Part III

Hardware-based attacks
Types of attacks, exploitation examples

Part II contents and objectives

- Types of attacks, exploitation examples.
- Examples of secure architectures and protections (counter-measures).
- Fault/error models.
- Impact on test techniques.

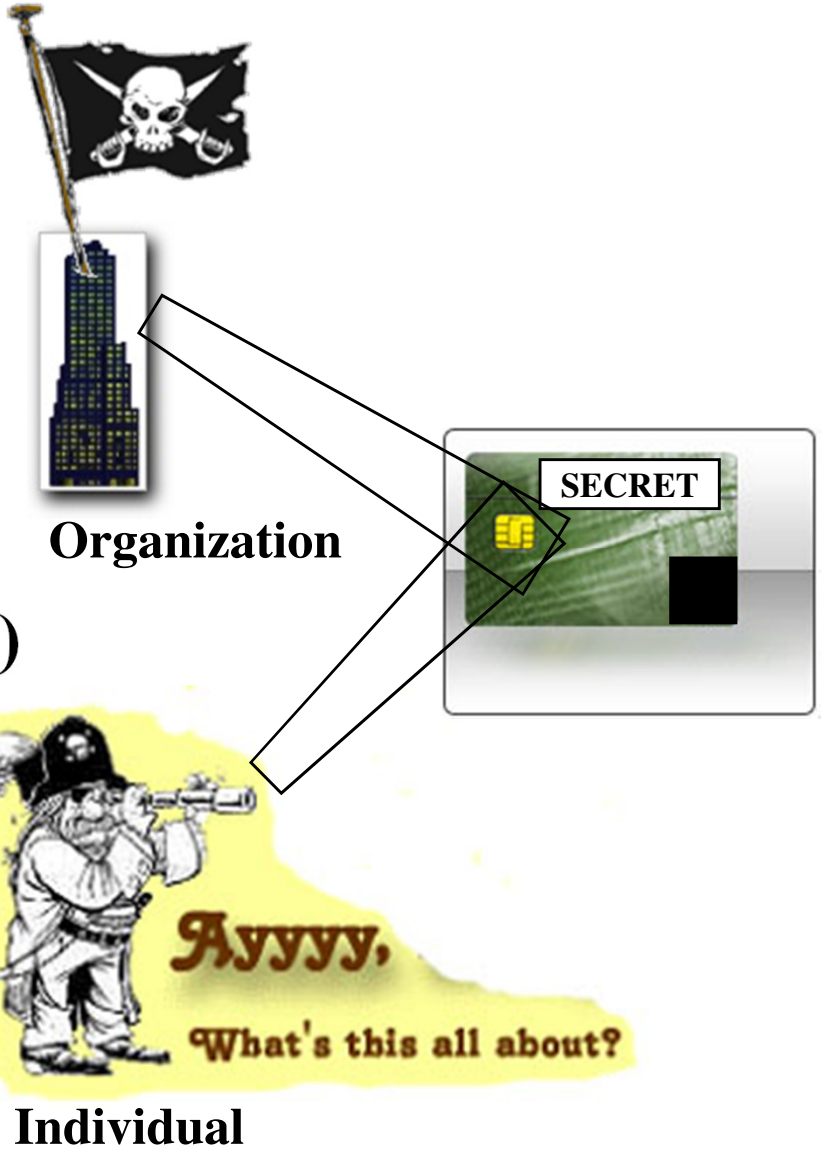
Why do you care about chip-level security ?

Outline adapted to practical work schedule …

- Understand the context and the security threats
- Understand the basics and limitations of protection techniques
- Prerequisites: basics in cryptography (M1 courses).

Types of attacks: global taxonomy

- Implementation based attacks
  - Invasive
    - Circuit modifications
    - Probing
  - Semi-invasive
  - Non-invasive
- Observation-based (passive, side channels)
- Perturbation-based (active, fault based)
- Note: we will not discuss here attacks based on software&networks threats (e.g. viruses, malicious applets)



Limitations of counter-measures

- Never 100% protection/security – just make the attacks harder
- Still an empirical selection process
  - No global optimization of overheads
  - A protection against a given attack can ease another one (e.g. error detecting codes w.r.t. power-based attacks)
- All types of possible attacks must be simultaneously addressed (the hacker will use the easiest way to success)

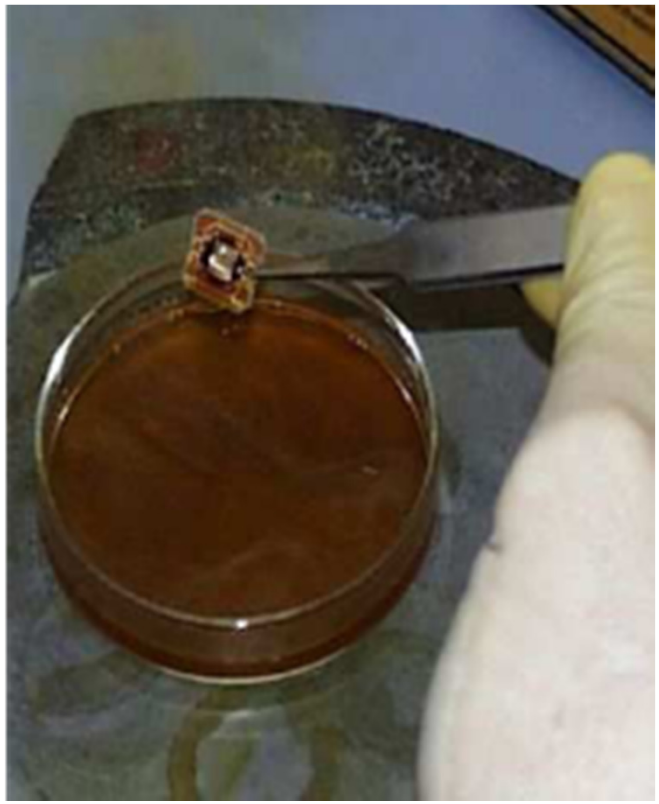
Implementation Attacks

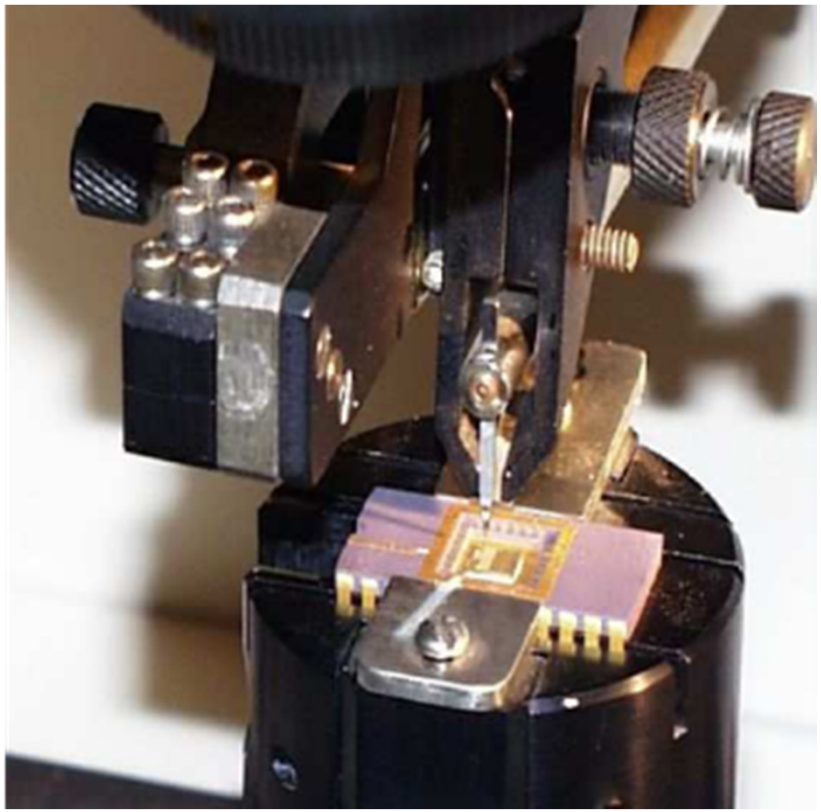
- **Invasive attacks on the circuit**

- Side Channel Analysis
  - Time Analysis
  - Power Analysis
  - EM analysis
  - …
- Test structures
  - Scan chains,
  - …
- Fault attacks
  - Voltage glitches,
  - Clock glitches,
  - Overvolting,
  - Downvolting,
  - Laser shots,
  - Harmonic EM,
  - Pulsed EM injection,
  - …

- **Which countermeasures?**

Depackaging / repackaging

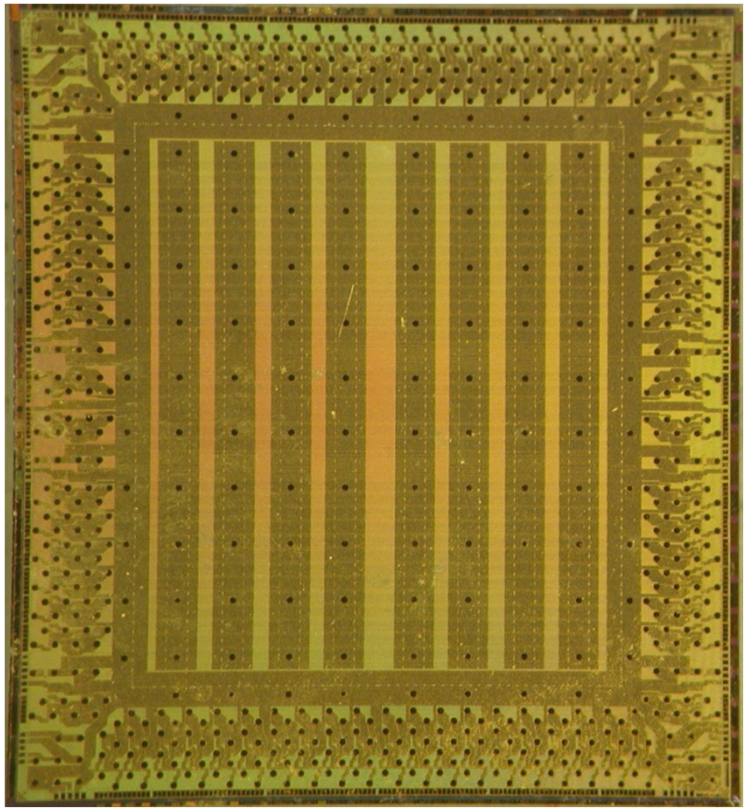




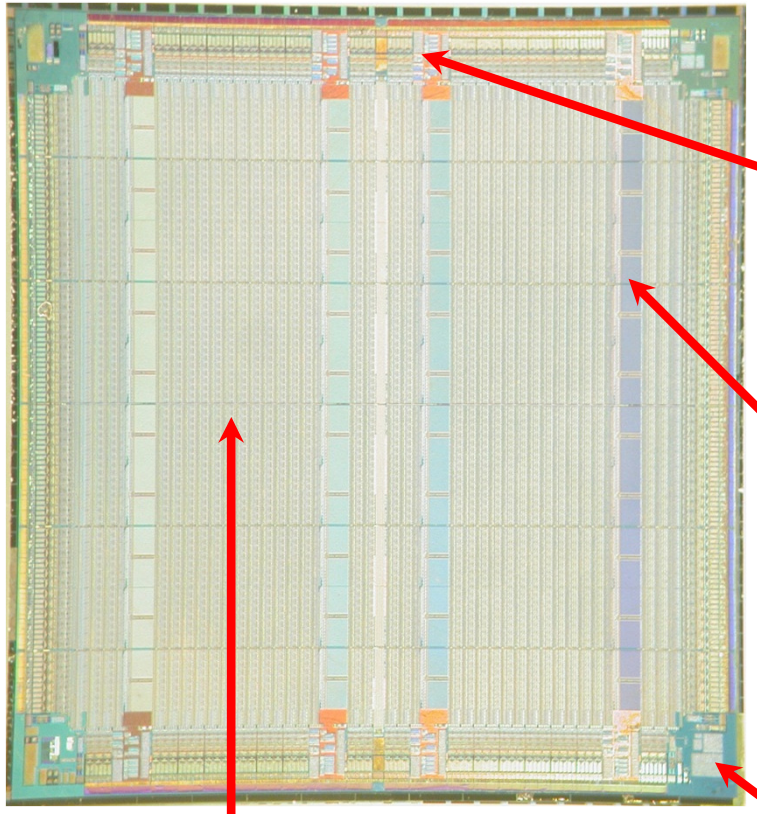
Sensors (light/UV) can be used to detect intrusion and stop functioning (or even destroy the critical contents)

Layer removal: example on a XC2V1000

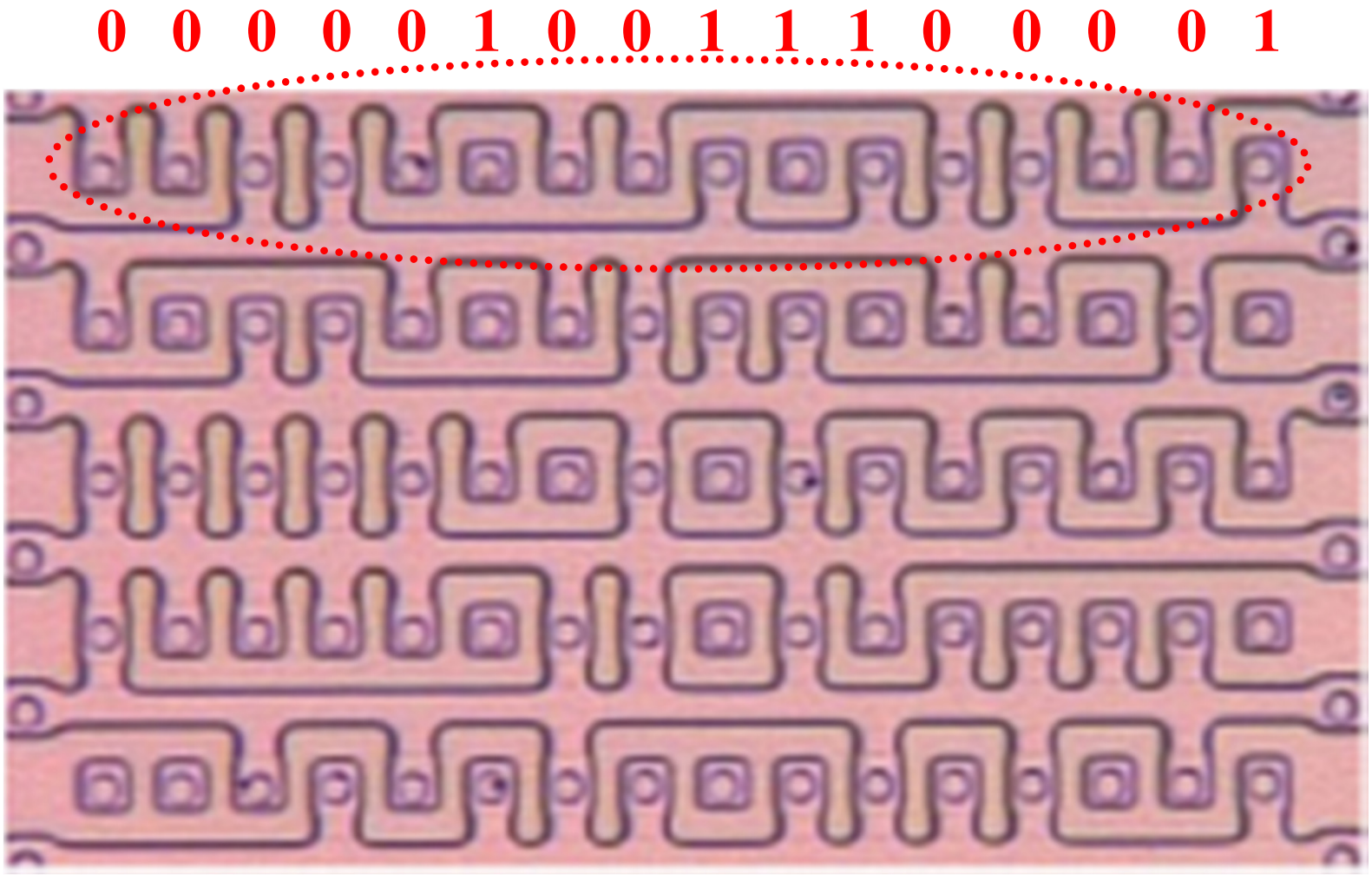
Before



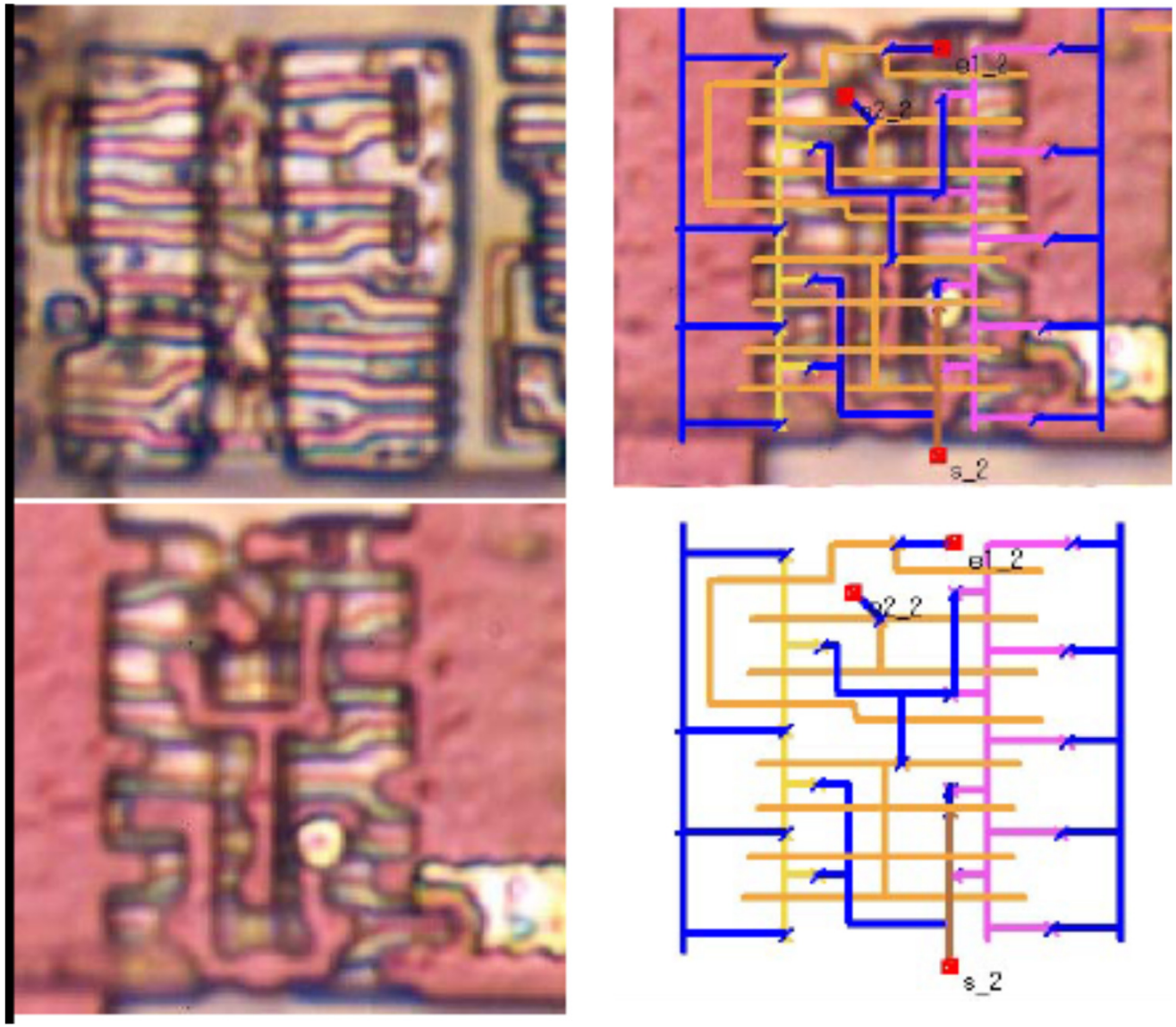
After - Poly-Silicon layer



Reading ROM memory contents …



Reverse engineering

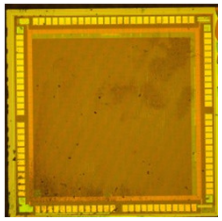


[Merle 2006]

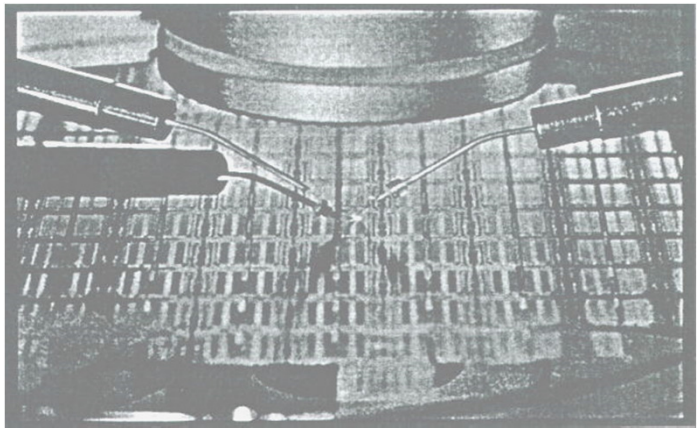
A. Merle, "Security testing for hardware product: the security evaluations practice", Minatec CrossRoad, May 2006

Layout observation and micro-probing

- Successive layer removal (chemical) => destructive !

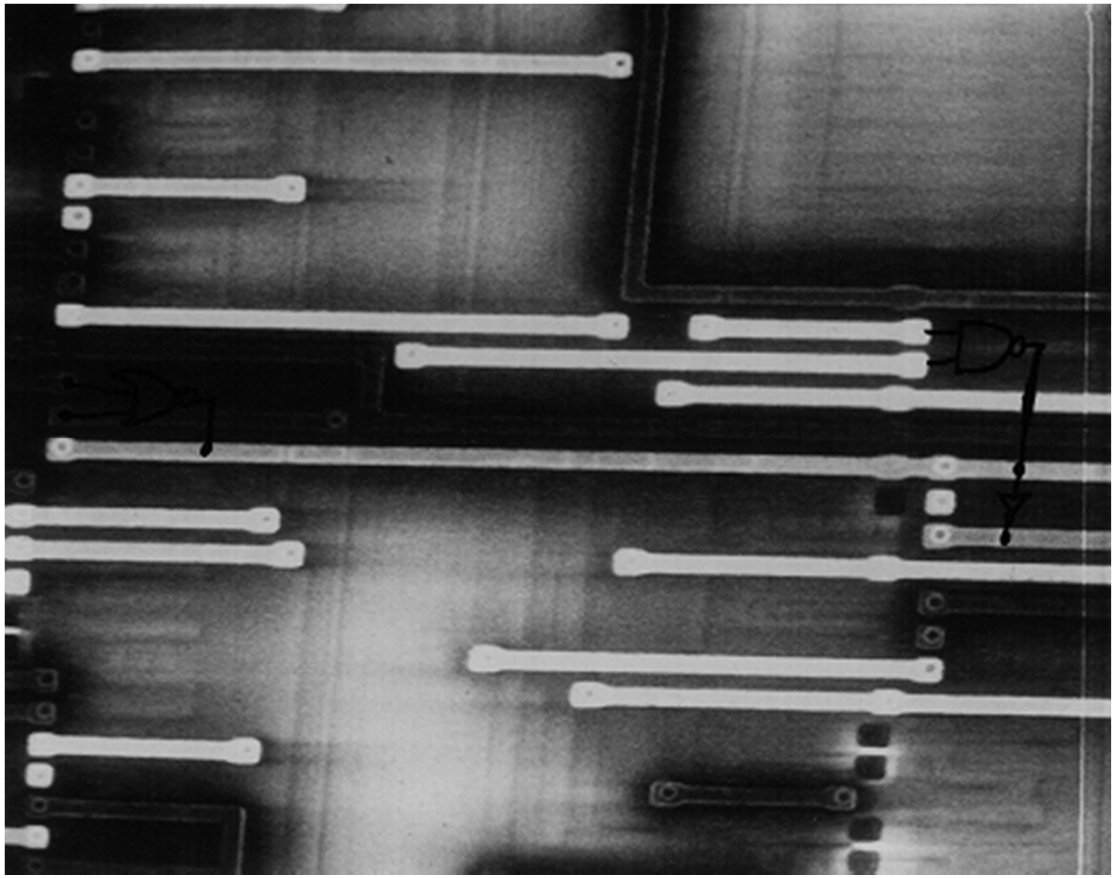


- Optical observation - more difficult in recent technologies (inherent shielding: dummies as covering layer …)



- Bus microprobing
  - Bus must be localized …
  - Number of probes …
  - Physical access required (partial layer removal)
  - Many data can be read on a single probing point … but difficulty to identify what those data are (functional meaning)

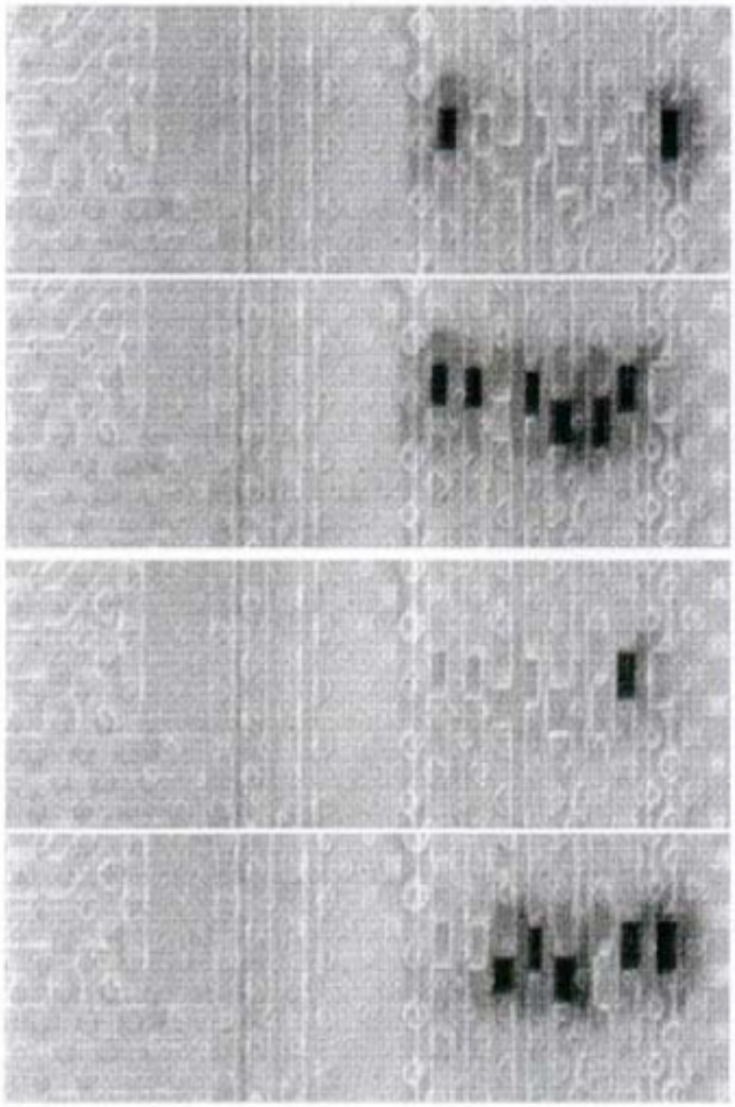
Voltage contrast microscopy (SEM)



Contactless, damage free probing of DC voltage within the IC

Usage example: could extract information from a Flash ROM storage cell

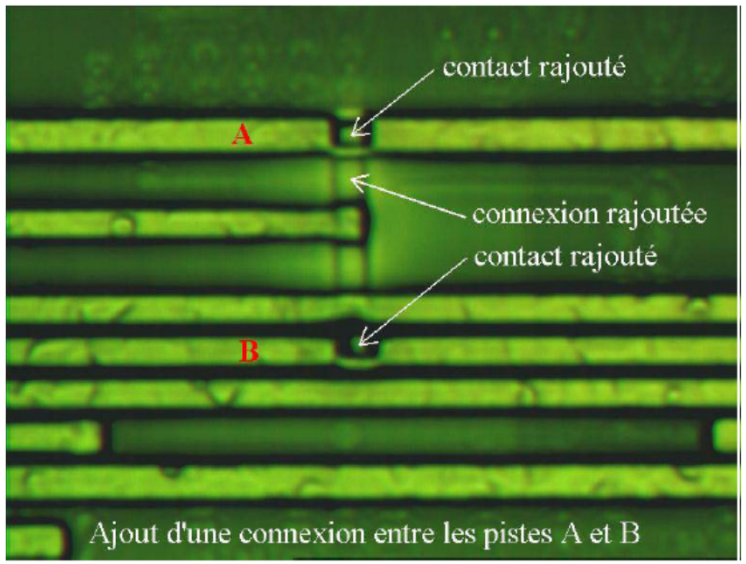
Dynamic analysis of data sent on a bus



[Merle 2006]

Other physical interventions

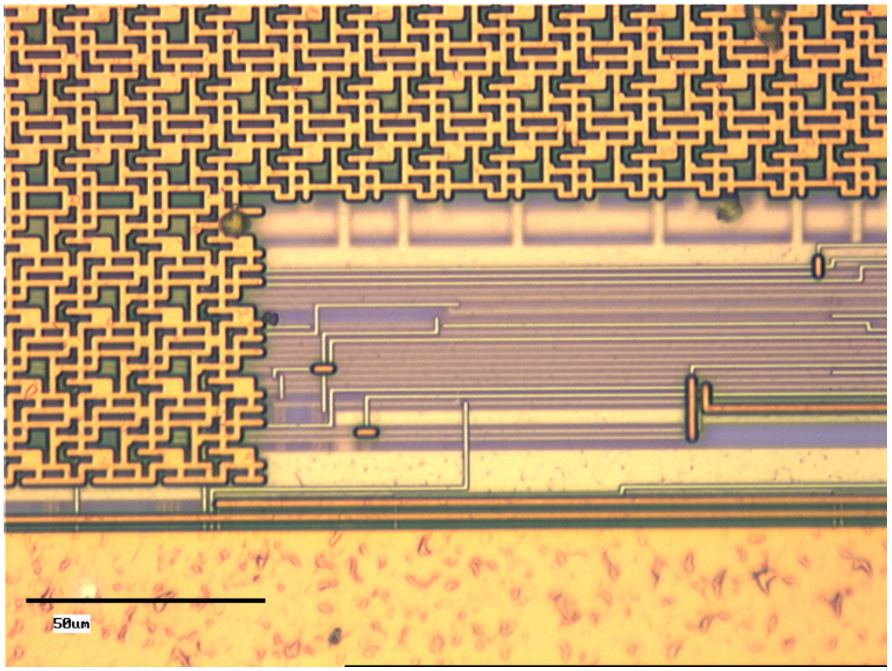
- Cutting internal connections
  - Protection disconnection
- Adding/repairing internal connections
  - Reconnecting a security fuse that prevents reading of the device
  - Reconnecting a cut scan access by boundary scan



Counter-measures against invasive attacks

- Avoid easy layout reading/exploitation
  - For optical observation after layer removal => e.g. memory scrambling
  - For probing => e.g. bus scrambling (or encryption)

- Protect against local probing
  - Protection grid (active) for passive probing
  - Detection signals on extra wires for active probing



Implementation Attacks

- Invasive attacks on the circuit

- **Side Channel Analysis**
  - **Time Analysis**
  - **Power Analysis**
  - **EM analysis**
  - **…**

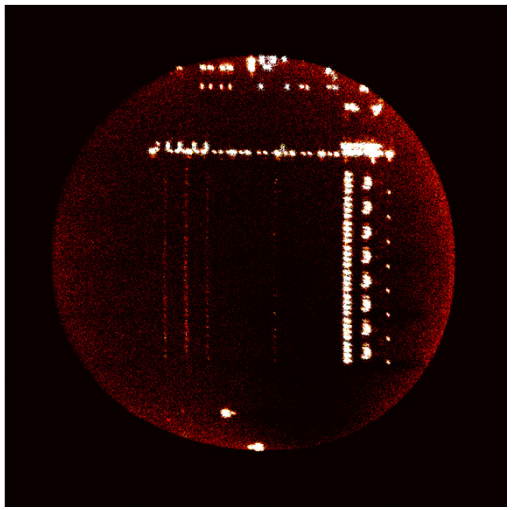
- Test structures
  - Scan chains,
  - …

- Fault attacks
  - Voltage glitches,
  - Clock glitches,
  - Overvolting,
  - Downvolting,
  - Laser shots,
  - Harmonic EM,
  - Pulsed EM injection,
  - …

- **Which countermeasures?**

Side channels

- **Execution time => Timing Attack (TA)**
- **Power consumption => Power Attacks (SPA / DPA / CPA)**
- **Electromagnetic emissions => Electromagnetic Attack (EMA)**
- **Sound, heat, photons …**
- **And many more ?**



Conditions of side channel exploitation

- Relationship between processed (secret) data and a given physical quantity that can be measured
  - It must exist …
  - … and a (good) model of the relationship must be available
- Known executed algorithm, so that predictions can be made
  - e.g. AES, RSA …
  - Security must no more be based on secret algorithms …
- Global secret used in independent pieces, so that exhaustive search can be done on separate parts
  - Replaces the complexity of a brute force attack by a much less easy search repeated on the several pieces of the secret

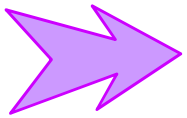
Side channel analysis

- **Simple Side Channel Analysis**
  - Makes use of characteristics that are directly visible in one measurement trace.
  - The secret key needs to have some simple, exploitable relationship with the operations that are visible in the measurement trace.
- **Differential Side Channel Analysis**
  - Looks for side channel differences that are not directly visible in one measurement trace => statistical methods
  - Targets one specific intermediate result (typically a selection function, i.e., an intermediate result at the beginning or end of the cryptographic algorithm) that shows up in a specific part of the measurement traces.
  - The result of the selection function depends on the known input/output data and a small number of hypotheses on the secret (key) value.
  - The outcome of the selection function leads to a partitioning of the overall measurement data for each hypothesis used.
  - For the correct key hypothesis, different statistical properties of the two partitioning sets are expected at that points in time which depend on the result of the selection function.

Timing attacks - principles

- Software level: e.g. If Then Else => execution time may be directly related to the tested condition (e.g. secret key bit !!)
- Hardware level: possible similar effect of condition tests in FSMs or data-related computation time in operators
- Security requires balanced execution paths and avoiding tests on critical (secret) values
  - Example:

```
If (x=1)
  then a:=a+b;
```



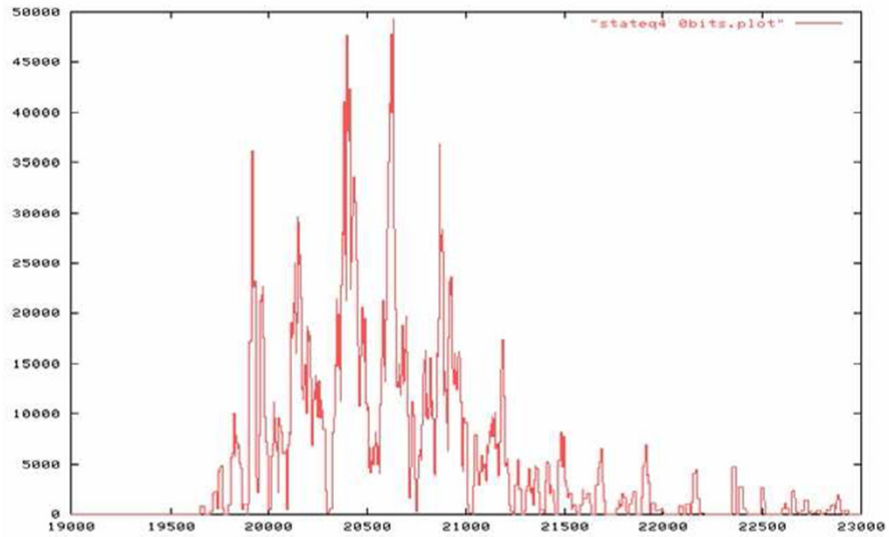
```
t[0]:=a;
t[1]:=a+b;
a:=t[x];
```

A practical example

- Timing attack on a PIN code
- Basic algorithm: successive comparison of the digits
- The more time before PIN rejection, the more digits are right …
- Successive exhaustive trials on each digit: for each one, the longest rejection time corresponds to the right value (since the next one has also been checked)
- PIN trials limitation necessary (+ better algorithm) !!!

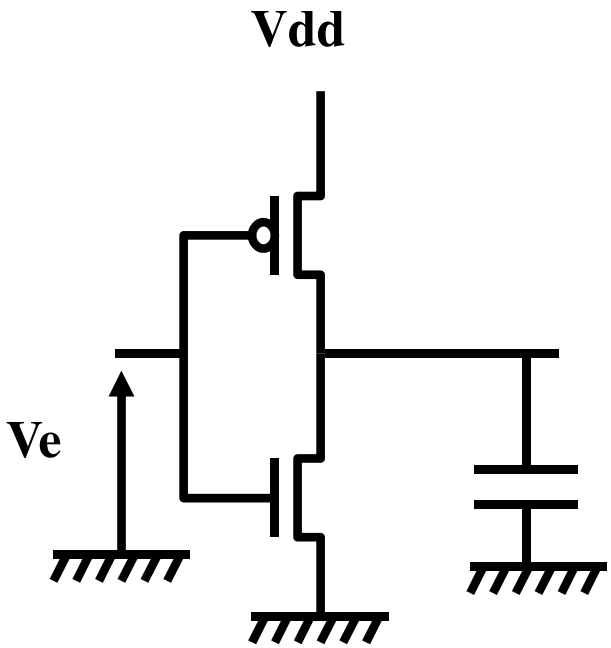
Architectural features and attacks

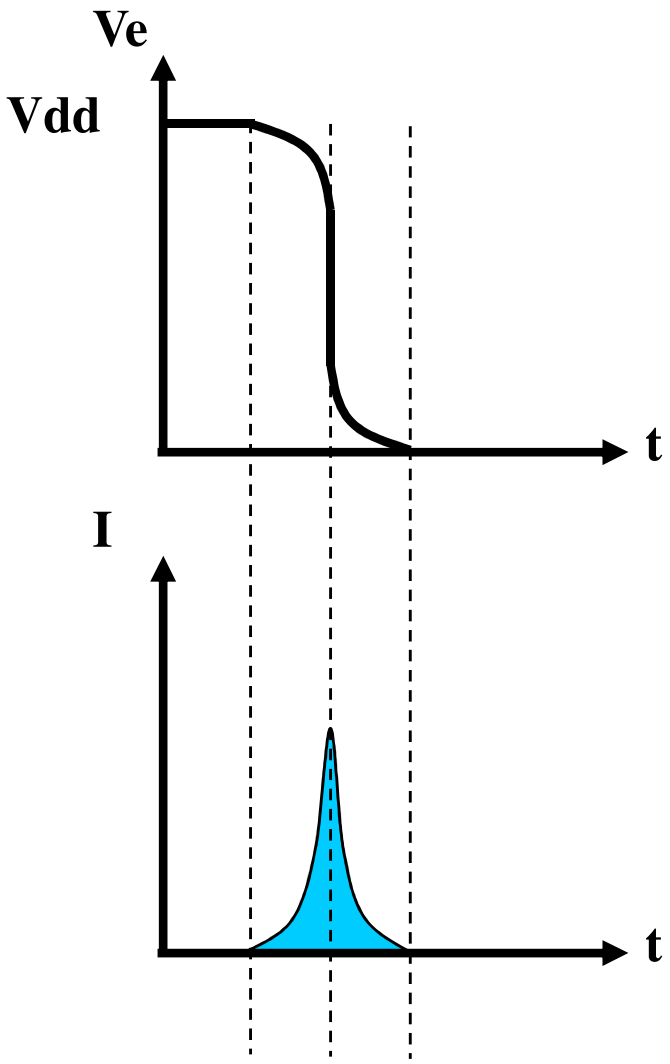
- Performance optimizations in processors can help timing attacks
  e.g. effect of HW optimizations in Pentium 4, for constant-time assembly-level implementation:



- Branch predictions (already executed branch has not the same execution time as a new branch)
- Cache memories (access to cache has not the same execution time as access to central memory)
- Performance counters provide accurate picture

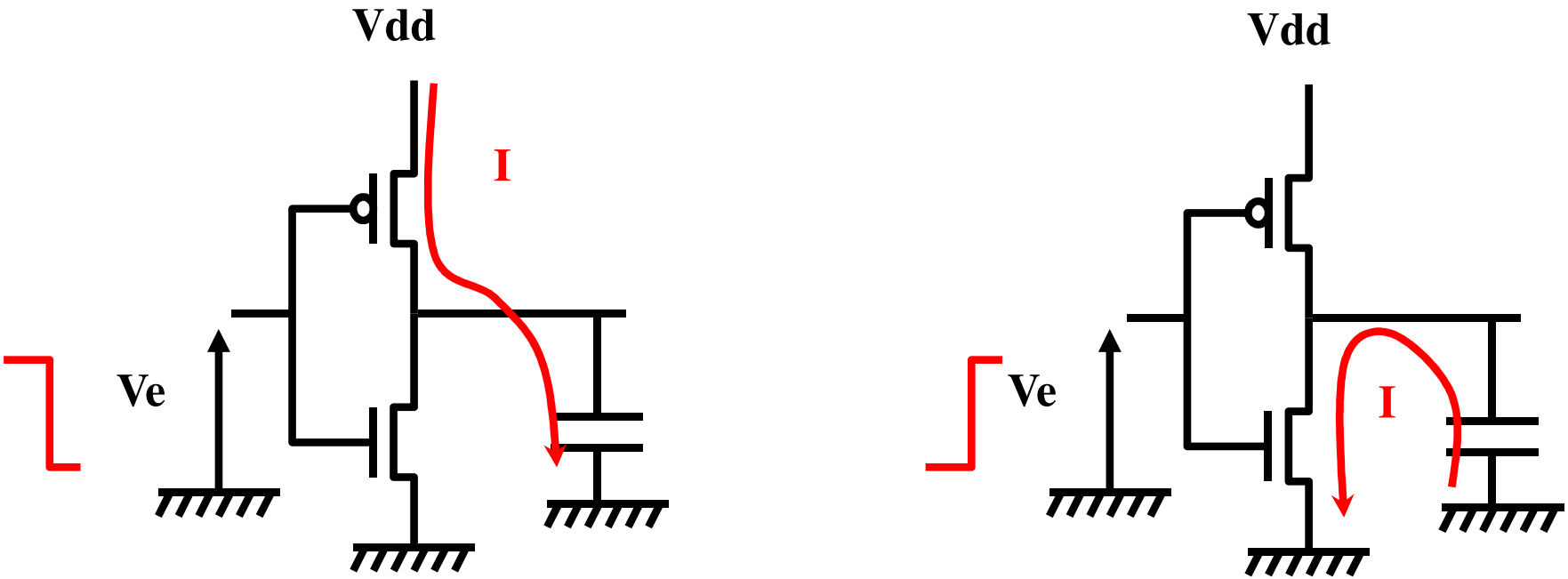
Power consumption in CMOS - Basics





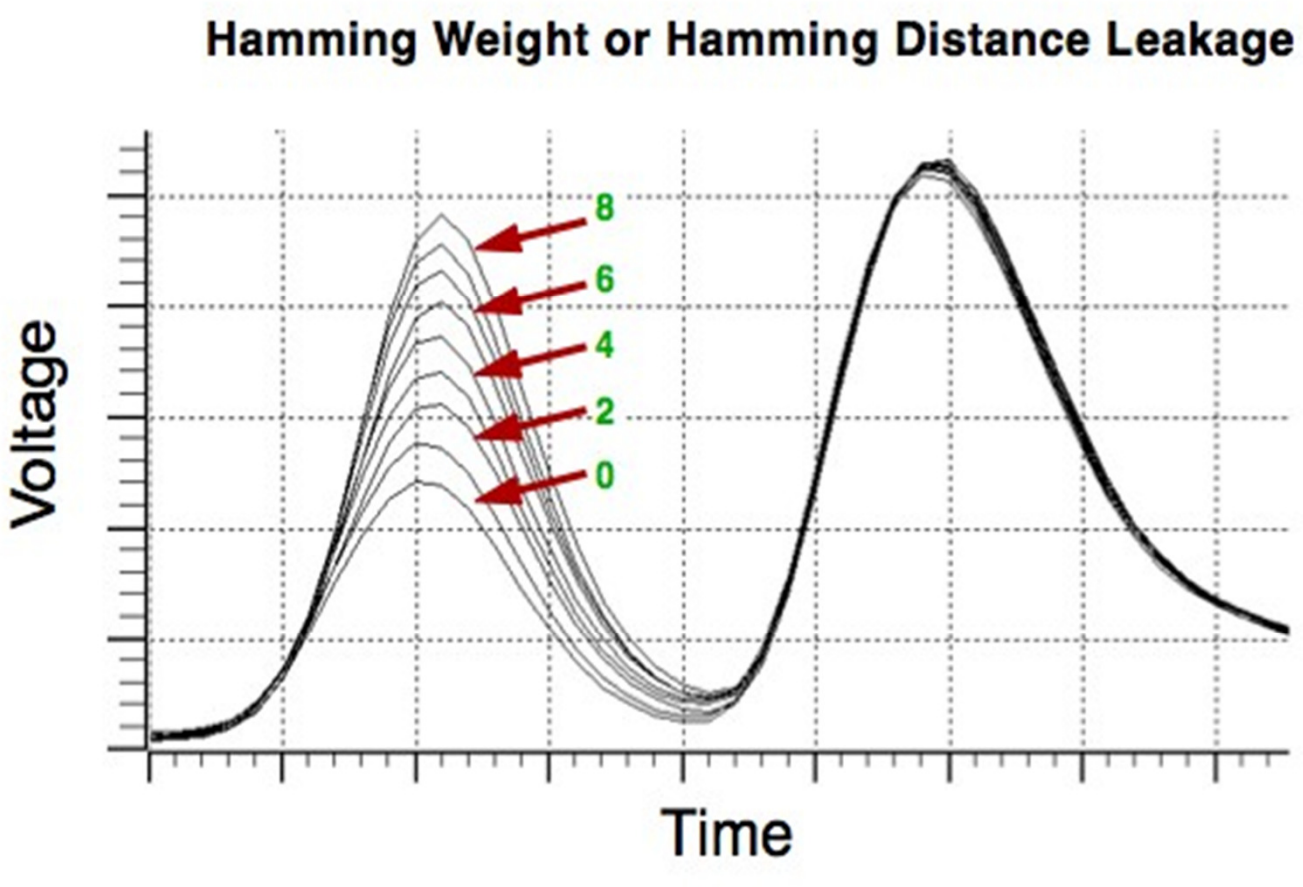
- Three components:
  - Static consumption
  - Dynamic consumption (logic commutations)
    - Short circuit current
    - Load charge/discharge

Power consumption in CMOS - Dissymmetry

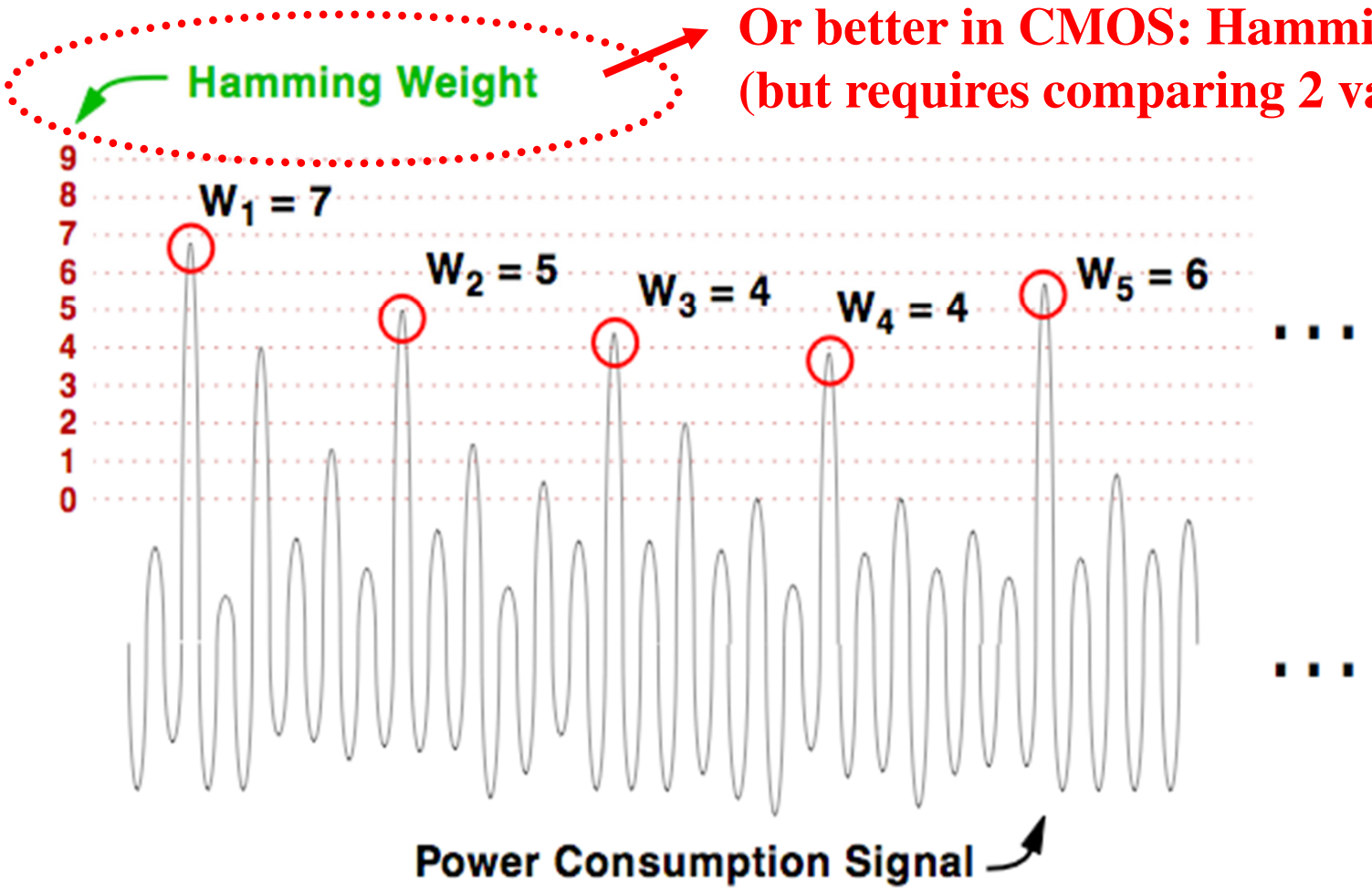


- Opposite logic commutation => different current signature
- Not the same current flow on Vdd and Vss
- Not exactly the same shape (value in time) due to either NMOS or PMOS conduction

Information leakage: power vs. transitions



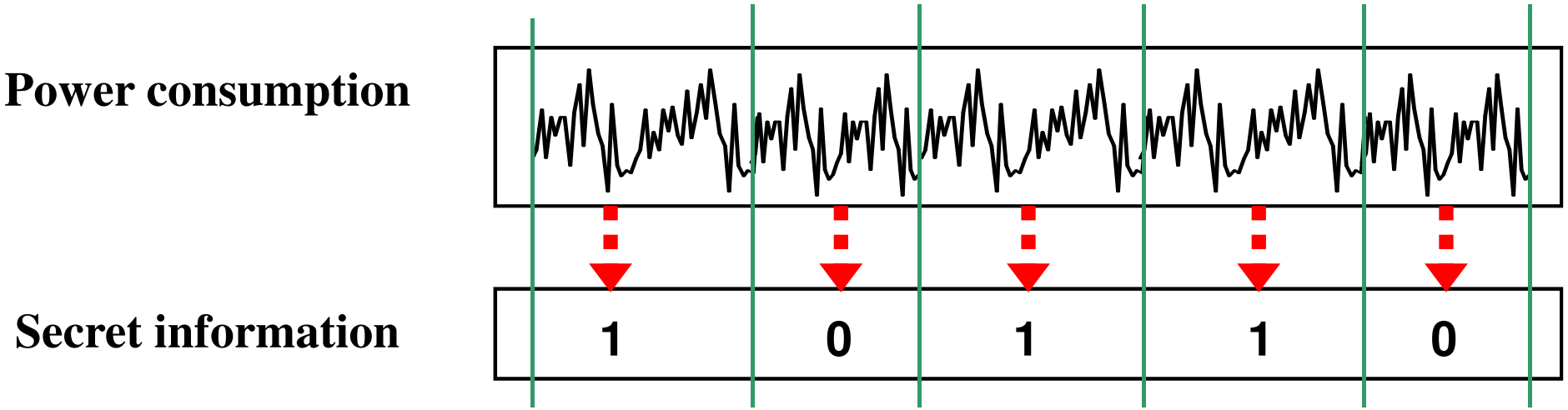
Reading Hamming weight from SPA trace



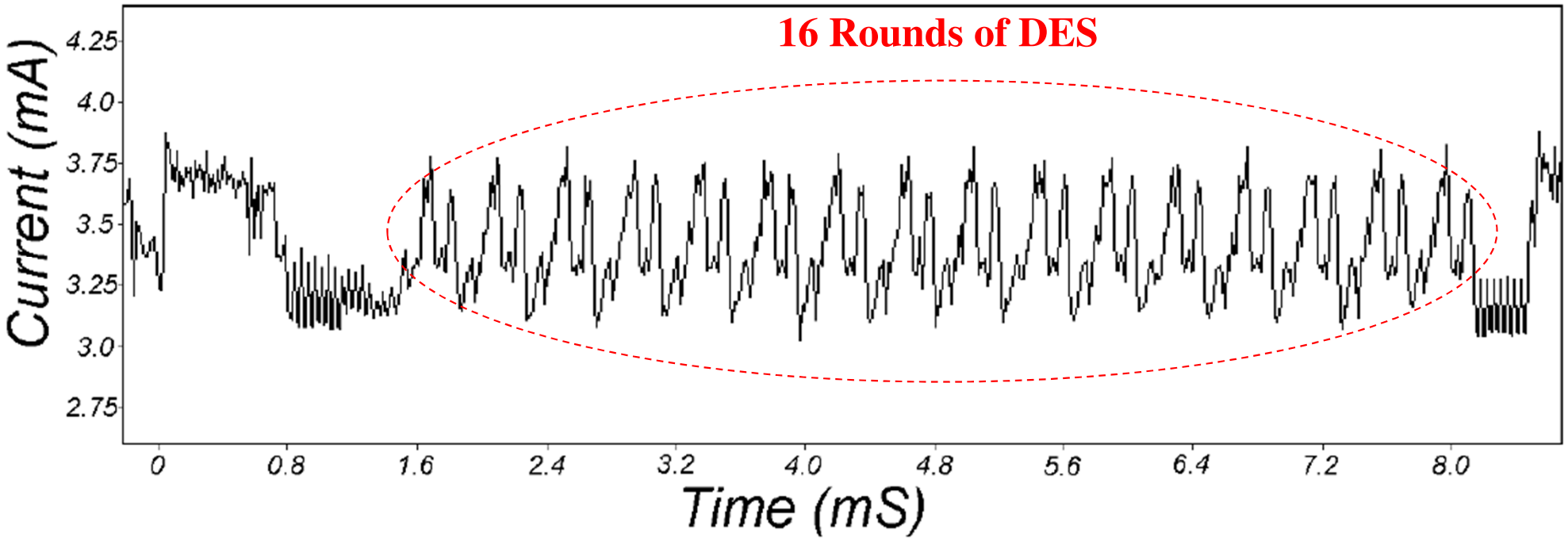
Requires to know at what time significant data must be observed …

SPA (Simple Power Analysis)

- Monitor the power consumption trace => signatures
- Find translation into
  - Executed instructions (software)
  - Manipulated data



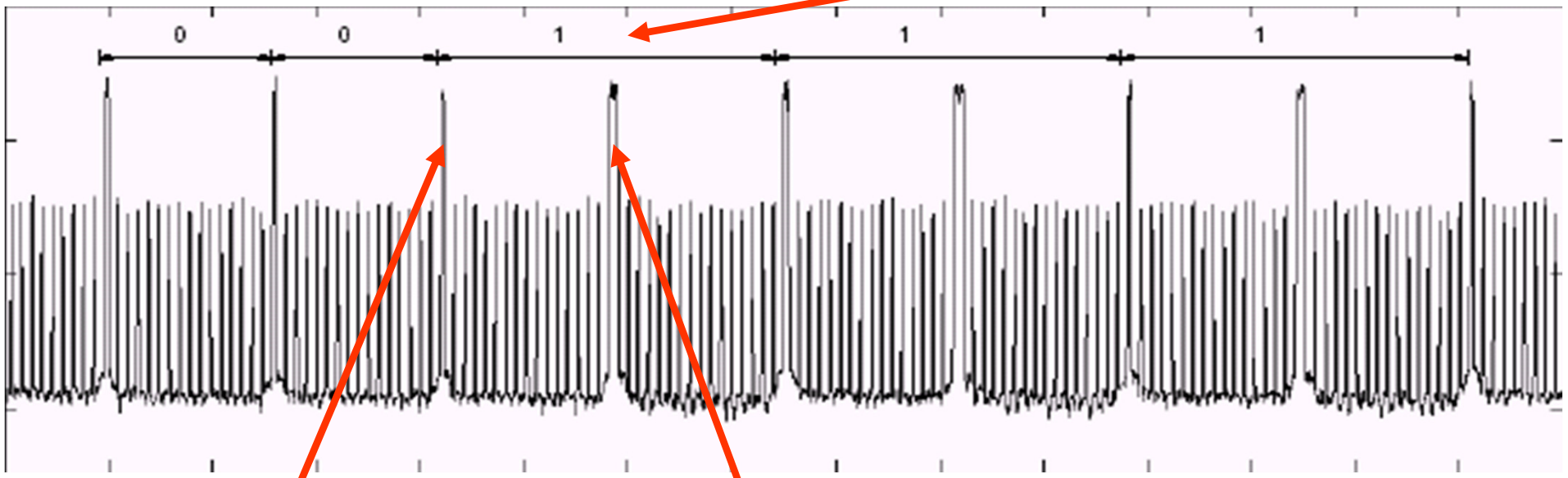
Example of SPA - DES



- May also be used to
  - Synchronize another attack (e.g. laser shot)
  - Measure the duration of some internal (intermediate) computations (TA)

Example of SPA - RSA

- RSA computation: supply current trace



Example of SPA - ECC

General scalar multiplication algorithm

Algorithm 1. Double-and-add algorithm

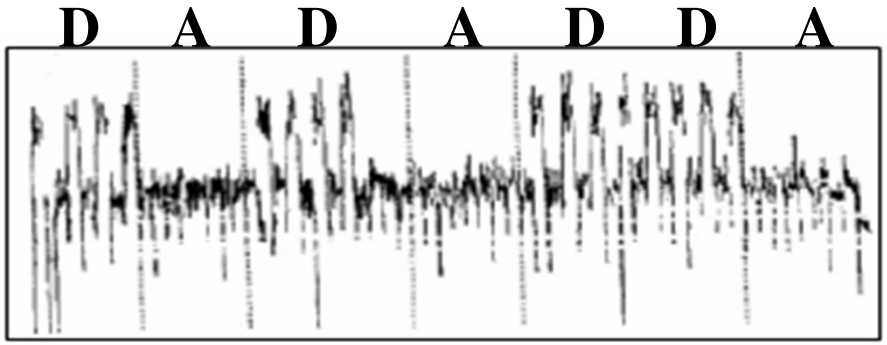
Input : A point P, and $d = (d_{n-1}d_{n-2}...d_1d_0)_2$, $d_{n-1} = 1$
Output : dP

1. $S = P$
2. For $i = n-2$ downto 0
   - 2.1 $S = 2S$
   - 2.2 If $d_i = 1$, $S = S + P$
3. Return(S)

d: secret exponent

Point Doubling (D):
Execution at each round

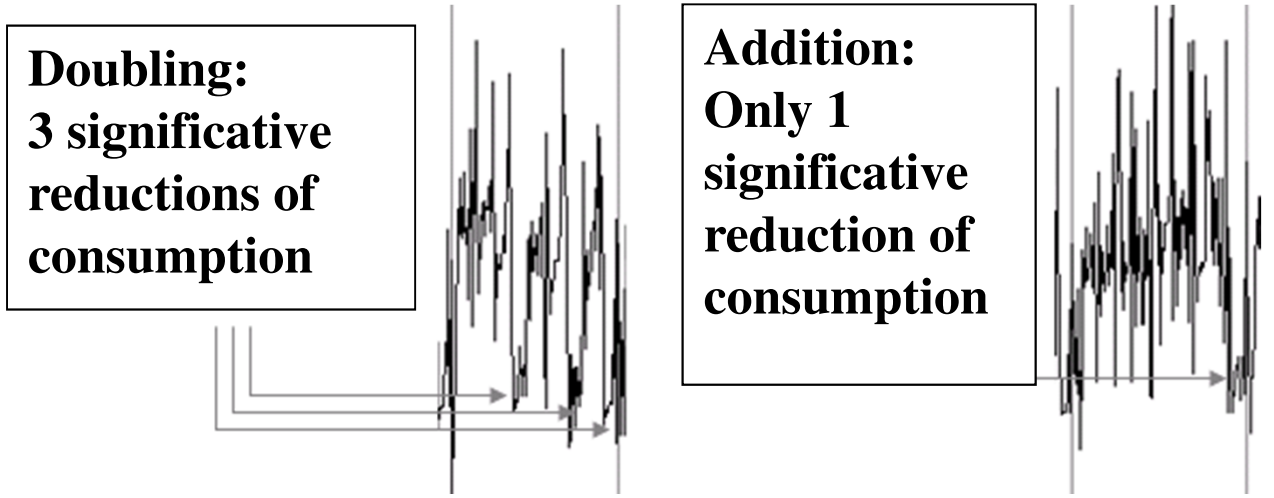
Point Addition (A):
Execution when bit value is "1"



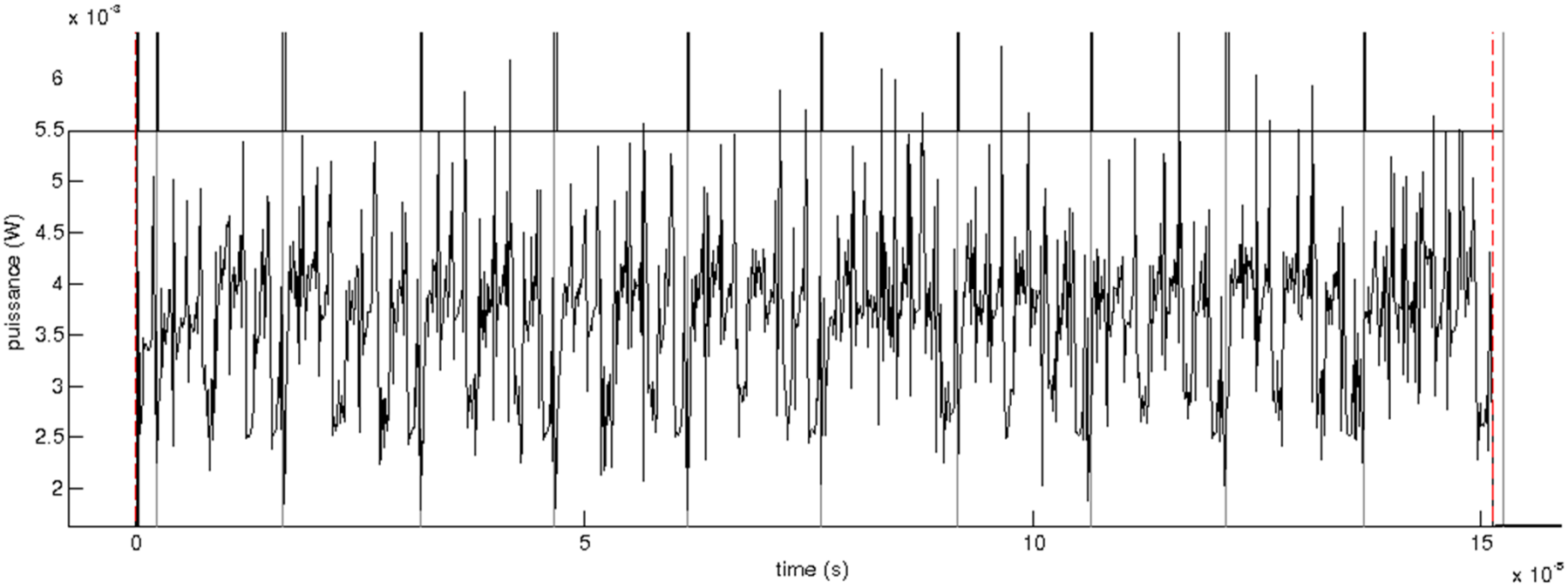
d = 1101

Simple computation

- Addition / doubling
- Curve with 270 points (private key between 1 and 269)
- Characterization of power consumption



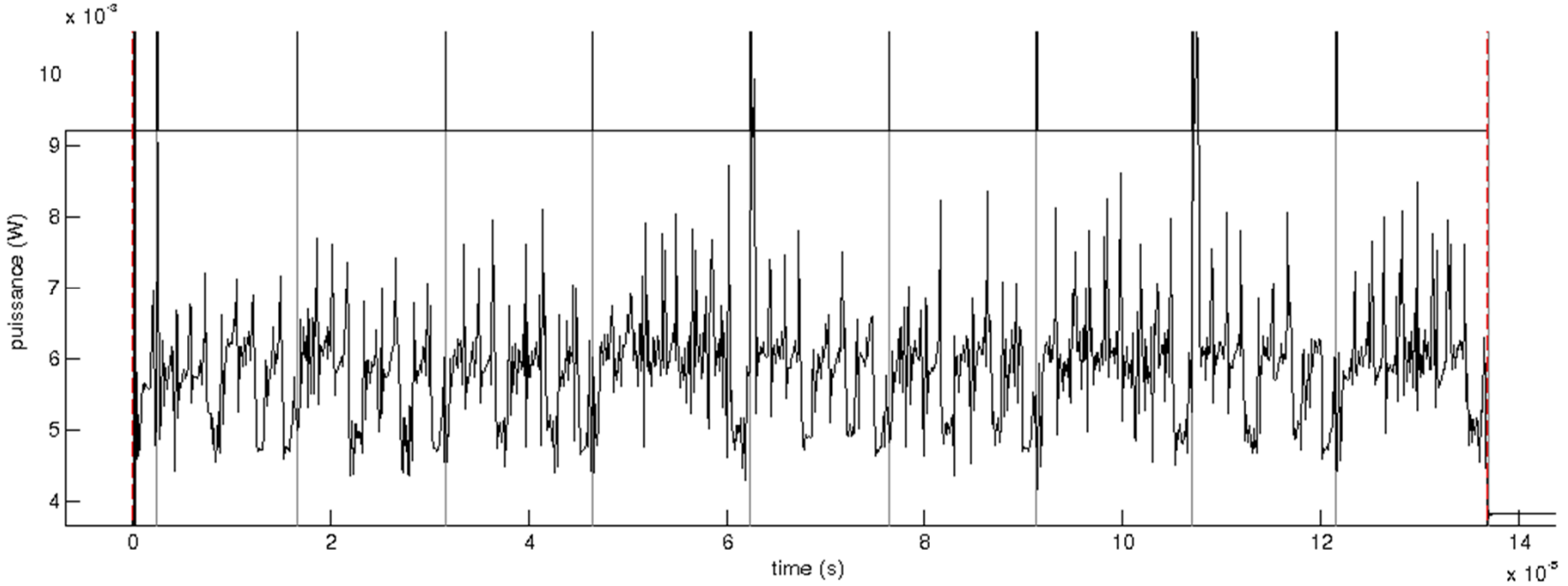
Which key?



- Add / Doubling? (or accumulator overwriting)
- What about MSBs?
- What is the private key?

Window-based computation

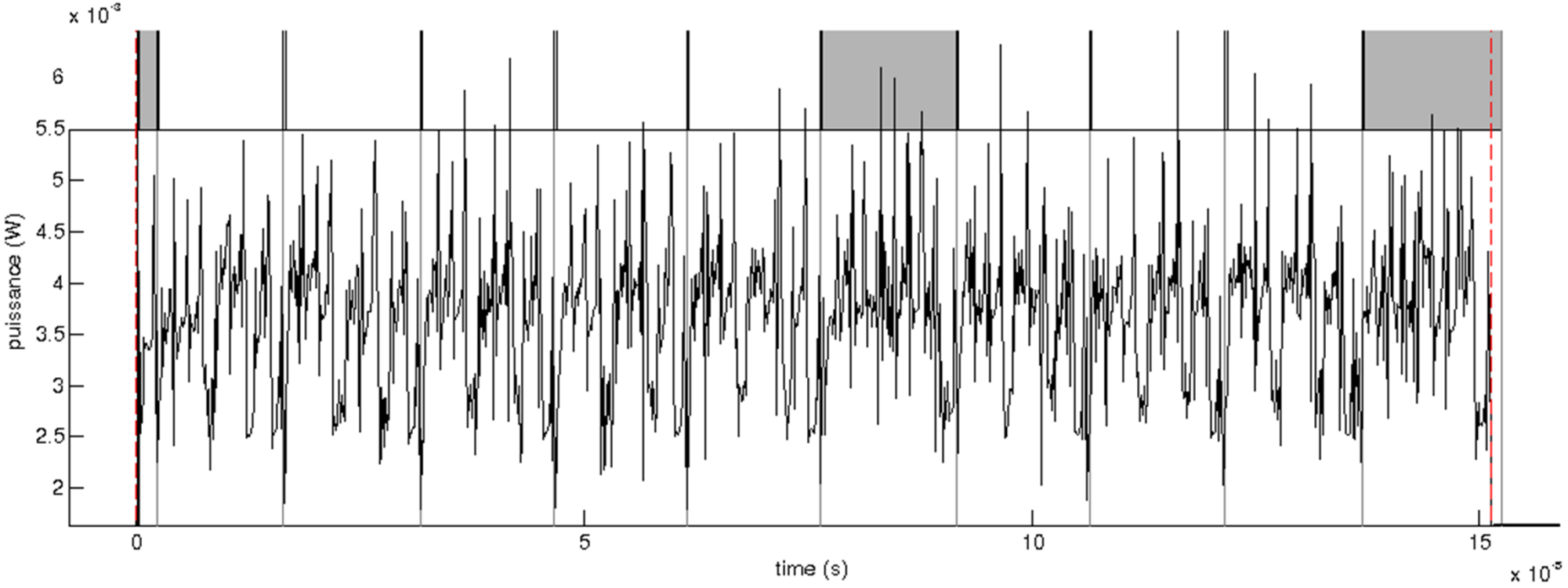
- Random window size: 2 or 3 bits (when necessary, 1 bit for LSB)
- Pre-computed values for additions on 2 or 3 bits



- Add / Doubling? (or accumulator overwriting)
- What about MSBs?
- What is the private key?

Answers (1) …

- **Key with simple computation: 8 doubling (+1 over-writing), key with 9 bits => MSB=1 => reading additions, key=100001001 (265)**

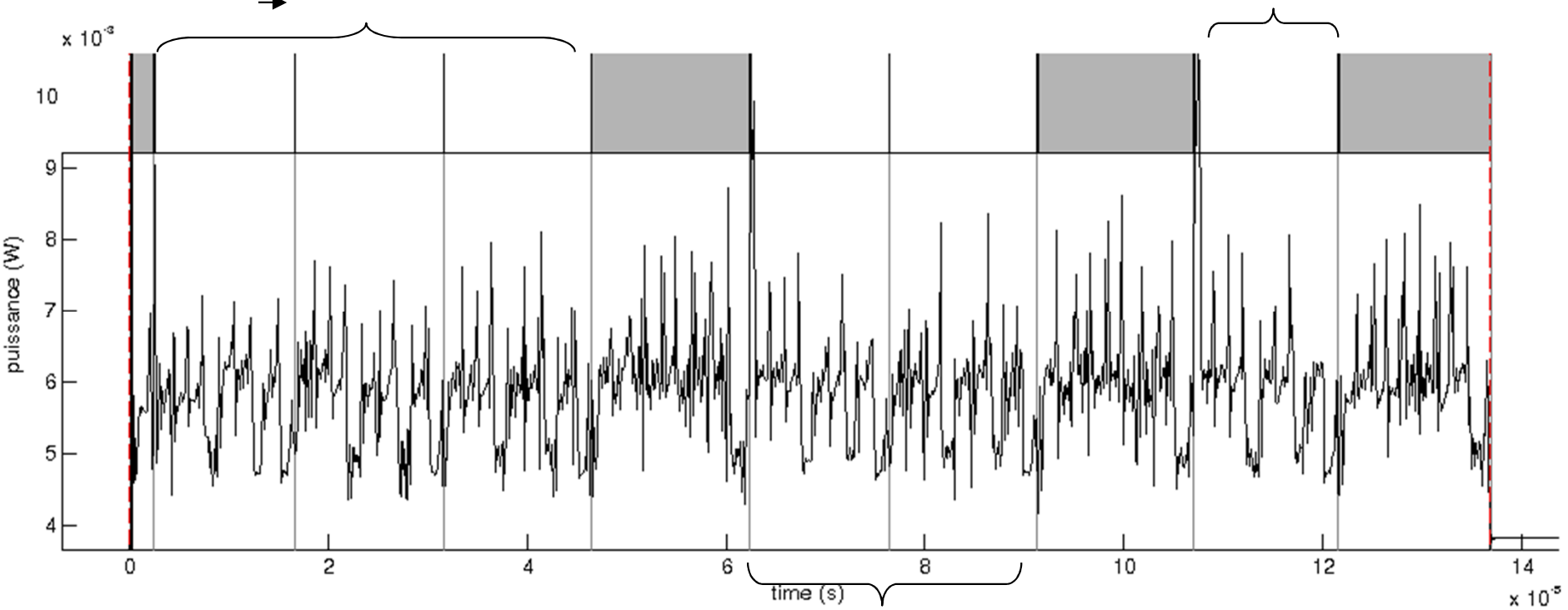


Answers (2) …

- Key with windowed computation: 4 windows, LSB=1

3 doubling before add => 3-bit window

1 doubling before add => 1-bit window

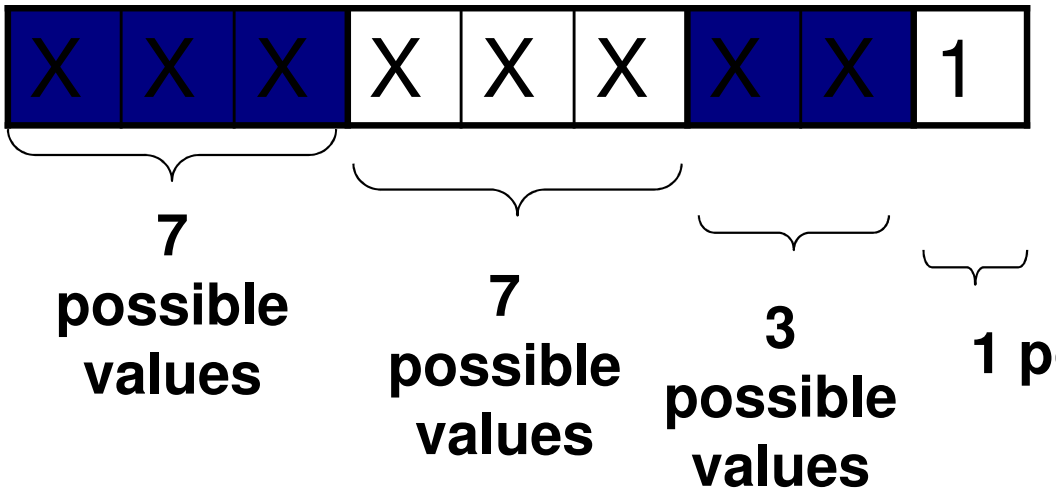


3 (non-null) bits before over-writing!

2 doubling before add => 2-bit window

Answers (3) …

- Key with windowed computation: 4 windows, LSB=1



4 non-null windows since over-writing or addition for each …

⇒ 7x7x3x1 possible keys (147 values, so 54% of all keys)

BUT key <= 269 so … how many possible keys?

Key was $(001\ 011\ 11\ 1)_2$=95

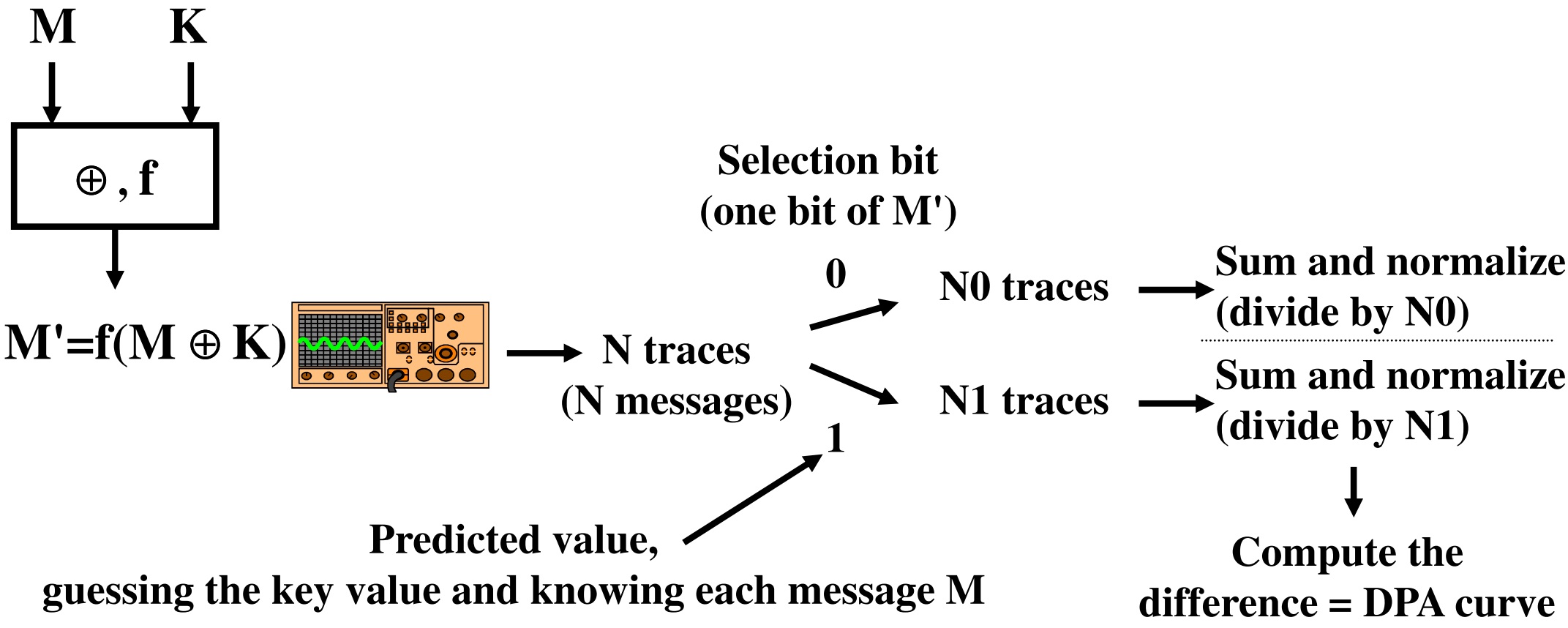
DPA (Differential Power Analysis)

- Introduced by P. Kocher (1998)
- Data collection phase + data analysis phase
- Basic procedure
  - Gather many power consumption curves
  - Assume a key value
  - Divide data into two groups (0 and 1 for chosen bit)
  - Calculate mean value curve of each group
  - Correct key assumption → not negligible difference
- Attack in pieces (divide and conquer - small parts of the key) – e.g. DES only 2^6 choices per Sbox (exhaustive search feasible)

DPA - Basics

- Modeling the power consumption: Hamming weight model
  - Typically measured on a bus, Y=aH(X)+b
    (Y: power consumption; X: data value; H: Hamming weight)
  - Other possibility: the Hamming distance model
    Y=aH(P ⊕ X)+b - Accounting for the previous value on the bus (P)
- DPA can be performed in any algo that has operation M'=f(M ⊕ K), where M, f are known and K is the segment key
- The waveforms are captured by a scope (many executions with many messages) - The selection bit will classify the wave
  - Hypothesis 1: bit is zero / Hypothesis 2: bit is one
  - The DPA waveform with the highest peak will validate the hypothesis
  - A differential trace is calculated for each bit
- Typical: 500K+ acquisitions …

DPA - Illustration



If function f only depends on a small number of bits of the key (typically Sbox), exhaustive search of this sub-key becomes easy

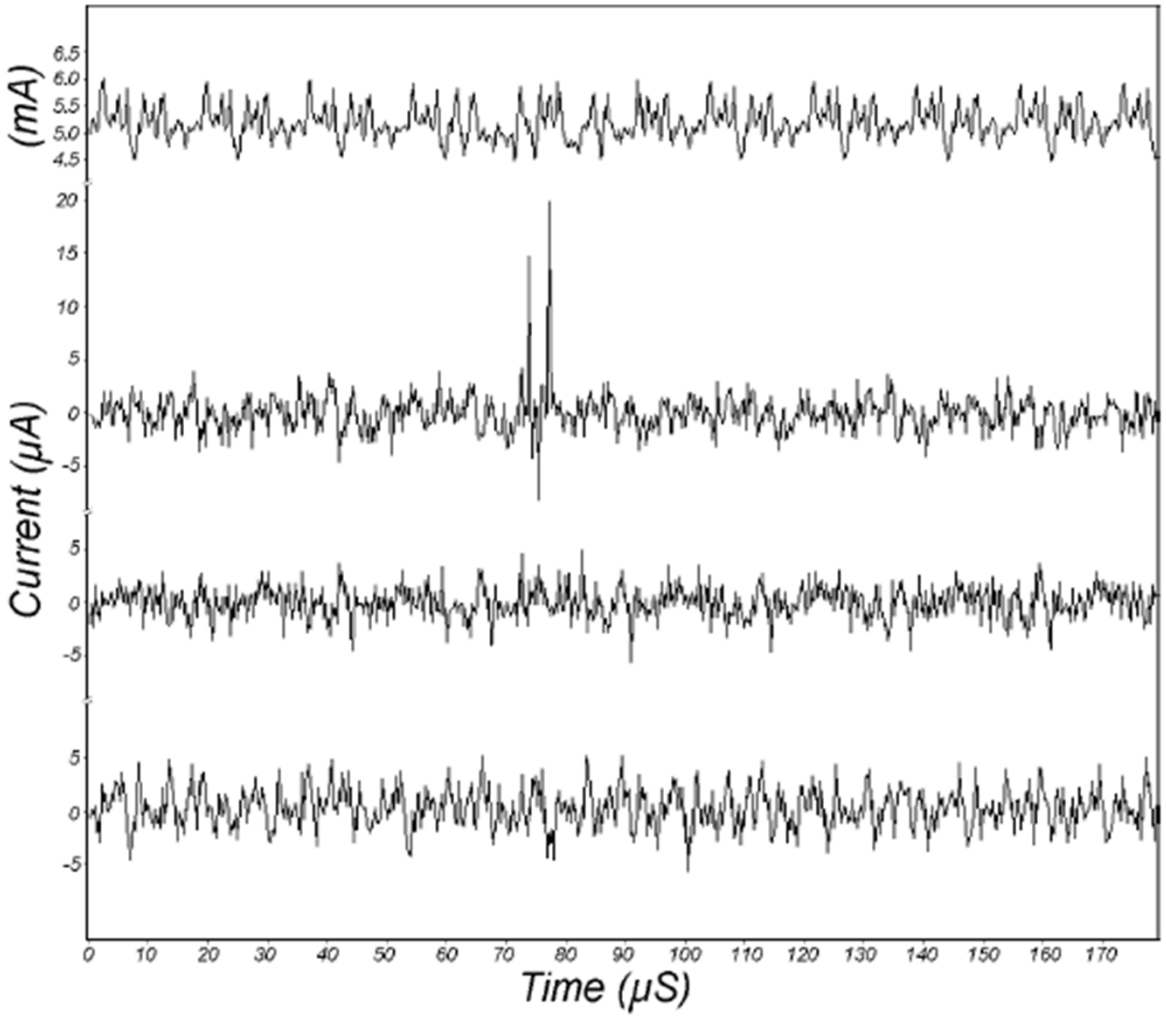
- 1 DPA curve per possible sub-key value
- highest peak shows the right guess (contrast or peak height proportional to $N^{1/2}$)

Example of DPA - DES

Average power consumption

Power consumption differential curve with correct key guess

Power consumption differential curves with incorrect key guess



Example of DPA - AES

- Number of computations ?
- 128-bit key (16 bytes) – Each byte used independently as a 8-bit Sbox input (after message XORing)
- 256 DPA curves to compute per byte, 16*256 to find the 128-bit key

$$2^4 * 2^8 << 2^{128} !!$$

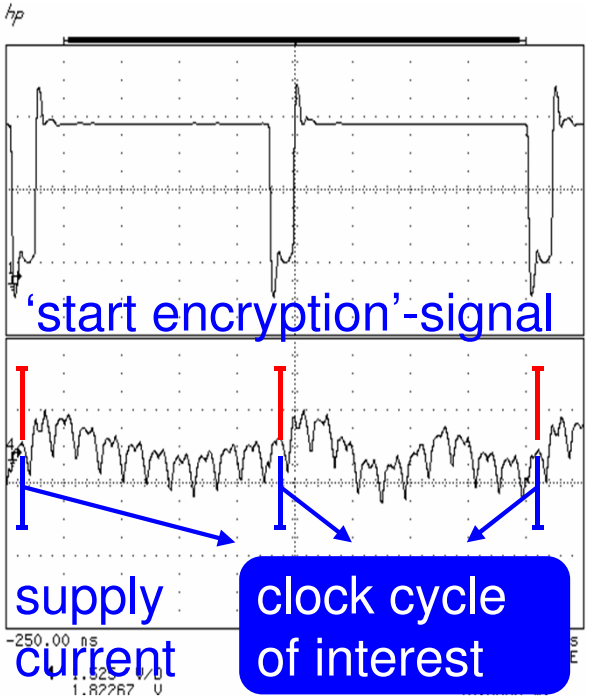
- But
  - Several peaks can be similar (or even a wrong guess may lead to a higher peak)
    - Algorithmic noise (implementation model)
    - Implementation characteristics
  - Comparing results for several selection bits may help (but do not always agree !)

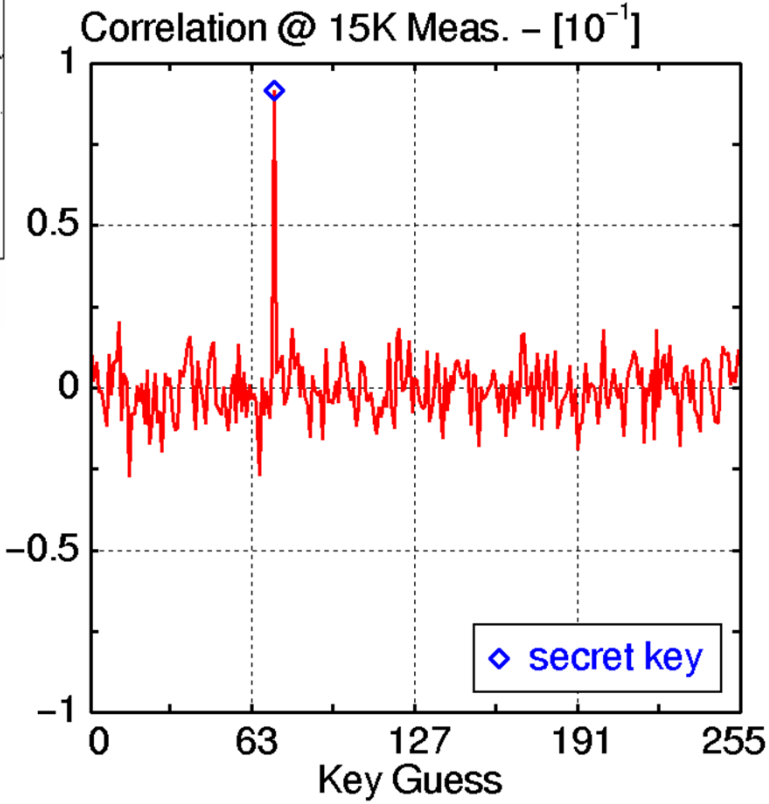
And attacks are more and more acute …

- Mono-bit vs. multi-bit DPA
- Second order power analysis
  - Rather than observing a single consumption time, focus on correlation between two points in time
  - Can be extended to 3rd, 4th order …
- CPA: Correlation Power Analysis
  - The equation for generating differential waveforms are replaced with correlations
  - Rather than attacking one bit, the attacker tries predicting the Hamming weight of a word
- Template attacks
  - A single (a few) sample may be sufficient (suited to stream ciphers, and cases with key re-use limited by system level protocols)

Power analysis - AES

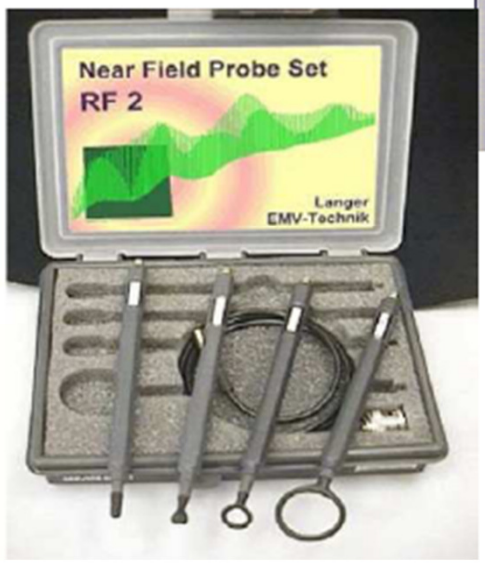
- Unprotected ASIC AES with 128-bit datapath, key scheduling
- Measurement: Ipeak in round 11
- Estimation: HamDistance of 8 internal bits
- Comparison: correlation
- Key bits easily found despite algorithmic noise
- 128-bit key under 3 min.

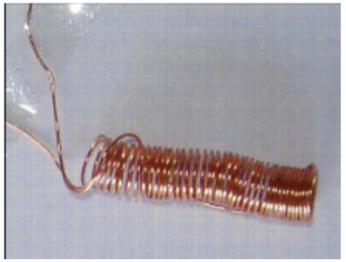




Source: K. Tiri, Intel

Electromagnetic attacks (SEMA – DEMA)

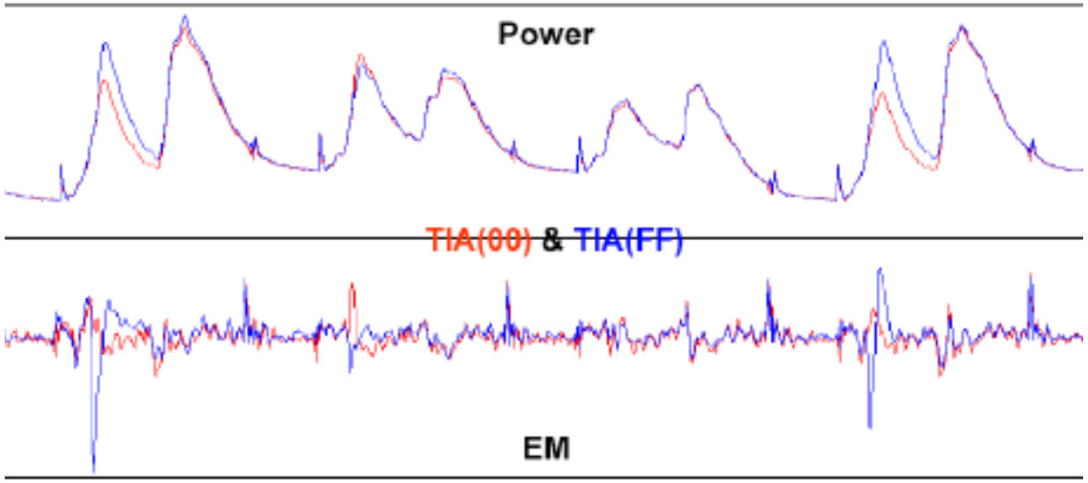




Same principles as power attacks, with a different measurement equipment giving more precise (more localized) internal measures and bypassing current smoothers … but experimentally more difficult to put in practice.

Comparison power/EM

- EM more local (depending on setup and probe), with wider bandwidth (~100 MHz power, ~1 GHz EM) => more data in the recorded signal

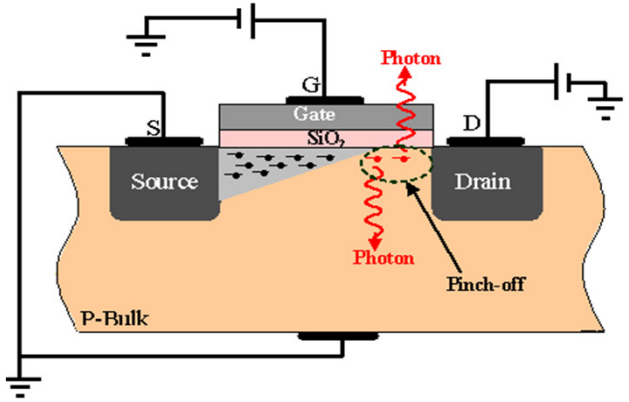


- No direct connection required for EM (e.g. better for FPGA PCB !)
- Magnetic field better for smartcard analysis than Electric field, but small amplitude and SNR => low noise amplification required + trade-off between precision (probe size) and signal amplitude
- Many parameters: probe size and orientation, measurement position on chip, distance to chip, time interval for analysis, …

EMA "success story" - example

- EAL4 product - AVA_VAN.5 robustness level
- Protected ECC crypto-processor with "add always" implementation or RSA with "multiply always"
- About 2 weeks of (manual/visual) EM measurement analysis by expert
- Broken key – due to difference in storage address (least significant bit) for the useful and useless additions or multiplications

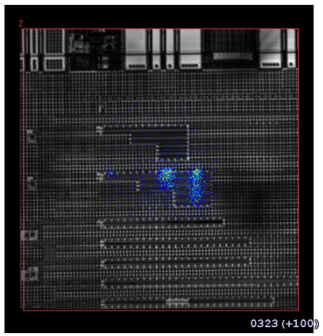
Attacks based on light emission (photons)



NMOS transistor

Failure analysis equipments
Hamamatsu – Tri PHEMOS

- Spatial analysis of photon accumulation
- Dynamic analysis techniques (TRE, PICA)
  - Signal propagation analysis
  - Functional analysis (function behavior - reverse engineering)



Courtesy J. Di-Battista (PhD defense - April 2011)

Counter-measures against SCA

- Attacker always needs an initial guess to compare his measurements to - Protection: make measurements unguessable or constant (data independent)
- Several levels
  - Algorithmic level
  - Architecture level
  - Gate level
  - …
- Examples
  - Avoid conditional execution
  - Use constant-time (i.e. worst-case time) programs
  - Use dedicated "non-leaking" logic
  - Randomize execution flow (SW or HW) – But resynchronization attacks are possible
  - Add noise… but this will only require more samples for a successful attack

Protections against power analysis

- **Hardware:**
  - **Main idea: balanced switching (dual rail, or N rails, with systematic simultaneous 1 -> 0 and 0 -> 1 transitions … on SAME load …)**

    Not so easy ! Specific design techniques/tools (dual functions), load balancing (P&R …), … but increasing process variations !

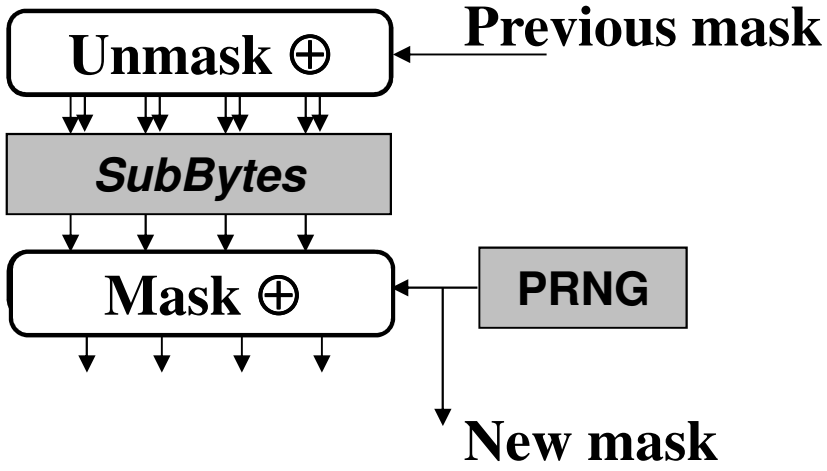
    + must avoid evaluation anticipation (internal Z nodes), …

  - **Other ideas:**
    - **Reduced synchronization by variable/random jitter on clock or some operations (not just noise !!)**
    - **Masking of actual algorithmic values (manipulated data are not directly correlated to the key value) – but still vulnerable to DPA**
    - **Add dummy operations (possibly randomly) => useless power consumption during sensitive operations**
    - **Random execution order for independent instructions or computations**
- **Software: Specific coding techniques (usually at assembly level)**

RTL Countermeasures – Masking

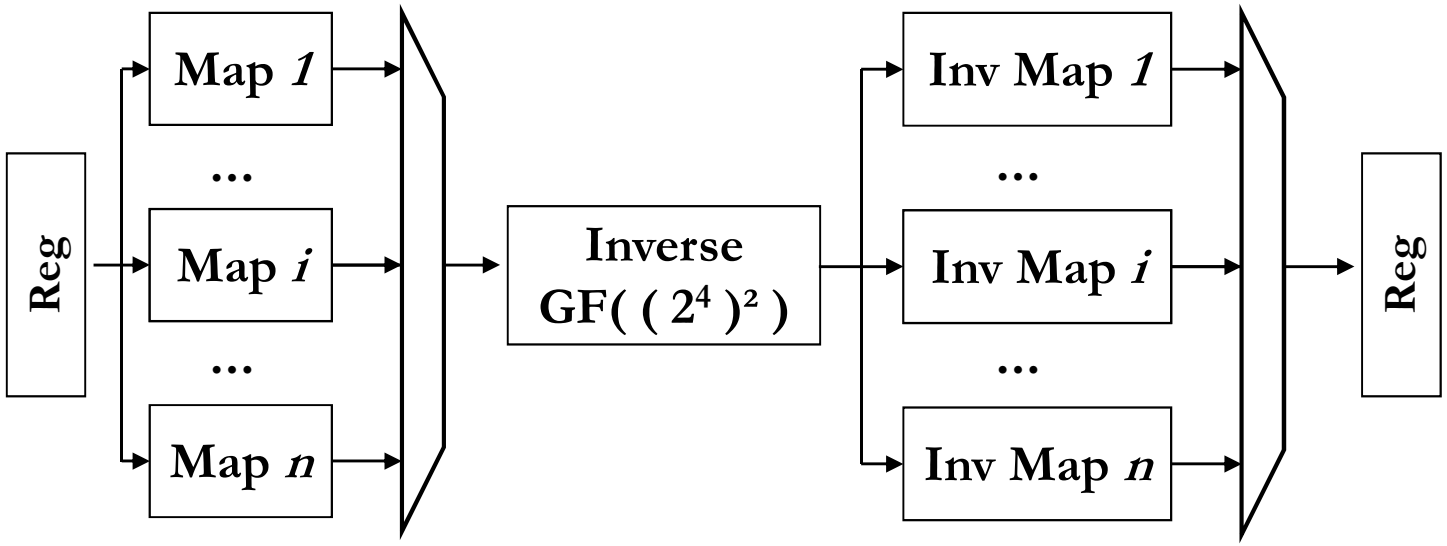
- Linear masking is a common low-cost masking scheme for linear data path (P network)
  - Masking at S-Box output
  - Unmasking at S-Box input
  - Change mask value as often as possible

- S-Box not protected
  - Use different (multiplicative) masking
  - Rotating masked S-Boxes



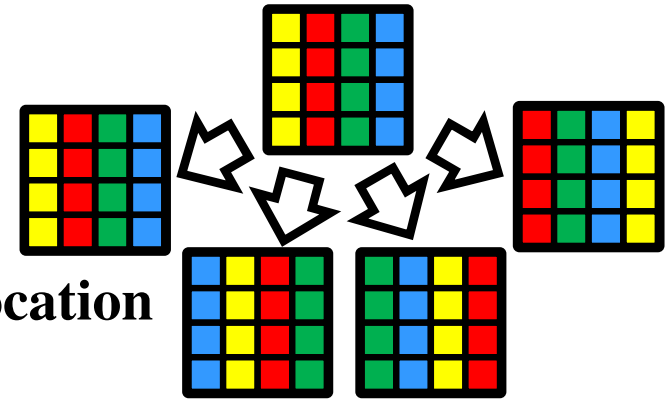
RTL Countermeasures – Dynamic Representation

- For each S-Box, implement several parallel composite field mappings
  - From 1 to 8 possible dynamic mappings for any composite field
  - Choose randomly at runtime (several granularities)
  - At output, choose the correct inverse mapping to get back the result
- Limited to S-Box data path
- Independence of mappings?
- Vulnerable to Zero- and One-value attacks (as multiplicative masking)

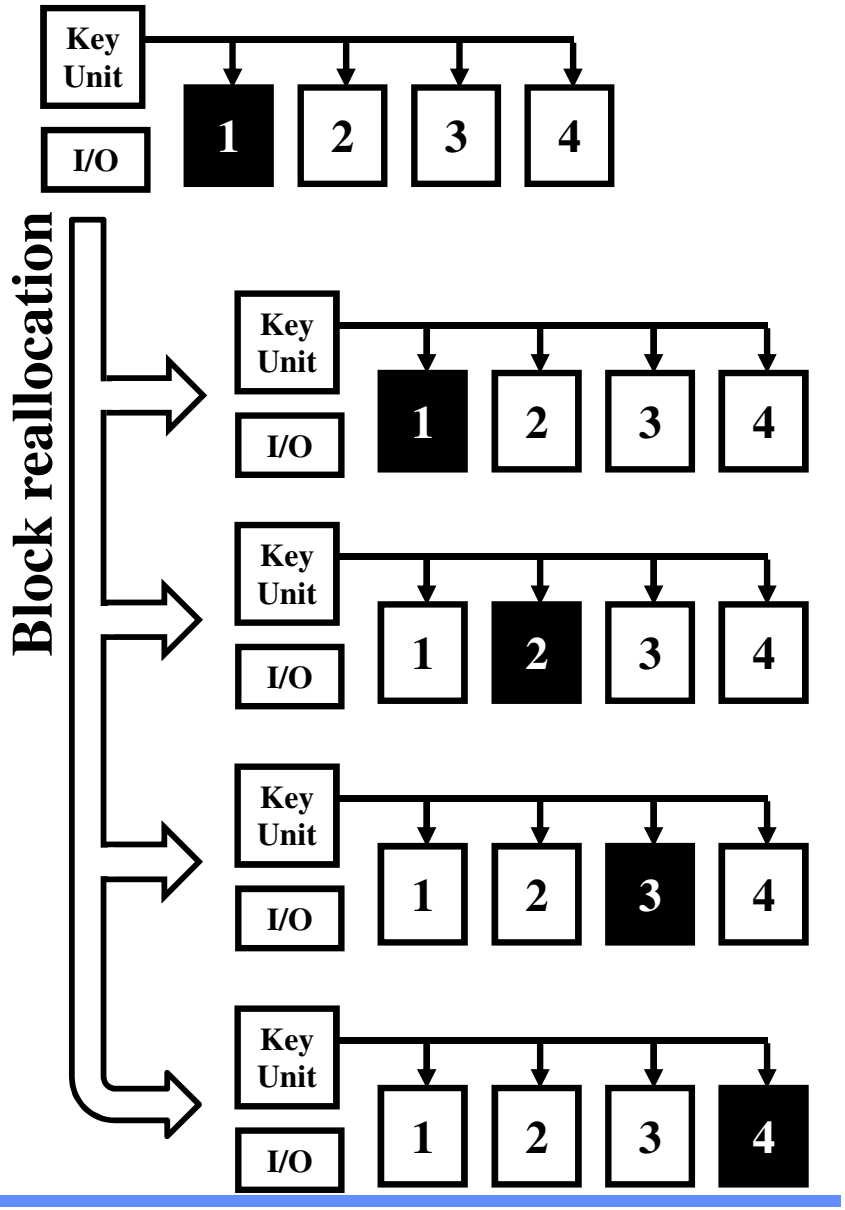


RTL Countermeasures for AES

- Random register renaming (sw)
- Correct order is restored before output
- Random data allocation
  - Block-wise
  - Column-wise
  - Byte-wise
- Several external constraints due to data dependency

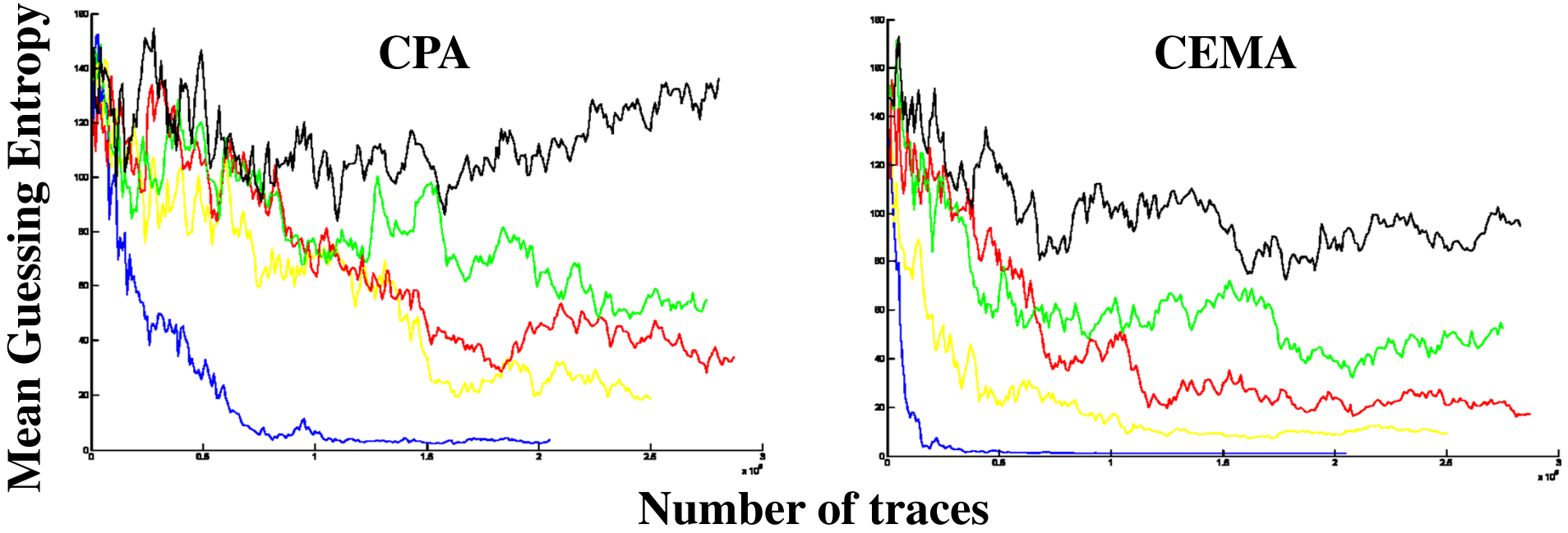


Column reallocation



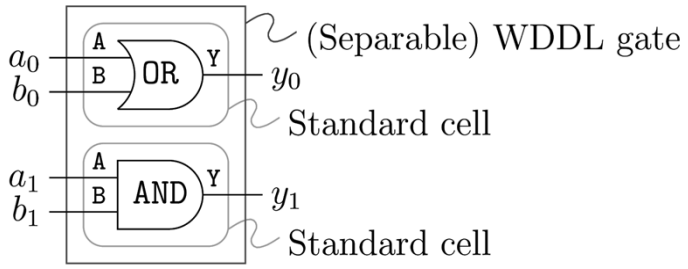
EM vs Power (again…)

Design	Power Analysis			EM Analysis		
	Key bytes found	Mean Guessing Entropy	# traces (x10^3)	Key bytes found	Mean Guessing Entropy	# traces (x10^3)
Basic	15	1	205	16	1	155
+ LinMsk	4	54	275	8	52	275
+ DynMap	5	34	287	9	17	287
+ D*R	7	19	250	12	9	250
+ All	0	136	283	0	94	283

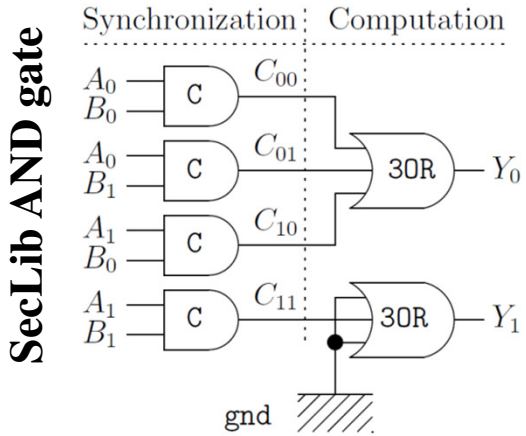


Dual Rail Logic

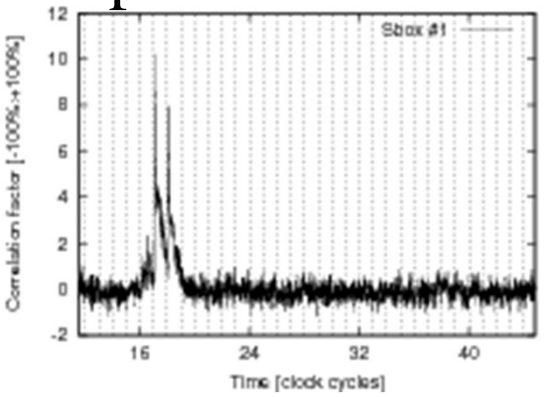
- **WDDL: Wave Dynamic Differential Logic**



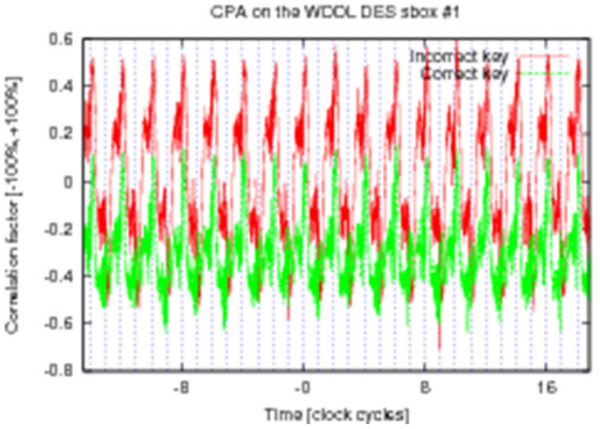
- **SecLib: full-custom balanced quasi-delay insensitive (QDI) cell library**



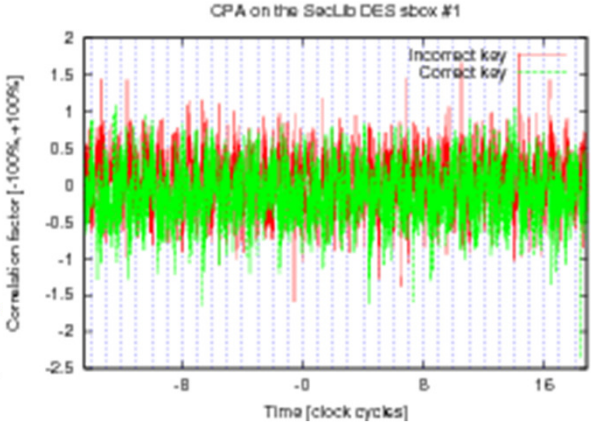
1. Unprotected version



2. WDDL resistance

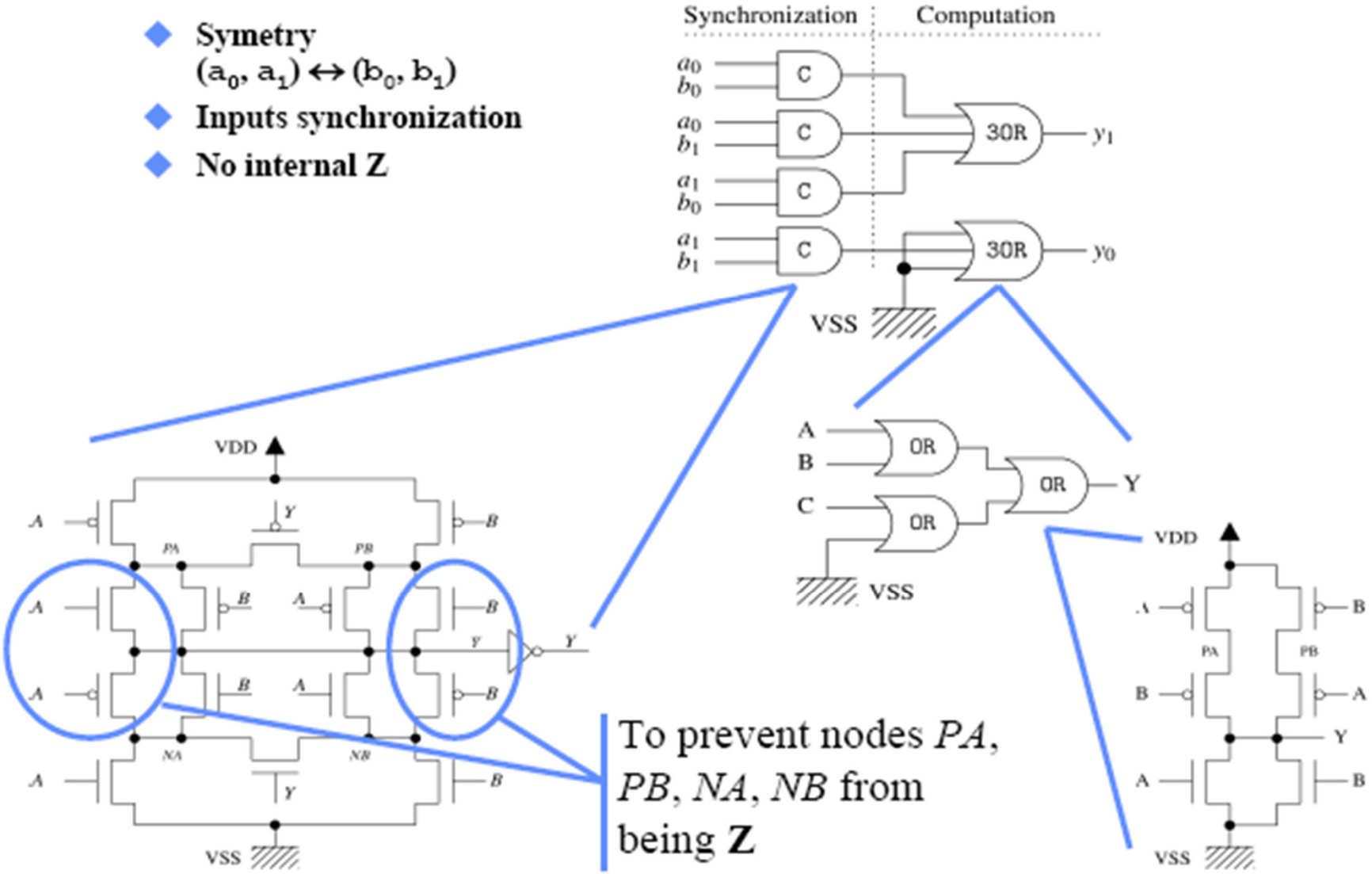


3. SecLib resistance



Example of secured NAND gate

- Symetry
  $(a_0, a_1) \leftrightarrow (b_0, b_1)$
- Inputs synchronization
- No internal Z



From ECoFac 2006 presentation – S. Guilley et al. (GET/ENST)

Multi-level counter-measures (SCA)

Level	Counter-measures against SCA (examples)
Gate (transistor level structure)	**Power-constant logic** or **dual-rail with precharge = DRL**). **Pwr-cst *in average* logic = RSL, M-DLP, *etc.***
P&R	Differential pairs routing
RTL	Constant Hamming weight encoding **=> DRL**
Algorithm	Dynamic masking, substitution box reconstruction: **cracked**
Architecture	S-RNG **(biased for low N)**, bus scrambling (**broken**), public key infrastructure in NoCs **(too expensive)**
System	Secret sharing, peripheral access after bus integrity verification

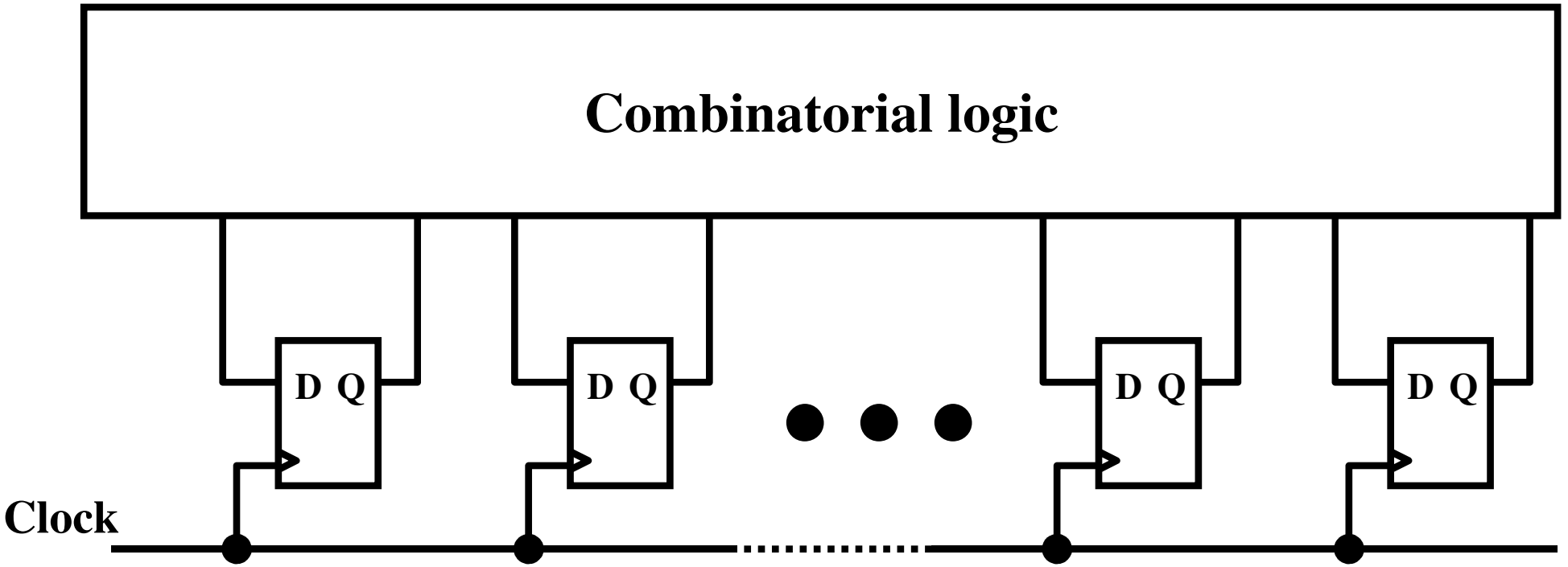
Implementation Attacks

- Invasive attacks on the circuit
- Side Channel Analysis
  - Time Analysis
  - Power Analysis
  - EM analysis
  - …
- Test structures
  - Scan chains,
  - …
- Fault attacks
  - Voltage glitches,
  - Clock glitches,
  - Overvolting,
  - Downvolting,
  - Laser shots,
  - Harmonic EM,
  - Pulsed EM injection,
  - …
- Which countermeasures?

Courtesy D. Hely [HOLT06]

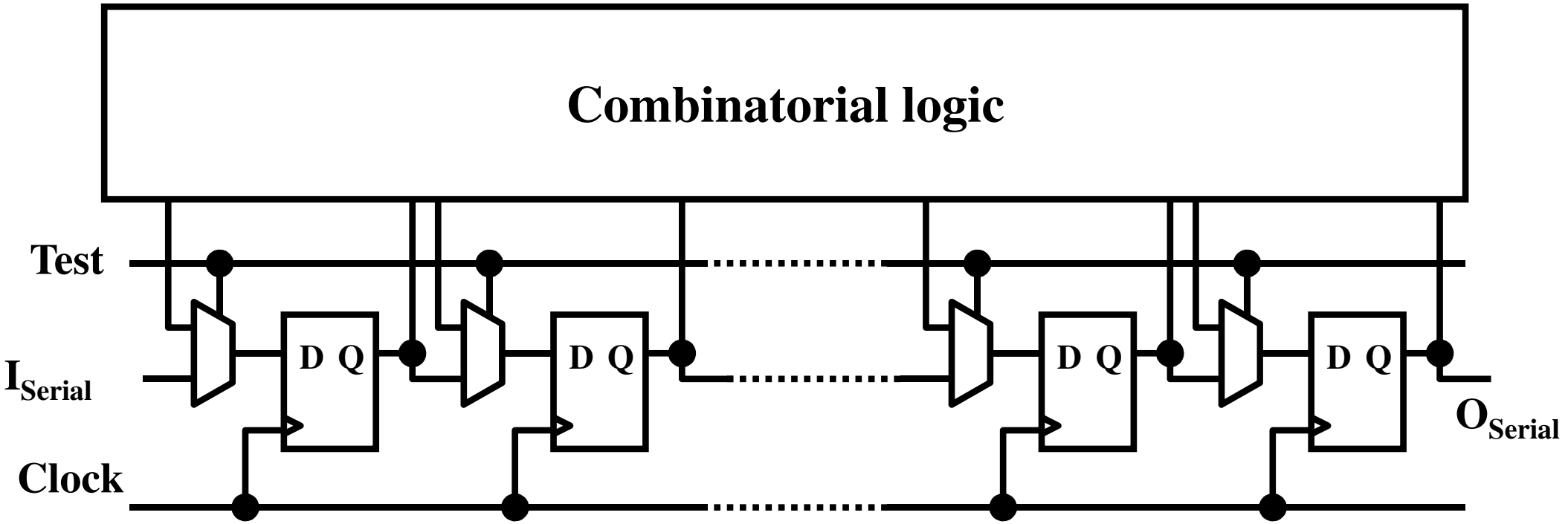
"Scanpath" implementation – basics (1)

Initial circuit:

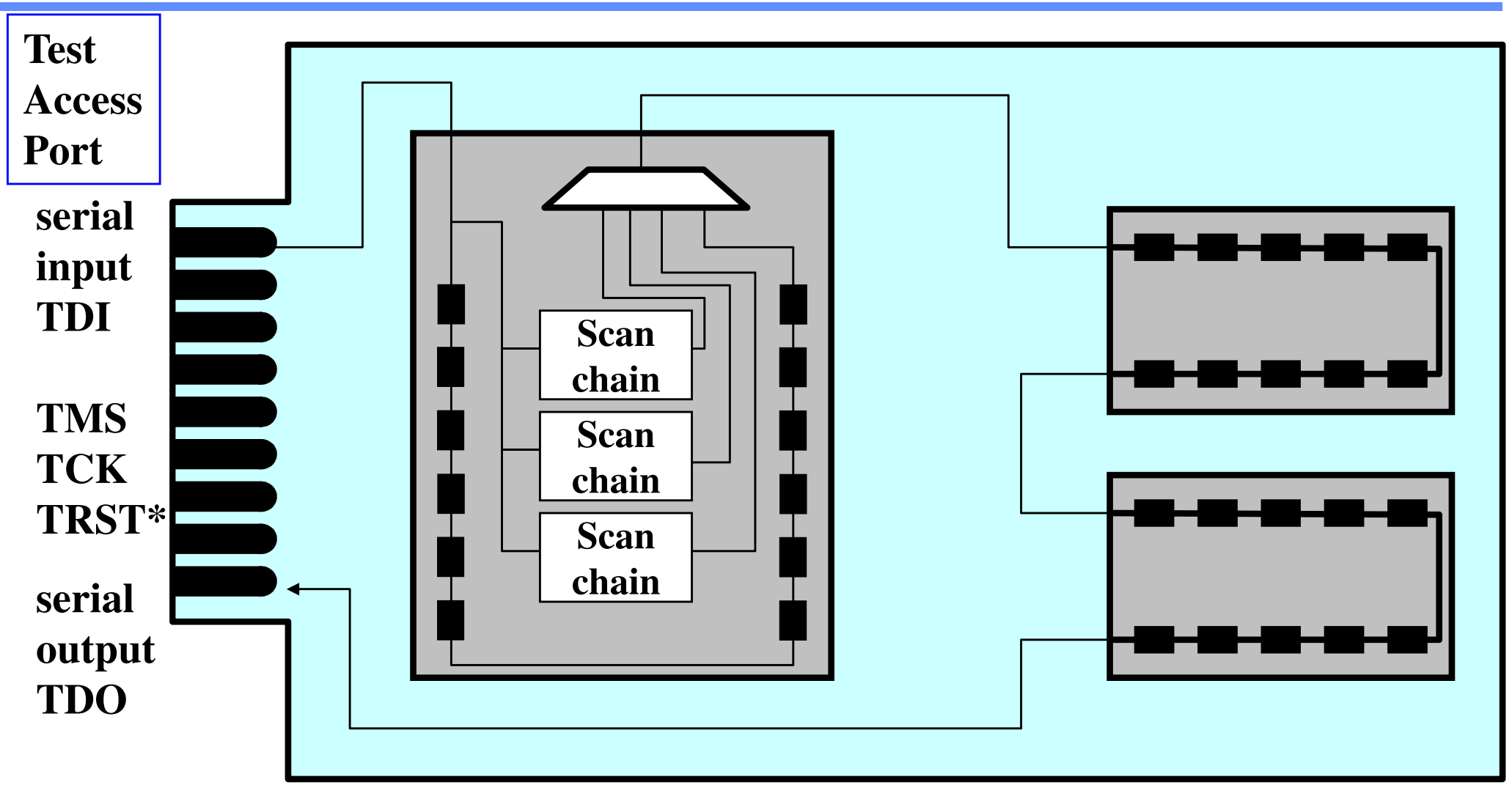


"Scanpath" implementation – basics (2)

Circuit with a single scan chain:

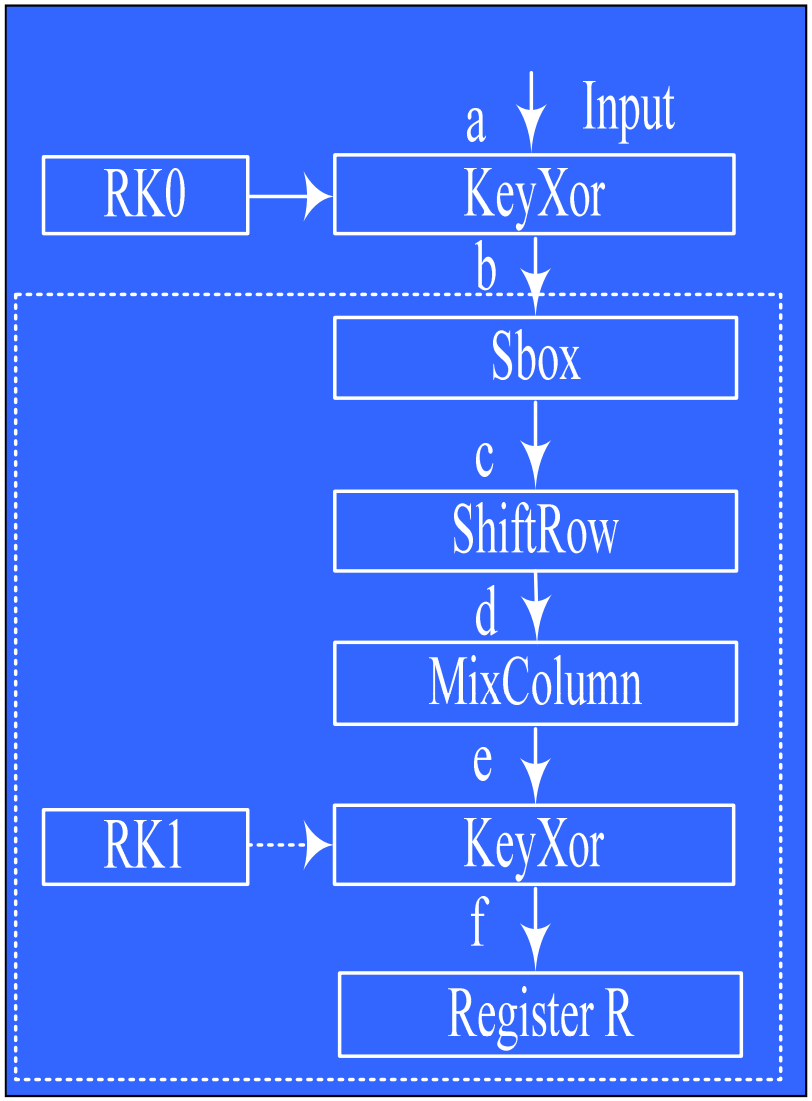


Boundary scan (JTAG)



Several usages: board test, in-situ circuit test, (remote) configuration

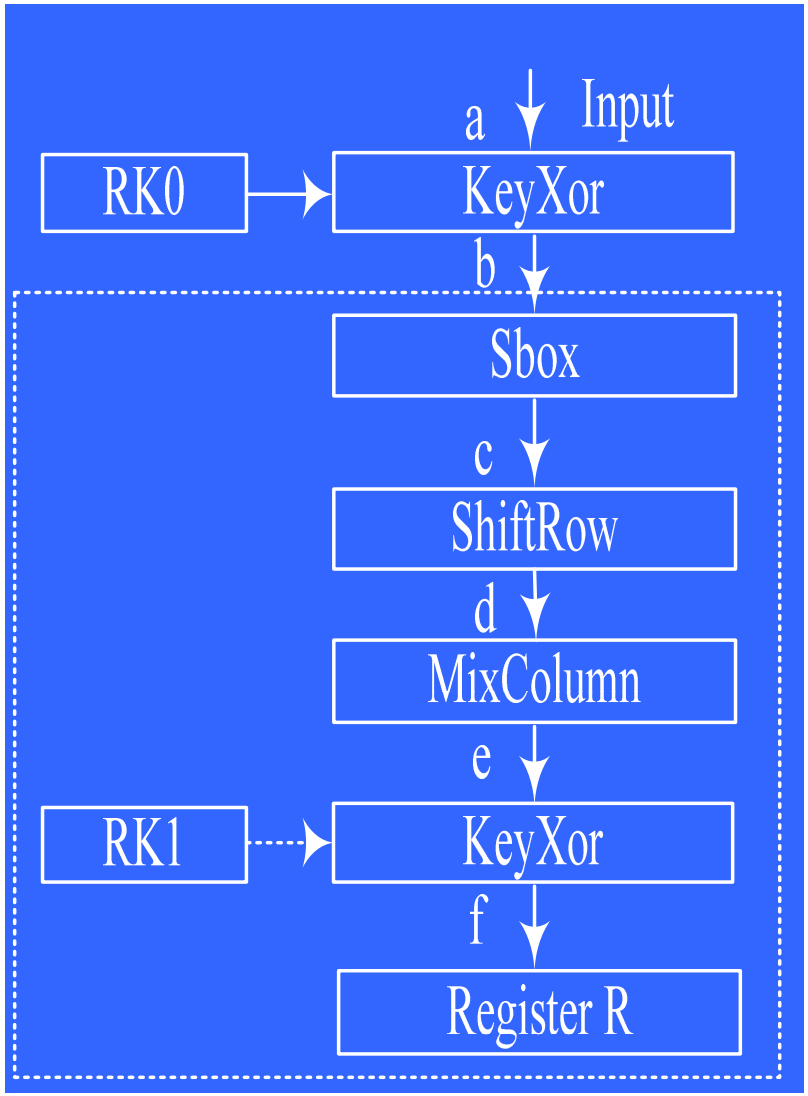
Scan-based attack on AES – Starting point



- Input is partitioned into 16 bytes a_{11},… a_{14}, a_{21},… a_{24}, a_{31},… a_{34}, a_{41},… a_{44}
- Register R is fed back to point b ten times with RK1 to RK10
- 128-bit Round register R is in scan chains
- The complexity of AES is reduced to one round
- Value of RK0 ?

Yang, Wu and Karri, "Secure Scan: A Design for Test Architecture for Crypto-chips", DAC 2005

Scan-based attack on AES – 2 steps

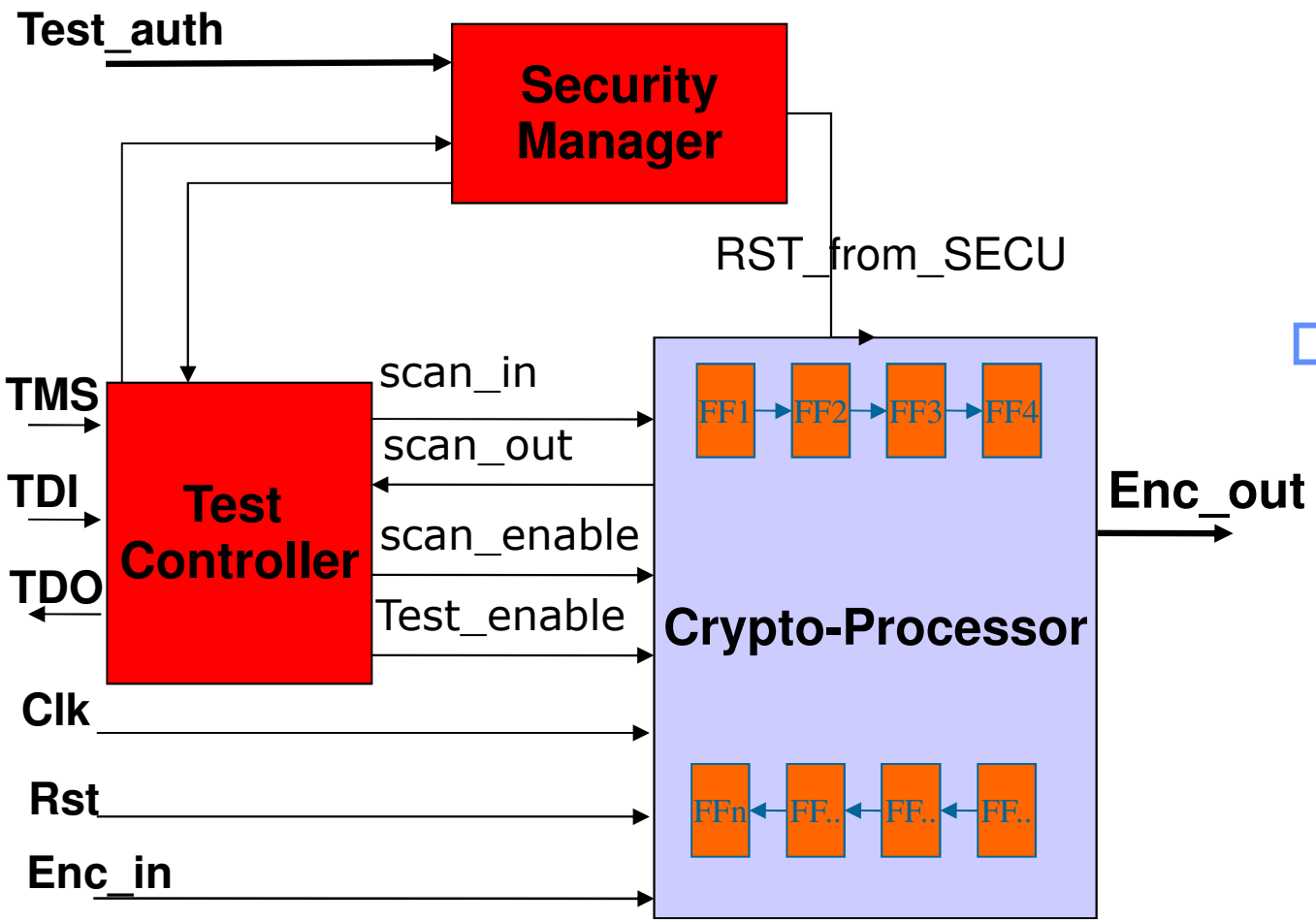


- **Determine scan chain structure**
  - The locations of flip-flops of R in the scan chains are unknown
  - Change in a11-> change in b11 -> change in c11 -> change in d10 -> change in ei0 -> change in fi0 -> 4 byte at R
  - On average, 15 patterns are enough applied at a11 to determine all the 32-bit in Register R (fi0) by comparing the scanned out bit streams
- **Recovering Round Key RK0**
  - Determining bit positions
  - Cancelling RK1 effect by differential analysis
  - RK0 obtained from a XOR b

Securing test modes …

- Secure test mode activation (secret key, …)
- Detection of unexpected shifts (due to probing)
- (Variable, random) scrambling of scan chains
- Dummy bits in scan chain (detection of invalid shifted patterns)
- …
- Avoid scan techniques when possible: self-test (BIST, SBST)

A typical system



- **System overview**
  - Security manager in charge of authentication
  - A scan chain driven by a test controller
- **Attacks**
  - Test Mode Attack: authentication is bypassed
  - System Mode Attack: Scan_enable activation, Scan_out observation

Courtesy D. Hely [IOLT06]

Securing the scan chain

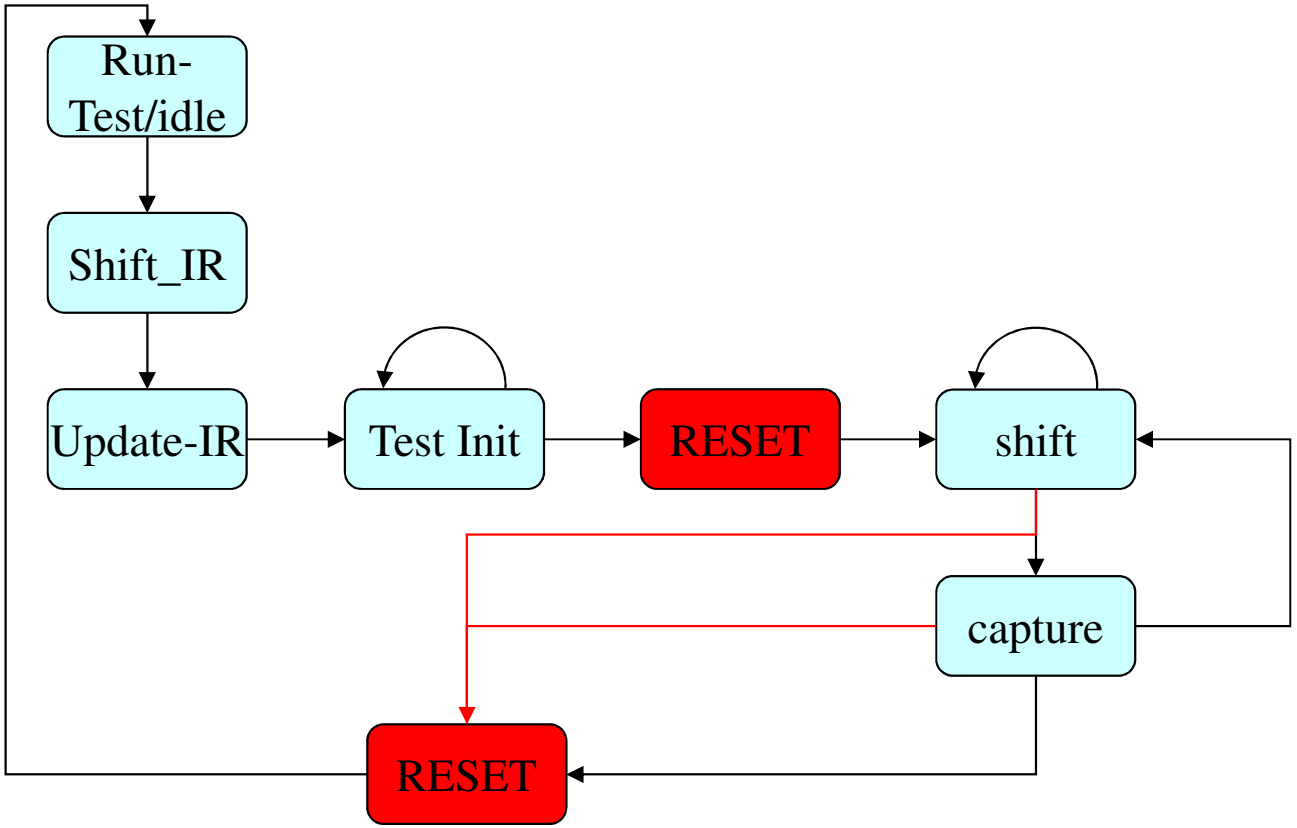
- Goal
  - No observation or control of the functional data processed by the secure system
- Principle
  - Prevent illegal scan shift operations
- Solutions
  - Test mode protection
    - Scan protocol
  - System mode protection
    - Scan chain scrambling
    - Scan enable tree protection
    - Spy FFs

protection against illegal usage of the test mode

protection against scan chain probing attacks

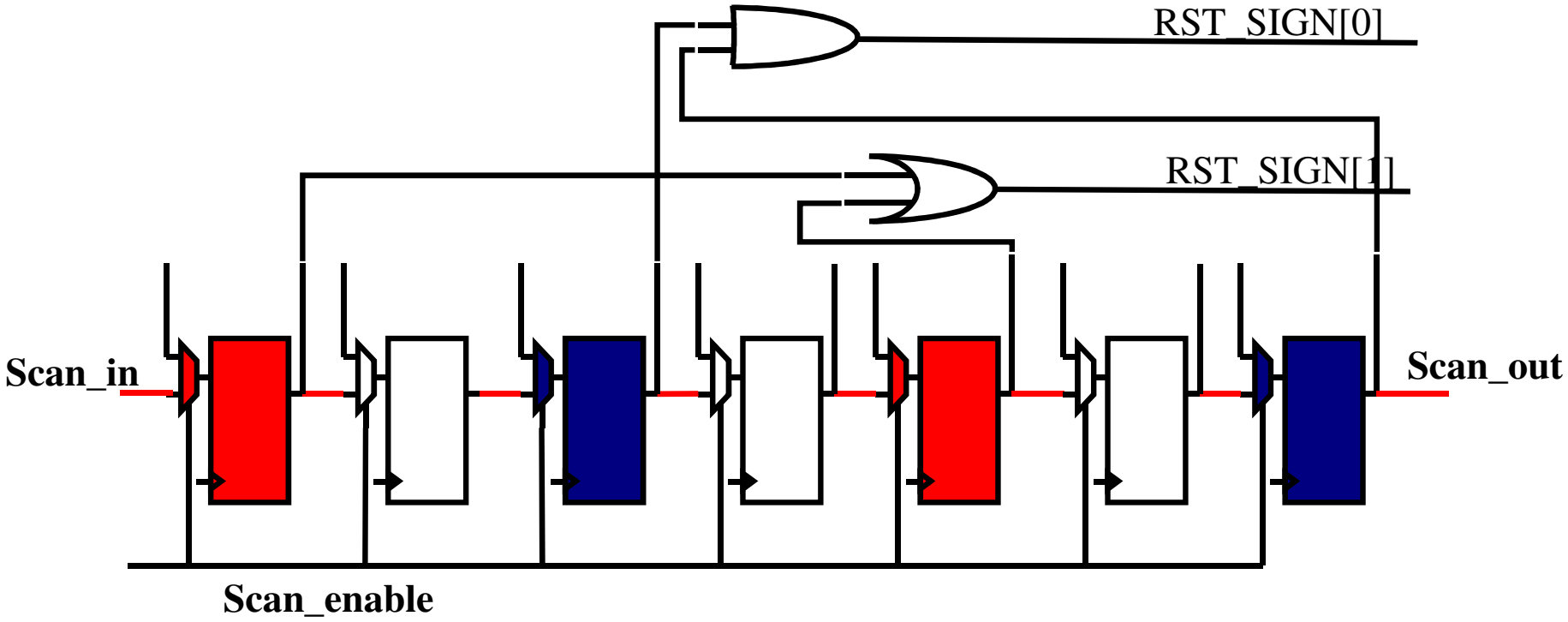
Test mode protection

- Secured Test Controller [He05]
  - Addition of two Reset states
  - Secret data are erased before scan operations
  - Scan data are erased before crypto-operations



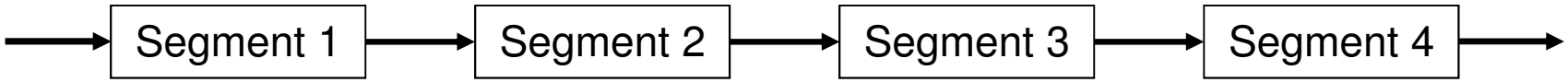
Test mode protection

- **Reset Verification**
  - **Static Verification (no scan chain activation)**
  - **Verification done of a subset of FFs**

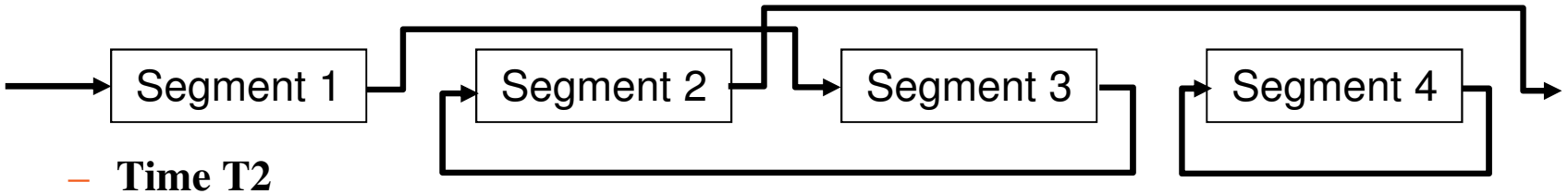


System mode protection

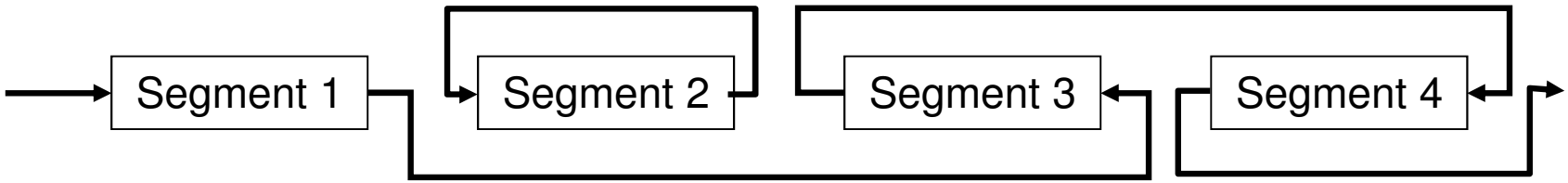
- **Scrambling method [He04]**
  - **Scan path with a prefixed segment organization during test mode**



  - **Scan path with random segment organization if shift during system mode**
    - **Time T1**

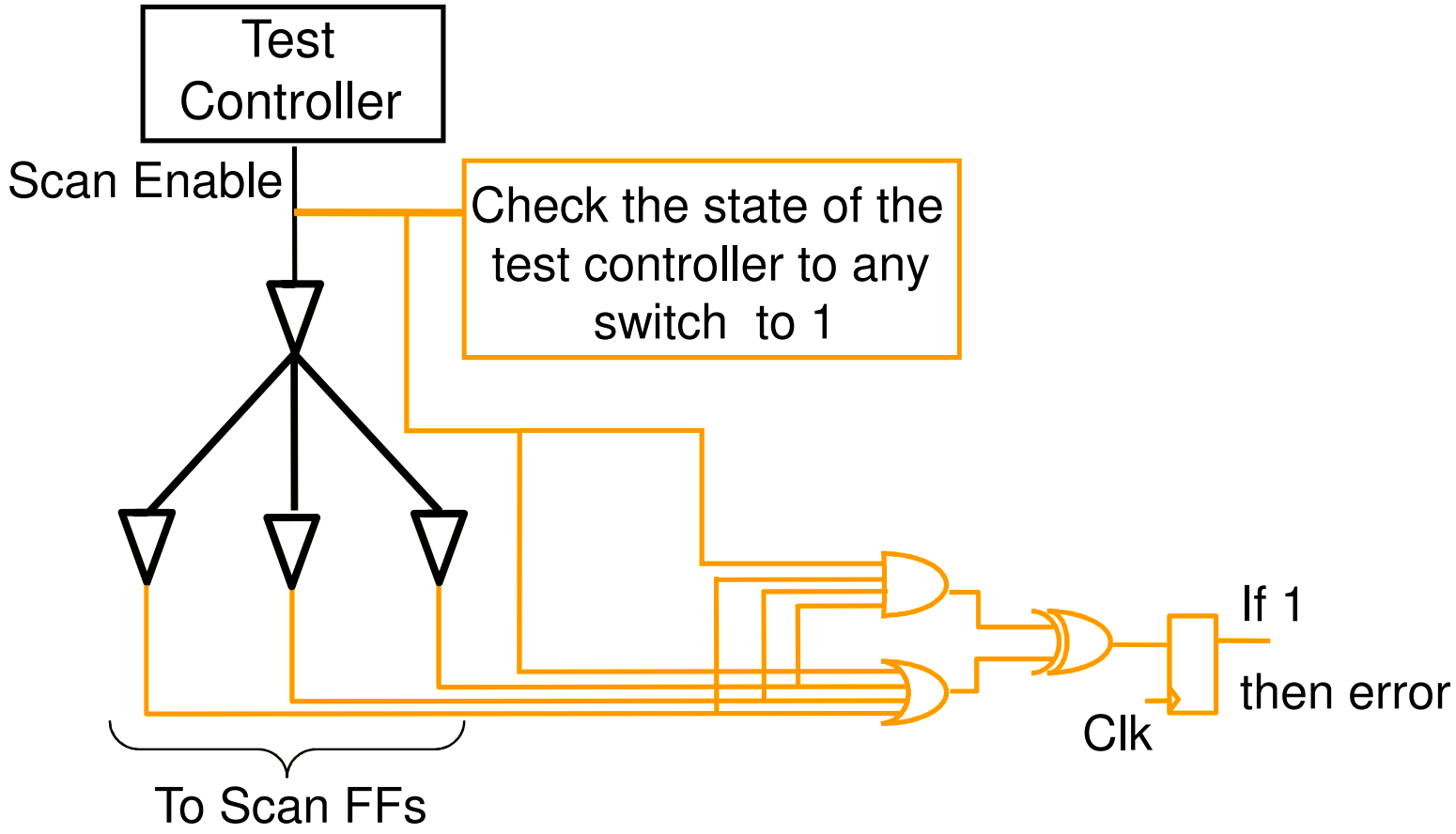


    - **Time T2**



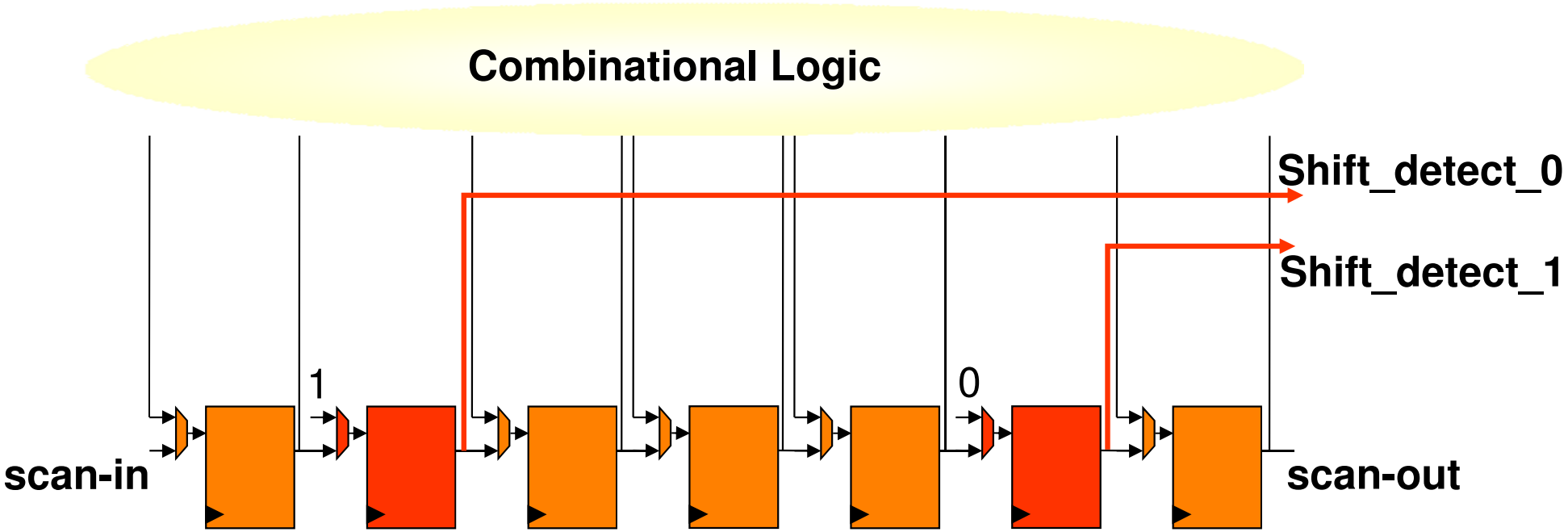
System mode protection

- **Scan-Enable Tree Protection [He05]**
  - **Compare the scan enable signals at different locations**



System mode protection

- Spy Flip-Flops [He06]
  - Include spy cells in the scan chain
  - Control the spy cells to a constant value
  - Observe the spy cells states



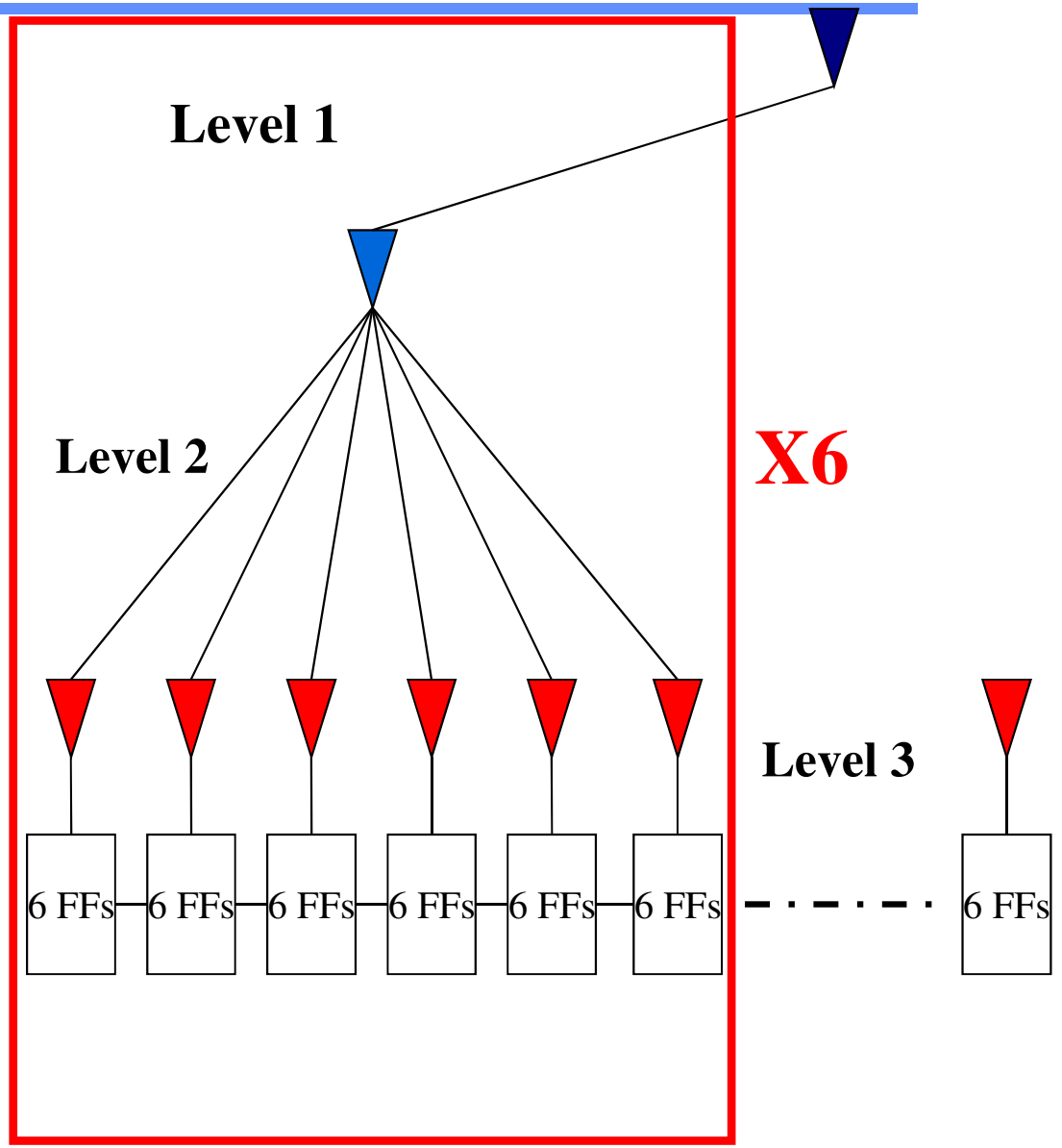
System mode protections - Comparisons

- Case study:
  - DES crypto-processor
  - Scan chain with 198 FFs
  - Attack described in [Bo04]
  - Secure test controller for the 3 architectures
  - Several system mode protections
    - Architecture A (SEI): Scan_enable integrity
    - Architecture B (Spy Cell): Shift detection flip-flops
    - Architecture C (Scrambling): Scan chain scrambling

- Criteria of comparison:
  - Design integration
  - Design reuse, Validation…
  - Test cost
  - Test time, test coverage…
  - Design cost
  - Area, Power, Timing closure…
  - Security
  - Efficiency

System mode protection - Parameters

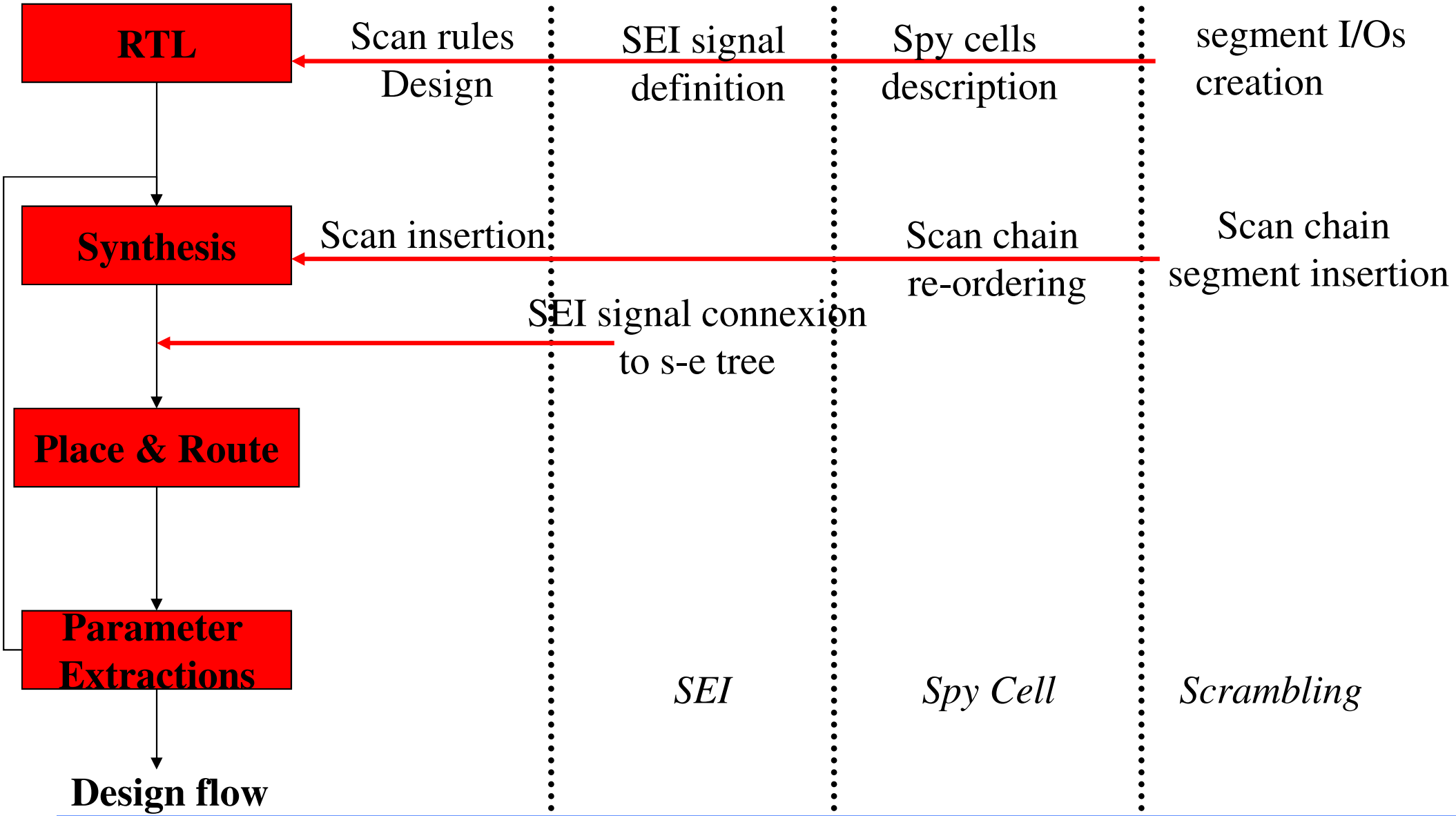
- System mode protection parameters depend on s-e tree bufferization
  - 3 level tree: 1 buffer drives 6 FFs
  - Parameters are chosen so that all brute attack on level 2 are overcome
  - At level 2 one buffer drives 36 FFs



System mode protection - Architectures

- Architecture A
  - SEI_INT signal is connected to the 6 braches of level 2 of s-e tree.
  - Connexion is made after synthesis (gate level modification)
- Architecture B
  - 6 spy flip-flops spread in the scan chain
  - Spy flip-flops inserted in VHDL description of the IP
  - Scan re-ordering to correctly spread them
- Architecture C
  - Scan chain divided into 6 segments
  - Random number generator is mandatory to provide alea to the scrambler

Comparison results - integration



Comparison results – test caracteristics

- Test Coverage: idem for the 3 architectures

- Test Data:

SEI	Spy	Scrambling	Without security
+1.75%	+3.5%	+0%	62 patterns

- Test Time:

SEI	Spy	Scrambling	Without security
+1.75%	+5.25%	+0%	12536 cycles

Comparison results – design costs

- Area overhead
  - IP: almost nul for SEI and Scrambling, very low for spy cell
  - Control mecanism in test controller =>Area increase

Arch.	Area.
SEI	*1359*
Spy	*1269*
scrambling	*2560*
Without security	*1100*
Area of test controller	

- Power Consumption
  - Almost nul for SEI ans Spy cell
  - Sensible for scrambling (+7% during a DES ciphering)

Comparison results – security

- Brute attack on scan chain
  - Any attacks on level 2 are detected
  - Immediate detection for SEI, after 2 clock cycles (average) for spy cell
  - DES Attack of [Bo04] simulated with scrambling on, attack fails

- Solution robustness
  - Architectures A and B rely on a reset operation after detection
  - Scrambling is more robust

[Bo04] Bo et al. "Scan-based Side-Channel Attack on Dedicated Hardware Implementations on Data Encryption Standard", International Test Conference (ITC 2004), pp 339-344

Comparison results – summary (circuit level)

		Scrambling	Scan enable	Spy cell
Insertion flow		RTL	RTL + place&route	RTL
Test	Test time	0%	1%	5%
Design	Area	0.2%	0.3%	1.8%
	power c.	7%	0%	0%
Security		+++	++	++

Architectures should be chosen according to application priority:

Time to design: spy cells

Power aware application: SEI and spy cells

High Security: scrambling

…

Built in Self Test (BIST)

- Avoid scan-based testing
- Symmetric ciphers are based on diffusion and confusion principles
  - Diffusion: every input bit should influence several output bits
  - Confusion: relating input, key and output should be as complex as possible
- Large observability …
  - Every input affects different outputs
- … and controllability of the internal state
  - Every output depends on several inputs
- Block ciphers can be excellent pseudo-random generators and signature analyzers
- 100% fault coverage after a few encryption cycles (~240) and only 3% area overhead

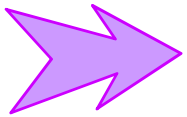
Flottes et al. "AES-based BIST: Self-test, Test Pattern Generation and Signature Analysis", DELTA'08

Implementation Attacks

- Invasive attacks on the circuit
- Side Channel Analysis
  - Time Analysis
  - Power Analysis
  - EM analysis
  - …
- Test structures
  - Scan chains,
  - …
- **Which countermeasures?**
- **Fault attacks**
  - **Voltage glitches,**
  - **Clock glitches,**
  - **Overvolting,**
  - **Downvolting,**
  - **Laser shots,**
  - **Harmonic EM,**
  - **Pulsed EM injection,**
  - **…**

Perturbation-based attacks

- Modifying the nominal working conditions
  - Temperature
  - Power voltage
  - Clock frequency (overclocking)
  - …



Indirect fault/error induction (e.g. timing errors)

Sensors (temperature/voltage) can be used to detect abnormal conditions

- Directly inducting faults/errors
  - External electrical perturbations – glitches (power, clock, …)

Sensors may be used on some critical lines + filtering (e.g. power line)

  - Optical perturbations – (focused) white flash light, laser, UV, X Ray
  - Electromagnetic sources, particles
  - …

Overclocking vs. clock glitch

- **Overclocking**
  - Faults may be induced at each cycle – no timing control
  - No spatial control
  - No identification of induced errors

- **Clock glitch**
  - Faults induced only at the attacked cycles – some timing control
  - Indirect spatial control and error identification due to
    - (1) the circuit structure: critical paths activated for a given internal state and given input values
    - (2) the attacked cycle, defining the internal state and the input values
    - (3) the clock shape (aspect ratio - duration of each level), to activate only some critical paths

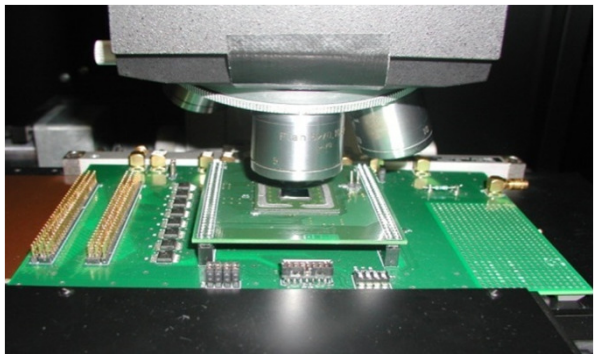
But both can be detected (by sensors) and glitches can be filtered (e.g. by PLLs)

Ideal fault induction technique …

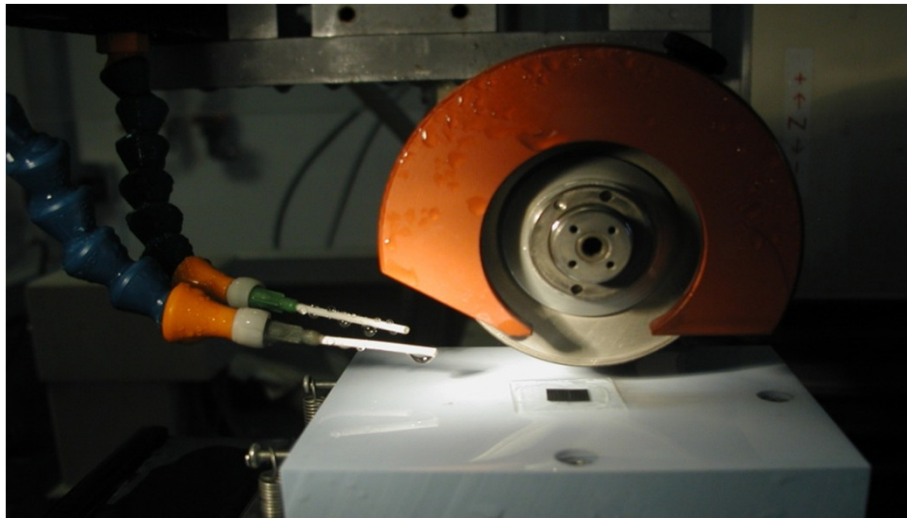
- Location control (x,y)
- Timing control (start, duration)
- Fault type control (stuck at, bit flip, etc.)
- Focalization control (number of faulty bits)
- Reproducibility
- Low cost
- Easy to develop and use

Laser-based attack: preparation

- Example on a Virtex chip (just unpackaged)
  - Substrate @ 790 µm
  - $\lambda \approx 900$ nm
  - P ≈ some Watts
  - ∅ = 40 µm and 8 µm
  - Backside attack => No error



- Substrate thinning
  - Die thinned by a mechanical process
  - Residual thickness of 30 µm $\pm$ 1 µm
  - Successful backside attacks



Fault-based attacks: basics

- In general a combination of perturbation and cryptanalysis

  => fault creation + (potential) exploitation

- Example of the Bellcore attack on CRT-based RSA
  - One correct encryption result
  - One erroneous result from the same encryption (using e.g. a laser)
  - A simple GCD computation … and the 1024-bit key is no more secret !
  - Few constraints on the injection: hit only one of the exponentiations

- Predictive robustness analysis: requires a good knowledge of injected faults

- FA (erroneous result directly exploited) – DFA (erroneous vs. correct result)

Some insights into Bellcore attack – RSA & CRT

- **Signing process of a message (assuming the whole message is used to sign)**
  - **p, q: two large prime numbers, $n = p*q$**
  - **d, e: two numbers in Z/nZ relatively prime to n and verifying $d*e = 1 \bmod \Phi(n)$, where $\Phi(n) = (p-1)*(q-1)$ is Euler function**
  - **Signing s is computed by $s = m^d \bmod n$**

- **Modified computation with CRT algorithm**
  - **d_p, d_q defined by $d_p = d \bmod (p-1)$ and $d_q = d \bmod q-1$**
  - **s_p, s_q computed with $s_p = m^{dp} \bmod p$ and $s_q = m^{dq} \bmod q$**
  - **a, b defined by $a = 1 \bmod p$, $a = 0 \bmod q$ and $b = 0 \bmod p$, $b = 1 \bmod q$**
  - **Final signing: $s = s_p*a + s_q*b$**

Some insights into Bellcore attack – DFA

- One single fault/error induction => computation of s_p (respectively s_q) is wrong

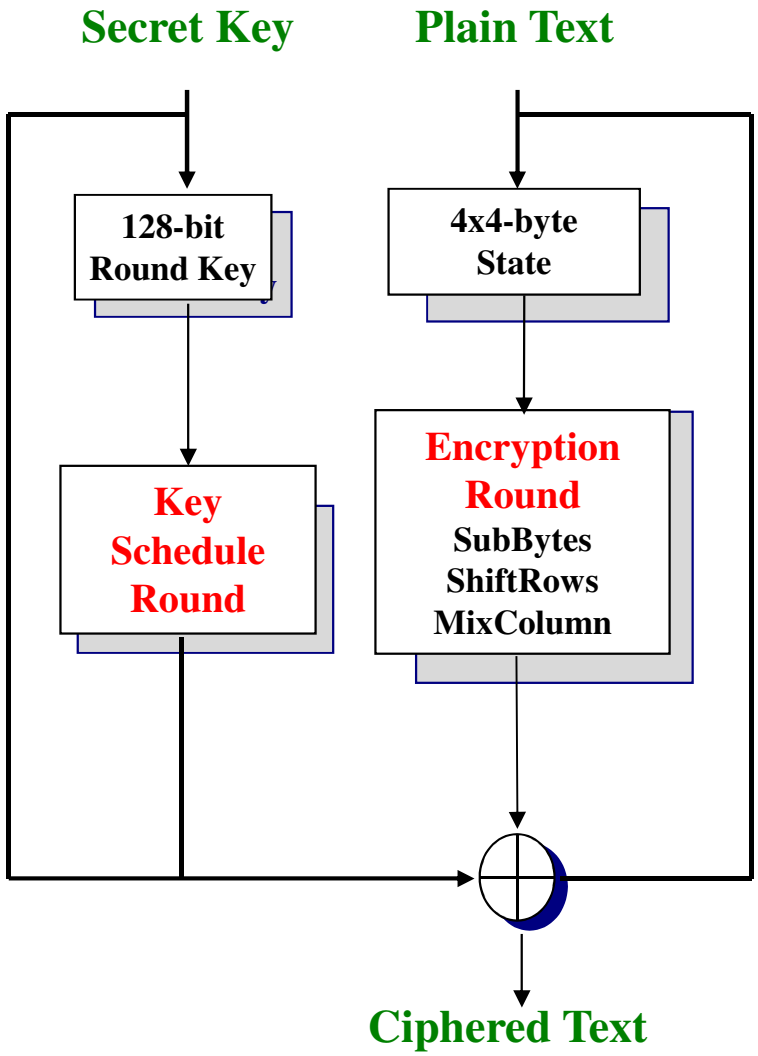
  => s_p' (respectively s_q') and wrong signature s' defined by $s' = s_p' * a + s_q * b$ (respectively $s' = s_p * a + s_q' * b$)

- $\Delta = s - s' \Rightarrow \Delta = a*(s_p - s_p')$ (respectively $\Delta = b*(s_q - s_q')$)

- Given that $a = 0 \bmod q$ (respectively $b = 0 \bmod p$), Δ is a multiple of q (respectively p), and a simple calculation of a Greatest Common Divisor (GCD) will factor n:
  $GCD(\Delta, n) = q$ (respectively $GCD(\Delta, n) = p$)

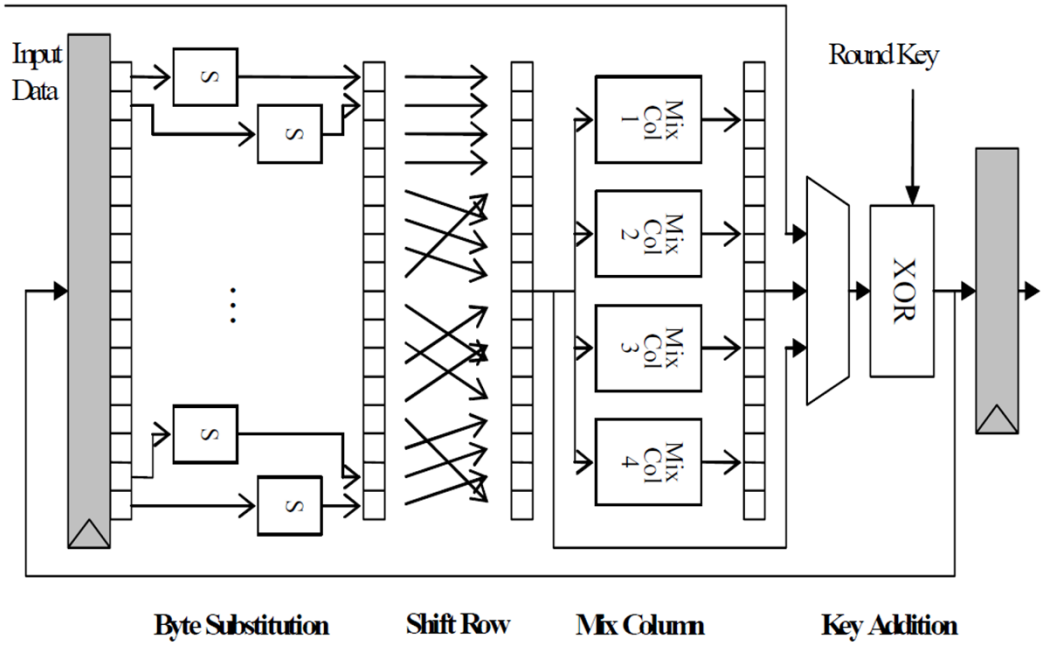
Advanced Encryption Standard

- Symmetric block cipher
  - Plain Text 128b,
  - Secret Key 128/192/256b
- SPN cipher
  - 10/12/14 rounds
- Round operations
  - SubBytes: nonlinear byte substitution
  - ShiftRows: row-wise word rotation
  - MixColumns: column-wise linear multiplication
  - Key Addition: XOR with round subkey

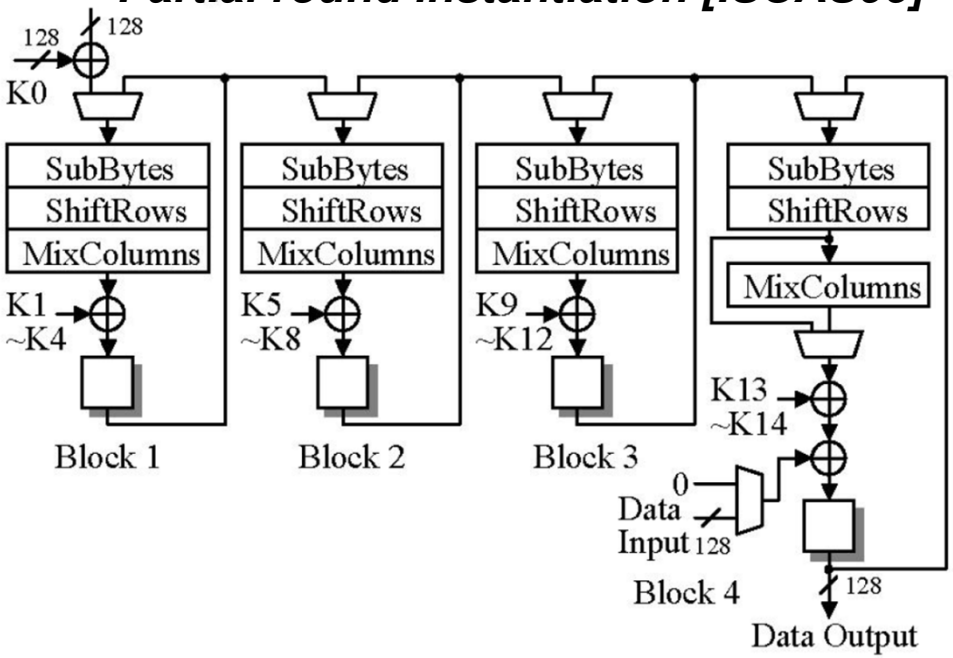


AES Hardware Implementations

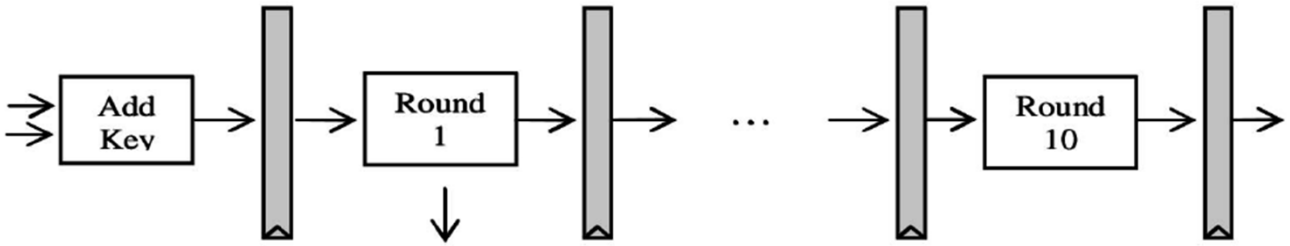
Round-iterated designs [GLSVLSI05]



Partial round instantiation [ISCAS06]

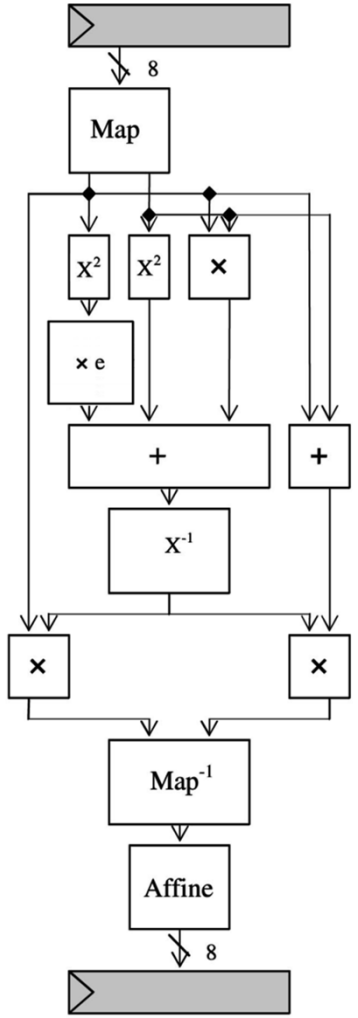


Fully unrolled architectures [TC 55/4]



S-Box Implementations

- Look-up table
  - 256B ROM
  - Combinational logic network
- Composite field
  - Mapping from $GF(2^8)$ to $GF(2^4)^2$
  - Smaller than LUT,
  - but slower
  - Several possible mappings
  - Easy pipeline



Fault Attacks on AES

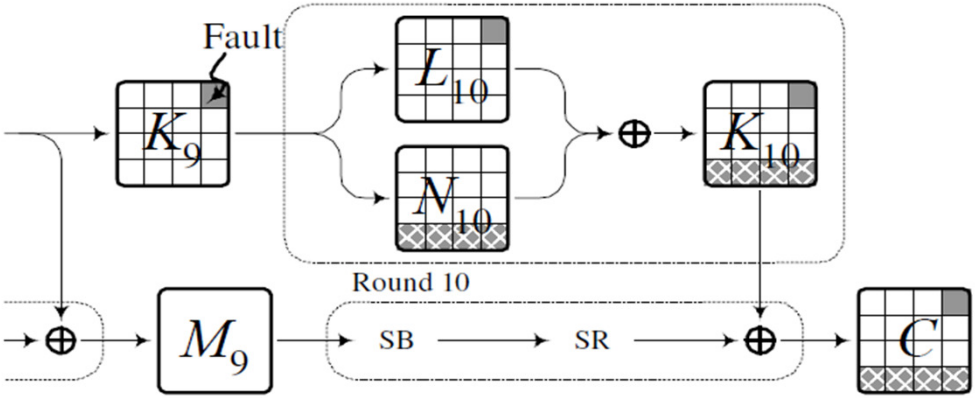
- C-safe error
  - Specific stuck-at injected at probabilistic location (byte-level)
  - Corrupted output → Error injected → Known value (~90k injections)

- Exploit S-Box & MixColumn scaling (Dusart et al, ACNS 2003)

$$s(x) + s(x + c \cdot \varepsilon) = \varepsilon$$

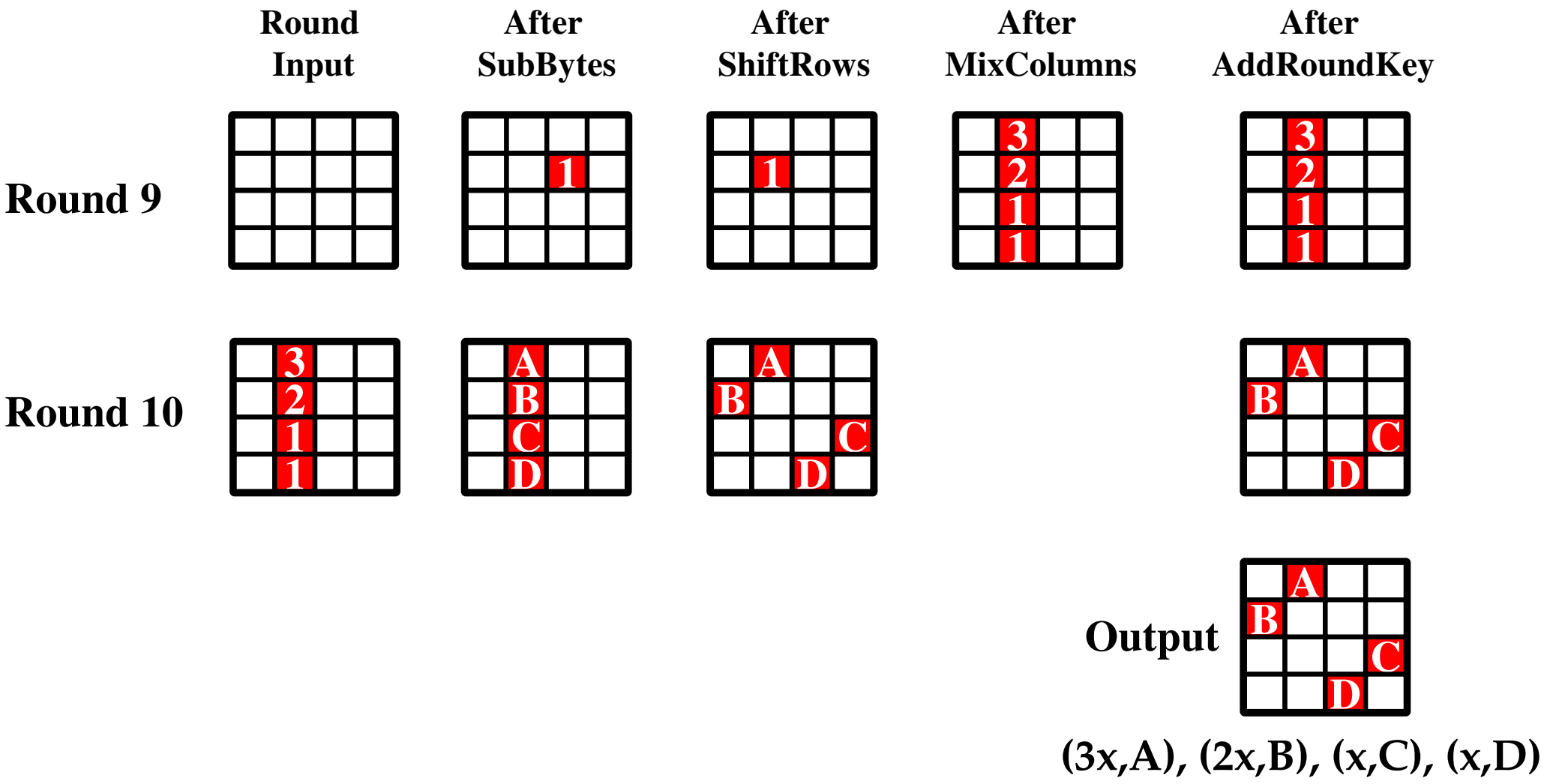
  - 50 faulty ciphertexts to recover a full 128-bit key

- Injection in last round key
  - Chen and Yen
  - About 44 fault injections

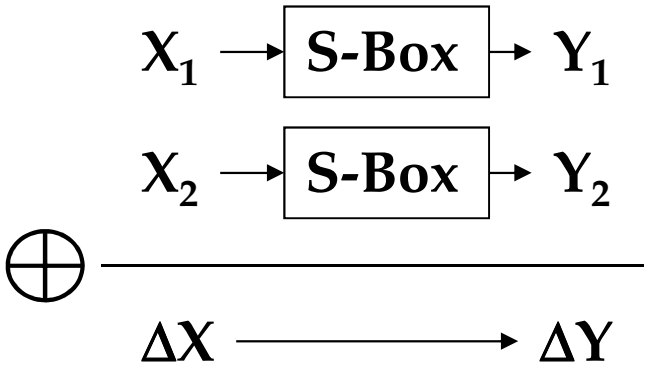


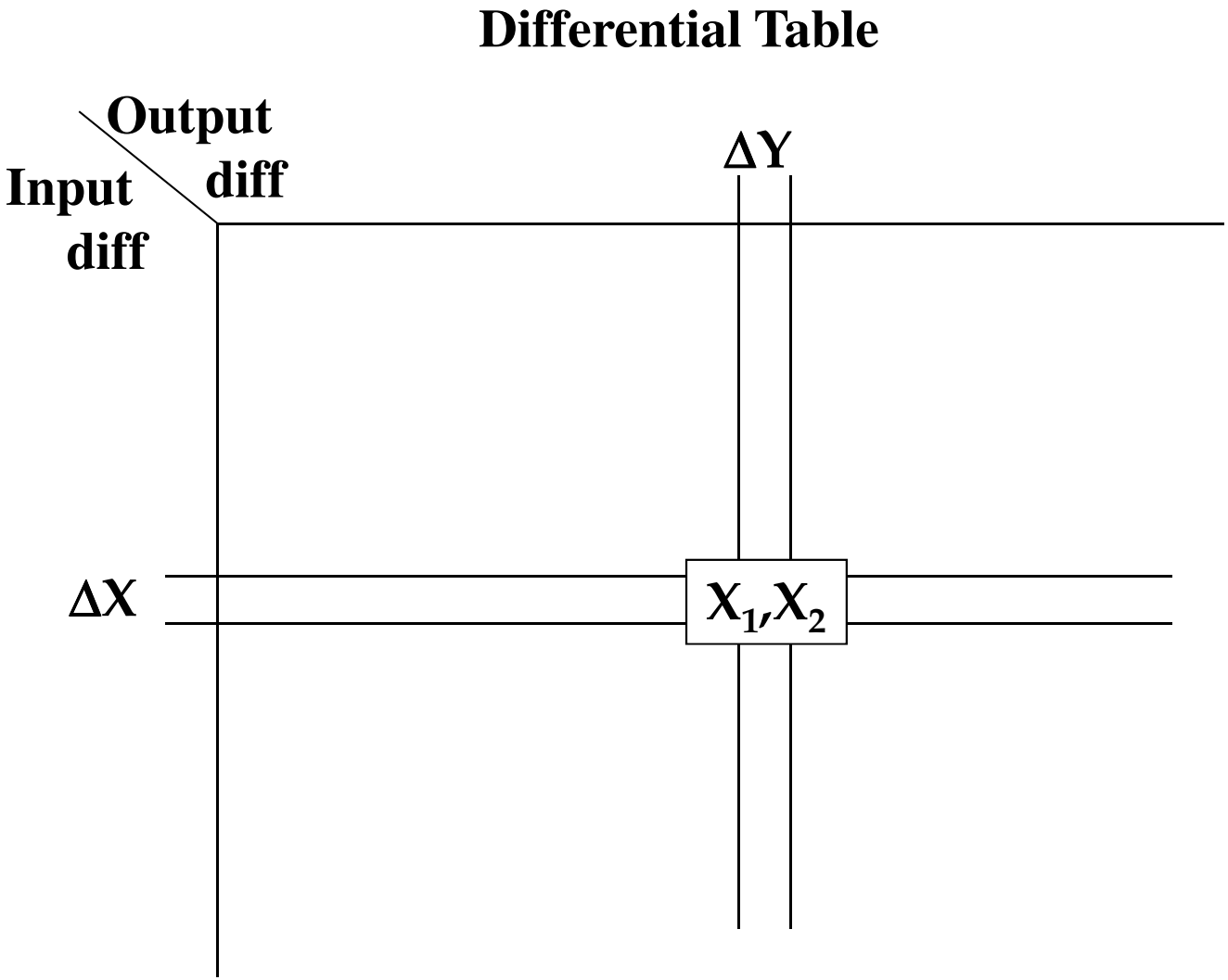
- Differential Fault Analysis…
  - Piret & Quisquater, [CHES 2003], see next slides

DFA on AES – Fault Propagation

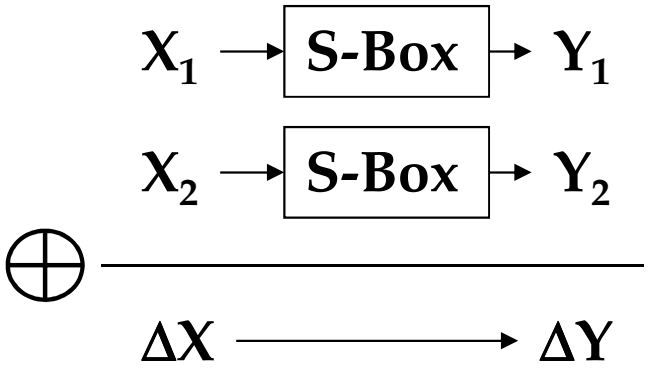


DFA on AES – Differential Fault Space

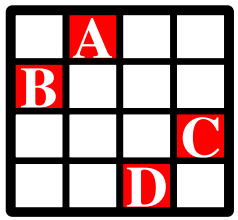




DFA on AES – Finding the Candidates



Output differences



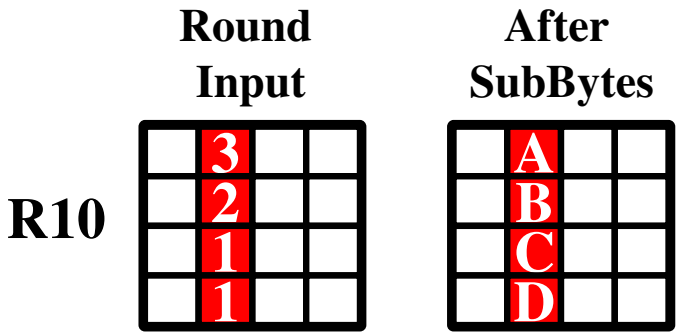
(3x,A), (2x,B), (x,C), (x,D)

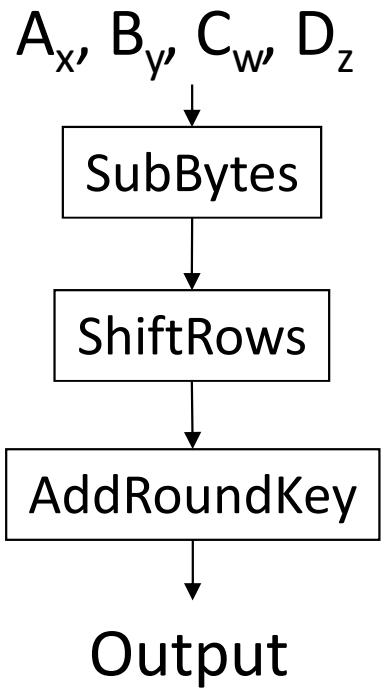
Differential Table

Input diff \ Output diff	C	B	D	A
x	C_1, C_2		D_1, D_2	
3x				A_1, A_2
2x		B_1, B_2		

DFA on AES – Finding the Key

- **For every possible error value**
  - Add candidates to list
  - Discard on at least one empty cell
- **Perform another fault injection**
  - Build a second set of candidates
- **Intersect the two lists**
  - Shared values are last S-Box inputs
- **Find last round key**
- **Compute reverse key schedule**
  - and find the secret key!





When is the attack successful ?

- When the fault is injected and the computation is corrupted ?

  Not yet !

- When the erroneous result is obtained ?

  Yes !

- So the erroneous result must
  - Either be hidden (but may give useful indications to the hacker)
  - Or be corrected
  - … Or the hacker may be misled (erroneous result replaced by another that cannot be exploited)

  - Note: delays induced by detection + recovery => information to the hacker

Use of induced errors: DFA, but not only !

- Round reduction, RNG output forcing => easy cryptanalysis
- PIN counter corruption => unlimited trials
- Hardware or software protection bypass …
- Behavior analysis ("safe-error" attack): if the attack effect is controlled, e.g. stuck-at-0 bit, a normal or abnormal behavior indicates detection or not of the attack, thus the initial value of the attacked bit (even without output result to analyze).
- Direct hacking: counter modifications (e.g. money in e-purse)

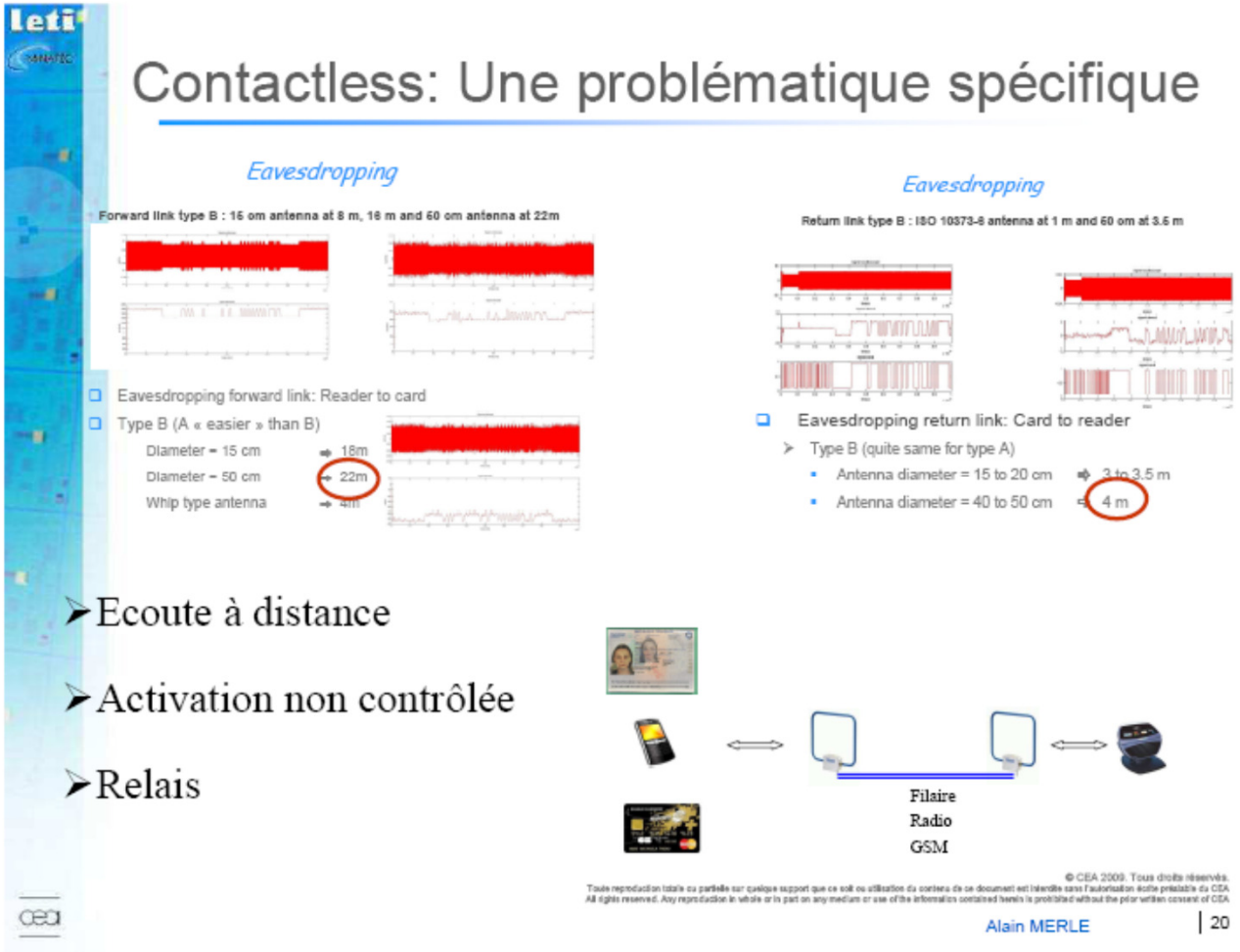
Induced errors – logic point of view

- The error type does not really depend on the physical perturbation technique => soft errors
  - Direct induction in memory elements: SEUs/MBUs
  - Induction in combinatorial logic: SET then propagation
  - Final effect: erroneous bit(s) in register(s)
- Multiplicity depends on the source and fault location
- Error models assumed in published attack schemes are not always realistic (w.r.t. current fault induction techniques) – e.g. one single particular bit forced at a given value at a given cycle during the computation …

A special case: reconfigurable hardware

- In case of SRAM-based or Flash-based FPGA
- Specific possible attack by "downgrade", i.e. replacing the FPGA configuration by an older one (correctly encrypted and with correct integrity checks) => may maintain functionality but remove security "patches" correcting previously identified security holes
- Need for accurate versioning !!

"New" concerns: wireless/long distance attacks



A. Merle, Presentation to the GdR SoC-SiP Security working day, Nov. 16, 2009

Other example: "long" distance recording of keyboard activations (creating carrier signal modulation)

RFID and security

- Smart phones such as Apple's iPhone and Google's Android are significant (unprotected) targets … (personal data, zombies …)

- … but not only:
  - One of the most common attacks on wireless networks is "war driving," in which hackers drive around a neighborhood, hunting for unsecured wireless nodes. In the latest twist on war driving, a security expert armed with a cheap RFID scanner and low-profile antenna managed to clone half a dozen electronic passports in an hour while cruising around Fisherman's Wharf in San Francisco.
  - The researcher who conducted this experiment asserts that the attempt at "war cloning" was successful because the type of RFID in the Homeland Security version of a passport emits a real radio signal, which could conceivably be tracked from a couple of miles away. Although no criminal hacks of passports or e-licenses have been detected to date, this insecure technology poses a strong risk for identity theft and invasion of privacy.[7]
  - 7. Kelly Jackson Higgins, "Drive-By 'War Cloning' Attack Hacks Electronic Passports, Driver's Licenses," Dark Reading, February 2, 2009, www.darkreading.com/security/privacy/showArticle.jhtml?articleID=213000321

http://embedded-computing.com/current-trends-cyber-attacks-mobile-embedded-systems

Protecting Secure Circuits

- "Classical" protection styles (but adapted), especially against faults/errors

- Specific protections (active)
  - Sensors (voltage levels, glitch, light, temperature, …)
  - On-chip encryption of data
  - …

Multi-level:
- Software
- OS
- Hardware

- Specific protections (design methodology)
  - Internal clock generation (problem with testing requirements !)
  - Restrictions in design styles (dynamic logic, …)
  - Shielding
  - Memory scrambling
  - …

Towards a dependable system (w.r.t. errors)

- Many approaches/techniques, a few major categories
  - Spatial redundancy, replication (massive redundancy)
  - Information redundancy (coding)
  - Timing redundancy

  +
  - Assertion-based on-line checking
  - Control-flow checking
  - etc. …

  } Hardware and/or software

- Choice depends on application constraints (overheads, …) and on dependability specification
  - Detection only
  - Tolerance by detection + recovery (retry, checkpointing, …)
  - Tolerance by masking (voting) … taking care of fault accumulation …

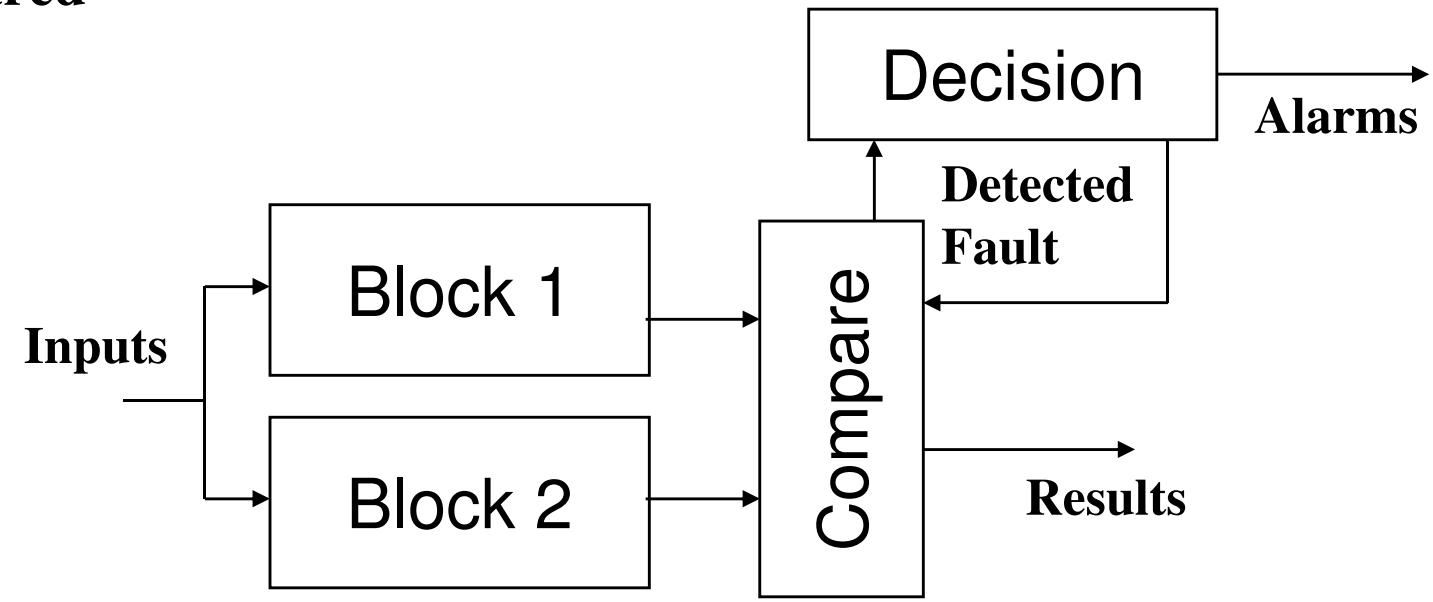
Multi-level counter-measures (FA)

Level	Counter-measures against FA (examples)
Gate (transistor level structure)	Various sensors, robust structures against SEUs
P&R	Coupling reduction
RTL	**Error detection codes: limitations**, timing redundancy
Algorithm	Precautions, e.g. CRT use for RSA
Architecture	Function fusions to mask intermediate results
System	Depends on security policy: memory erasing, process or OS-level error recovery

Hardware Redundancy

Basic Redundancy

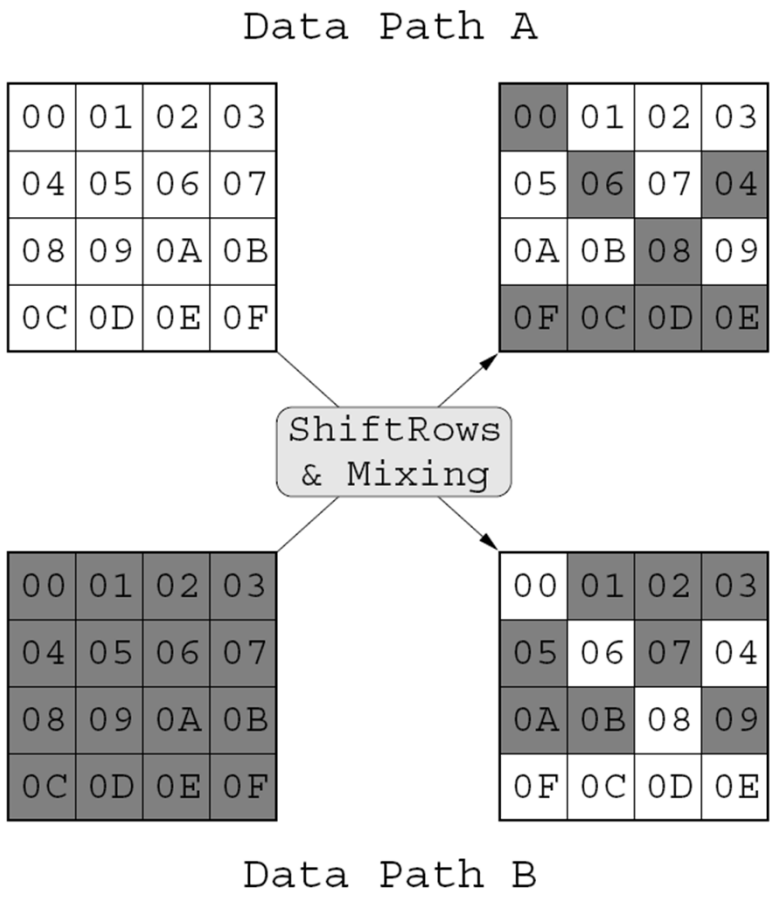
- Multiple [different] functional blocks are implemented and outputs are compared



- ☺ Effective against transient or destructive faults
- ☹ Overhead >100% in area and power
- ☹ Even greater correlation between data and power consumption
  - Easier DPA !!!

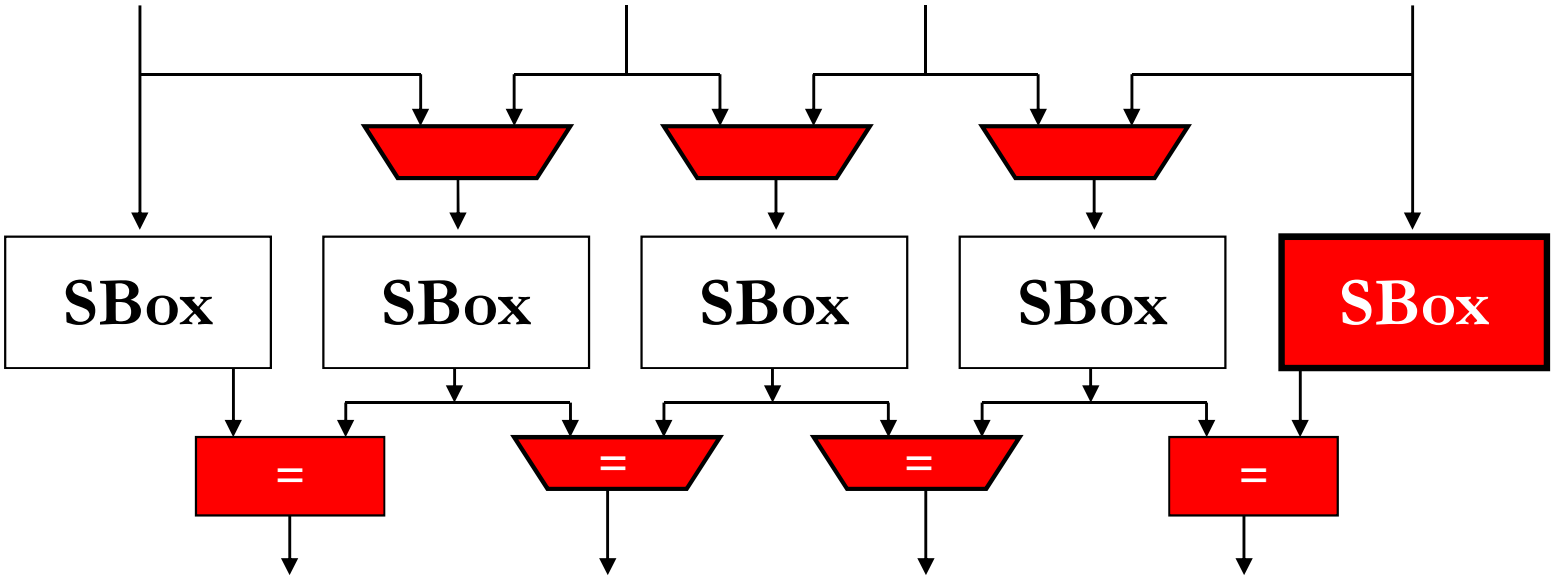
Redundancy and Diffusion

- Hardware redundancy for fault detection
- Data contamination to avoid fault exploitation
- Same advantages and same problems than simple duplication
  - Greater security if final validation step is compromised



[Joye, Manet, Rigaud. IET Inf Sec 2007]

Partial Redundancy (1/2)



- Dynamic comparison with additional component

☺ Limited overhead and good detection capability of permanent faults

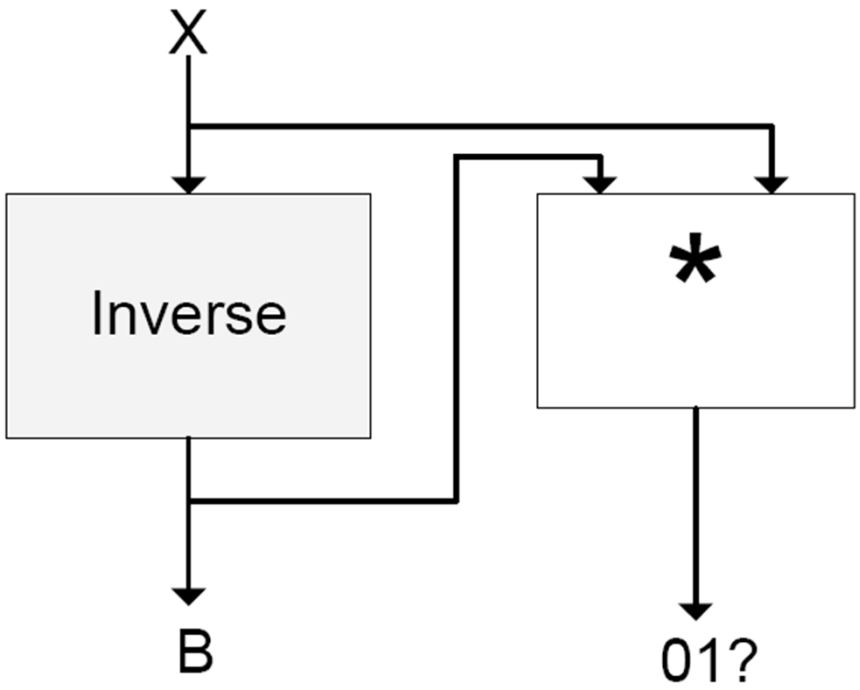
☹ Limited detection capability of transient faults

[Di Natale, Flottes, Rouzeyre. WDSN 2007]

Partial Redundancy (2/2)

- **AES S-Box:**
  - **Most complex and expensive component**
  - **Non linear**
  - **Based on computation of the multiplicative inverse**
- **Detection based on the computation of the multiplicative inverse**
  - **Add a GF multiplier**
  - **Configurable width**

☹ **Coverage limited to S-Box**
☹ **Coverage dependent on redundancy and overhead**
☹ **Linear code required for the rest of the circuit**



[Kulikowski, Karpovsky, Taubin. CARDIS 2004]

Hardware Redundancy - Summary

- Two (or more) independent functional blocks are used and the outputs are compared
  - Protections can be limited to a specific unit
  - Data paths can be mixed
- Effective against transient and permanent faults
- **Very expensive in terms of area**
- No throughput reduction
- Particularly vulnerable to side-channel analysis

Information Redundancy

Using error detecting codes (EDCs)

- May be an answer to fault-based attack detection, before security policy activation
- Choice of code: efficiency (w.r.t. a characterized threat) vs. overheads
- Algorithm impact: codes may be or not well suited (complexity of code prediction vs. number of checkers)
  - Identify the common components in (symmetric) block ciphers
  - Associate EDCs to data block and develop a code prediction rule for (possibly) each operation
  - Evaluate the suitability of a code to the whole cipher (i.e., overhead and error coverage)

A few following slides from L. Breveglieri, I. Koren, P. Maistri, "Detecting Faults in Integer and Finite Field Arithmetic Operations for Cryptography", FDTC: workshop on Fault Diagnosis and Tolerance in Cryptography, Supplemental Volume Proc. of the Conference on Dependable Systems and Networks, pages 361-367, June 2004

Why prediction rules ?

- **High coverage with low-order errors**
  - **Codes often provide 100% coverage of single bit errors**
- **With high-order errors, coverage depends on redundancy**
  - **Output code and data match randomly**
- **Expected hardware overhead smaller than duplication**
  - **EDCs need a code *generator*, a (single) *comparator* and *propagation units* implementing the prediction rules**
- **EDCs are cheaper when simple prediction rules are available for the whole encryption process**
  - **The check bits are generated at the beginning and validated at the end of the process**
  - **Checkpoint frequency can be increased for higher coverage**

Operations in symmetric ciphers

Ciphers	XOR	AND,OR	+, -	×	Sbox	Rot	Shift	Perm	× mod G(x)
Camellia	✓	✓			✓	✓		✓	
DES	✓				✓			✓	
IDEA	✓		✓					✓	
MARS	✓		✓	✓	✓	✓		✓	
RC5	✓		✓			✓			
RC6	✓		✓			✓		✓	
Rijndael	✓				✓			✓	✓
Serpent	✓				✓	✓	✓		
Twofish	✓		✓		✓	✓		✓	✓

- XOR: Every cipher
- AND, OR: Camellia only
- +: often used
  - -: in encryption, only MARS
- x: slow and area-consuming
  - IDEA uses uncommon modulus
- Rotations: even data-dependent
- Shift: Serpent only
- Permutation: provides confusion
- Polynomial x: Rijndael and TwoFish, over GF(2^8)
- S-Box: non-linear

EDC granularity

- Symmetric ciphers operate on different word size (8, 16, 32 bits)
- Code granularity should not be larger than operand size
  - The code should be validated and regenerated with each operation (e.g. substitution tables)
- Finer code adds further complexity and overhead
  - Detection rate improves
  - Prediction rule may become more complex

Matching EDCs to operations (1/3)

- Parity is more suited to logical operations; the prediction rules are...
  - eXclusive OR: ...the XOR of the input parities
  - Rotation: ...parity is unchanged, if the code is at the same level of the operation
  - Shifting: ...must consider bits leaving and entering the word
  - Polynomial multiplication: …if defined over GF(2^n), it is easily predictable when one of the operands is known a priori
  - Data-dependent operations (RC5, RC6) are obviously more complex
- Addition and multiplication must consider all the carries that are required to compute the result

Matching EDCs to operations (2/3)

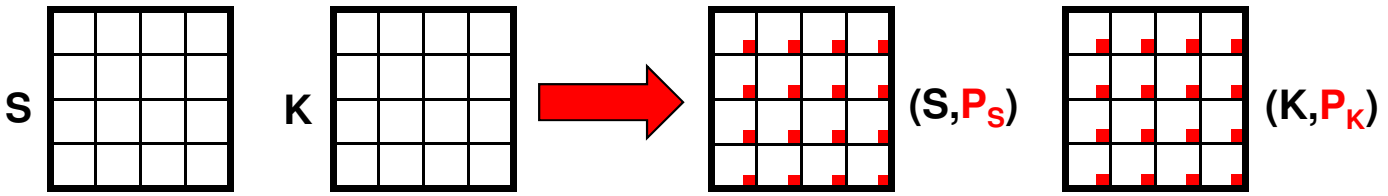
- Residues are more suited to arithmetic operations; the prediction rules are...
  - Addition: ...the sum of the input residues
    - but overflow needs correction!
  - Multiplication: ...the product of the input residues
    - but most significant (and neglected) bits need a corrective term
  - Shifting: ...it can be seen as a multiplication by a power of 2
  - eXclusive OR: …the sum of the input residues, but a correction term is needed

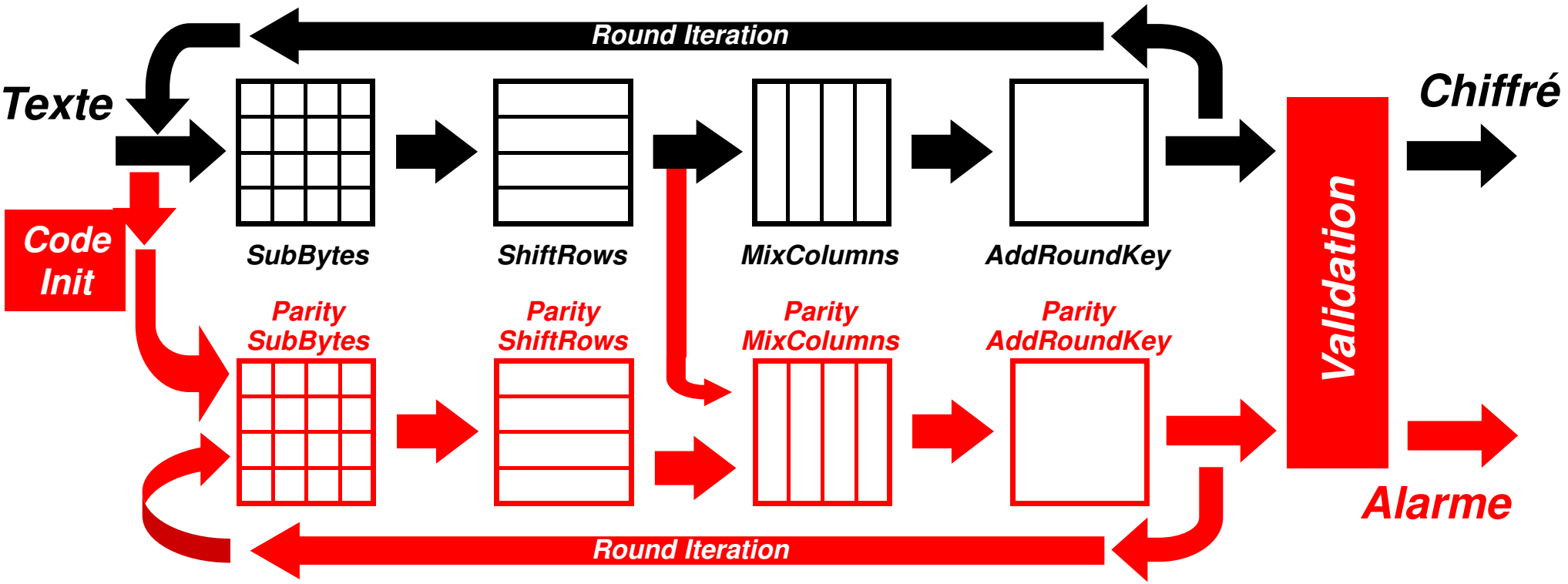
- Prediction of the code after polynomial multiplication over GF(2^n) is expensive

Matching EDCs to operations (3/3)

- Some operations are not suited to parity codes:
  - Logical AND and OR: prediction is much more expensive than duplication
  - Validate the code, protect by duplication and generate the code from scratch
- Some operations MAY BE suited both to residue and parity:
  - Substitution boxes: the output code is stored together with the result (if ROM-based or constant table impelmentation); the input code is used for implicit validation
  - Address protection by concatenating check bits introduces a large overhead (1 additional bit doubles the table size)
    - Use custom address decoding unit to reduce the area overhead

An example: Multiple parity for AES





[Bertoni et al. TC 2003]

Choosing the proper EDC

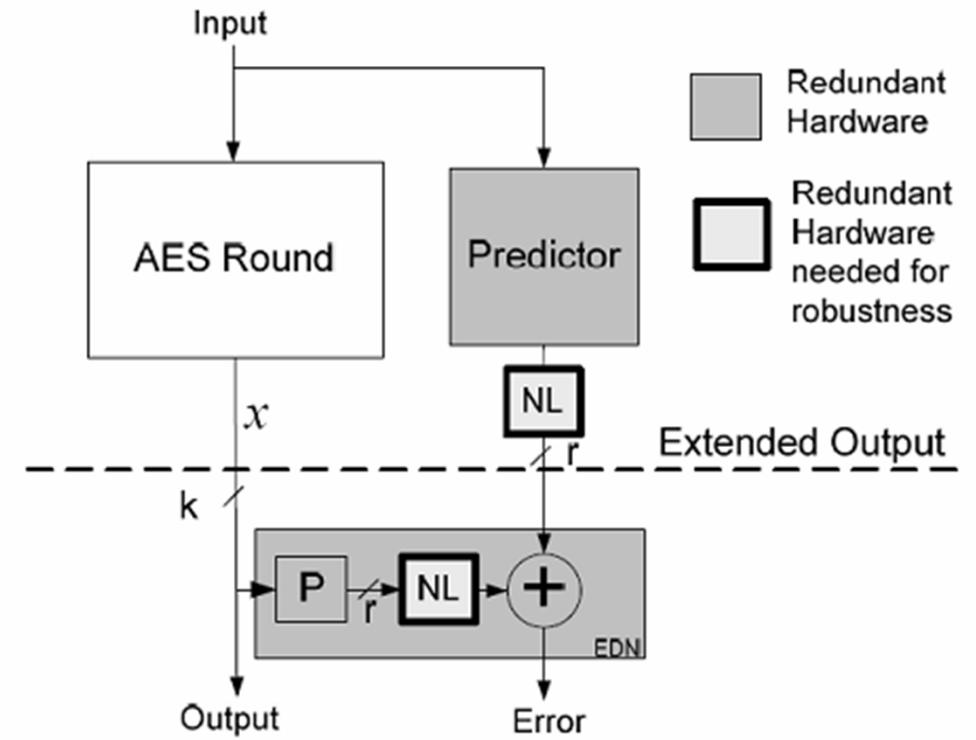
Cipher	$\oplus$	$\wedge,\vee$	$+,-$	$\times$	Sbox	Rot	Sh	Perm	$\times$ mod G(x)
Camellia	✓	✓			✓	✓		✓	
DES	✓				✓			✓	
IDEA	✓		✓	✓				✓	
MARS	✓		✓	✓	✓	✓		✓	
RC5	✓		✓			✓		✓	
RC6	✓		✓	✓		✓		✓	
Rijndael	✓				✓			✓	✓
Serpent	✓				✓	✓	✓		
Twofish	✓		✓		✓	✓		✓	✓

Cipher	Suggested code	Cipher	Suggested code
AES	Parity, per byte	RC5	Parity or residue
DES	Parity	RC6	Residue
IDEA	Residue, but expensive	Serpent	Parity, per byte
MARS	Residue, but expensive	Twofish	Parity, per byte

Nonlinear Codes

- Detection by linear codes depends only on error value
  - Potential vulnerability
- Detection by **non**linear codes depends on error and **data** values
  - More uniform detection rate
  - Attacker cannot choose a specific error value to inject

☺ Very high detection rate
☺ Very difficult injection of controlled faults
☹ Significant overhead (+70%)



Theme Variations

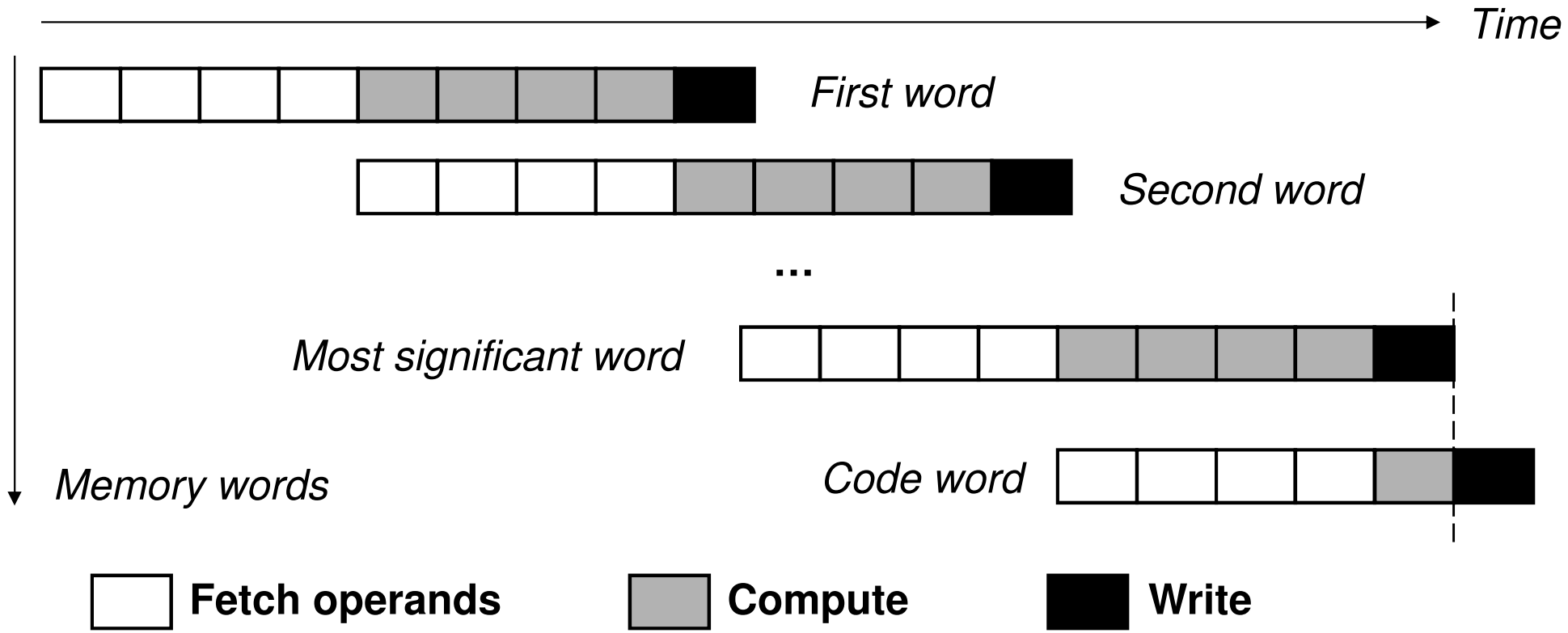
Authors	Publication	Notes
Yen, Wu	TC 2006	AES data path protected by CRC redundancy code
Kermani, Masoleh	DFT 2006	Parity prediction of S-box with GF decomposition
Cota et al.	ISCAS 2008	Hamming and Reed Solomon codes
Di Natale et al.	IOLTS 2007	Parity prediction of S-box taking into account both input and output
Kermani, Masoleh	CHES 2008	Parity prediction of S-box with decomposition with normal basis
…		

Public-Key Cryptosystems

- Fault attacks may affect also asymmetric cryptosystems
  - CRT-RSA can be broken with just one fault
- Verification of encrypted message needs a new encryption process (exponentiation)
  - Too slow
  - It must be done within the chip: no software solution, because the faulty output cannot be made available!
- Asymmetric systems are based on modular arithmetic
  - Computation errors can be easily identified by means of EDCs
  - Nonlinear codes can also be used [Gaubatz et al., FDTC 2006]

An example: RSA

- *Detection and correction of transient errors in a hardware implementation of the RSA cryptosystem [...] can be done efficiently and reliably with acceptable time and area costs equivalent to an increase in the size of the modulus by one digit or less* **[C. Walter, CHES 2000]**



Timing Redundancy

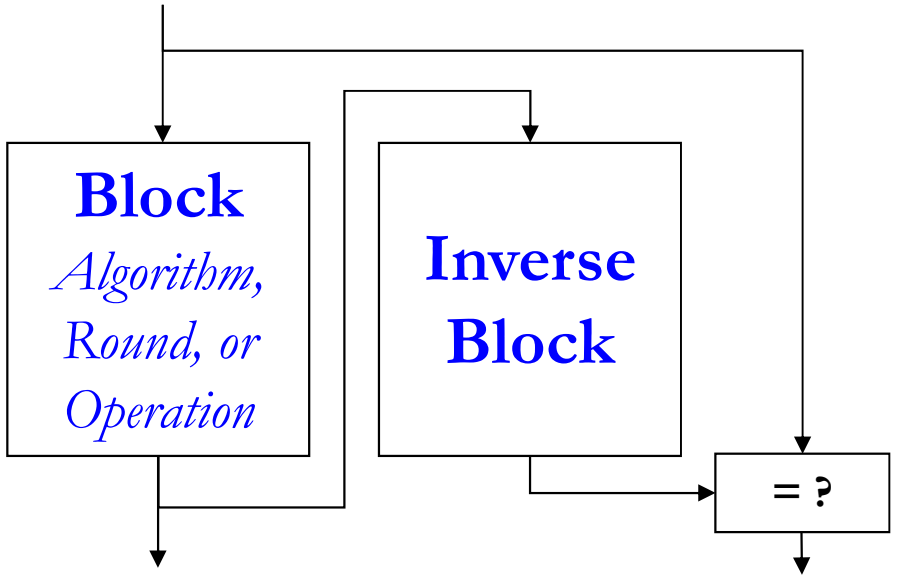
Inverse Computation

- **Based on the different paths for encryption or decryption**
- **3 levels of granularity: algorithm, round, operation**
  - **Overhead, throughput reduction and detection latency**
- **Inverse function is computed and the result is compared to the initial value**

😐 Decryption needed

☺ Transient and permanent faults

☹ Latency and overheads



[Karri et al. DAC 2001]

Involution Ciphers

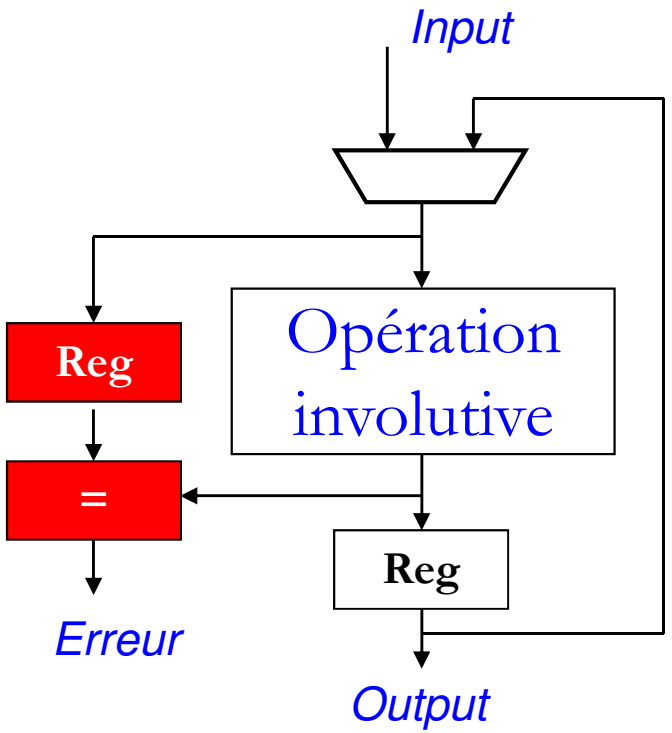
- **Involution cipher: encryption and decryption functions are the same**
- **Same idea than inverse computation…**

☺ Decryption logic is already there

☺ Transient and permanent faults

☹ 100 % computation time overhead

It may be improved by pipelining…

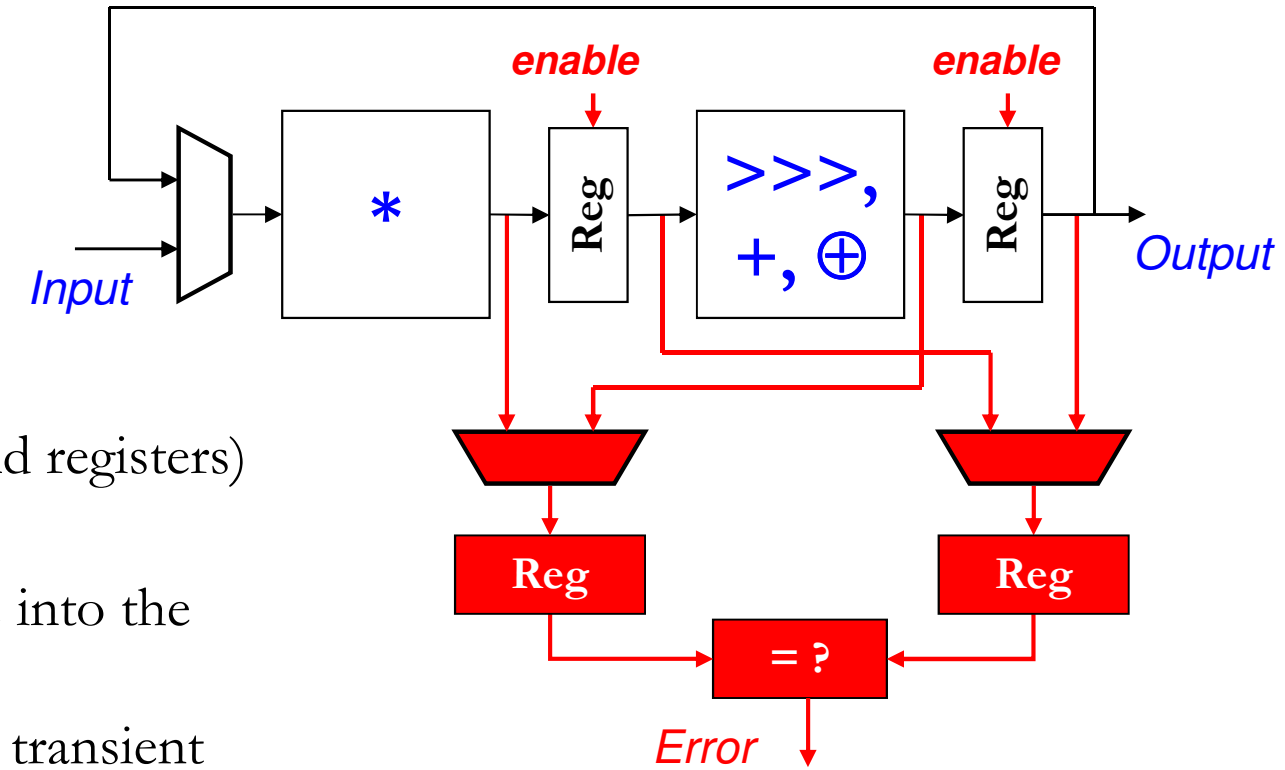


[Joshi et al., CHES 2004]

Pipeline Redundancy

- Unused pipeline stages are used to redo the same computation twice

Architecture RC6

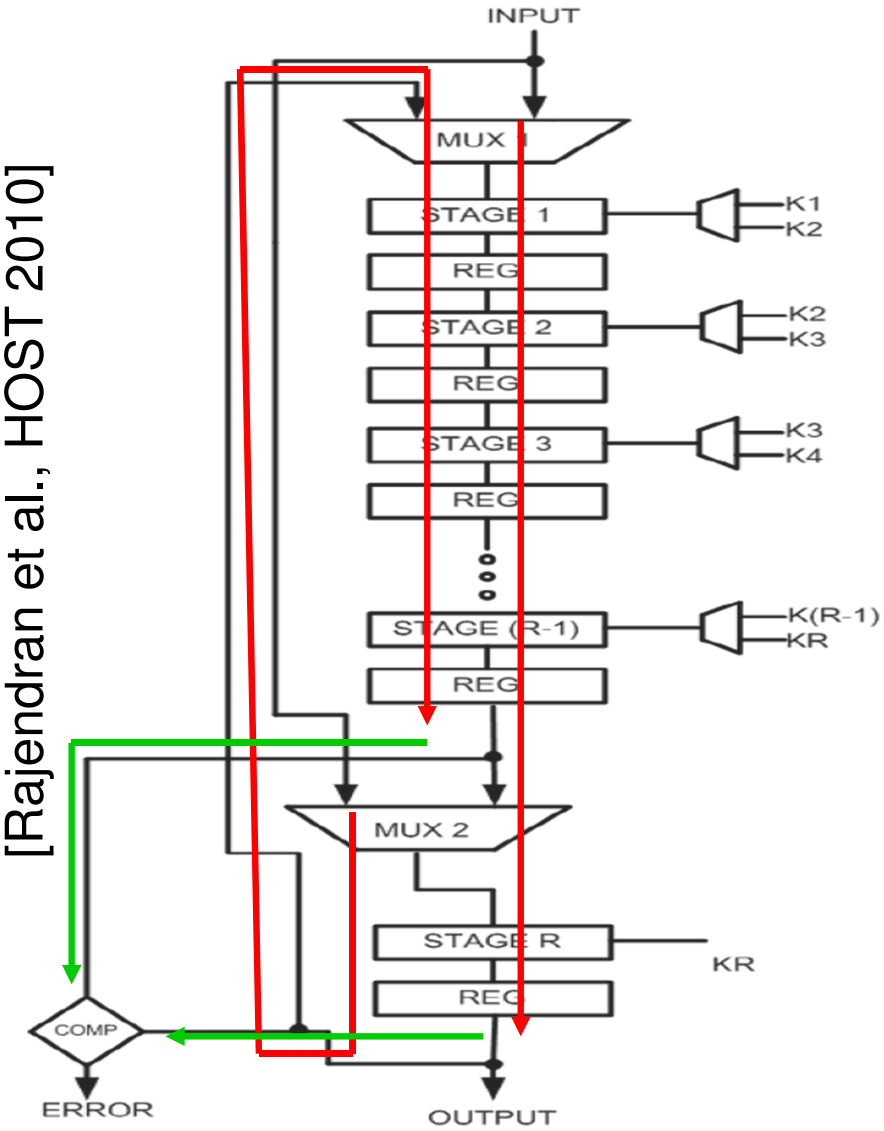


- ☺ Detection of transient faults
- ☺ Limited overhead (control and registers)
- ☺ Reduced detection latency
- ☺ Easy to integrate the pipeline into the design flow
- ☹ Neither permanent, nor long transient faults (>1 cycle)
- ☹ Partially unused pipeline is needed

[Wu, Karri. DFT 2001]

SLICED

[Rajendran et al., HOST 2010]



Stage / Cycle	1	2	3	4	5	6
1	R(1,1)					R(1,1)
2	R(2,1)	R(2,1)				
3	R(1,2)	R(3,1)	R(3,1)			R(1,2)
4	R(2,2)	R(2,2)	R(4,1)	R(4,1)		
5	R(1,3)	R(3,2)	R(3,2)	R(5,1)	R(5,1)	R(1,3)
6	R(2,3)	R(2,3)	R(4,2)	R(4,2)	R(5,1)	R(6,1)
7	R(1,4)	R(3,3)	R(3,3)	R(5,2)	R(5,2)	R(1,4)
8	R(2,4)	R(2,4)	R(4,3)	R(4,3)	R(6,2)	R(6,2)
9	R(1,5)	R(3,4)	R(3,4)	R(5,3)	R(5,3)	R(1,5)
10	R(2,5)	R(2,5)	R(4,4)	R(4,4)	R(6,3)	R(6,3)

- ☺ Transient faults
- ☺ Simple permanent faults
- ☺ Limited overhead (multiplexers)
- ☹ Frequency and throughput reduction
- ☹ Iterative unrolled implementation needed

What was not so good so far…

- Error detection codes for AES are either expensive (non-linear networks) or inefficient against malicious faults (parity)
- Spatial/information redundancy may increase correlation with power consumption and EM emissions, thus favoring side-channel attacks
- Temporal redundancy:
  - Process repetition involves high performance overhead
  - Pipeline implementation requires fast system clock and significant area overhead (+50%), but …
  - … the global system may work at reduced frequency, thus affecting the global throughput

Inverse Computation with Pipeline

- Inverse computation combined with pipeline redundancy

☺ Detection of transient and permanent faults

☺ Limited overhead

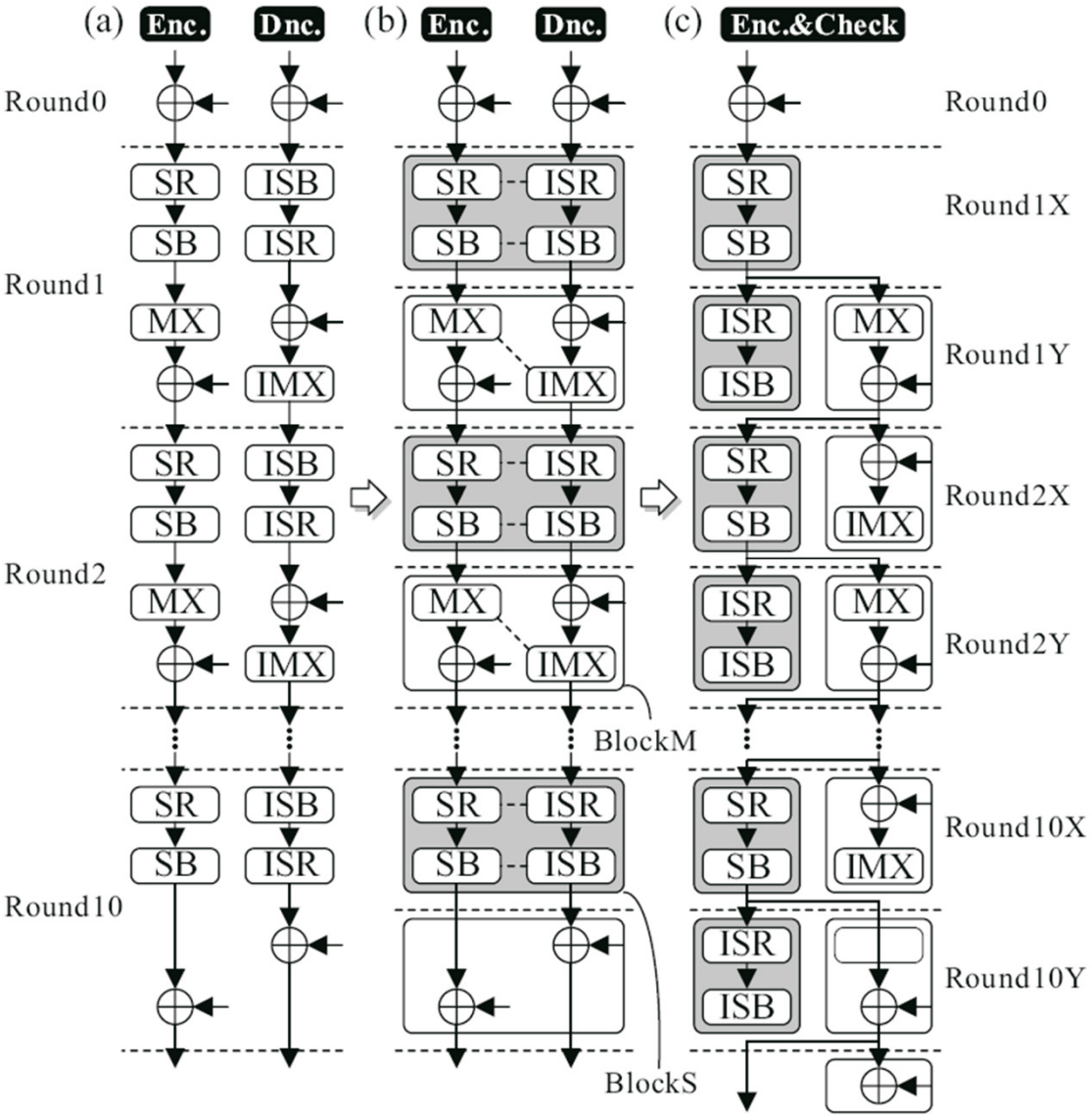
☺ Low detection latency

😐 Quite easy to integrate into the design flow

😐 Decryption logic needed

😐 Pipeline needed

😐 Adaptability to other ciphers to be verified



[Satoh et al. CHES 2008]

Part III

Protection limitations
Ex. DFA vs. DPA on registers

Actually protecting a circuit

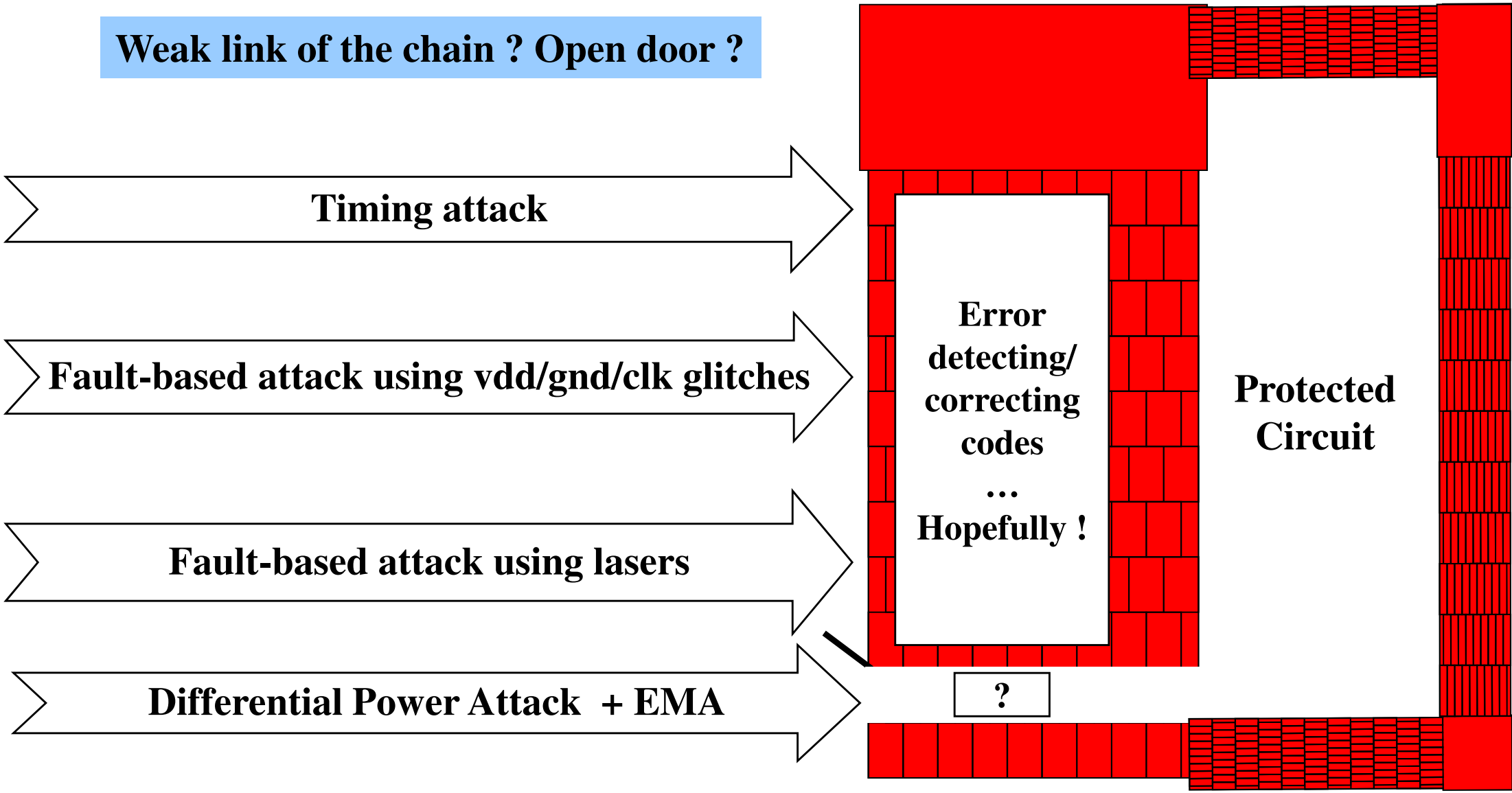
- Counter-measures usually proposed to protect against one type of attacks: Is it sufficient?
- Example of evaluation practice against faults:
  - Theoretical computation of the fault coverage
  - Simulation/emulation campaigns of fault injections
  - What about the impact of the added countermeasures on other types of attacks ?
- Issue: evaluate the global security of the protected device
  - Against Faults
  - Against Side Channels
  - Against …

Using error correcting/detecting codes

- One of the typical methods of protection
  - Well established theory – many codes available, protection efficiency against errors is theoretically well known
  - Timing attack: no extra sensitivity due to protection in synchronous circuits (at least for many classical codes)
- Concerns
  - Error assumptions – choice of code, and consequences on implementation costs
  - Power attacks: effect of the added checking bits ?
  - Electromagnetic attacks: derived from the power consumption

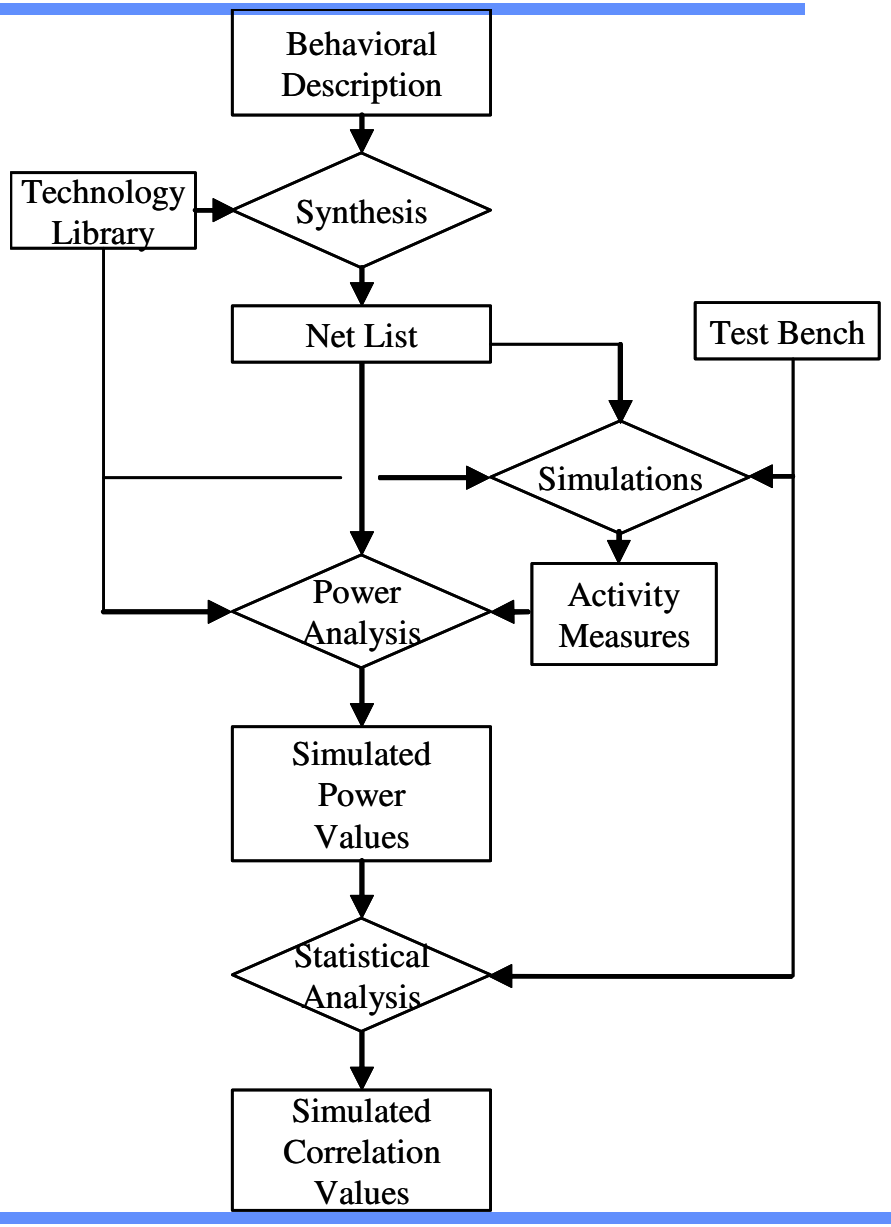
Error detection: secured circuit ?

Weak link of the chain ? Open door ?



Power attack - sensitivity analysis flow

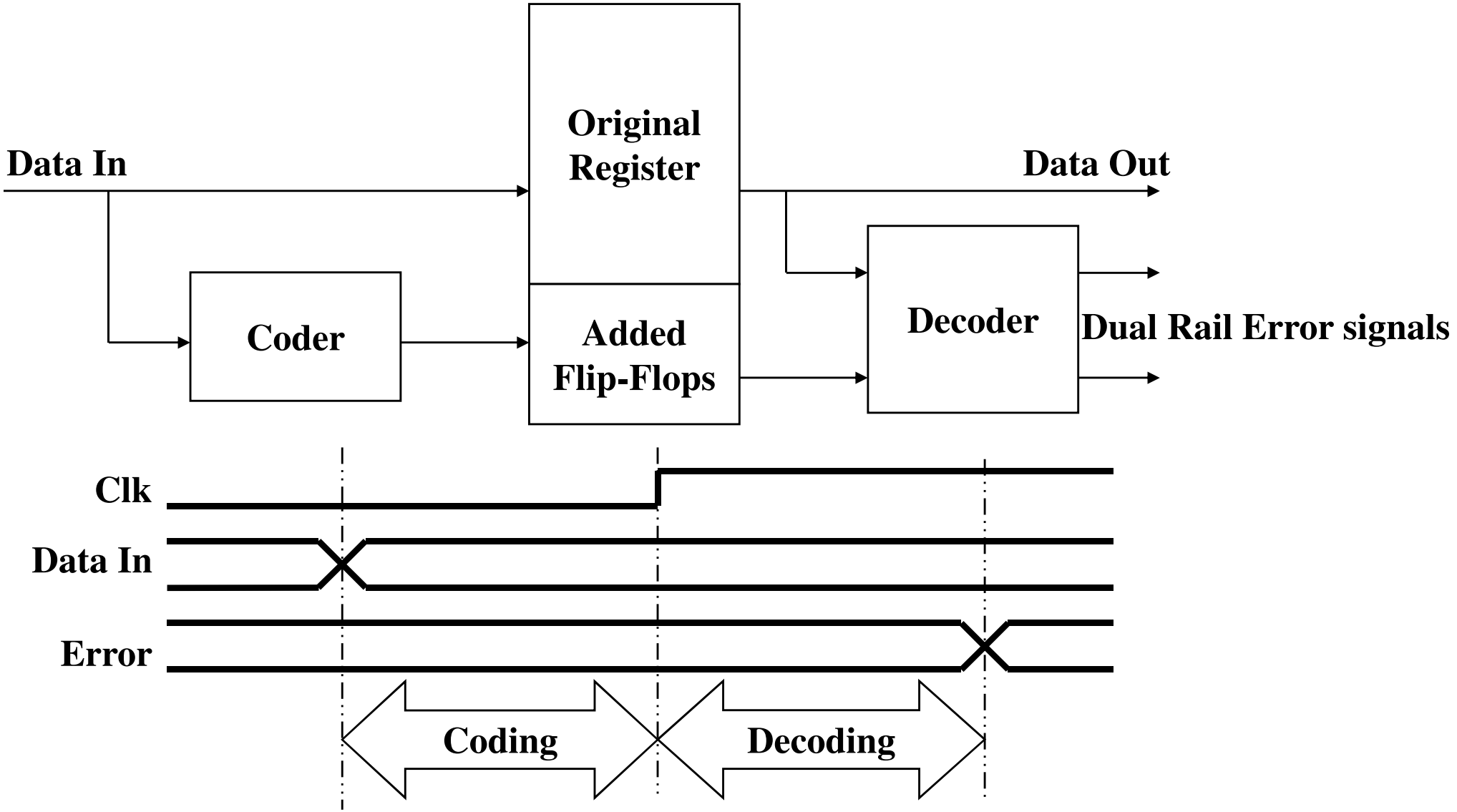
- **Activity measures**
  - **Transistor-level: precision**
  - **Gate-level: simulation time**
- **Power estimation**
  - **Using activity measure**
  - **After/before P&R (precision)**
- **Statistical post processing**
  - **Formatting of the raw data**
  - **Statistical analysis (Matlab)**



Error detecting codes - a case study

- 8/16-bit registers
- 5 versions of the registers
  - No code
  - Simple Parity (one parity bit)
  - Double Parity (one parity bit for odd index bits)
  - Complementary Parity (both odd and even parity)
  - Berger Code (counts how many 0s)
- Technology: AMS C35 Corelib
- Exhaustive simulations for a null starting value (gate level)
- Correlation between data Hamming weight and the power consumption

Hardened register



Case study: cost and efficiency

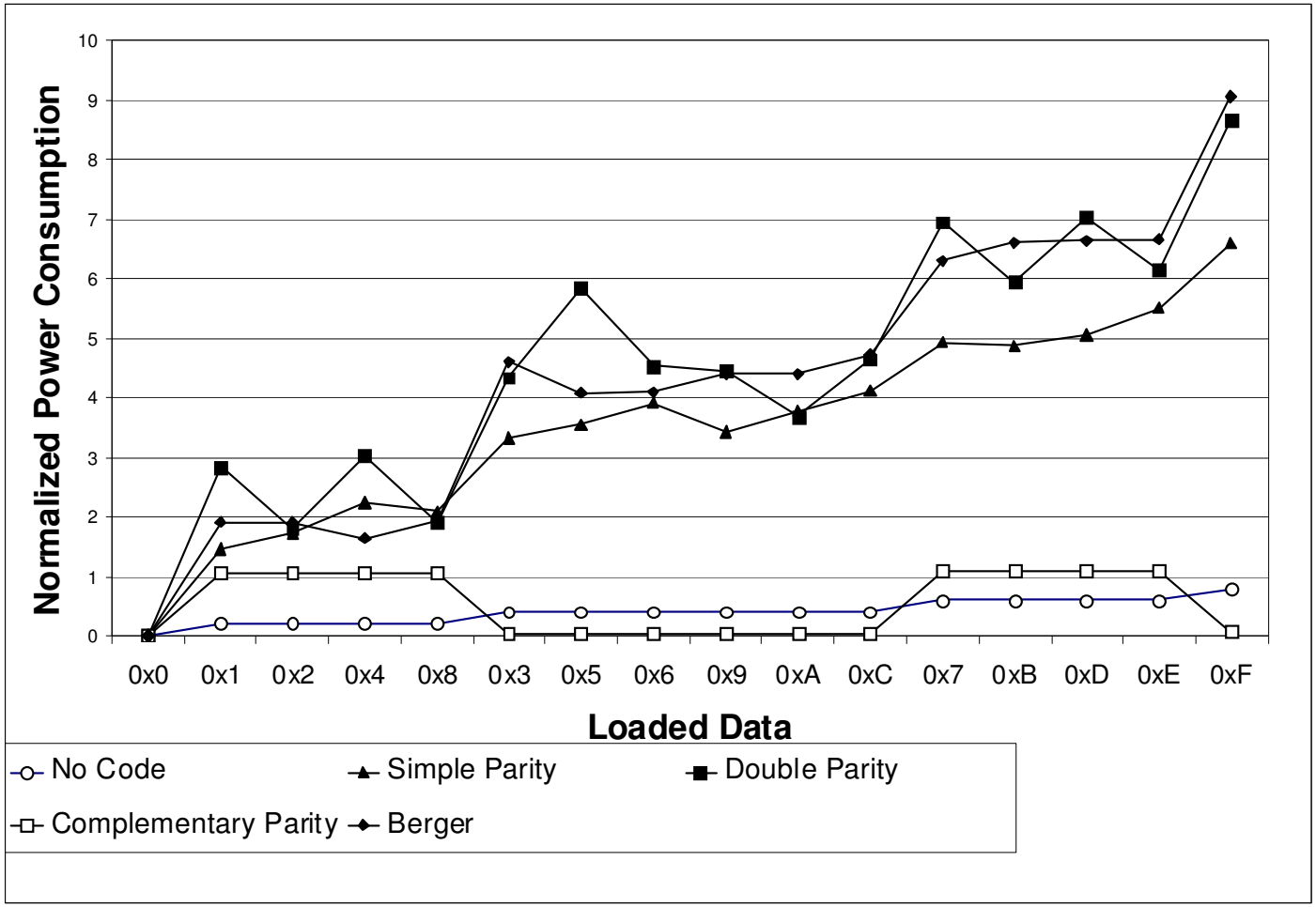
- Quite high overheads
  - Decoder in dual rail
- Detection probability
  - Unknown multiplicity
  - Not only unidirectional errors
  - Uniform distribution of the errors
- Trend to have lower cost for bigger registers
  - Except Berger

Code	Area overhead 8-bit / 16-bit	Detection probability 8-bit
No code	+0% / +0%	0%
Simple parity	+120% / +105%	50%
Double parity	+159% / +127%	74%
Complementary parity	+230% / +207%	75%
Berger	+358% / +1727%	93%

- Best trade-off cost/detection:
  - Double parity
  - Complementary parity

Case study: consumed power

- Normalized consumed power
  - Not very easy to use
- Almost linear dependence
  - Except for complementary parity



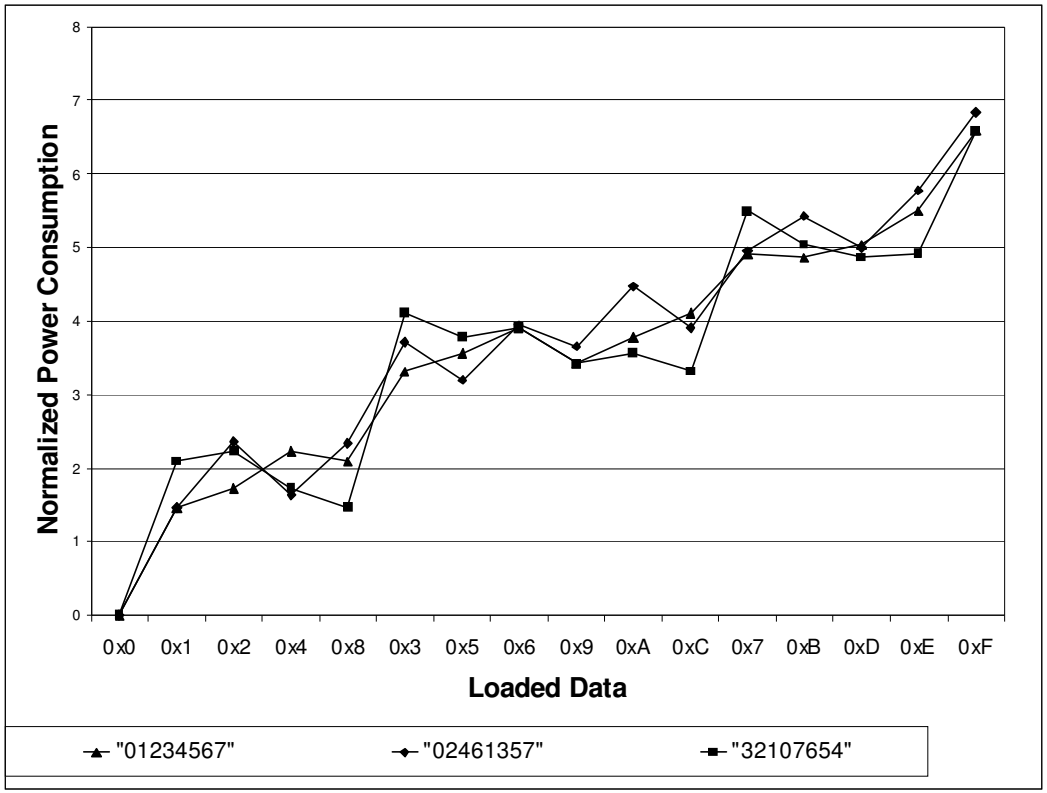
Case study: correlation values

Code	Correlation (Encoder) 8-bit / 16-bit	Correlation (Register) 8-bit / 16-bit	Correlation (Decoder) 8-bit / 16-bit	Correlation (Global) 8-bit / 16-bit
No code	n/a	1 / 1	n/a	1 / 1
Simple parity	0.995 / 0.965	0.927 / 0.945	0.955 / 0.949	0.977 / 0.970
Double parity	0.998 / 0.995	0.799 / 0.819	0.909 / 0.926	0.952 / 0.956
Complementary parity	**0.012** / 0.030	**0.334** / 0.537	**0.005** / 0.024	**0.056** / 0.162
Berger	0.989 / 0.849	0.999 / 0.970	0.993 / 0.929	0.994 / 0.935

- **Majority of the codes**
  - **Very high correlation**
  - **Almost linear dependence between data weight and power consumed**
- **Correlation increase with the size of the register**
  - **Less redundancy bits in proportion**
- **Complementary parity**
  - **Very low correlation (almost constant consumption)**
  - **Implementation with dual XOR/XNOR trees**

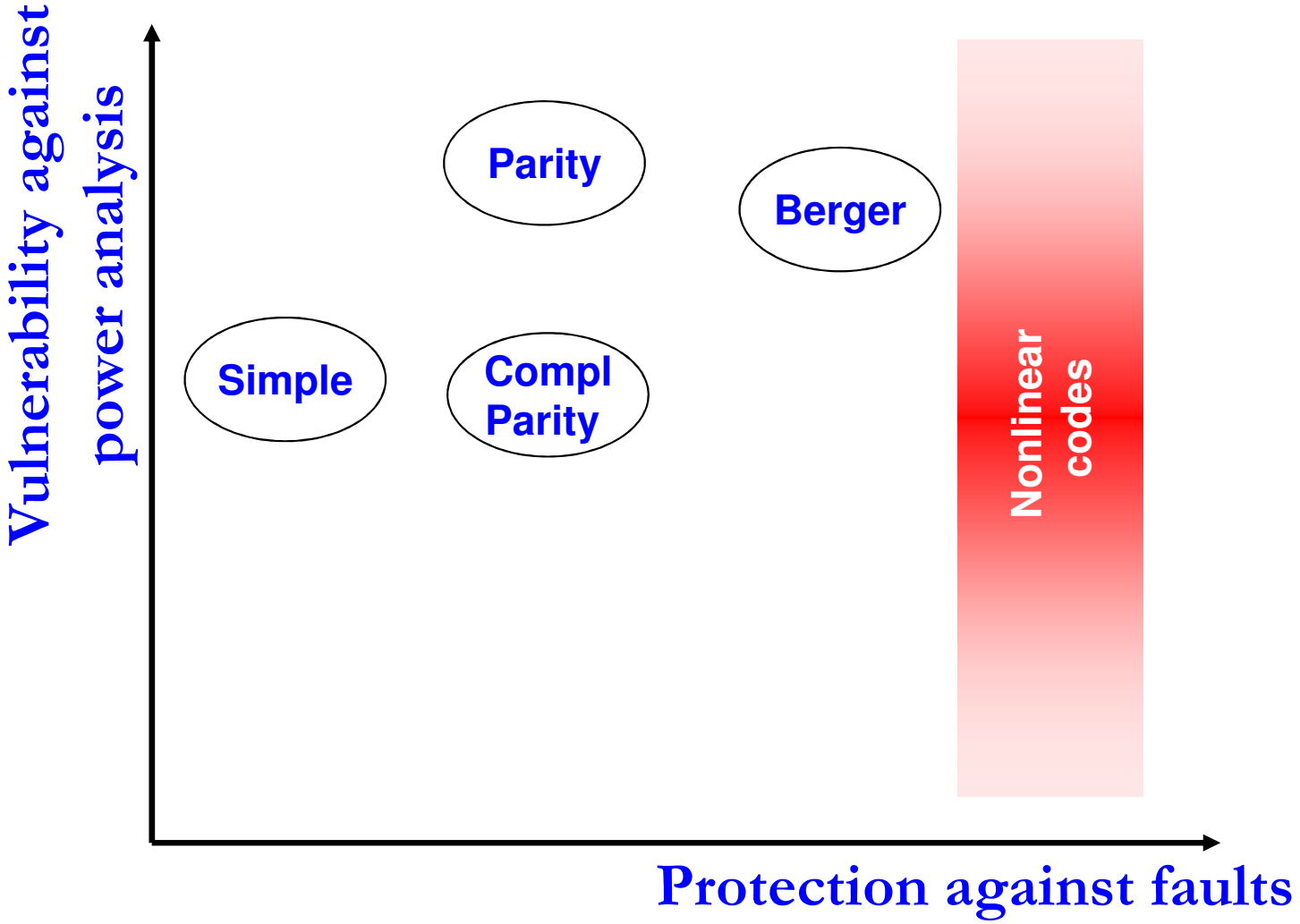
Case study: implementation influence

- Power for 3 versions of a simple parity code
  - Permutation of the bits on the decoder XOR/XNOR trees
- All inputs are not equivalent
  - Different values of power
  - Small changes in correlation
- Implementation must be done carefully to limit this effect



Code comparisons

- Complementary parity: best trade-off ?
- Attention must be given to both the design of the protection and its implementation

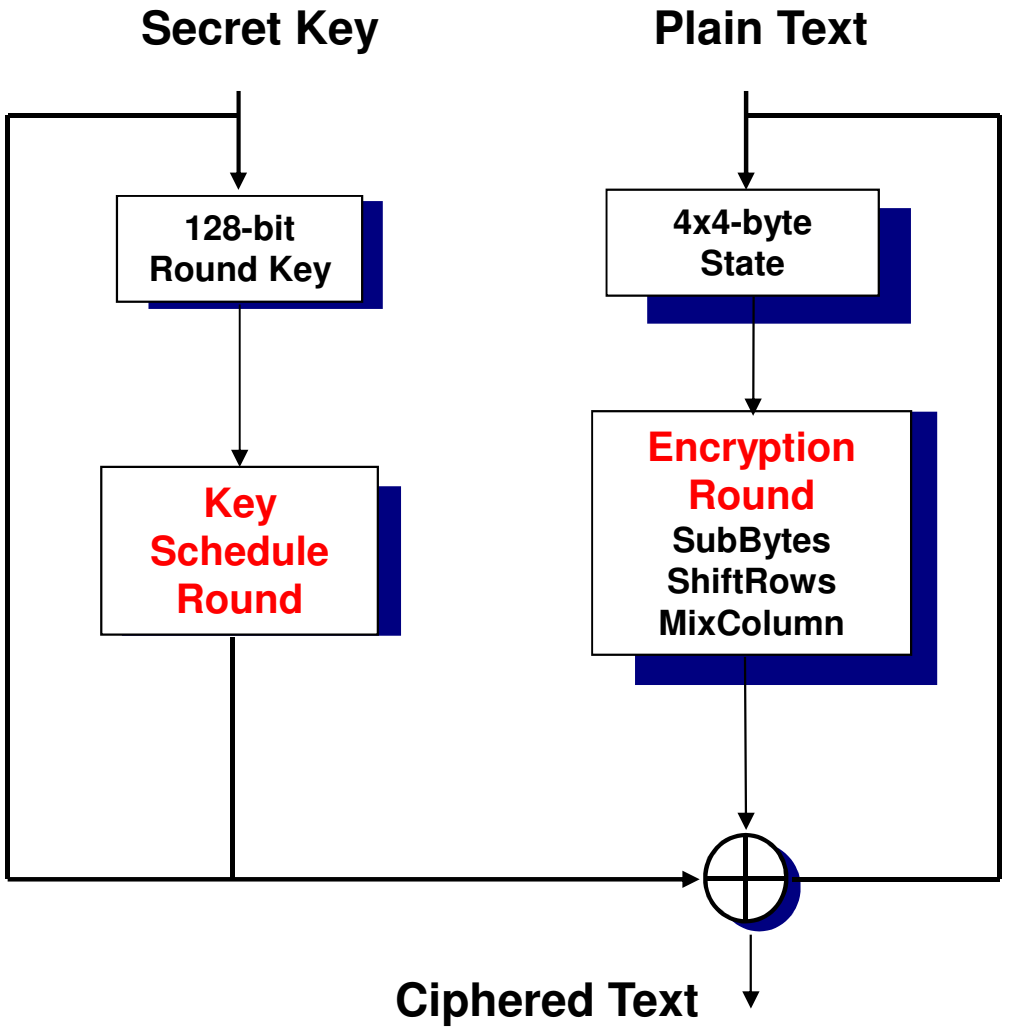


Part II

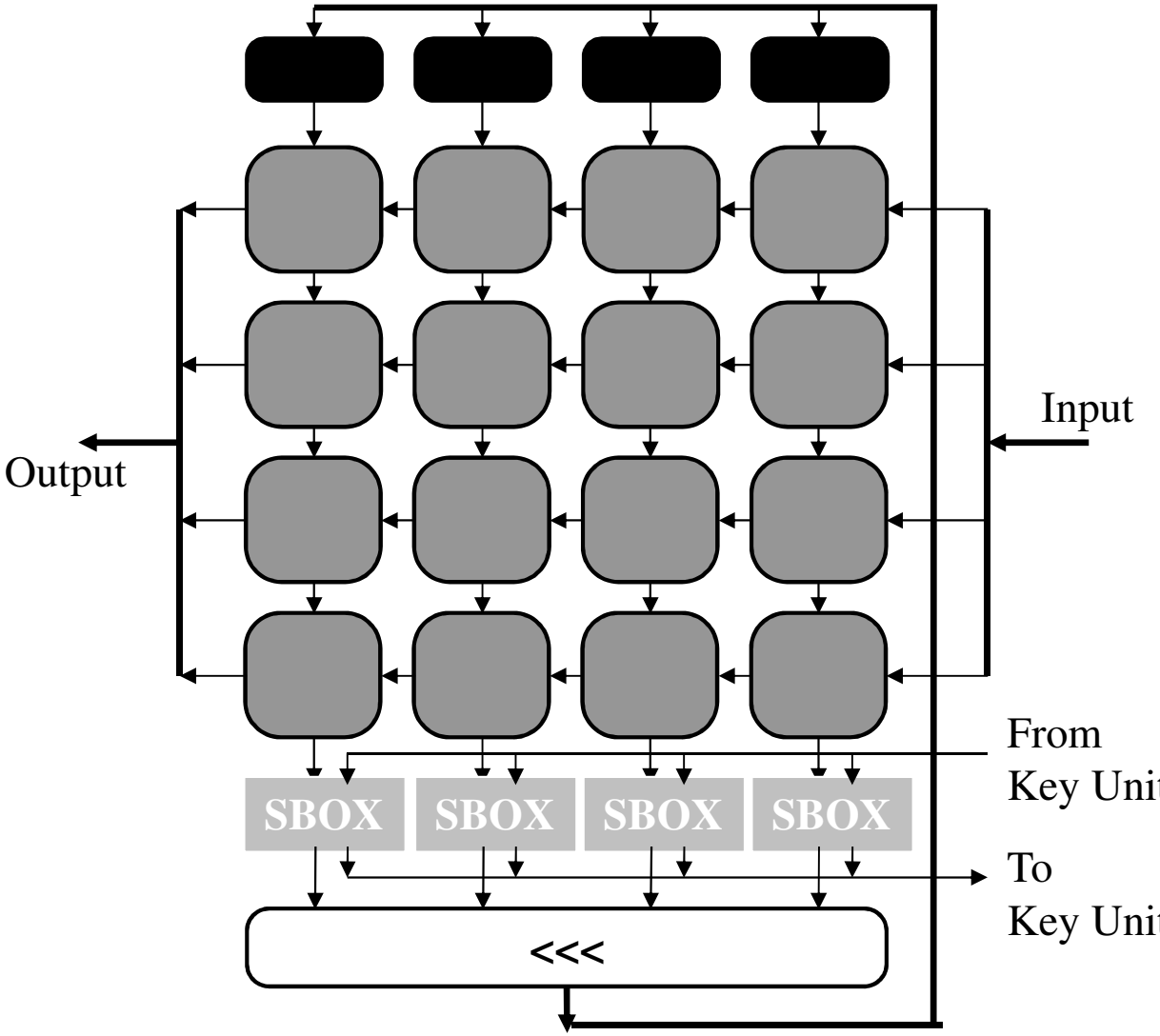
Protection optimization
Ex. An optimized AES architecture with timing redundancy

AES – the algorithm

- 128-bit data input
- 128/192/256-bit secret key
- Round-based (10 rounds for 128):
  - SubBytes: Non-linear byte substitution
  - ShiftRow: Word rotation
  - MixColumn: linear word transformation
  - AddRoundKey: modulus-2 addition with round key
- Encryption and Key Schedule use the same basic operations
  - Decryption uses inverse operations of encryption



AES – a data-path architecture



- 32-bit data-path
- 4 Substitution Boxes
- 16 GF Multipliers for MixColumns
- 6 clock cycles per round
- On-the-fly key unrolling (using shared S-Boxes)

MixColumns, AddRoundKey, State

2-stage SBox

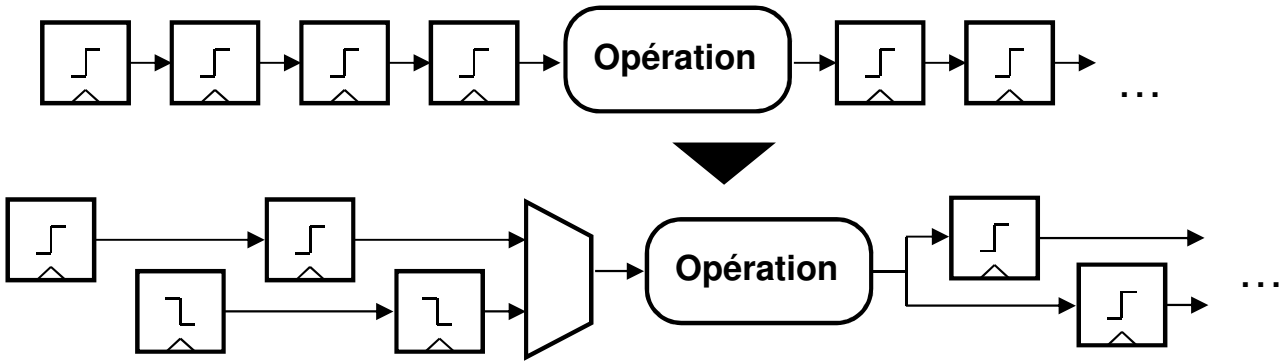
Register layer

Combinatorial logic

8-bit signal

32-bit signal

Double Data Rate (DDR)



- ☺ Surcoût acceptable
- ☺ Fautes transitoires longues
- ☹ Conception délicate
- ☹ Fautes permanentes

Cycle	1	2	3	4	5
Classical pipeline	D1	D2 D1	D3 D2	D4 D3	D4
Pipeline redundancy	D1	D1 D1	D2 D1	D2 D2	D2
DDR Redundancy	D1 D2	D3 D4 D1 D2	D1 D2 D3 D4	D3 D4 D1 D2	D3 D4

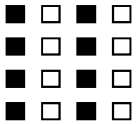
[Maistri, Leveugle. TC 2008]

- ✓ 2x the throughput at same frequency
- ✓ Small area overhead for DDR logic
- × More complex routing
- × Overhead for error detection

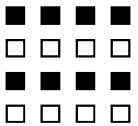
Data alignment in AES

- The data alignment phase partitions the register space into two classes:
  - Registers triggered by ascending clock edge
  - Registers triggered by descending clock edge

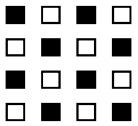
- Alignment can be done:



By columns: registers in the same columns share the clock alignment



By rows: registers in the same rows share the clock alignment



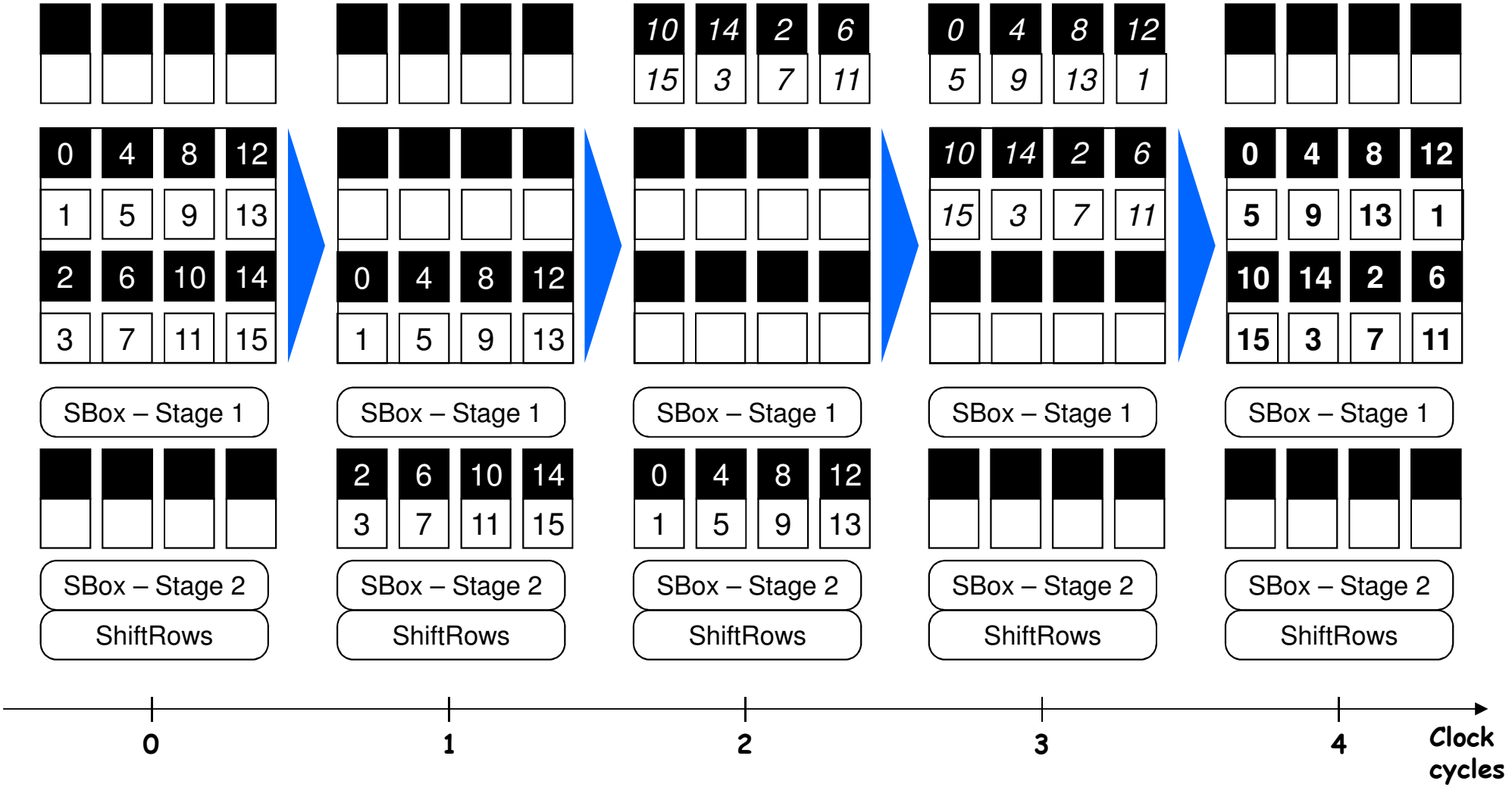
By checkers: elements of the partitions are interleaved both in columns and rows, like a chess board

Synchronization

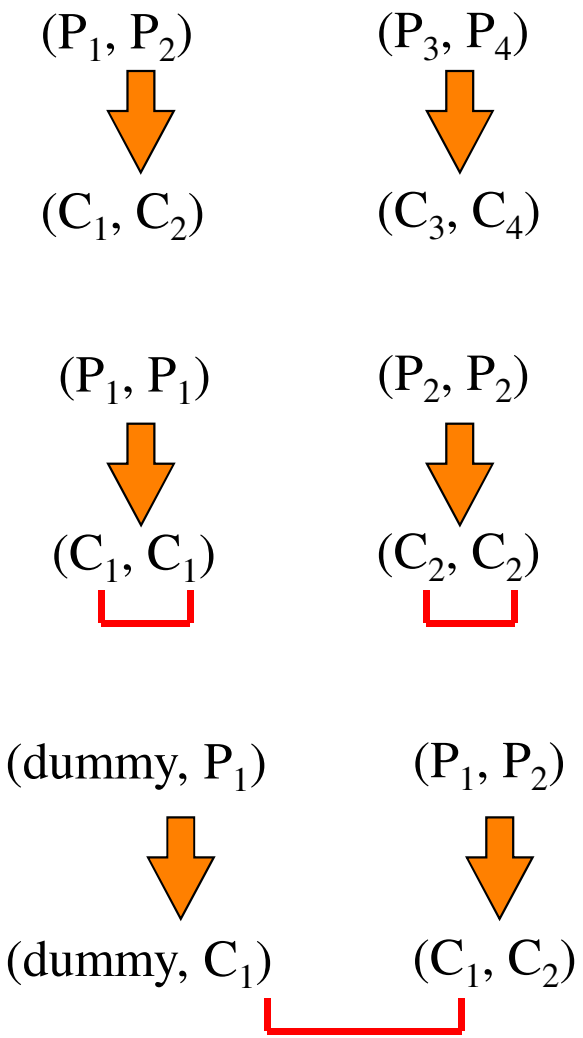
- **DDR computation can be employed when we have scarce resources, high parallelism and no data dependency**
  - **In our design, SBoxes are the scarce resources**
  - **Row rotation is performed while moving data during non-linear substitution (collateral data-dependence)**
  - **Row-wise DDR alignment is thus chosen**

- **In AES, all operations are independent on each byte, but the *MixColumns* operation**
  - **MixColumns are not a scarce resource (each byte is computed locally), but values have to be stable (i.e., a latch is used)**

DDR Round Encryption



Operation modes



Single: the unit uses the DDR computation to improve its throughput and no check is performed on data

Double: the unit uses the DDR computation to compute each round twice, checking for inconsistencies

Interleaved: like the *Double* mode, but the first and second repetition are processed together with two different (consecutive) blocks in ECB mode, which share the encryption key

Parity vs. Pipeline vs. DDR

	Parity	Pipeline*	DDR
Area	+ 23%	+ 50%	+ 36%
Throughput reduction	- 3%	- 18%	- 15% Single - 55% Double
Notes	One parity bit per byte	Temporal redundancy w/ unused pipeline stage	Temporal redundancy w/ both clock edges
Pros	High throughput and covers all single faults	(Almost?) all 1-cycle faults should be detectable	Same computations are very far in time; 2x Thr. @ same freq.
Cons	No even-order faults Poor on fault attacks	No guarantees if faults last more than one cycle, altering more bytes	Lower frequency; more complex design than pipeline

* RC6 implementation

Protections beyond DDR: controller

Reduce the number of possible targets	**Simplify the controller removing redundant registers, which store signals that could be instead computed on the fly**
Protect sensitive targets	**Specific registers are duplicated to ensure correct behavior (e.g., counters, state registers)**
Validate state encoding	**If any FSM falls into an unused state encoding, computation stops and they both return to the reset state**
Verify state transitions	**If an FSM performs an erroneous transition (e.g., Idle > Output), the error signal is raised and the machines go back to their reset state**

Implemented DDR architectures

- Medium (represented in previous figures)
  - 4 S-Boxes, 16 linear multipliers
  - Synchronization driven by state rotation
  - Horizontal data alignment: synchronization layer required for column-wise linear multiplication
  - 64 cycles/block
- Small
  - 4 S-Boxes, 4 linear multipliers
  - Synchronization driven both by state rotation and linear multiplication
  - Vertical data alignment: synchronization layer required for row-wise state rotation
  - 100 cycles/block

Software simulations

- Parity code was previously validated by algorithmic simulations
  - Computation model was used to further simplify the robustness analysis
  - Confirmed expected results: only those faults injecting an odd number of bit errors in at least one byte were detectable
- Fast and simple but…
  - Errors were injected only at the beginning of a round (actually a negligible limitation)
  - Only computation variables could be altered, no possible intervention on control flow
  - Software algorithm can not be an accurate model of a hardware implementation

Injection campaign

- Location
  - Linear multiplication layer
  - Barrel shifter (row rotation)
  - Non-linear substitution layer: inner and outer stage
  - Control unit
  - AES is highly regular: only one target element for each data path location
- Timing
  - Every computation clock cycle
  - Input or output phase were not considered
  - From 1 up to several clock cycles (twice the length of a round)
- Value
  - According to the laser-injected fault model, the error value is not controllable
  - Exhaustive search of all error values is carried on for each targeted area (e.g., all byte values for a byte target)

Implementation and injection results

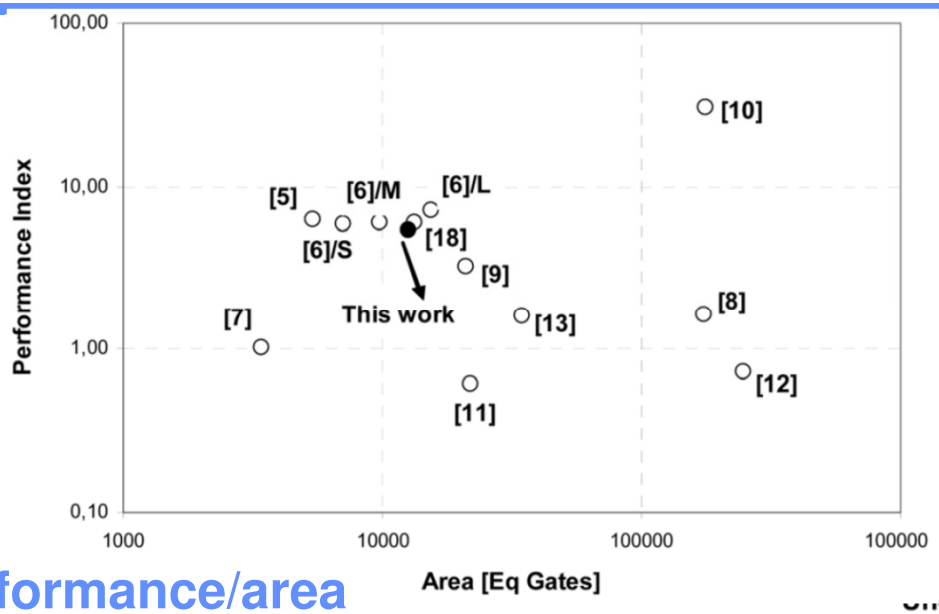
Architecture	Area Overhead	Throughput Reduction	Coverage Byte Errors in Datapath
Multiple Parity Bits Bertoni et al., TC '03	33%	3%	~67%
Inverse Process Karri et al., DAC '01	19% - 38%	23% - 61%	100%
Pipeline Recomputation Wu and Karri, DFT '01	50%	18%	~100%
Single Parity Bit Karri et al., CHES '03	18% - 24%	NA	~67%
Non-linear Code Karpovsky et al., DSN '04	77%	13%	~100%
DDR	36%	15% - 55%	~100%

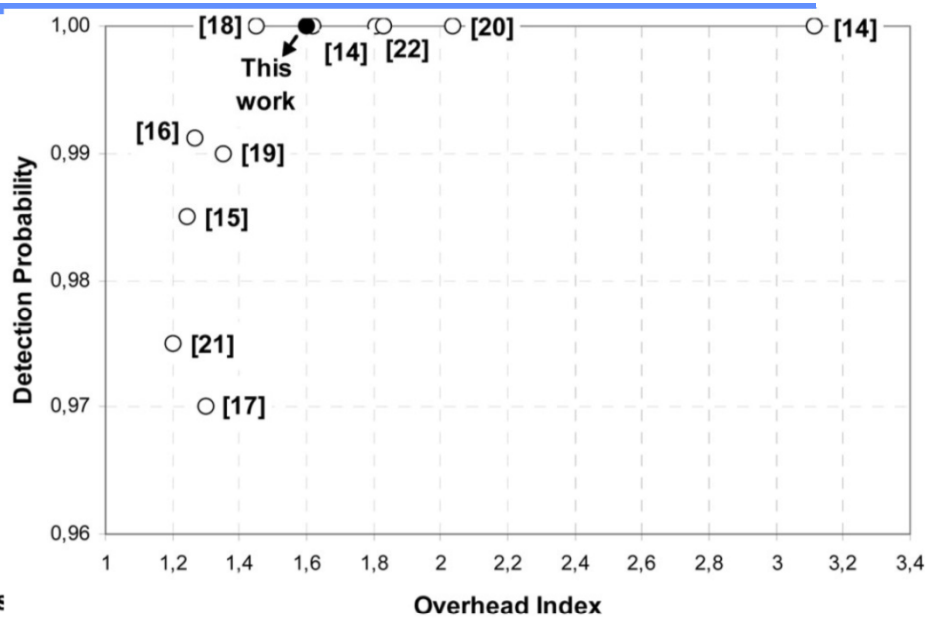
Protection in the control unit

One-Hot Encoding		Result Class [%]			
Target	Size (bits)	Silent	Undetected	False Pos	Detected
Low FSM	**9**	**4.36**	**0.20**	**4.92**	**90.52**
High FSM	**19**	**0.00**	**16.30**	**1.87**	**81.63**
Inner Signals	**6**	**15.74**	**84.26**	**0.00**	**0.00**

Reduce the number of possible targets	**The controller has been simplified removing superfluous registers**
Protect sensitive targets	**Specific registers are duplicated to ensure correct behavior (e.g., counters, state registers)**
Validate state encoding	**If the FSMs fall into a non-existing state encoding, they return to the reset state**
Verify state transitions	**If an FSM performs an erroneous transition (e.g., Idle > Output), the error signal is raised**

AES DDR – Results [TC'08]

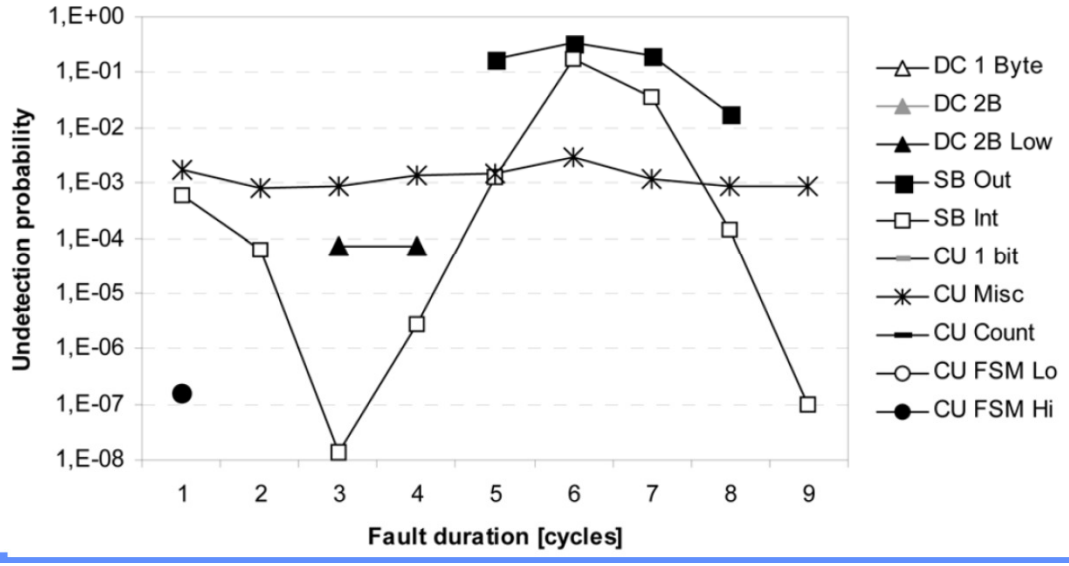




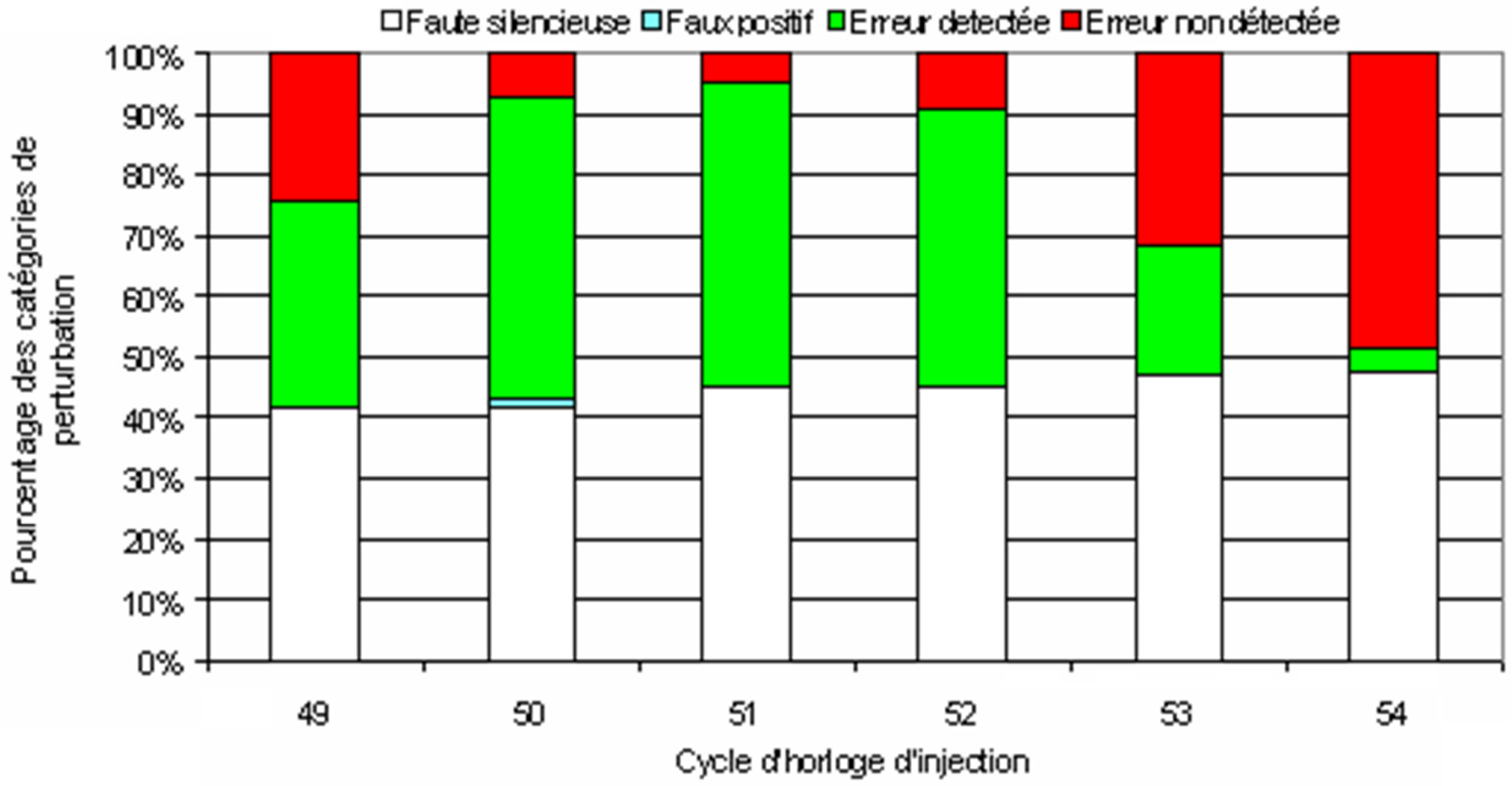
Performance/area characteristics

Detection vs. overheads

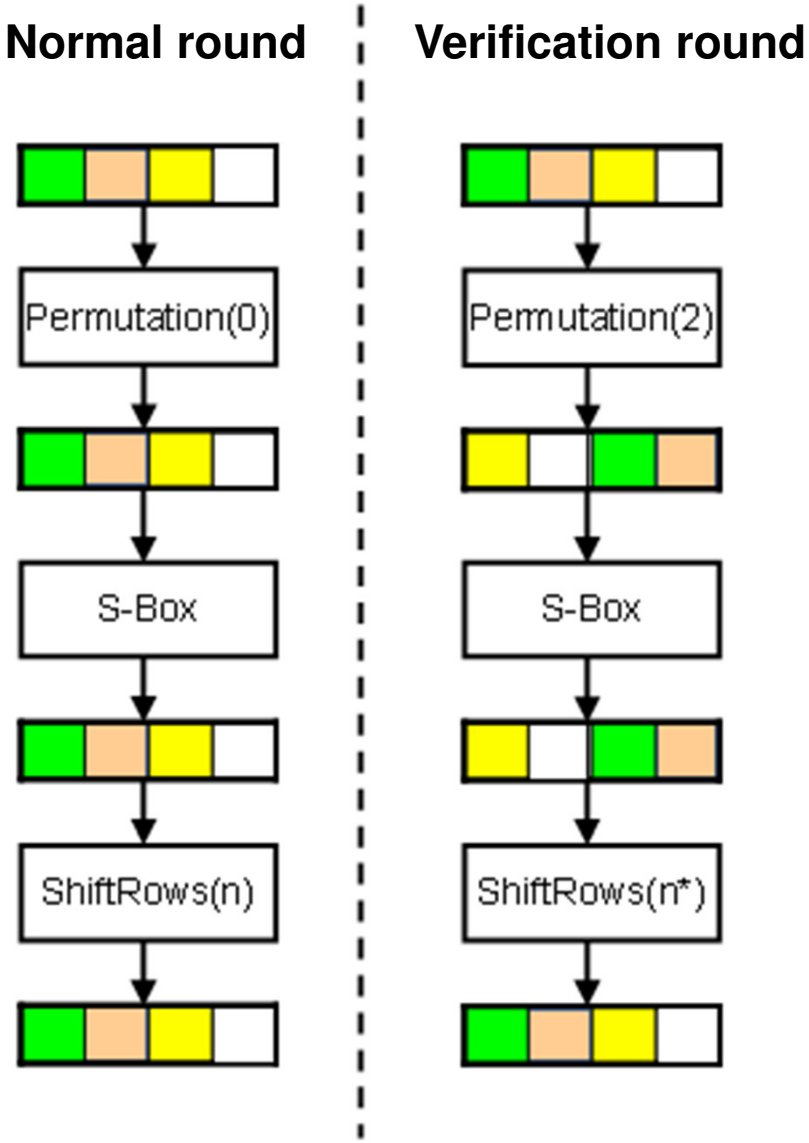
Undetected faults vs. duration



DDR on FPGA – Laser Injections



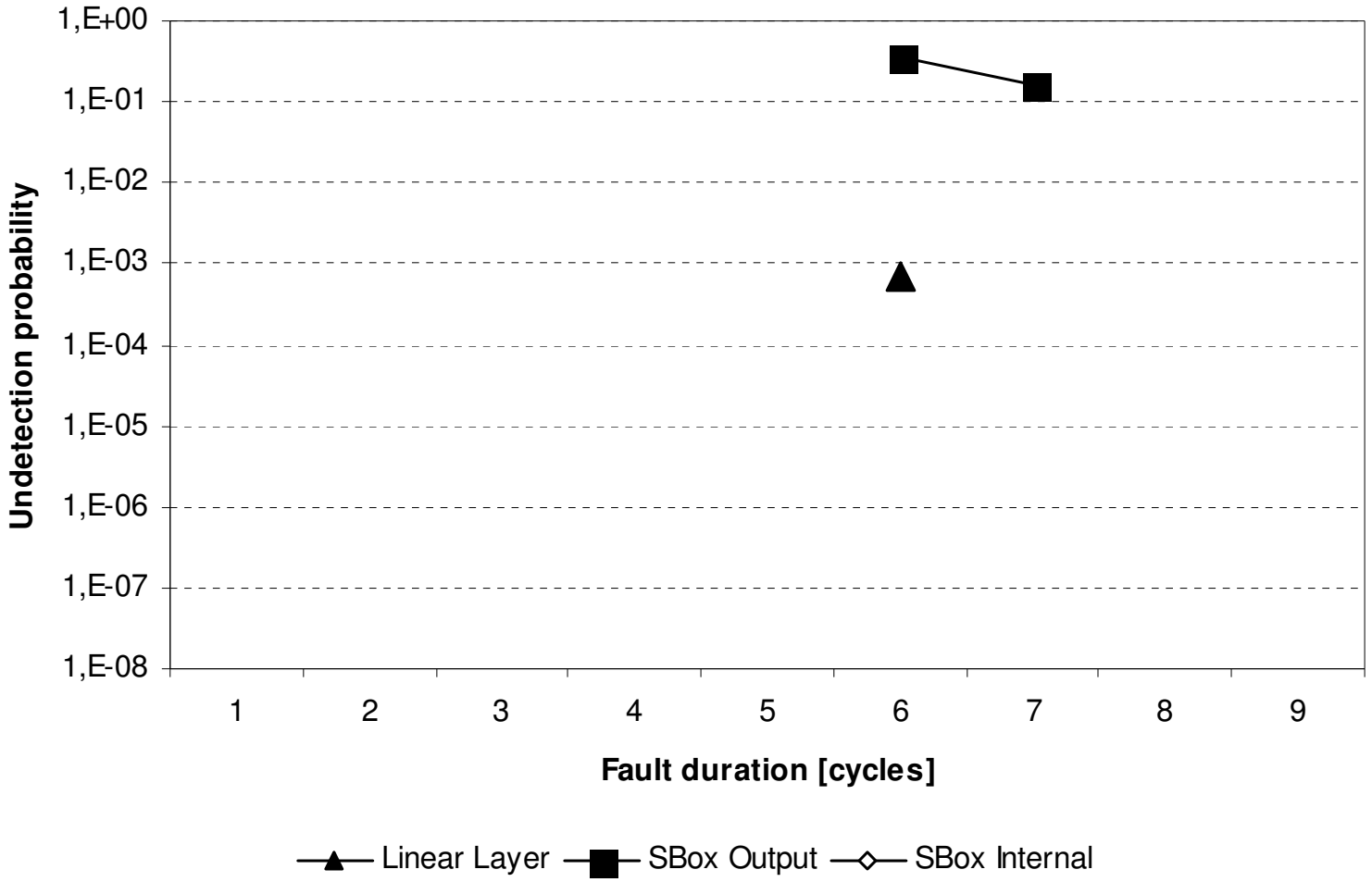
DDR and Resource Reallocation



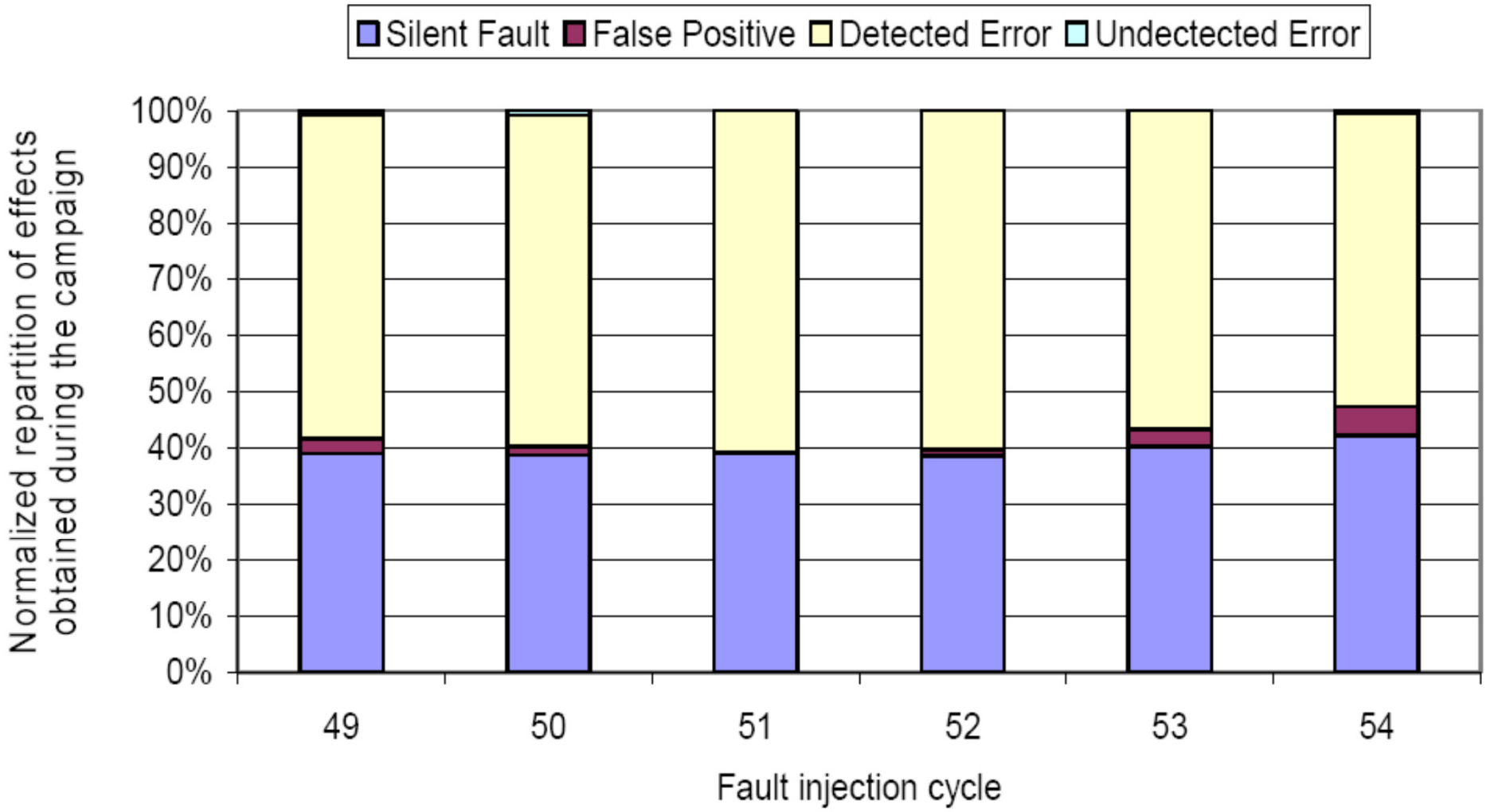
Results	Percentage
Shots with no effect	54,6 %
Modified configuration	45,4 %
Silent Faults	39,6 %
False Positives	2,3 %
Detected Errors	57,9 %
Undetected Errors	0,3 %

An improved counter-measure

- **Taking into account remanent (or permanent) faults**
  => resource reallocation between normal and verification rounds



Laser attacks on last DDR protected version



Overview of efficiency vs. overheads [P. Maistri, IOLTS 2011]

Type	Description	Permanent Faults		Multiple transient	Area overhead	Throughput reduction
		Single	Multiple			
Hardware	Duplication	100%	☺	☺	> 100 %	-
	Partial red	☺	☺	☹	25 – 35 %	-
Information	Byte/Word Codes	100 %	> 99 %	50 – 99 %	20 – 30 %	3 %
	Block Codes	98 %	☹	☹	18 – 24 %	~ 0 %
	Nonlinear code	$1 - 2^{-56}$	$1 - 2^{-56}$	☺	77 %	13 %
Time	Process repetition	0 %	0 %	☺	~ 0 %	50 %
	Involution ciphers	100 %	☺	☺	~ 0 %	50 %
	SLICED	100 %	☺	☺	~ 0 %	64 %
	Pipeline Red	0 %	☹	☺	50 %	18 %
	DDR Red	0 %	☹	☺	36 %	15 – 55 %
	DDR++	> 90 %	☺	☺	~ 36 %	~ 15 – 55 %
Mixed	Inverse Operation	100 %	☺	☺	19 – 38 %	23 – 61 %
	Inv Pipeline	100 %	☺	☺	-21 – 24 %	~ 25 %

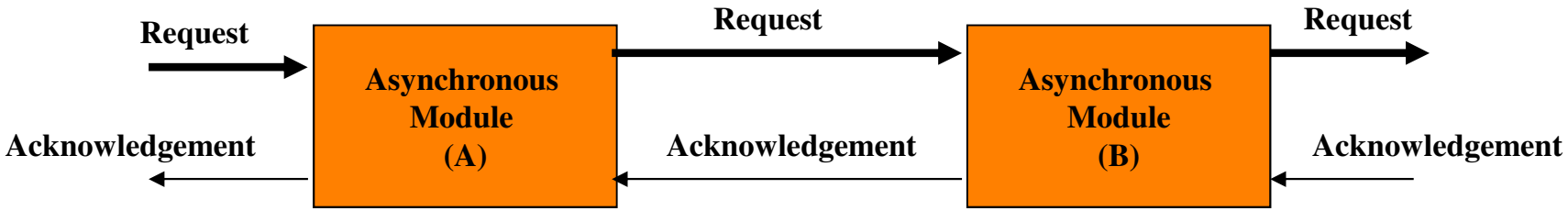
Temporal Redundancy - Summary

- Repetition of the same or of the inverse computation
- Detection of transient faults
  - Permanent faults detected by inverse computation
- Area overhead and throughput reduction acceptable when implemented with a pipeline
- No significant vulnerability against side channel attacks
  - if the basic architecture is protected itself, of course

Influence of design style
Example:
synchronous vs. asynchronous logic

Asynchronous Logic: Quasi Delay Insensitive circuits

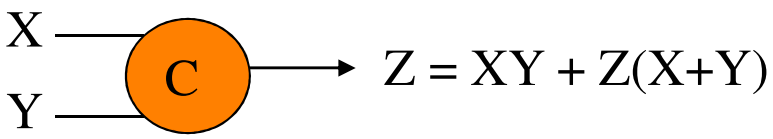
Basic structure of a synchronous circuit



Handshake-based communication between modules (QDI)

Intrinsically more robust to perturbations (delay faults) and SCA (better current profile)

QDI memory cell: the Muller gate (C element)



Truth table:

X	Y	Z
0	0	0
0	1	Z^{-1}
1	0	Z^{-1}
1	1	1

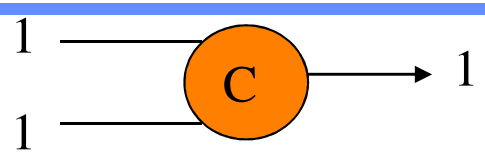
This gate is used to implement both the data paths and the protocol specification.

QDI circuits sensitivity criterion =>

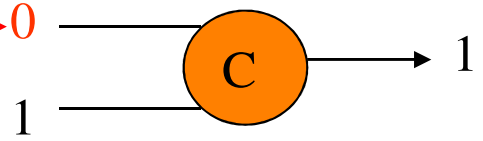
Ability of Muller gates to memorize/filter a transient fault

QDI circuits: specific sensitivity criterion

‘Set’ state:
(non sensitive)

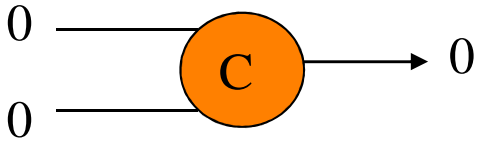


Single fault at input → 0

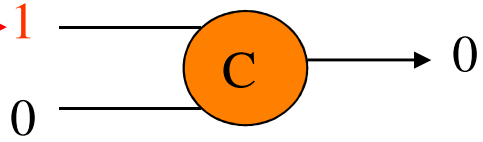


Fault is filtered !

‘Reset’ state:
(non sensitive)

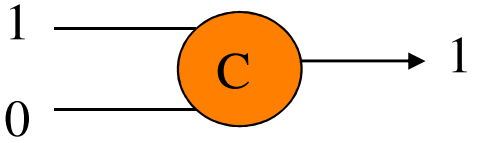


Single fault at input → 1

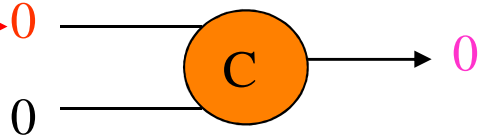


Fault is filtered !

Sensitive to ‘1’:

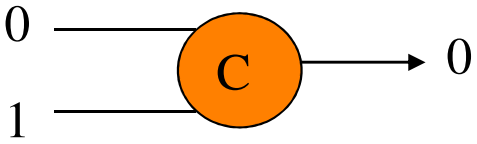


Single fault at input → 0

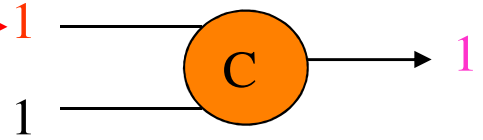


Fault is memorized !

Sensitive to ‘0’:



Single fault at input → 1

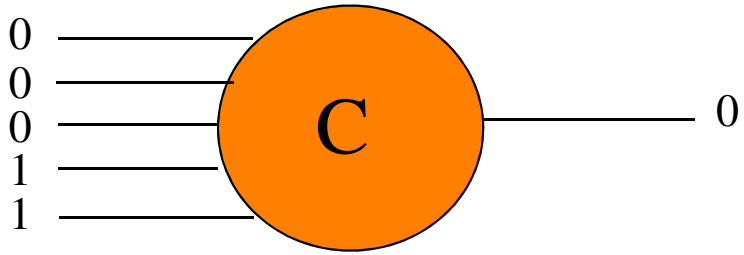


Fault is memorized !

Y. Monnet, M. Renaudin, R. Leveugle, "Asynchronous circuits sensitivity to fault injection", 10th IEEE International On-Line Testing symposium, Funchal, Madeira, Portugal, July 12-14, 2004, pp. 121-126

Generalization of the sensitivity criterion

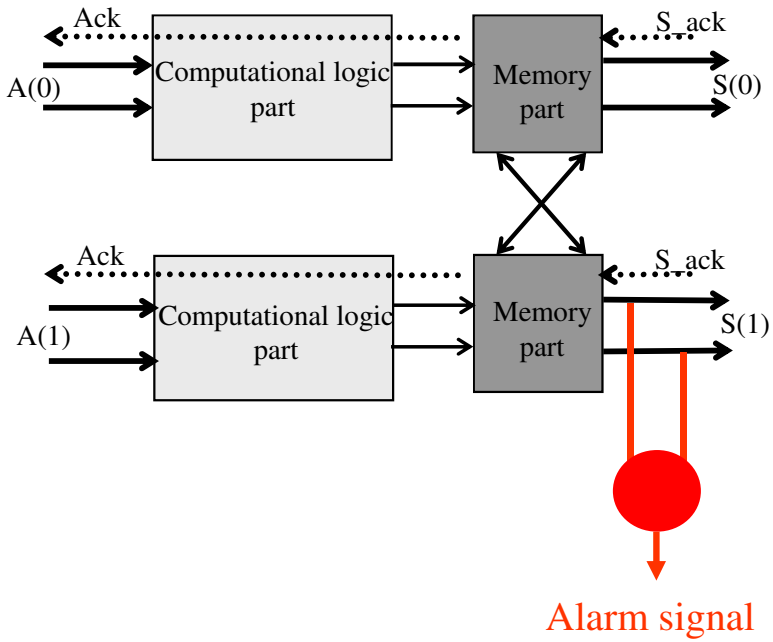
N-input Muller gate: sensitivity to [1..M] erroneous inputs



A “3-sensitive to 0” Muller gate

QDI circuits: specific protections

- **Objective :**
  - Protecting logic blocs against transient faults
  - Detecting wrong code generation
- **Rail synchronization technique**
  - Improves the tolerance against transient faults
  - Improves the wrong code detection when combined with alarm cells
- **Alarms cells**

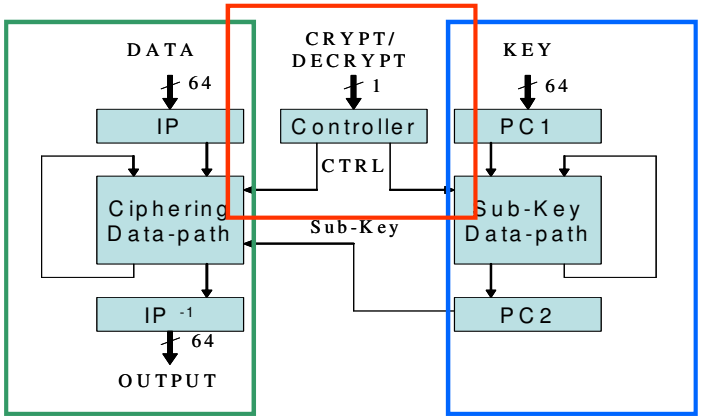


Y. Monnet, M. Renaudin, R. Leveugle, N. Feyt, P. Moitrel, F. M'Buwa Nzenguet, "Practical evaluation of fault countermeasures on an asynchronous DES cryptoprocessor", 12th IEEE International On-Line Testing symposium, Como, Italy, July 10-12, 2006, pp. 125-130

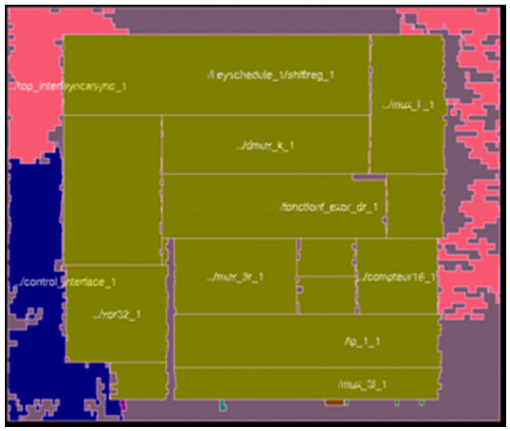
Asynchronous DES crypto-processors

Controller module

Data path module



Sub-Key module



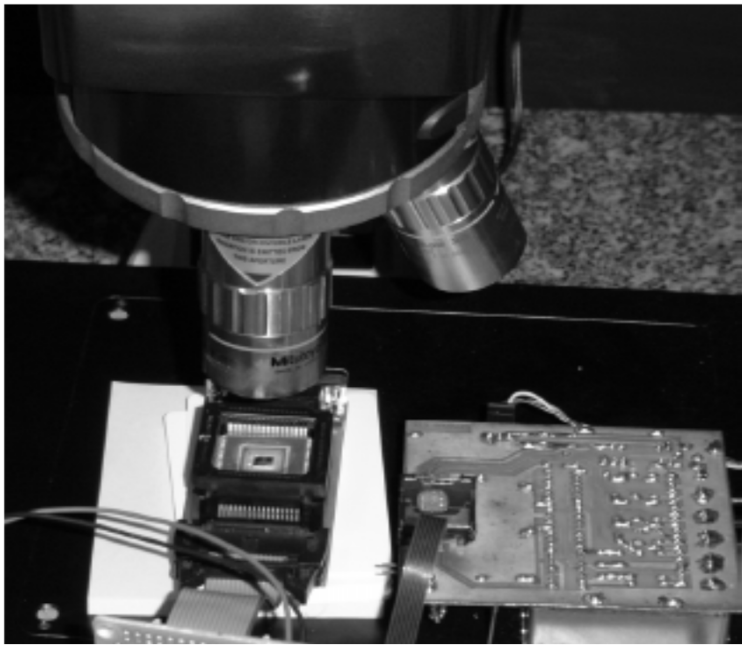
Low cost of hardening techniques:
about 10% overhead in terms of speed and area,
and negligible overhead in terms of power consumption

130 nm STmicroelectronics CMOS process

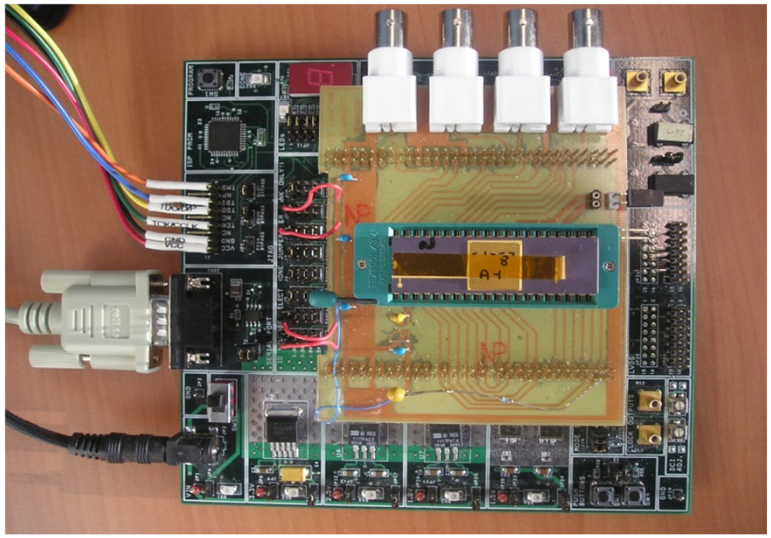
Practical attacks: laser setup

Laser characteristics:

- Green Laser
- 6 ns pulse
- Tunable spot size (220 µm²)
- Tunable energy (0.8 pJ/µm²)



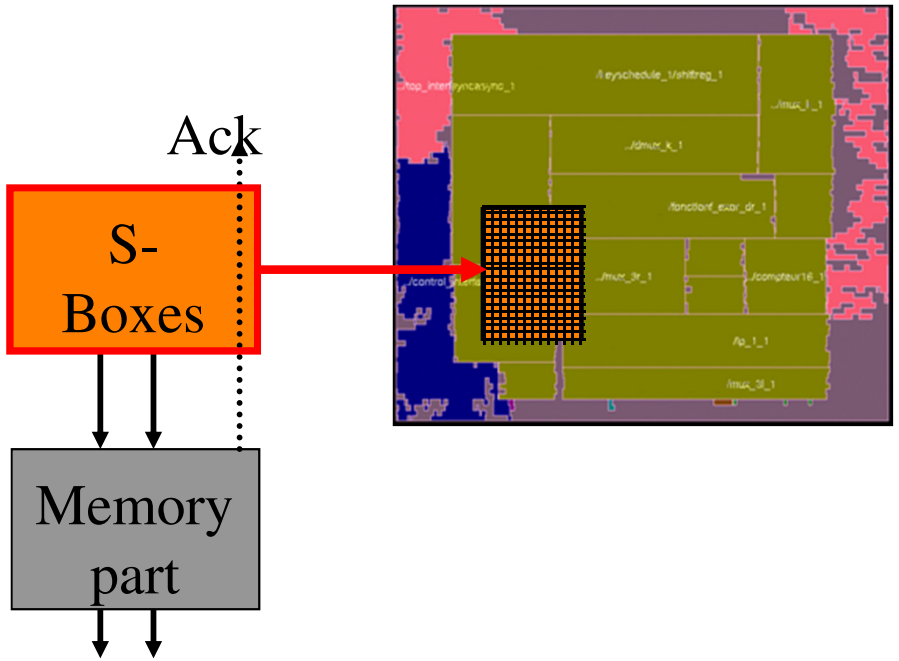
Gemalto laser platform



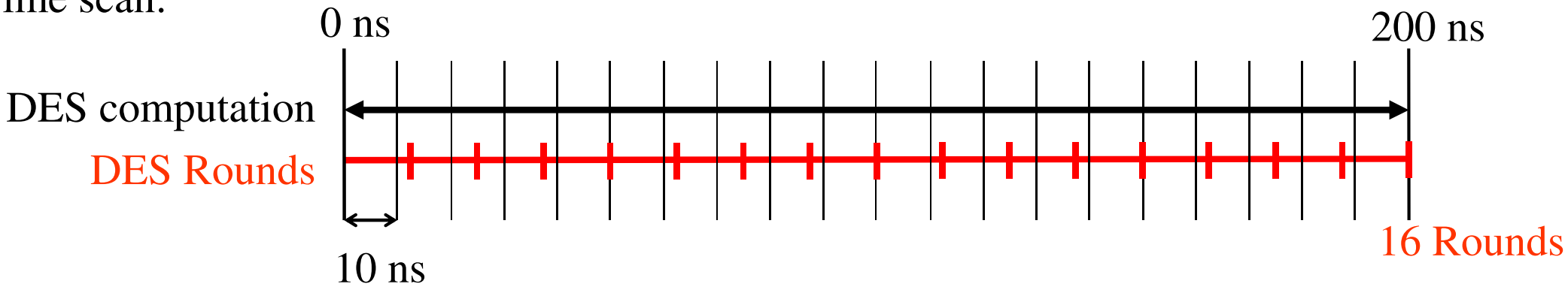
Laser Board for the DES

Campaign specification (example)

Spatial scan in S-Boxes
(non-linear substitution tables)



Time scan:



Results on Sbox

- **Asynchronous logic characterization:**

  Only 17% of the shots lead to an observable error

  > Delay insensitivity ?
  > Asynchronous circuit properties ?

SBOX	Number of shots	Number of errors
Reference	5600	955 (17%)
Hardened	5600	377 (7%)

- **Fault tolerance of the hardened DES is improved by 2.5**

- **Detection:**
  - **in unhardened version means deadlock (no end signal)**
  - **in hardened version means deadlock or alarm**

SBOX	Detected errors	Undetected errors
Reference	642 (67%)	313 (33%)
Hardened	377	0 (0%)

Prevention is better than cure

- **Instead of detecting the error, avoid the injection:**
  - **Filters on power supply**
  - **Sensors for glitches and overvolting**
  - **Temperature sensors**
  - **Light sensors**
  - **Metallic shielding**
  - **Memory encryption**
  - **Address scrambling**
  - **…**