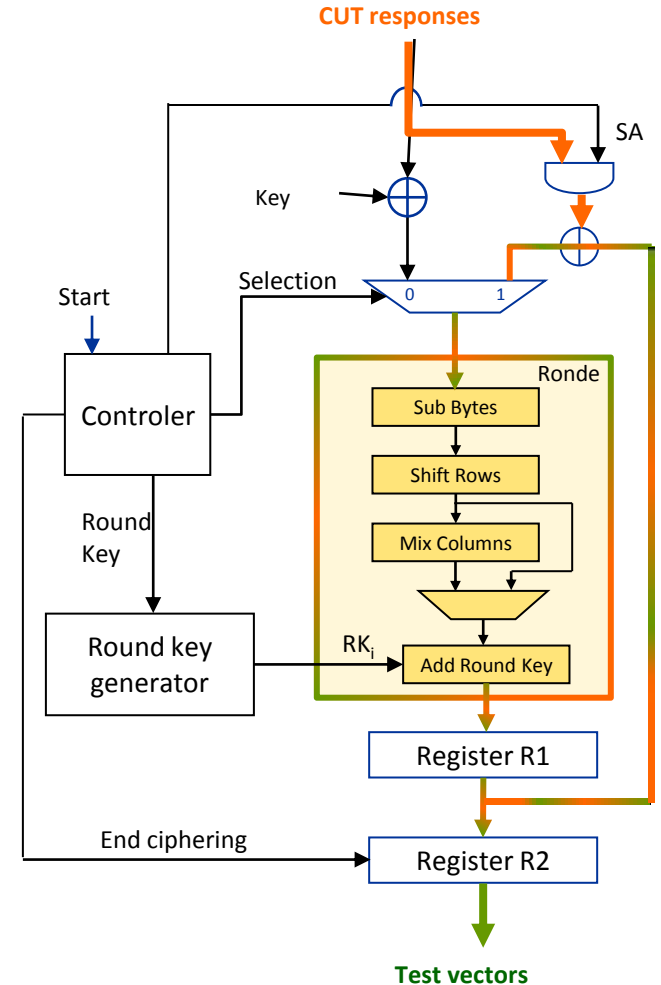
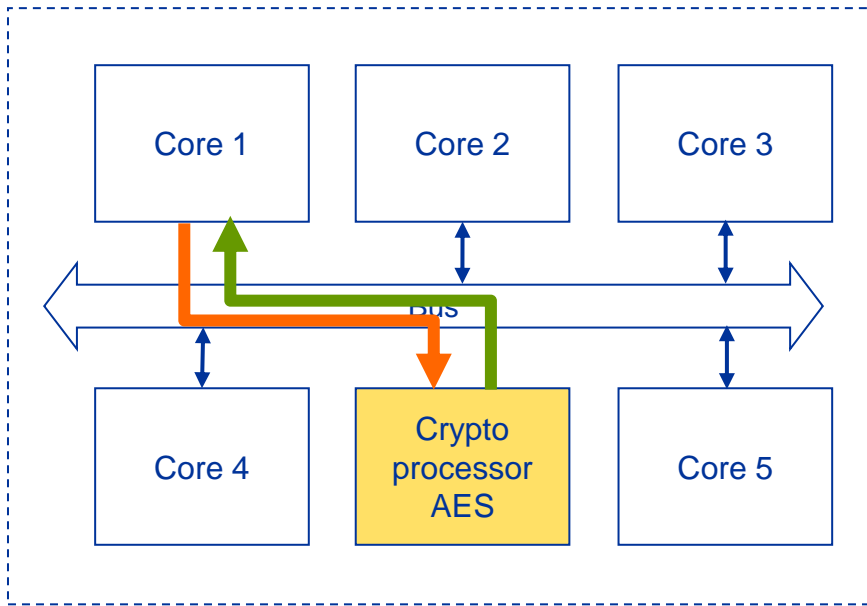
$$P(M_{128}) \xrightarrow{n \rightarrow \infty} \frac{1}{2^{128}} \approx 10^{-40}$$

■ MISR

$$P(M_n) = \frac{2^{n-1} - 1}{2^{m+n-1} - 1}$$

$$P(M_{128}) \xrightarrow{n \rightarrow \infty} \frac{1}{2^{128}} \approx 10^{-40}$$

Simultaneous TPG and Compaction



		AES	AES generator	AES compactor	AES Self-test	AES 4 modes
Round	-SubBytes	803 734
	-ShiftRows	0
	-MixColumns	59 847
	- AddRoundKey	49 945
Controler		6 345	+ 5.72%	+ 8.72%	+ 6.58%	+ 9.58%
Key generator		301 162	+ 0.015%	+ 0.015%	+ 0.015%	+ 0.015%
Glue logic		153 620	+ 0.04%	+ 17.95%	+ 0.04%	+ 18.36%
TOTAL		1 374 655	+0.03%	+2.05%	+0.04%	+2.10%

Overhead 2.1%

		AES	AES generator	AES compactor	AES Self-test	AES 4 modes
Round	-SubBytes	803 734
	-ShiftRows	0
	-MixColumns	520 217
Control						58%
Key generation						15%
Glue logic						36%
TOTAL		1 374 655	+0.03%	+2.05%	+0.04%	+2.10%

For comparison :

Implementing a LFSR \Rightarrow 3.67%

Implementing a BILBO \Rightarrow 7.64%

Overhead 2.1%

III Concurrent testing & fault attacks



■ Concurrent testing

Detect malfunctioning during execution

- permanent faults
- transient faults (SEU,)
- ECC techniques

■ DFA attacks

Inject transient faults during encryption (laser, power glitch, ...)

- process normal encryption
 - process faulty encryption
- (right place & right time)
-
- A diagram showing a right-facing curly bracket grouping the two bullet points above. To the right of the bracket is a plus sign in a circle (
- \oplus
-), followed by a grey arrow pointing to the right, and the word "Key".
- $\oplus \longrightarrow$ Key



DFA Example : attack on 1 bit

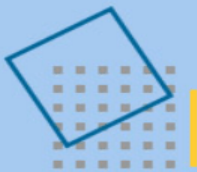
LIRMM

■ Fault/Error

- 1 bit-flip just before last SubByte
- Erroneous byte at output

■ Attack

- 2 AES executions, one regular, one faulty, with same (unknown) clear text
- xor of the 2 results, only a single **byte** differs
- The position of the byte gives the byte subject to the attack (InvShiftRows)
- Hypothesis on the faulty bit, compute the results
 - $C = SB(X) \oplus K_{10}$
 - $D = SB(X \oplus e) \oplus K_{10}$
 - $\Delta = SB(X) \oplus SB(X \oplus e)$
- Other runs (if necessary), to decide between candidates X
- Practically : less than 5 faults to retrieve the whole key

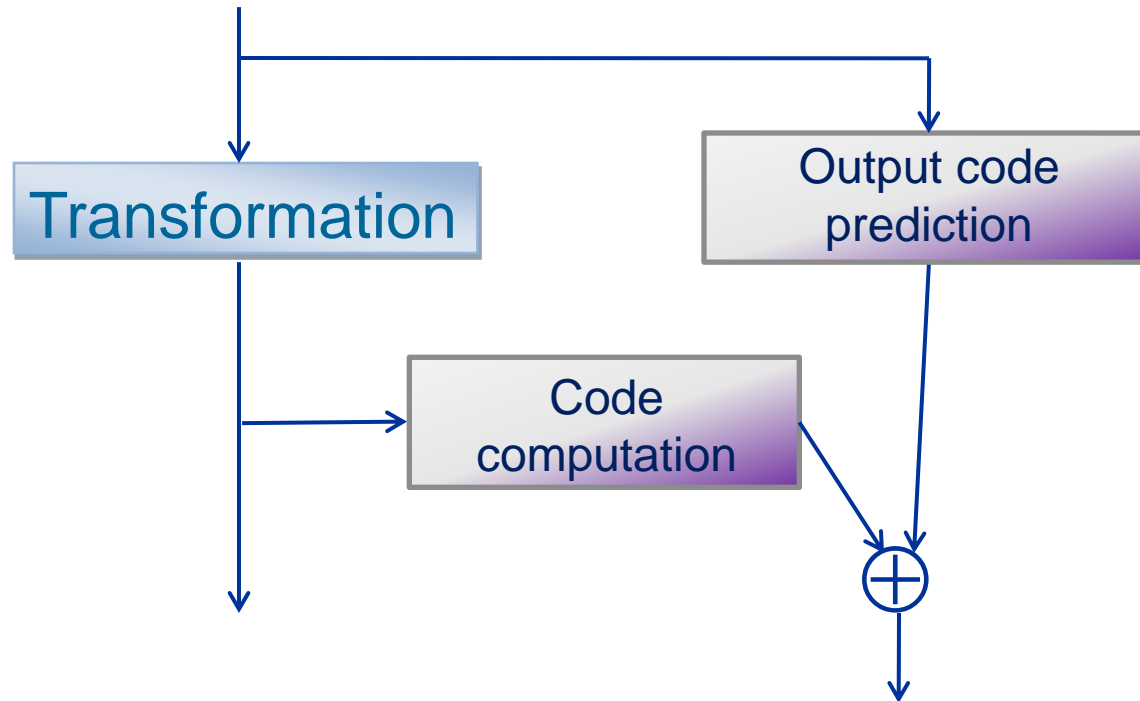


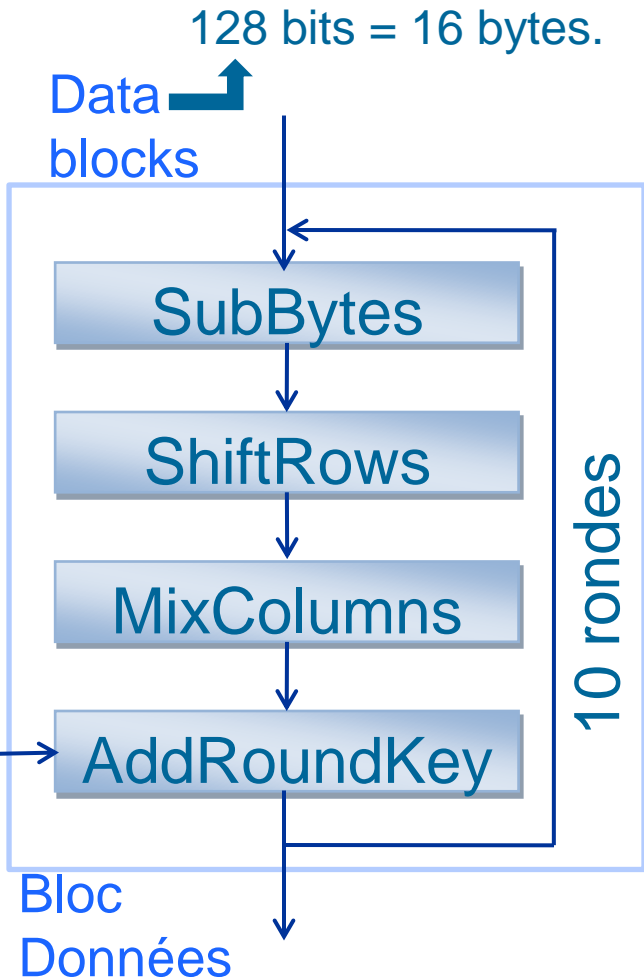
Transient faults detection

LIRMM

■ Redundancy

- Temporal / spatial redundancy
 - e.g. compare (decrypt(encrypt(M)) , M)
 - double computation
 - spatial
 - temporal (e.g. DDR)
- Information redundancy (spatial)
 - ECC techniques

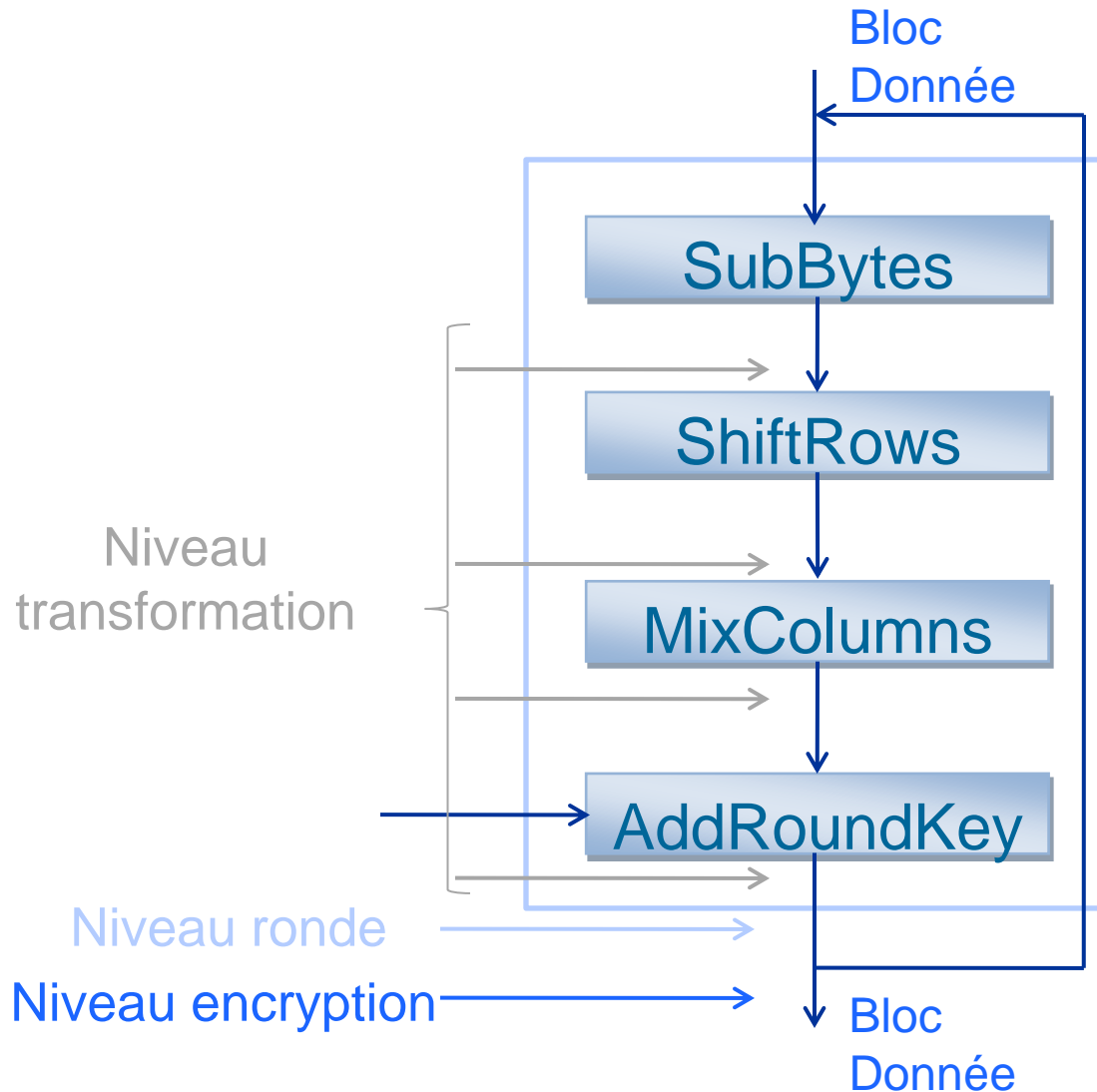


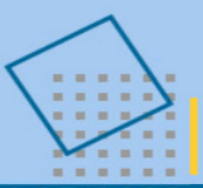


$$State = \begin{bmatrix} S_0 & S_4 & S_8 & S_{12} \\ S_1 & S_5 & S_9 & S_{13} \\ S_2 & S_6 & S_{10} & S_{14} \\ S_3 & S_7 & S_{11} & S_{15} \end{bmatrix}$$

MC: Opération *linéaire* sur chaque colonne; Multiplication des matrices: $M \cdot State$.

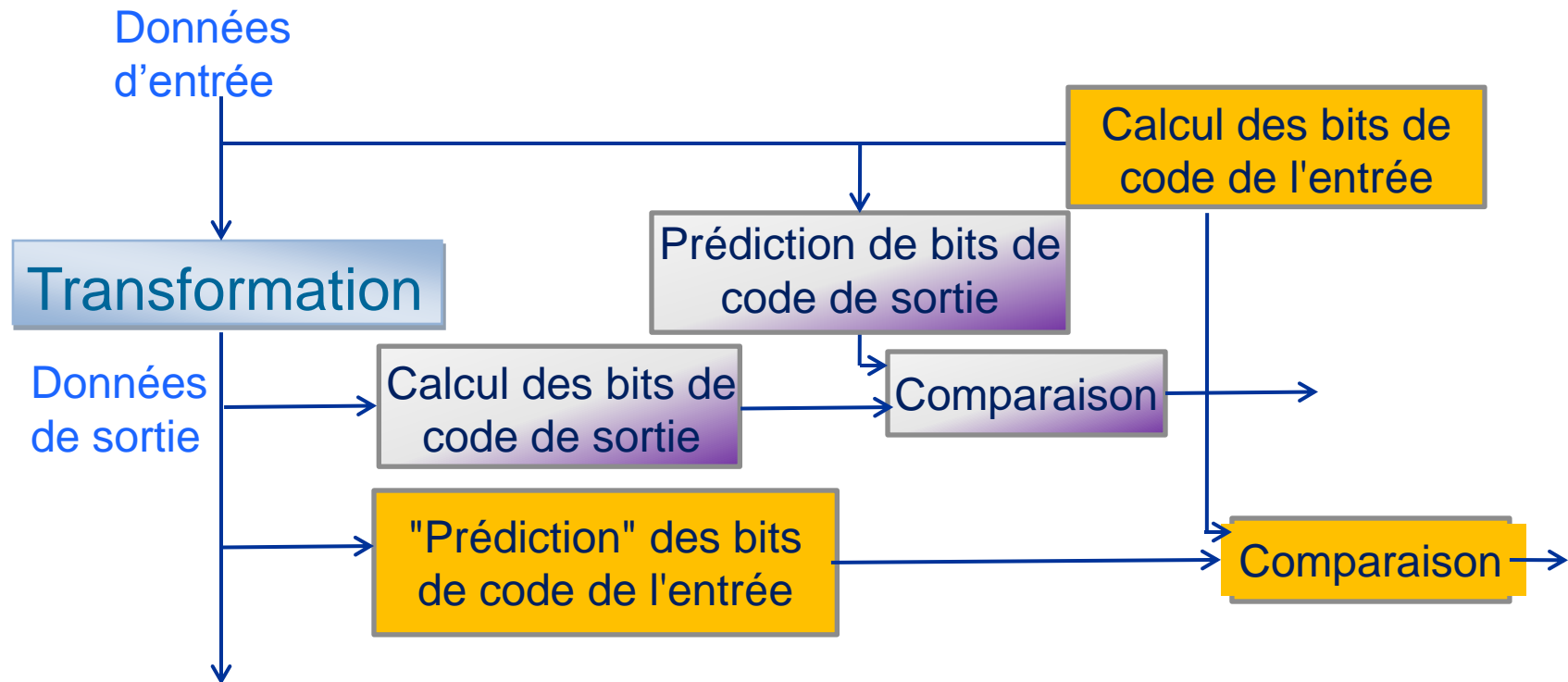
Ad: Opération *linéaire* sur octets; un OU exclusif entre les données et la clé: $State + K$.

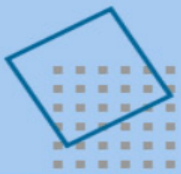




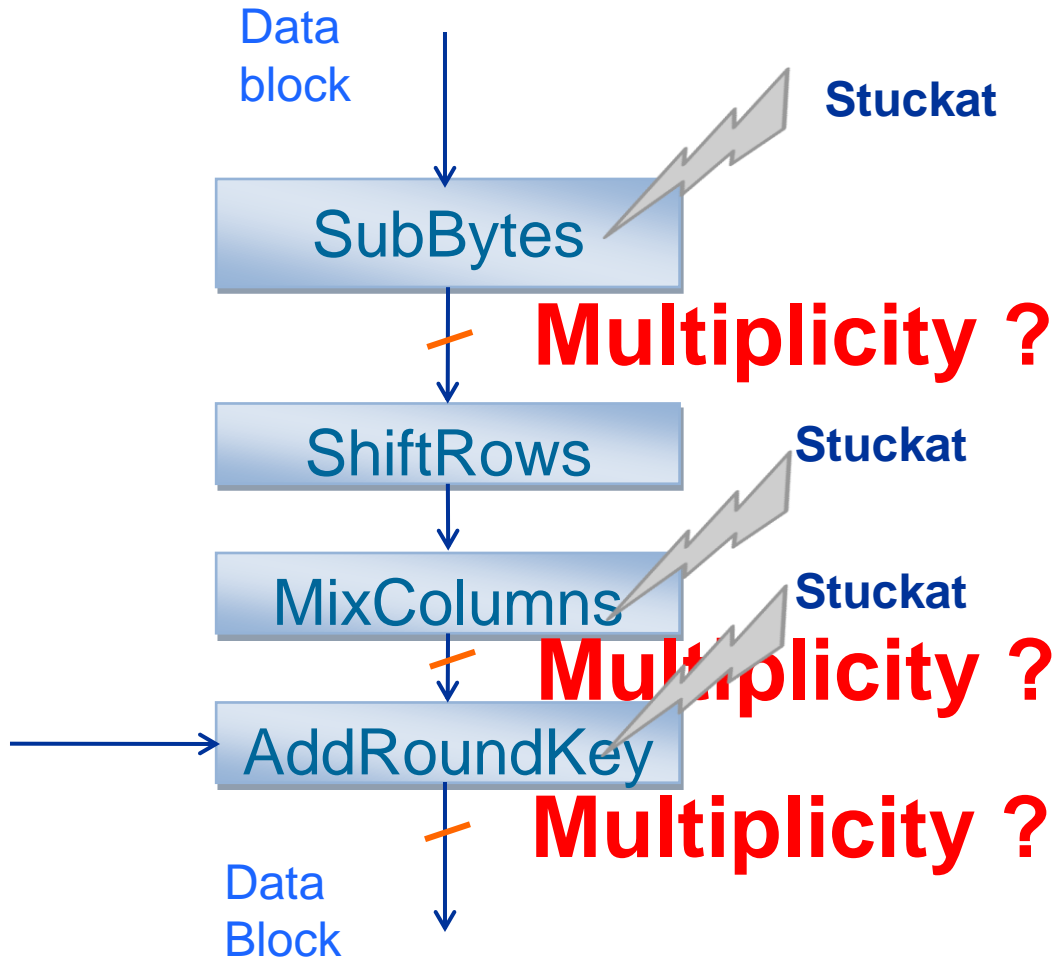
- Coding : parity bits
 - 1 parity bit for 128 data bits
 - 1 parity bit for each data byte
 - 1 parity bit for each state "column"

- Non linear codes (28 code bits for 128 data bits)
 - hyp : Evenly distributed errors





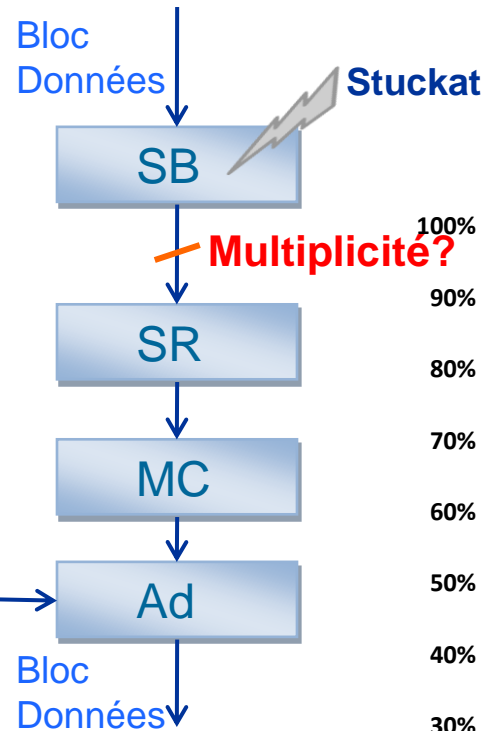
Faults & error multiplicity



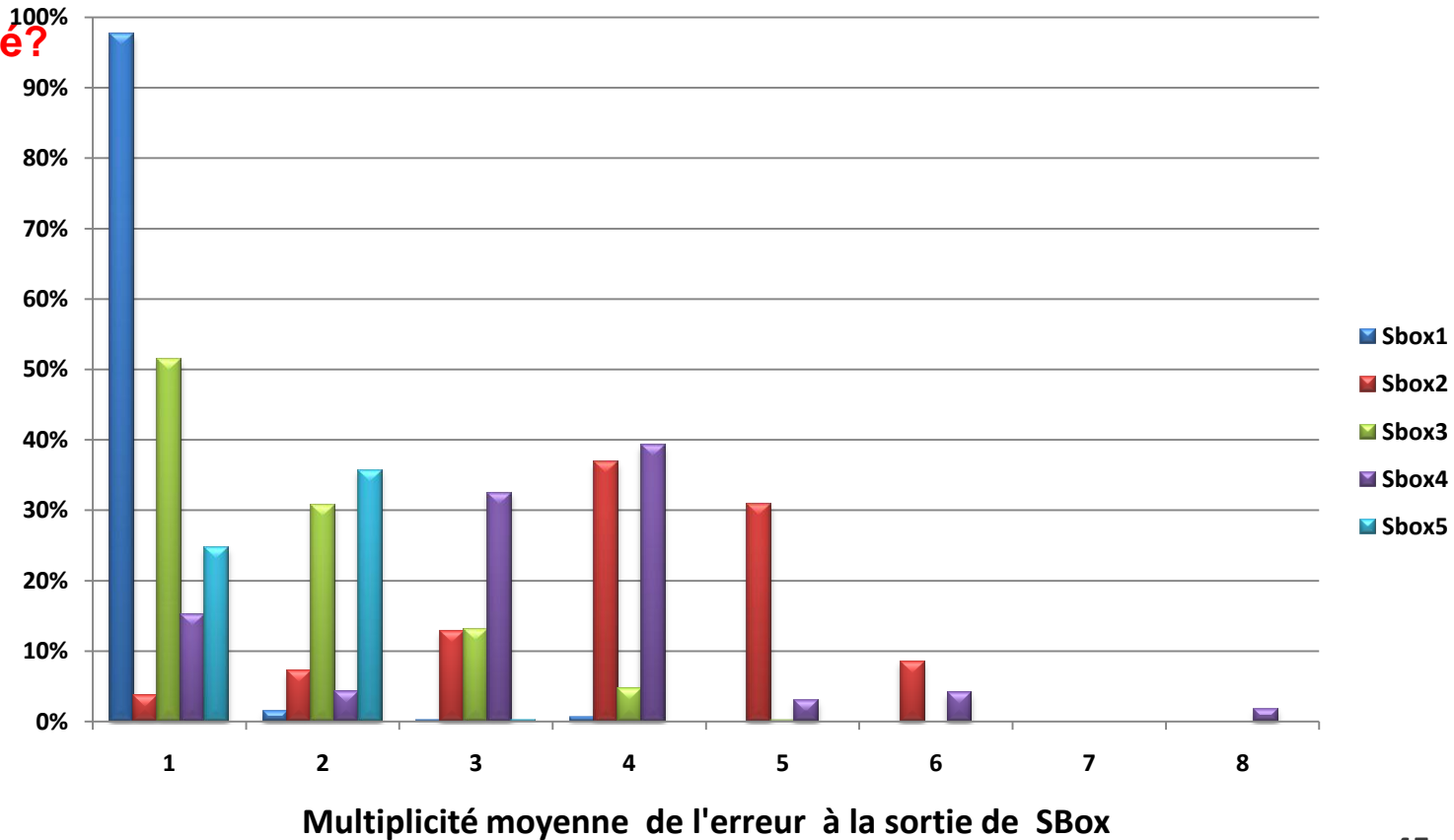
Pour **MC**: suite à sa structure (action sur colonnes), l'erreur peut se propager à 4 octets à la sortie.

Implémentation de l'opération **Ad**: 128 portes XOR.
Pour le modèle de faute sélectionné => multiplicité d'erreur d'ordre de 1.

Fautes et multiplicité d'erreur



Fautes simples



■ Corrélation

- Modèles de fautes / moyen d'attaque
- Fautes / implantation / erreurs
- DFA en fait DEA (error)
- Fautes / Erreurs "exploitables"

■ Fautes / Codes