Circuits électroniques de cryptographie attaques et protections

Sylvain GUILLEY.

Institut TELECOM / TELECOM-ParisTech
CNRS – LTCI (UMR 5141)

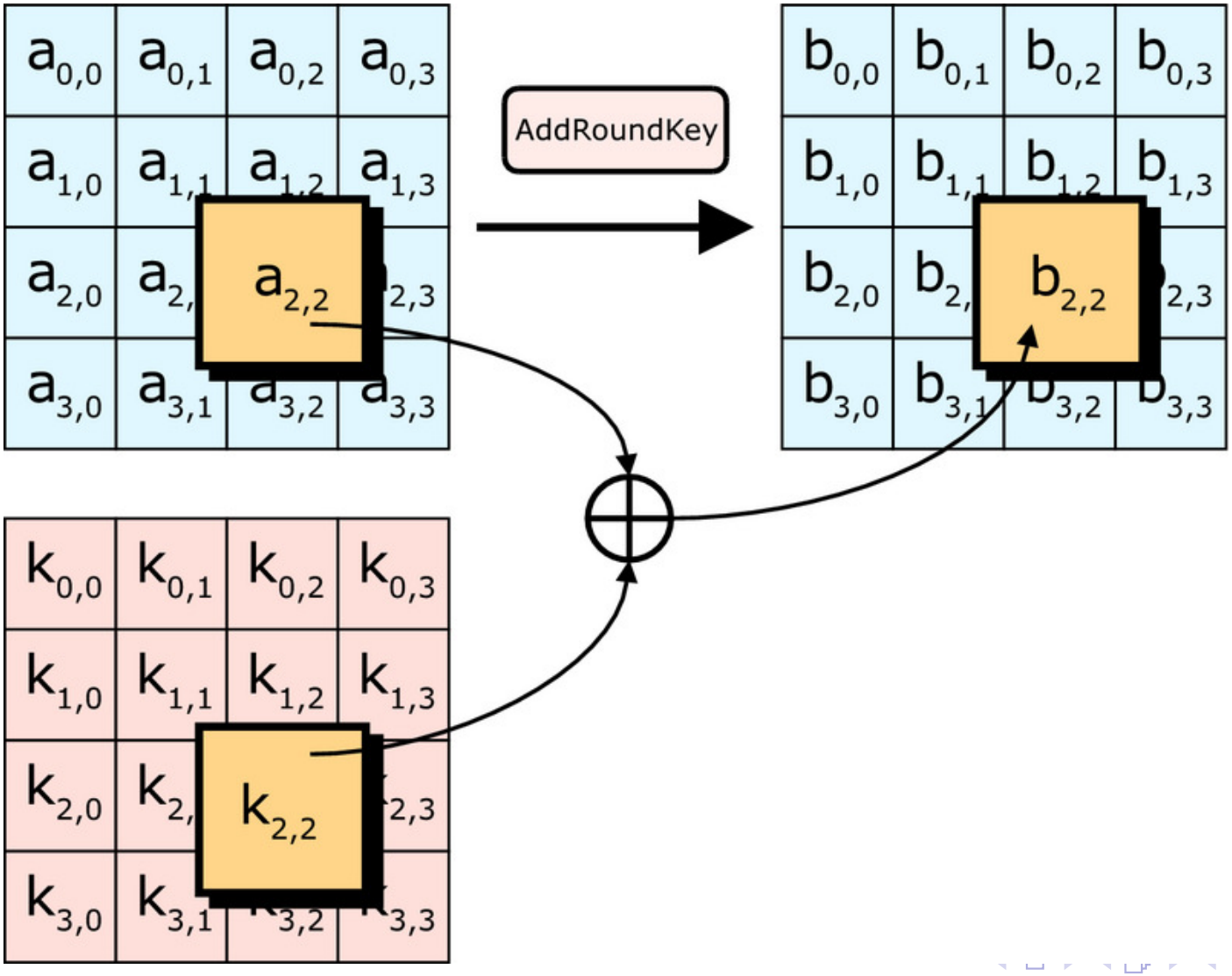
ENS L3, "systèmes numériques"

Presentation Outline

Presentation Outline

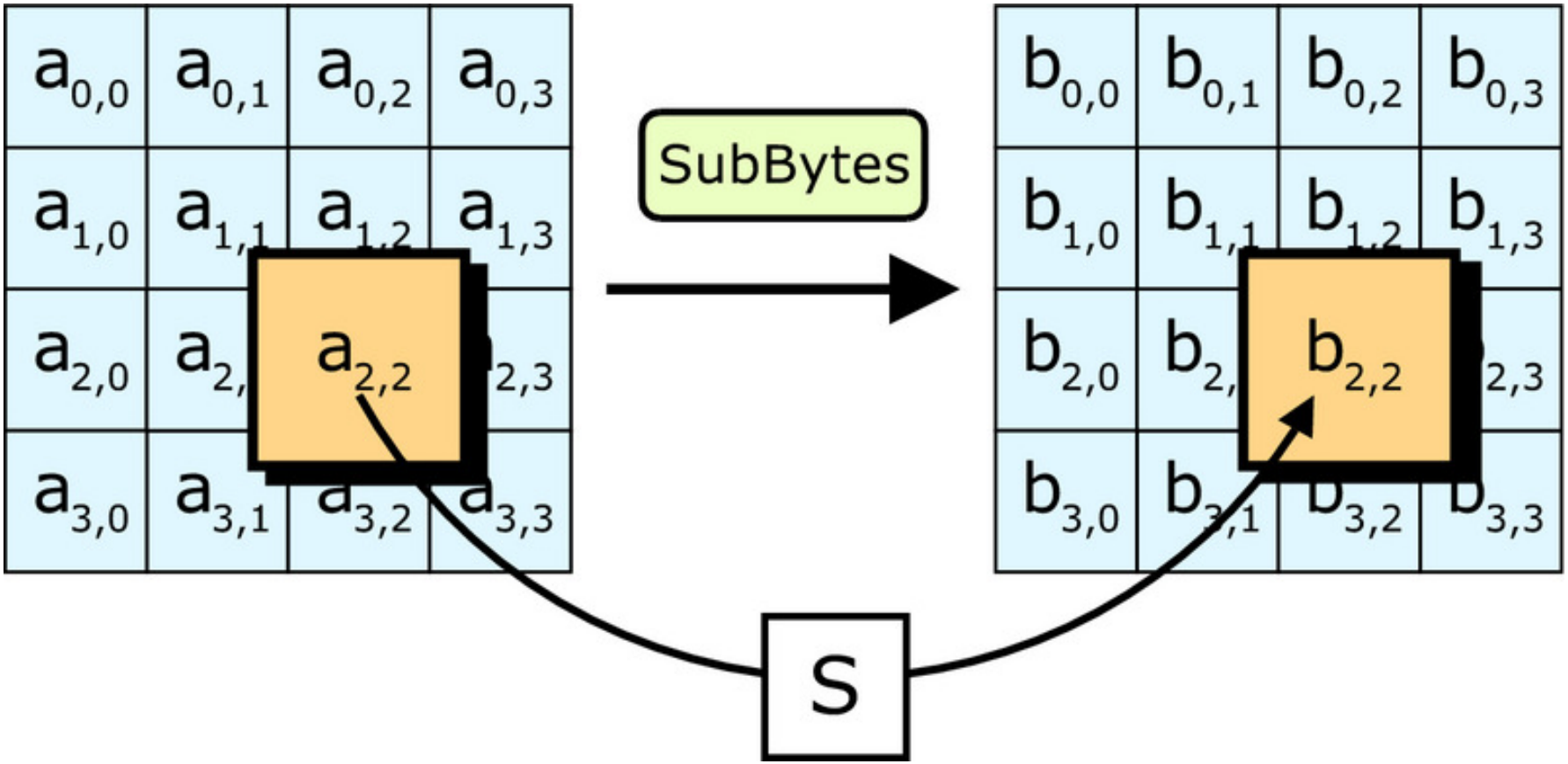
AddRoundKey

Bit-wise XOR



SubBytes

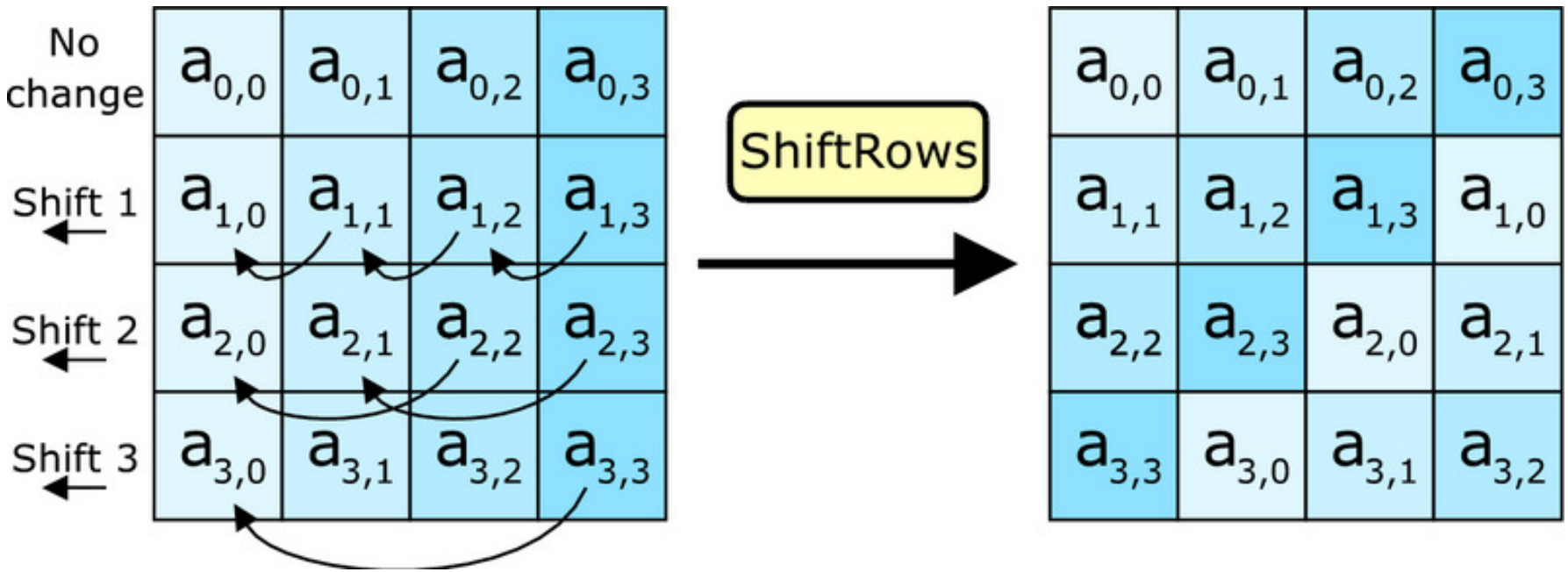
Understand "Substitute Bytes"



$S(x) \doteq a \cdot x^{-1} + b$ in $\mathrm{GF}(2)[X]/X^8 + X^4 + X^3 + X + 1$ if $x \neq 0$ or $S(0) \doteq b$ otherwise.

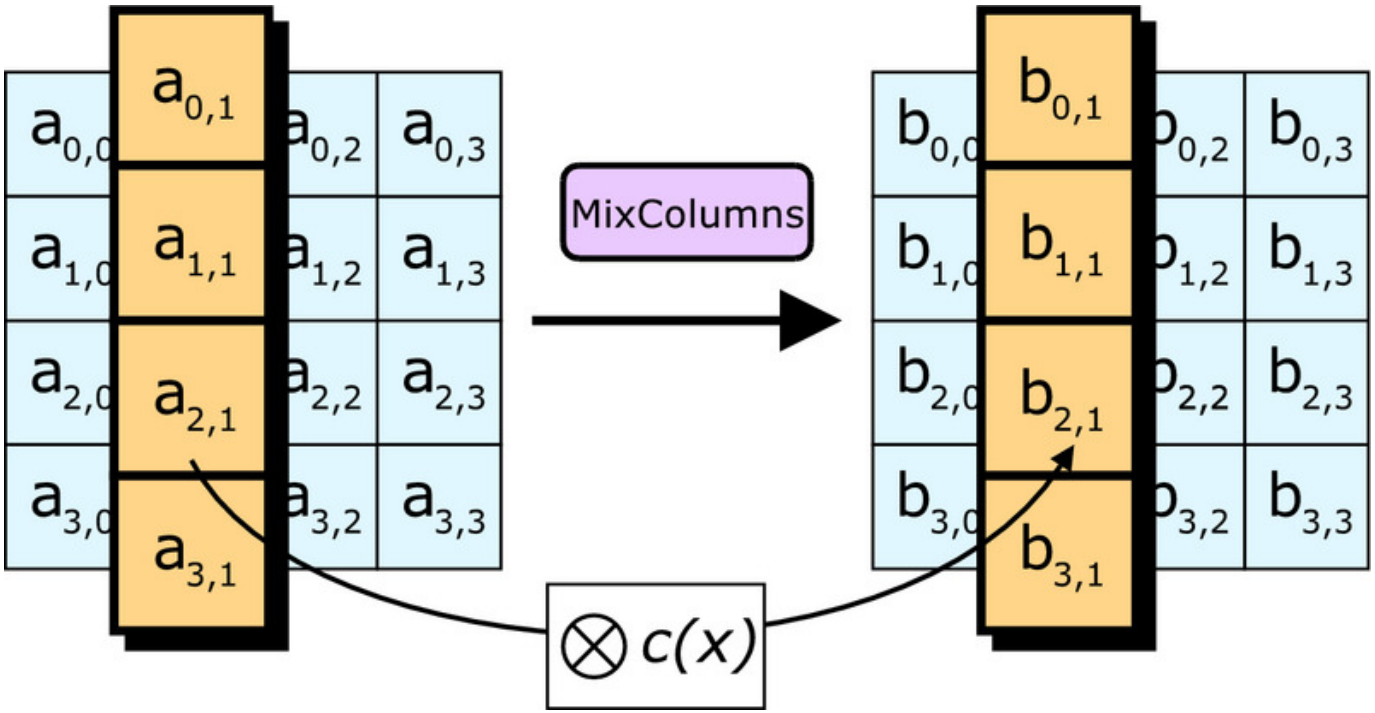
ShiftRows

Data independent circular shift



MixColumns

Intra-column mixing



Implies $\texttt{xtimes}(x) = \{02\}x$. Then MixColumns represents columns in $\mathrm{GF}(2^8)[X]/X^4 + 1$. MixColumns and InvMixColumns are the multiplication by :

$$\begin{cases} \text{MixColumns :} & a(x) &= \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}\,, \\ \text{InvMixColumns :} & a^{-1}(x) &= \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}. \end{cases}$$

Encryption

Encryption

- AddRoundKey `// Isolated`
- SubBytes
  ShiftRows
  MixColumns
  AddRoundKey `// n times ...`
- SubBytes
  ShiftRows
  AddRoundKey `// Last round`

Encryption *versus* Decryption

Encryption

- AddRoundKey `// Isolated`
- SubBytes
  ShiftRows
  MixColumns
  AddRoundKey `// n times ...`
- SubBytes
  ShiftRows
  AddRoundKey `// Last round`

Decryption

- AddRoundKey `// Was last round`
  InvShiftRows
  InvSubBytes
- AddRoundKey `// n times ...`
  InvMixColumns
  InvShiftRows
  InvSubBytes
- AddRoundKey `// Was isolated`

Note :
As such, encryption and decryption are not alike. One idea would be to swap InvShiftRows $\leftrightarrow$ InvSubBytes and AddRoundKey $\leftrightarrow$ AddRoundKey.

Encryption *versus* Decryption

Encryption

- AddRoundKey `// Isolated`
- SubBytes
  ShiftRows
  MixColumns
  AddRoundKey `// n times ...`
- SubBytes
  ShiftRows
  AddRoundKey `// Last round`

Decryption

- AddRoundKey `// Was last round`
  InvSubBytes
  InvShiftRows
- AddRoundKey `// n times ...`
  InvMixColumns
  InvShiftRows
  InvSubBytes
- AddRoundKey `// Was isolated`

Explanation :
InvShiftRows and InvSubBytes commute : they operate at a different granularity. InvShiftRows works on bytes whereas InvSubBytes transforms bytes.

Encryption *versus* Decryption

Encryption

- AddRoundKey `// Isolated`
- SubBytes
  ShiftRows
  MixColumns
  AddRoundKey `// n times ...`
- SubBytes
  ShiftRows
  AddRoundKey `// Last round`

Decryption

- AddRoundKey `// Was last round`
  InvSubBytes
  InvShiftRows
- InvMixColumns
  AddRoundKey `// n times ...`
  InvShiftRows
  InvSubBytes
- AddRoundKey `// Was isolated`

Explanation :

$$\underset{②}{\text{InvMixColumns}}\ (\text{state} \underset{①}{\oplus} \text{K}) = \underset{①}{\text{InvMixColumns}}\ (\text{state}) \underset{②}{\oplus} \text{InvMixColumns}(\ \text{K}\ )\ .$$

Coding Styles

C language : struct aes_128_enc { /* ... */ };

```
aes_128_enc* encrypt( aes_128_enc* this )
{
   this = KeySchedule( AddRoundKey( this ), 0 ); // Updating the key
   for( int round=1; round<Nr; ++round )
   {
      this = KeySchedule( AddRoundKey( MixColumns( ShiftRows( SubBytes( this ) ) ), round );
   }
   return AddRoundKey( ShiftRows( SubBytes( this ) ) );
}
```

C++ language : class aes_128_enc { /* ... */ };

```
aes_128_enc& aes_128_enc::encrypt()
{
   AddRoundKey().KeySchedule( 0 ); // Updating the key
   for( int round=1; round<Nr; ++round )
   {
      SubBytes().ShiftRows().MixColumns().AddRoundKey().KeySchedule( round );
   }
   SubBytes().ShiftRows().AddRoundKey();
   return *this;
}
```

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \begin{pmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{pmatrix}$$

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \begin{pmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{pmatrix} \begin{pmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{pmatrix}$$

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} c_{0,j} \oplus \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} c_{1,j} \oplus \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} c_{2,j} \oplus \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} c_{3,j}$$

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \begin{pmatrix} 02 \\ 01 \\ 01 \\ 03 \end{pmatrix} b_{0,j} \oplus \begin{pmatrix} 03 \\ 02 \\ 01 \\ 01 \end{pmatrix} b_{1,j+1} \oplus \begin{pmatrix} 01 \\ 03 \\ 02 \\ 01 \end{pmatrix} b_{2,j+2} \oplus \begin{pmatrix} 01 \\ 01 \\ 03 \\ 02 \end{pmatrix} b_{3,j+3}$$

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \bigoplus_{k=0}^{3} (\text{MC}_k)\, b_{k,j+k}$$

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \bigoplus_{k=0}^{3} (\text{MC}_k) S(a_{k,j+k})$$

Optimization on 32-bit Machines (see [FD01, GW08])

Notations :

$$(a_{0\leq i\leq 3,0\leq j\leq 3}) \xrightarrow{\text{SB}} (b_{i,j}) \xrightarrow{\text{SR}} (c_{i,j}) \xrightarrow{\text{MC}} (d_{i,j}) \xrightarrow{\text{ARK}} (e_{i,j})$$

Computation of column $0 \leq j \leq 3$ **in one round** :

$$\begin{pmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{pmatrix} = \begin{pmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{pmatrix} \oplus \bigoplus_{k=0}^{3} \text{Te}_k(a_{k,j+k})$$

Te_k are four memories of 256 words of 32 bits.

OpenSSL AES code

```
/* map byte array block to cipher state and add initial round key: */
s0 = GETU32(in     ) ^ rk[0];
s1 = GETU32(in +  4) ^ rk[1];
s2 = GETU32(in +  8) ^ rk[2];
s3 = GETU32(in + 12) ^ rk[3];

/* round 1: */
t0 = Te0[s0 >> 24] ^ Te1[(s1 >> 16) & 0xff] ^ Te2[(s2 >>  8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[ 4];
t1 = Te0[s1 >> 24] ^ Te1[(s2 >> 16) & 0xff] ^ Te2[(s3 >>  8) & 0xff] ^ Te3[s0 & 0xff] ^ rk[ 5];
t2 = Te0[s2 >> 24] ^ Te1[(s3 >> 16) & 0xff] ^ Te2[(s0 >>  8) & 0xff] ^ Te3[s1 & 0xff] ^ rk[ 6];
t3 = Te0[s3 >> 24] ^ Te1[(s0 >> 16) & 0xff] ^ Te2[(s1 >>  8) & 0xff] ^ Te3[s2 & 0xff] ^ rk[ 7];
/* round 2: */
s0 = Te0[t0 >> 24] ^ Te1[(t1 >> 16) & 0xff] ^ Te2[(t2 >>  8) & 0xff] ^ Te3[t3 & 0xff] ^ rk[ 8];
s1 = Te0[t1 >> 24] ^ Te1[(t2 >> 16) & 0xff] ^ Te2[(t3 >>  8) & 0xff] ^ Te3[t0 & 0xff] ^ rk[ 9];
s2 = Te0[t2 >> 24] ^ Te1[(t3 >> 16) & 0xff] ^ Te2[(t0 >>  8) & 0xff] ^ Te3[t1 & 0xff] ^ rk[10];
s3 = Te0[t3 >> 24] ^ Te1[(t0 >> 16) & 0xff] ^ Te2[(t1 >>  8) & 0xff] ^ Te3[t2 & 0xff] ^ rk[11];

/* ... */

/* apply last round and map cipher state to byte array block: */
s0 =
        (Te2[(t0 >> 24)       ] & 0xff000000) ^
        (Te3[(t1 >> 16) & 0xff] & 0x00ff0000) ^
        (Te0[(t2 >>  8) & 0xff] & 0x0000ff00) ^
        (Te1[(t3      ) & 0xff] & 0x000000ff) ^
        rk[0];
PUTU32(out     , s0);

/*                */
```

RSA for signature (+ use PKCS#1)

- **Public parameters** : $N = pq$, where p, q are two large primes, e
- **Private parameters** : p, q, $d = e^{-1} \mod (p-1)(q-1)$
- $H(m)$ is the hash of the message to be signed

Signature

- $S = H(m)^d \mod N$

Verification

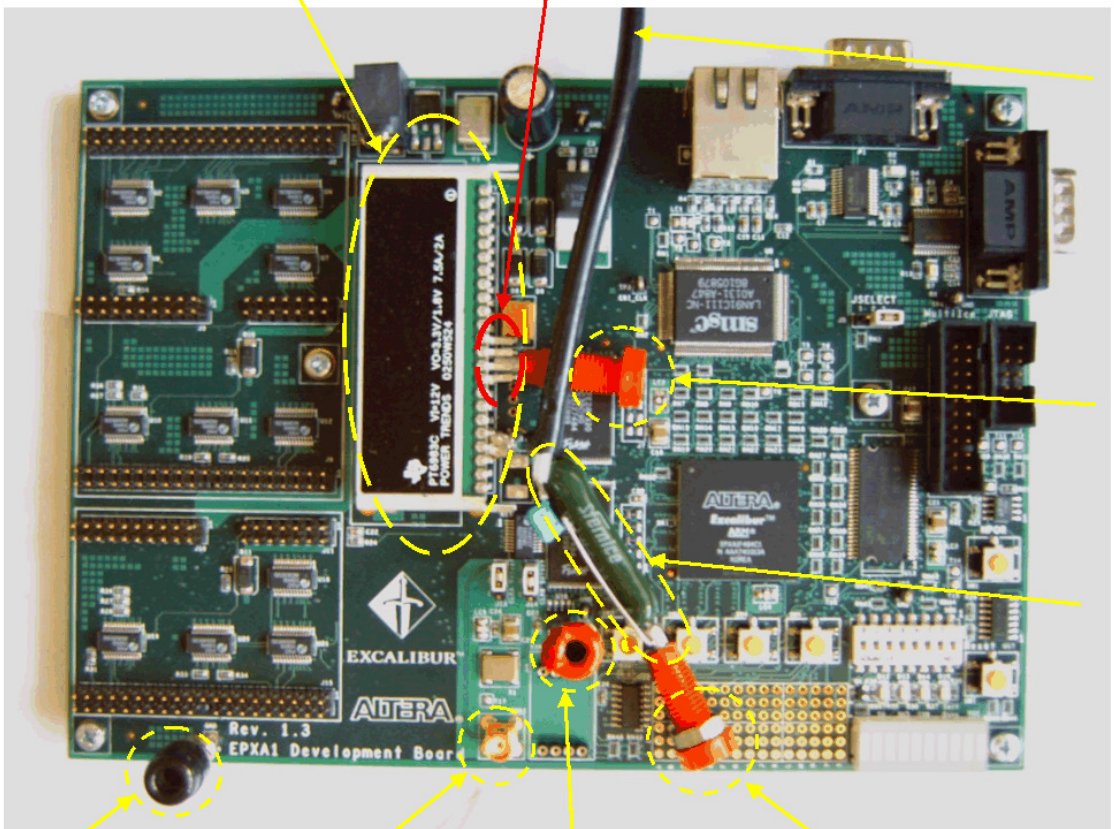
- $S^e \stackrel{?}{=} H(m) \mod N$

Presentation Outline

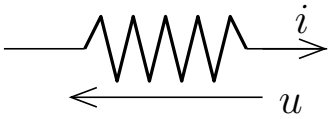
Overview of attacks

	Side-channel attack	Fault attack
RSA	1	1
AES	100	2

ALTERA Excalibur evaluation board "customized for DPA"

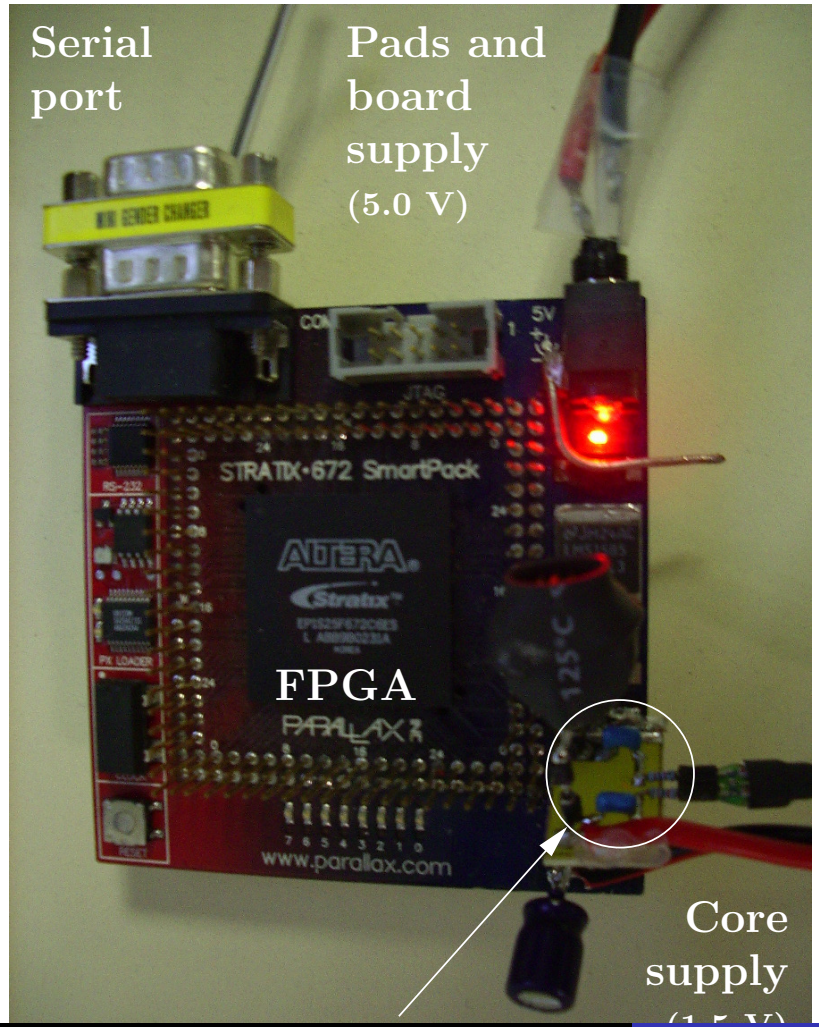


Resistor:

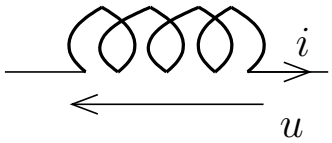


$u = Ri$, where R is the resistance.

Parallax ALTERA Stratix board "customized for DPA" [GSD+08]



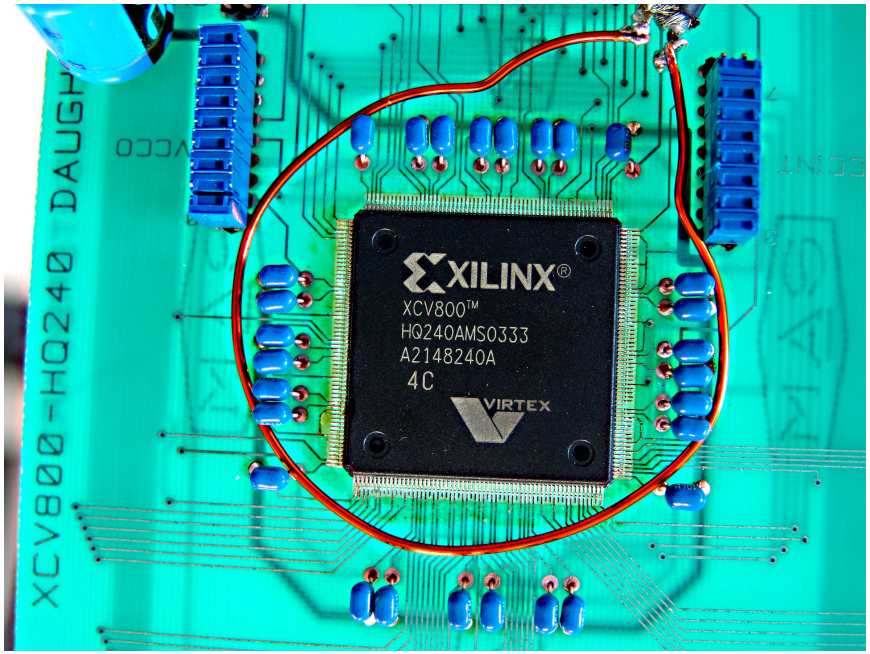
Inductor:



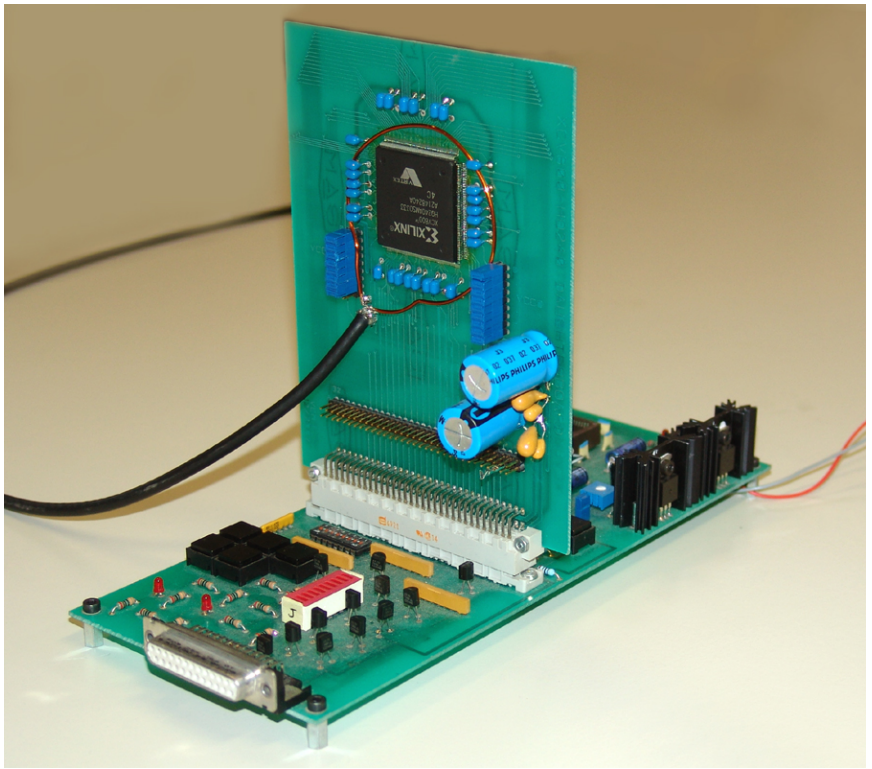
$u = L\frac{\partial i}{\partial t}$, where L is the inductance.

XCV800 home-made board suitable for **global** EMA

Antenna

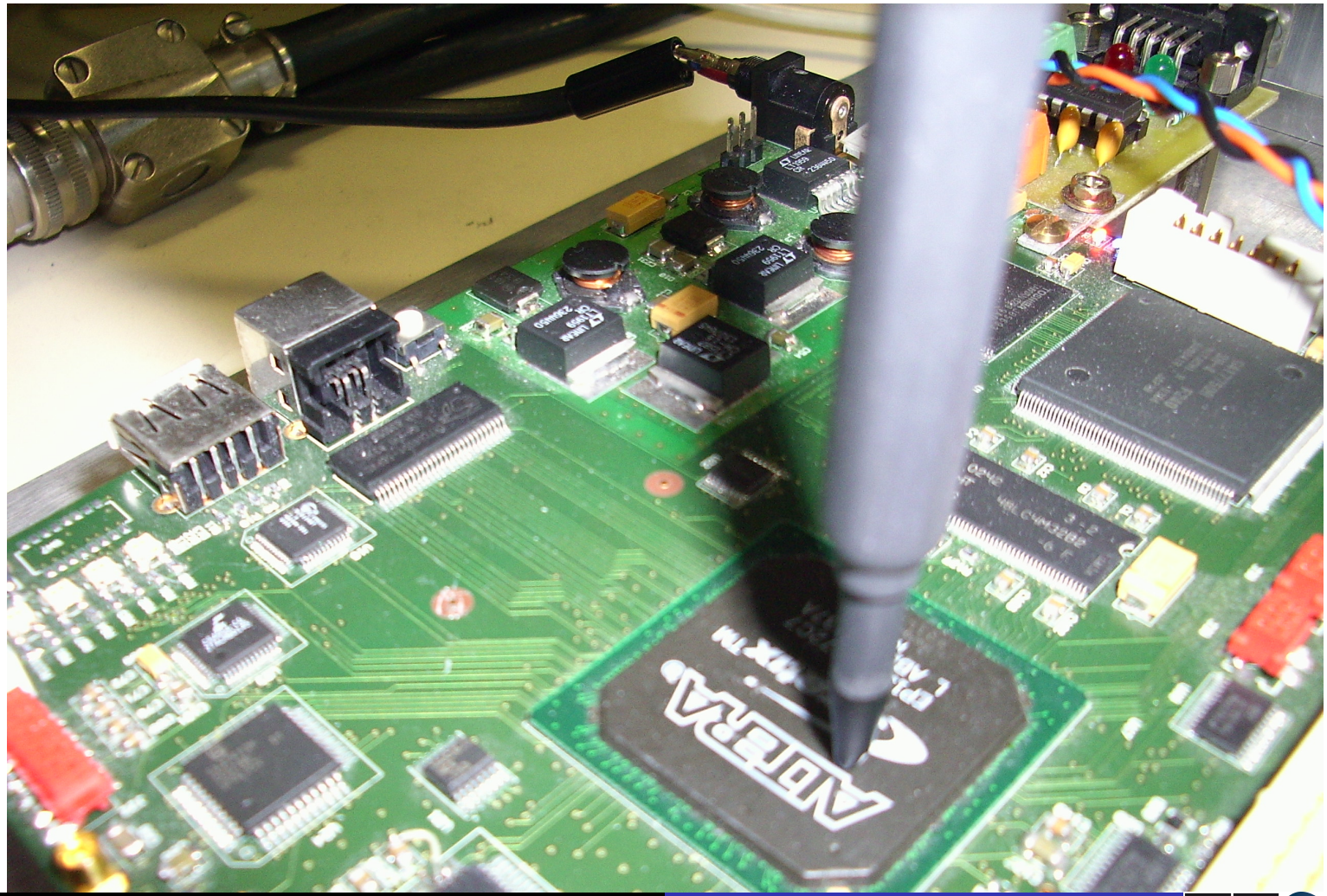


Acquisition setup

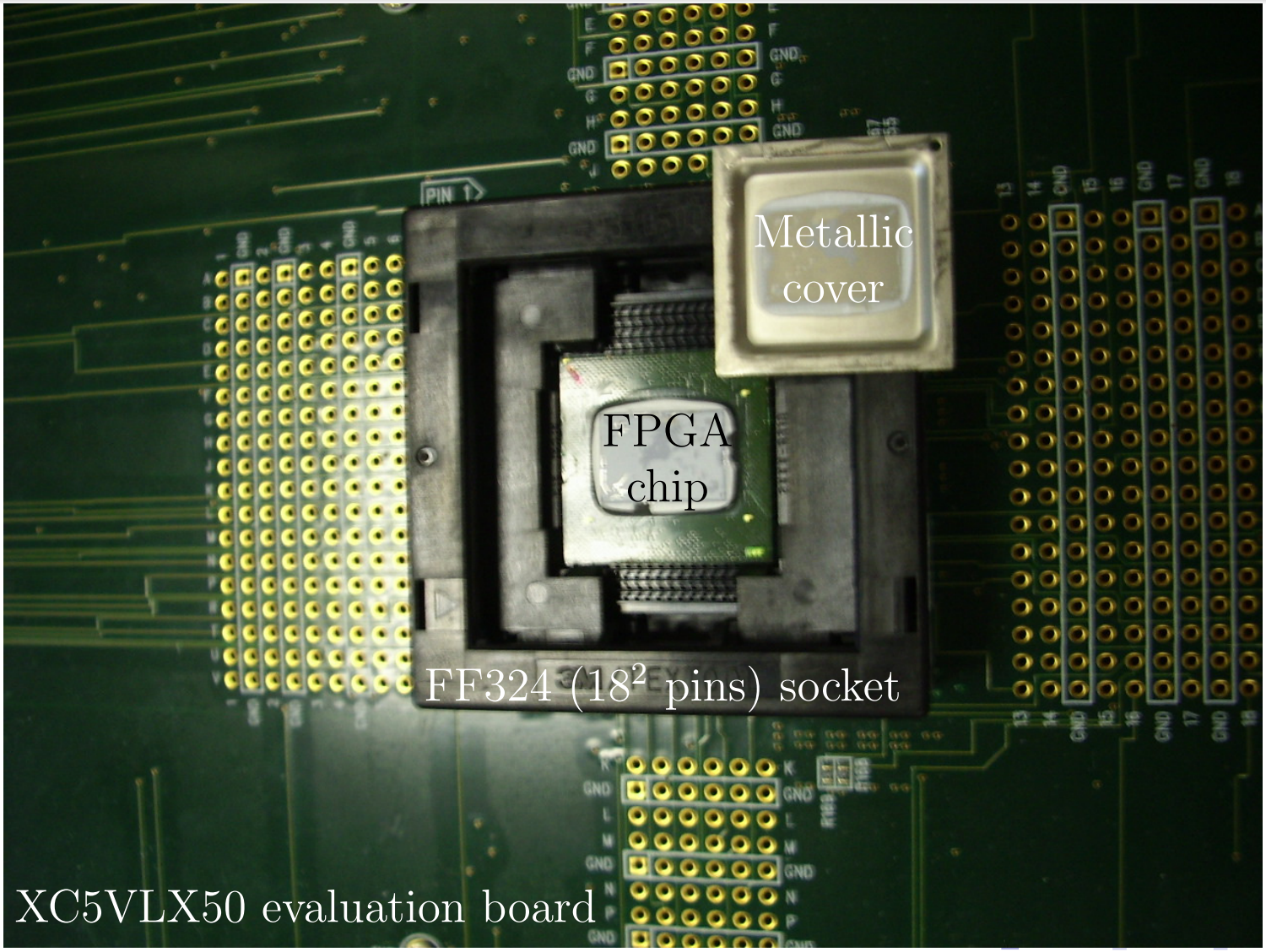


Pictures are courtesy of ESAT, Katholieke Universiteit Leuven, Belgium, (Elke De Mulder [MBO+05]).

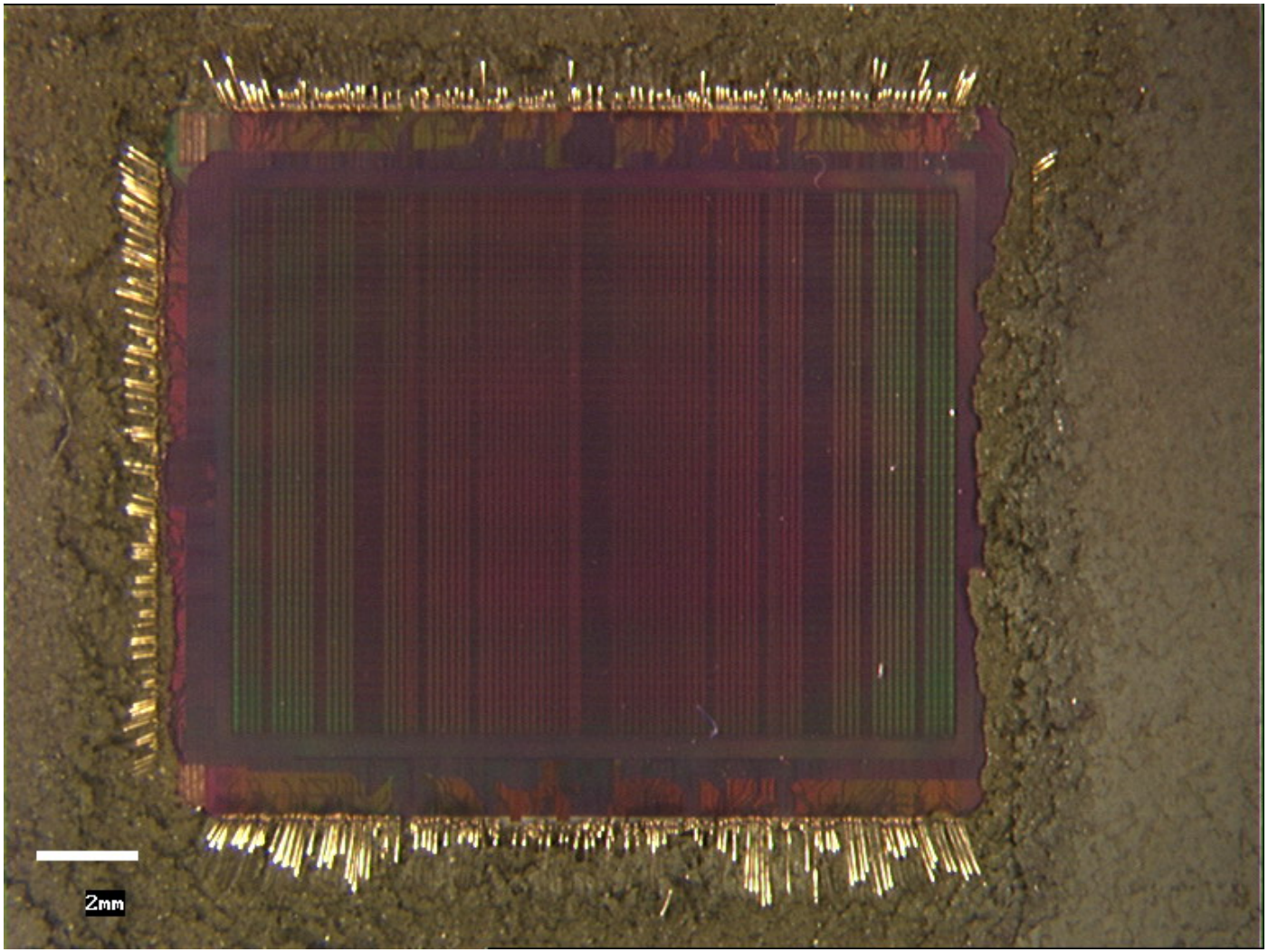
In-house ALTERA Stratix "as is" suitable for **local** EMA [SGM09, SGD+09]



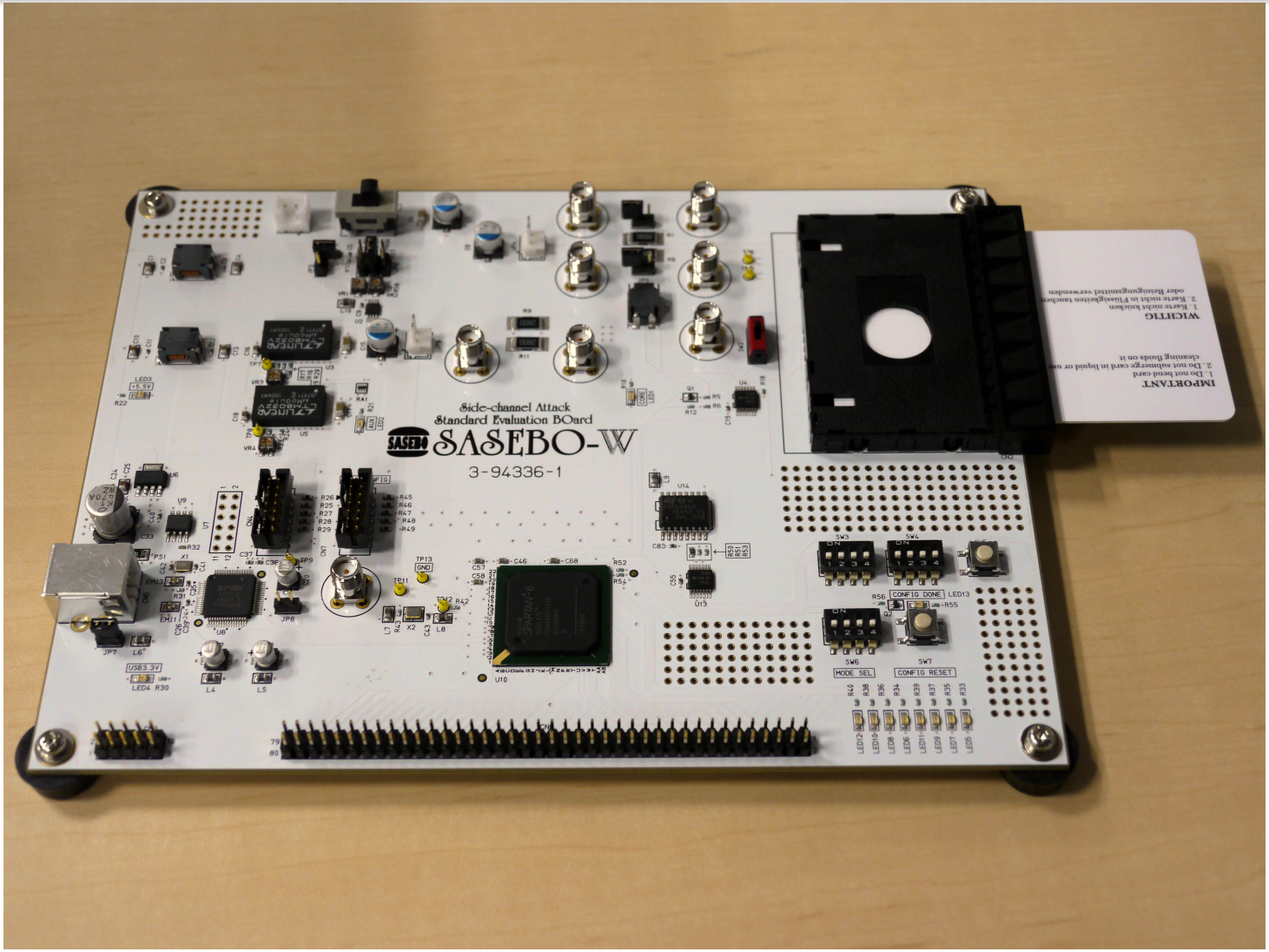
XILINX Virtex-5 evaluation board "customized for EMA"



ALTERA Stratix with chemical preparation for EMA



New SASEBO-W for hardware and software evaluations



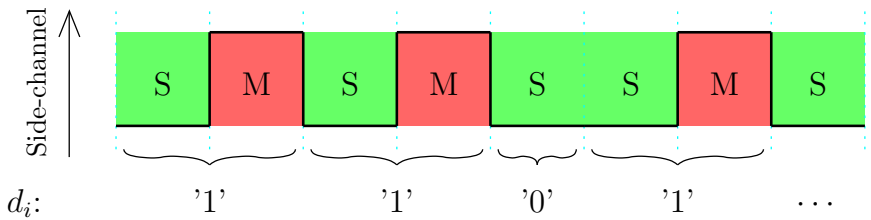
Timing Attacks & Simple Power Analyzes (SPA)

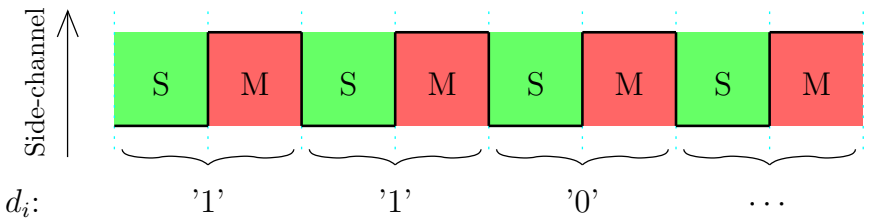
Square-and-Multiply RSA

Inputs : M, d
$R = 1$;
for $i = |d| - 1$; $i \geq 0$; $i--$ **do**
  $R = R^2$;
  /* Unbalanced branching */
  if $d_i == 1$ **then**
    $R = R \times M$;
  end if
end for
Return $R = M^d$;

Montgomery ladder exponentiation

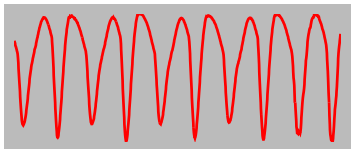
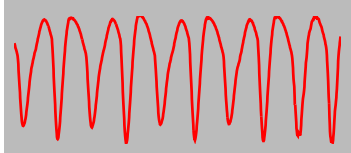
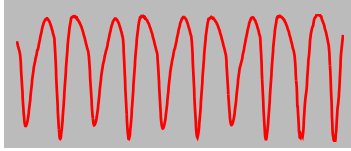
Inputs : M, d
$R = 1$; $S = M$;
for $i = |d| - 1$; $i \geq 0$; $i--$ **do**
  /* Balanced branching */
  if $d_i == 1$ **then**
    $R = R \times S$; $S = S^2$;
  else
    $S = S \times R$; $R = R^2$;
  end if
end for
Return $R = M^d$;



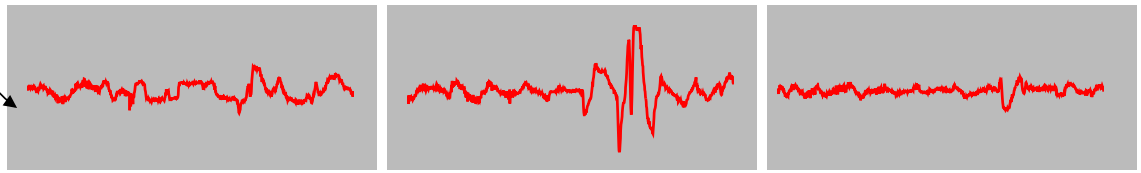


SCA Example on the Last Round of AES

(Observation O, ciphertext C) | Model M: $\forall k_{0,0} \in \{\texttt{00}, \texttt{01}, \cdots, \texttt{ff}\}, \mathrm{ISB}(c_{0,0} \oplus k_{0,0}) \oplus c_{0,0}$

1:	 ab 20 4f 0c e7 31 c2 66 89 07 a0 fd 5a 59 8d 73	$\mathrm{ISB}(\texttt{ab} \oplus \texttt{00}) \oplus \texttt{ab} = \texttt{a5}$ $\Rightarrow M = \mathrm{HW}(\texttt{a5}) = \mathbf{4}$	$\mathrm{ISB}(\texttt{ab} \oplus \texttt{01}) \oplus \texttt{ab} = \texttt{c9}$ $\Rightarrow M = \mathrm{HW}(\texttt{c9}) = \mathbf{4}$	$\mathrm{ISB}(\texttt{ab} \oplus \texttt{ff}) \oplus \texttt{ab} = \texttt{56}$ $\Rightarrow M = \mathrm{HW}(\texttt{56}) = \mathbf{4}$
2:	 09 11 8d 32 d6 91 dd 84 54 e4 89 e2 07 aa 31 05	$\mathrm{ISB}(\texttt{09} \oplus \texttt{00}) \oplus \texttt{09} = \texttt{49}$ $\Rightarrow M = \mathrm{HW}(\texttt{49}) = \mathbf{3}$	$\mathrm{ISB}(\texttt{09} \oplus \texttt{01}) \oplus \texttt{09} = \texttt{b6}$ $\Rightarrow M = \mathrm{HW}(\texttt{b6}) = \mathbf{5}$	$\mathrm{ISB}(\texttt{09} \oplus \texttt{ff}) \oplus \texttt{09} = \texttt{df}$ $\Rightarrow M = \mathrm{HW}(\texttt{df}) = \mathbf{7}$
q:	 2c 83 82 29 10 83 91 00 df 84 64 ec a2 91 c6 76	$\mathrm{ISB}(\texttt{2c} \oplus \texttt{00}) \oplus \texttt{2c} = \texttt{6e}$ $\Rightarrow M = \mathrm{HW}(\texttt{6e}) = \mathbf{5}$	$\mathrm{ISB}(\texttt{2c} \oplus \texttt{01}) \oplus \texttt{2c} = \texttt{d6}$ $\Rightarrow M = \mathrm{HW}(\texttt{d6}) = \mathbf{5}$	$\mathrm{ISB}(\texttt{2c} \oplus \texttt{ff}) \oplus \texttt{2c} = \texttt{85}$ $\Rightarrow M = \mathrm{HW}(\texttt{85}) = \mathbf{3}$

ρ

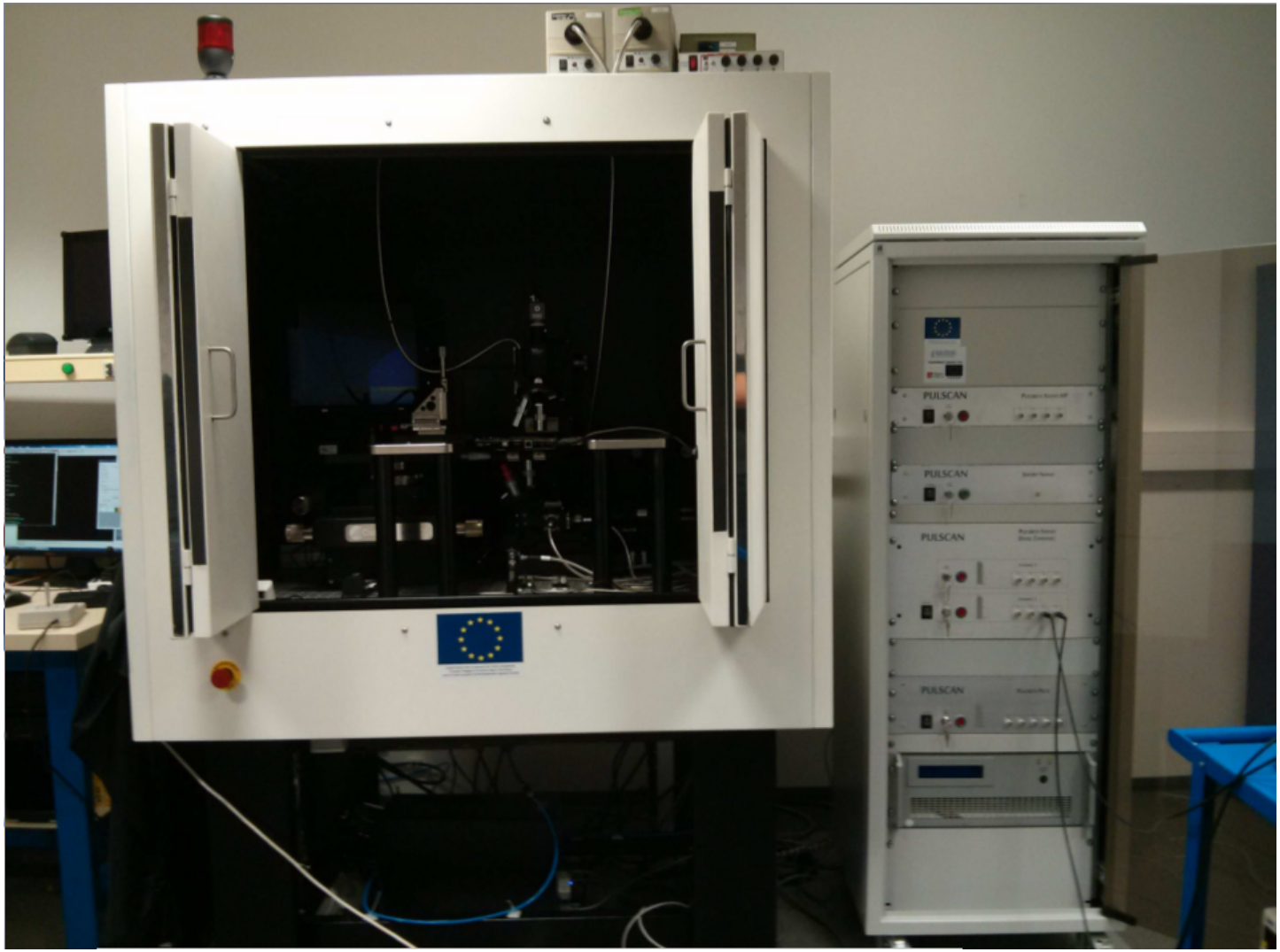


The couple $(t, k_{0,0})$ that maximes the differential curves indicates the most leaking time and key.

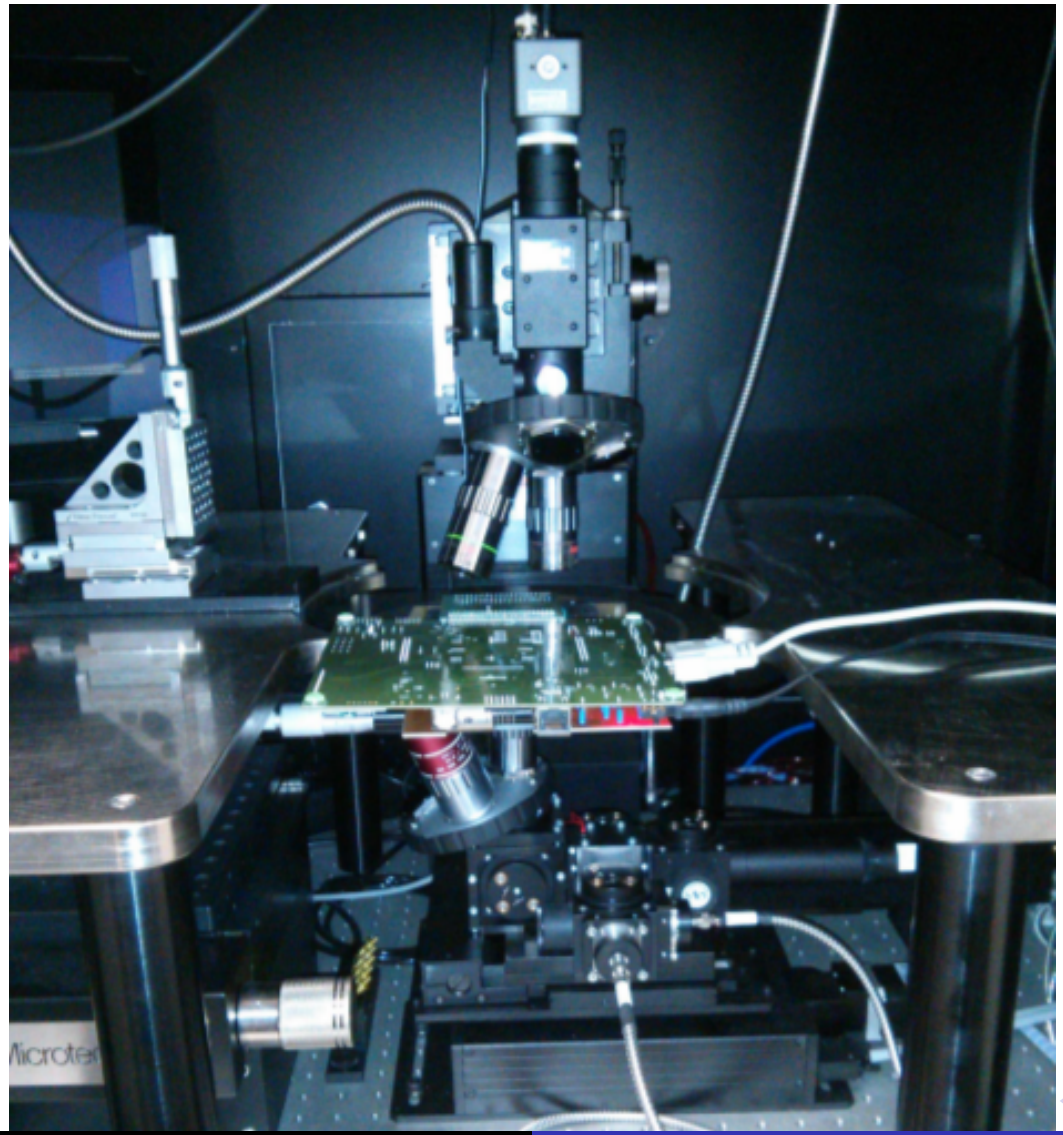
Realignement

[WISTP '11, Guilley et al.] [GKLD11]

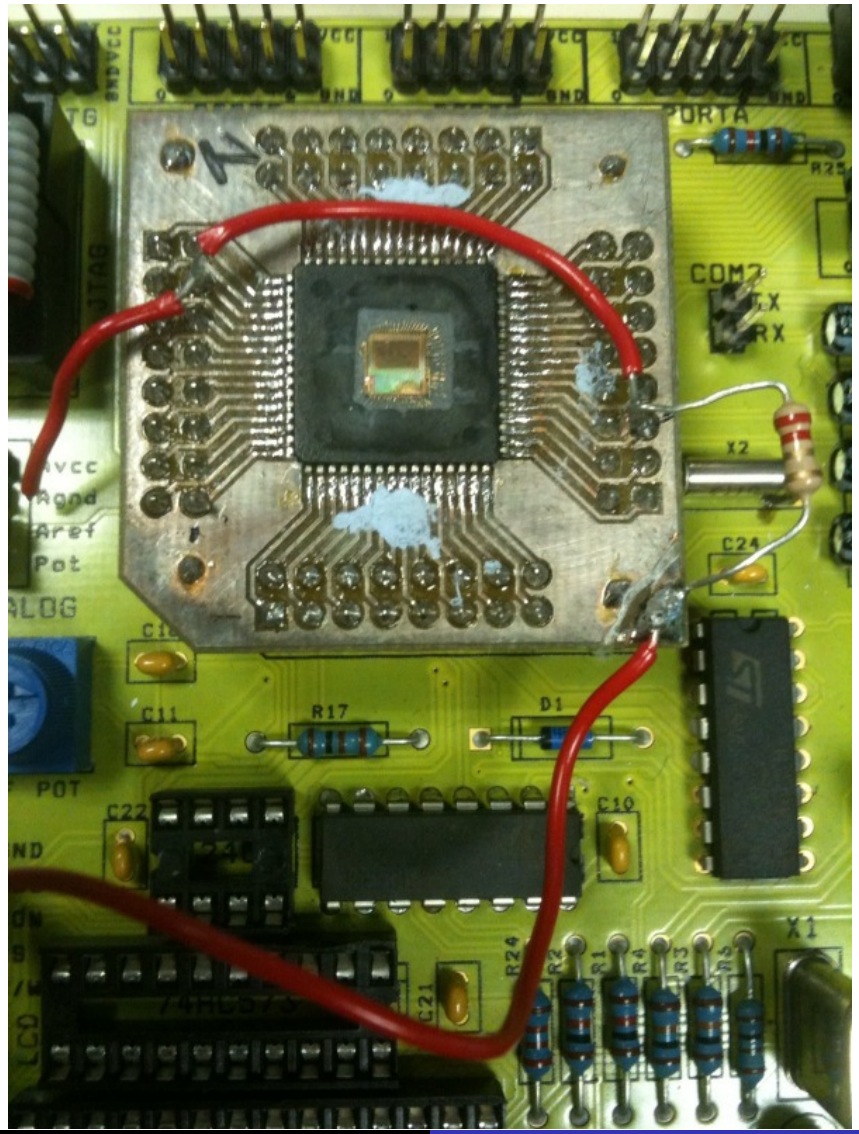
Laser station



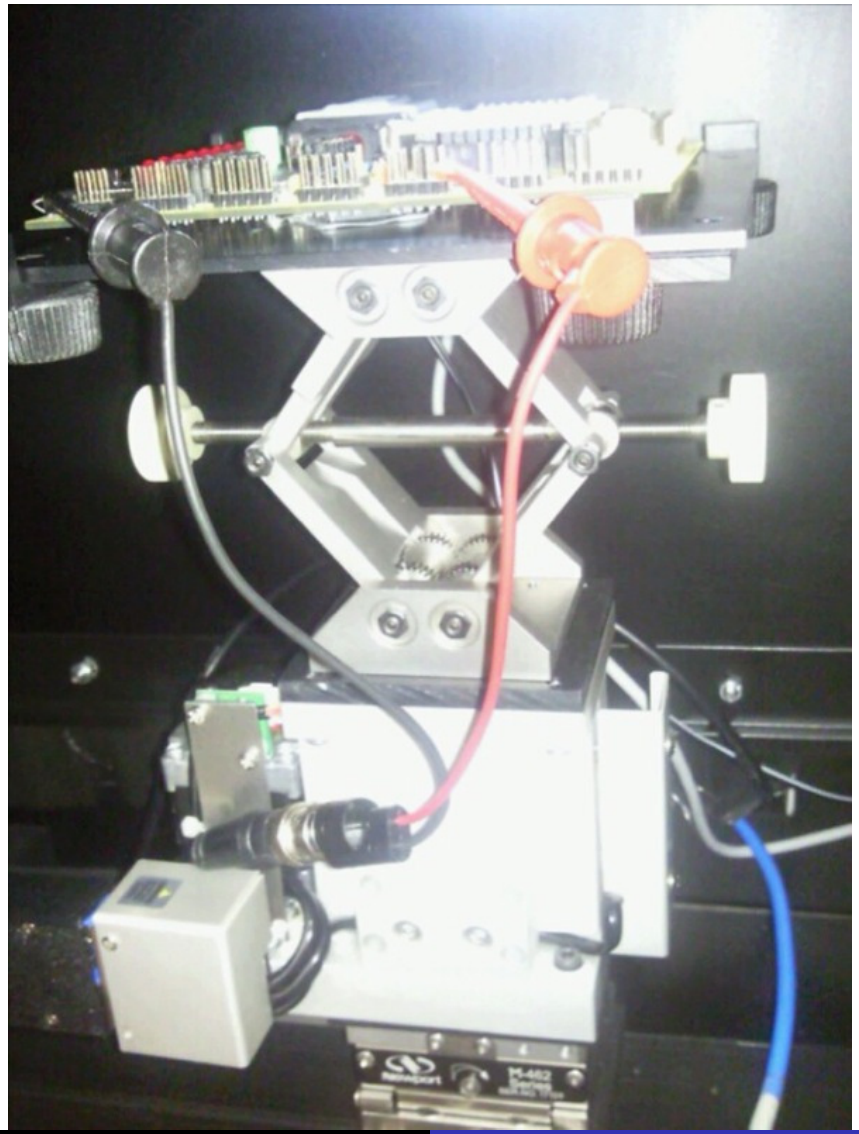
Laser station



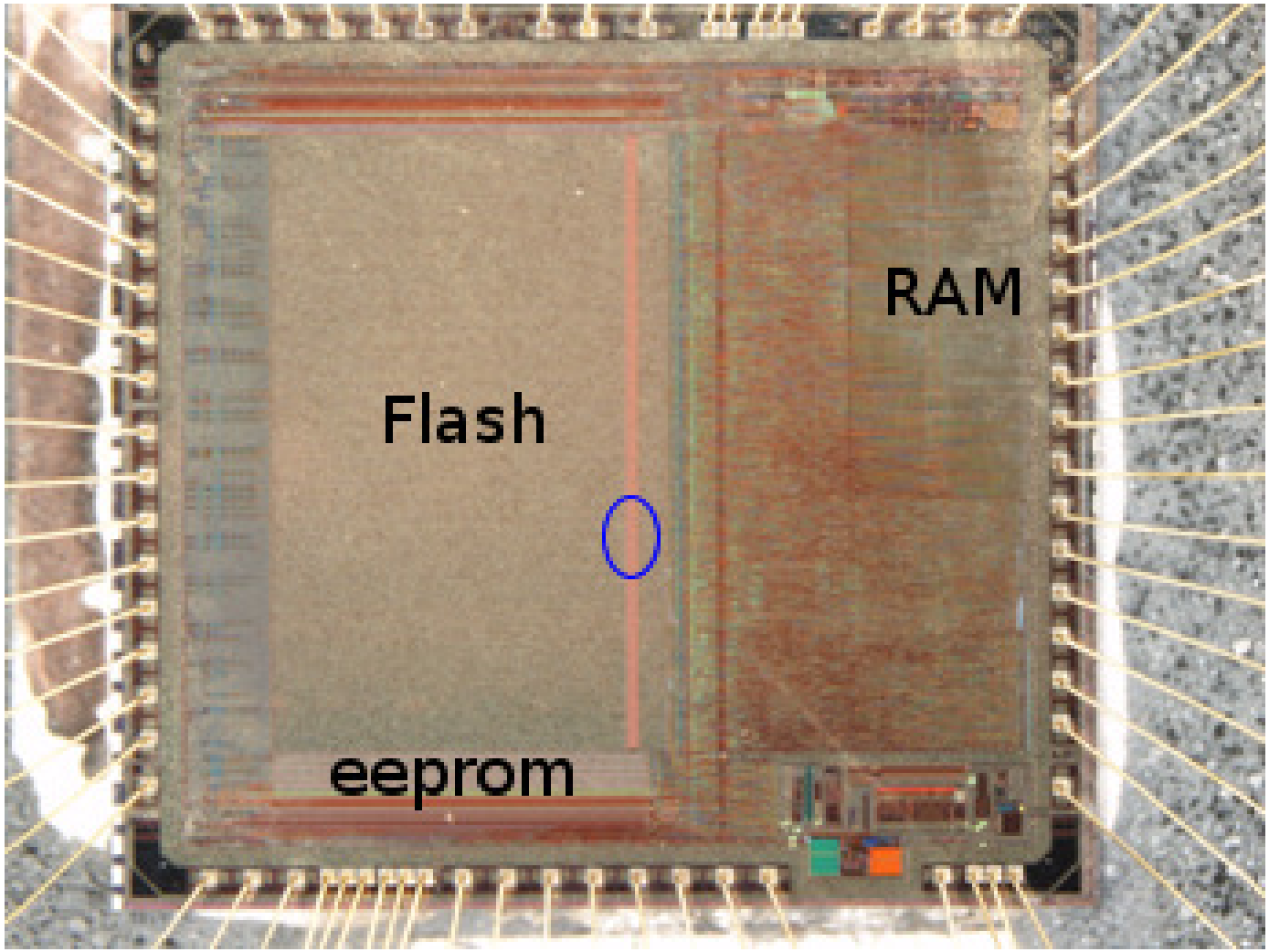
[1/3] Example @ TELECOM-ParisTech ..... PCB



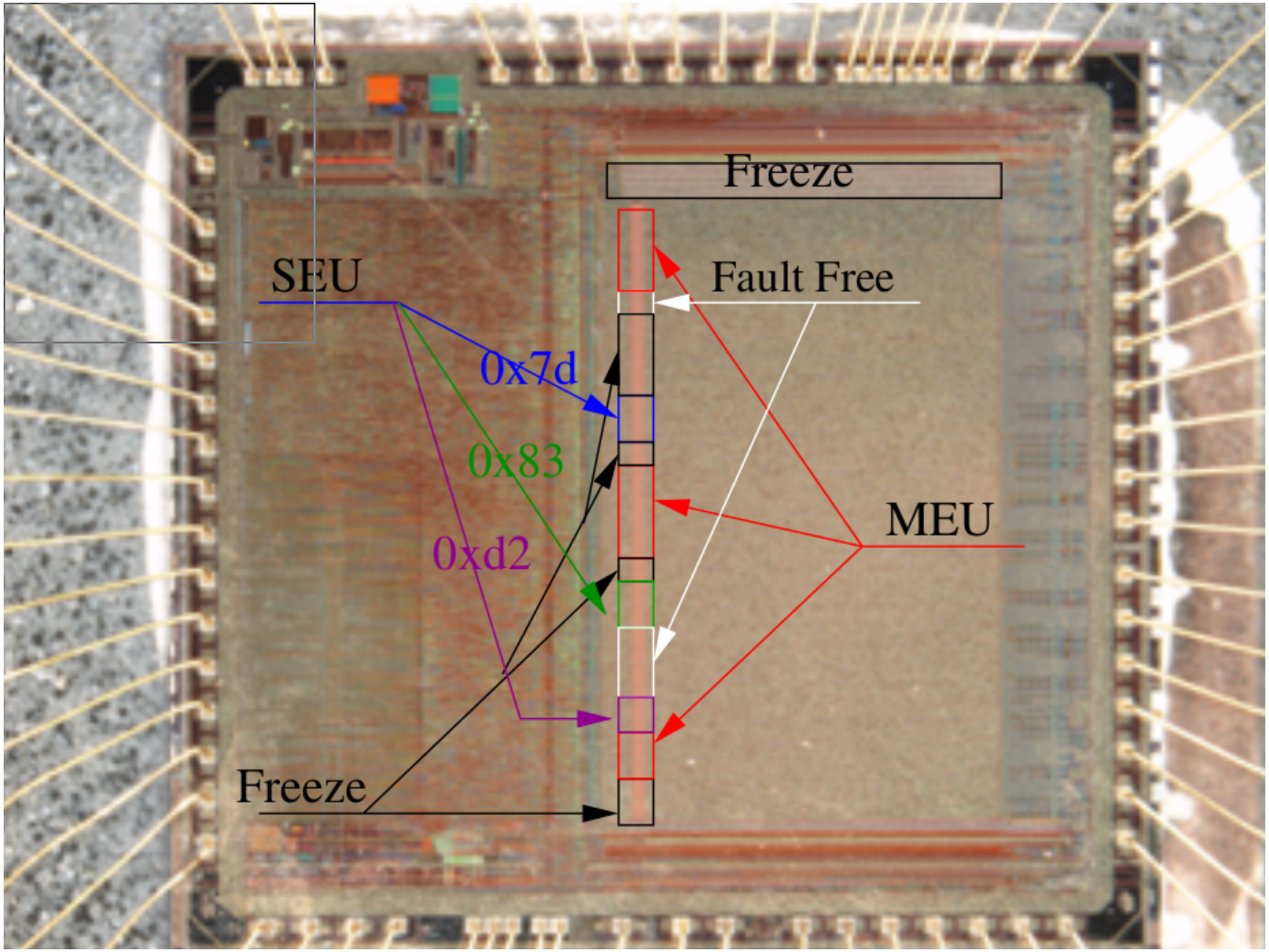
[2/3] Example @ TELECOM-ParisTech ..... Setup



[3/3] Example @ TELECOM-ParisTech . . . . . ASIC



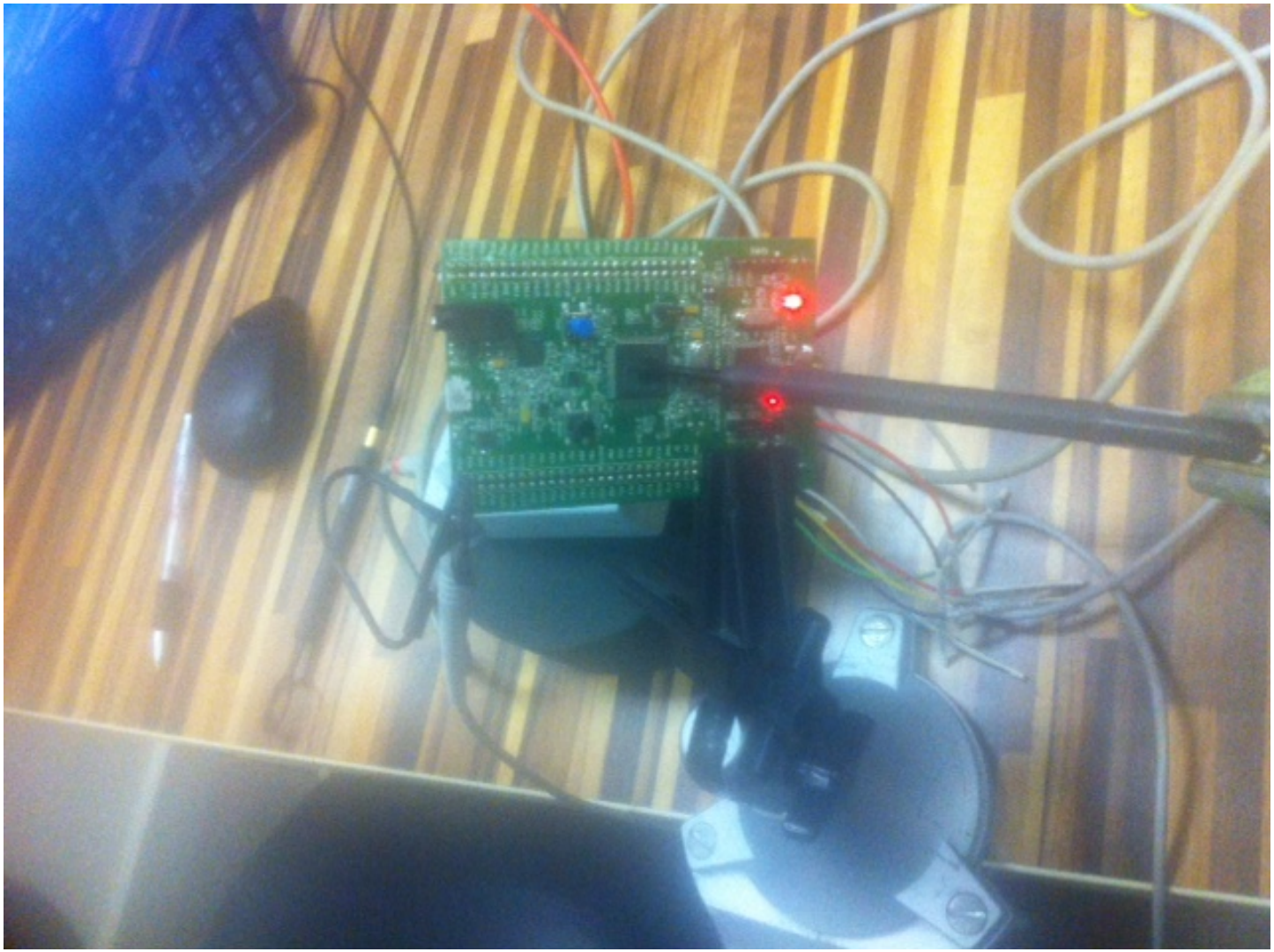
[3/3] Example @ TELECOM-ParisTech ..... ASIC



EMI disturbance system

Example of setup for EMI

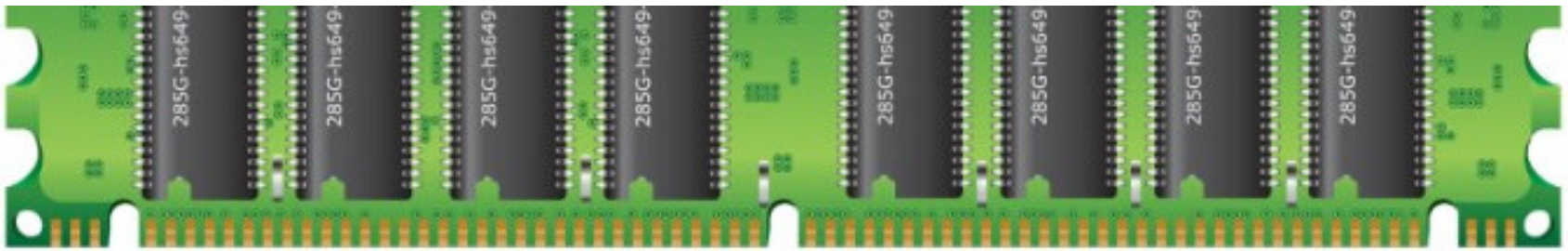
Example of setup for EMI

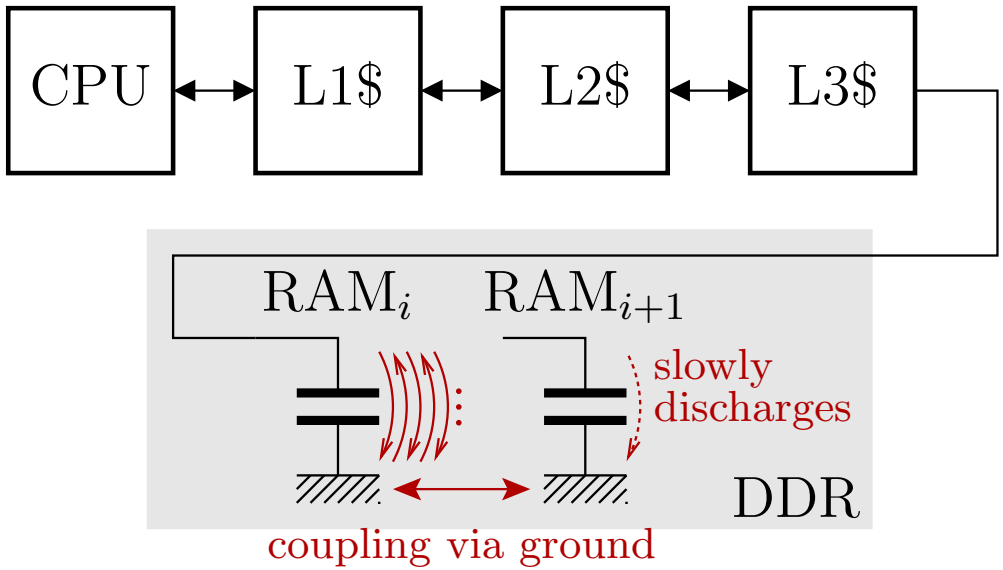


Example of setup for EMI

Flipping Bits in Memory Without Accessing Them

http://users.ece.cmu.edu/~yoonguk/papers/kim-isca14.pdf [KDK+14] — RowHammer.





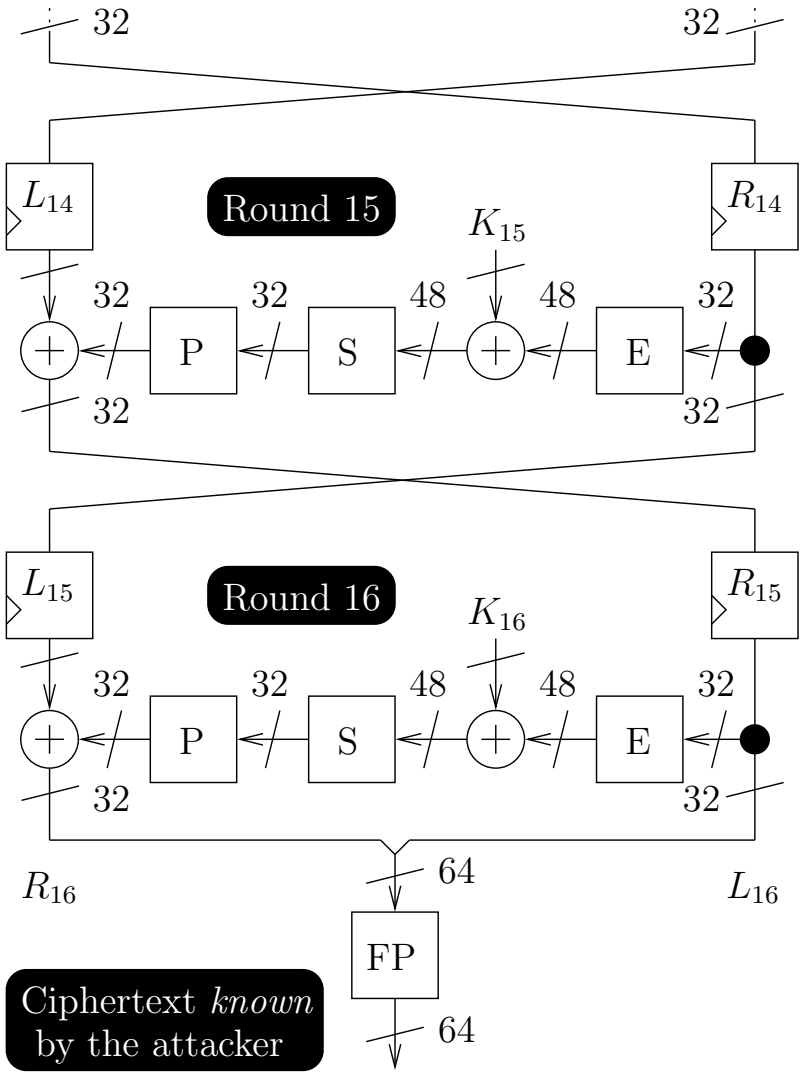
Bellcore attack against RSA signatures with CRT

- Signature S of message x : $S \doteq x^d \mod N$, with $N = p \cdot q$.
- Using Chinese Remainder Theorem (CRT), the signature can be simplified as :
  - $S_1 = x^{d \mod \phi(p)} \mod p = x^{d \mod (p-1)} \mod p$ and
  - $S_2 = x^{d \mod \phi(q)} \mod q = x^{d \mod (q-1)} \mod q$, both operations working on half bitwidth.
- The signature is obtained back using the two constants :

$$\begin{cases} a = 0 \mod q \\ a = 1 \mod p \end{cases} \quad \text{and} \quad \begin{cases} b = 1 \mod q \\ b = 0 \mod p \end{cases}$$

- $S = a \cdot S_1 + b \cdot S_2 \mod N$.
- Now, if S_1 happens to be faulty : $S_1 \rightarrow \widehat{S_1}$ for whatever reason,
- $\gcd(S - \widehat{S}, N) = \gcd(a \cdot (S_1 - \widehat{S_1}), N) = q$.

DFA Attack Setting



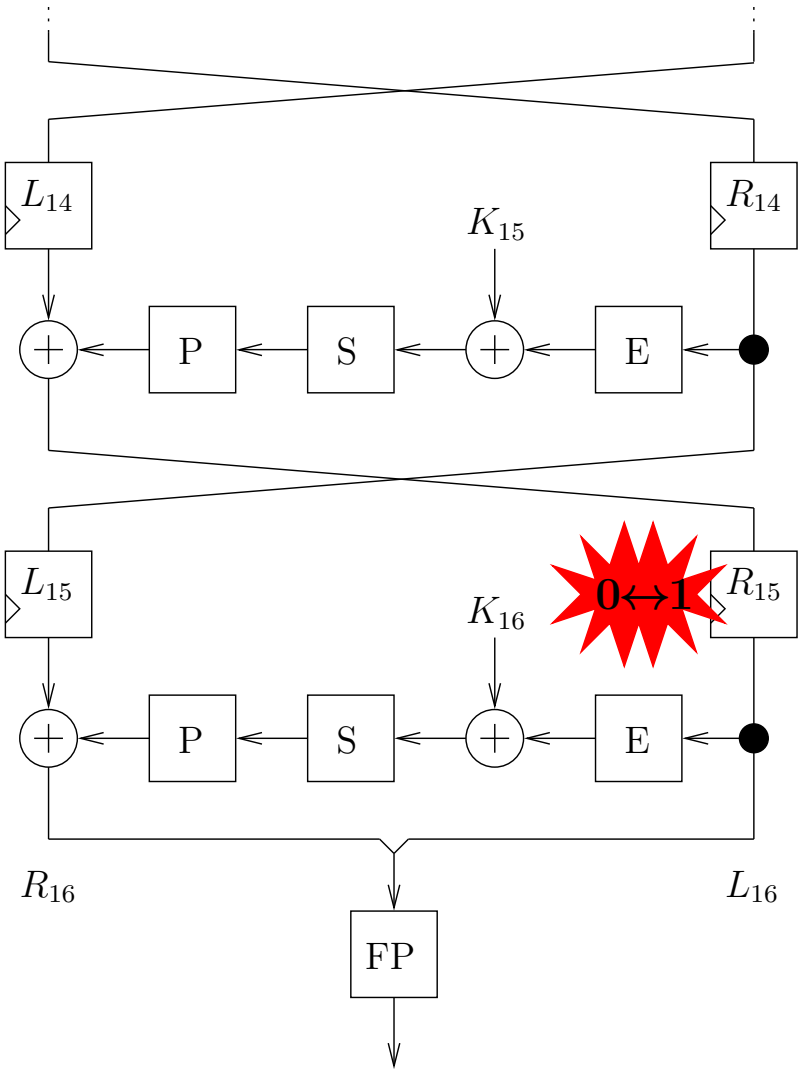
DFA Assumptions

- Unrolled implementation.
- Single bit-flips on any right register R_i, for $i \in [1, 16]$.
- Ciphertext-*only* attack.

DES Properties

- All the DES constitutive boxes, but the S, are linear : $f(x \oplus a) = f(x) \oplus f(a)$, for $f \in \{\mathsf{Id}, \mathsf{P}, \mathsf{E}, \mathsf{FP}\}$.
- $L_{16} = R_{15}$.

One Fault Occurs in R_{15}



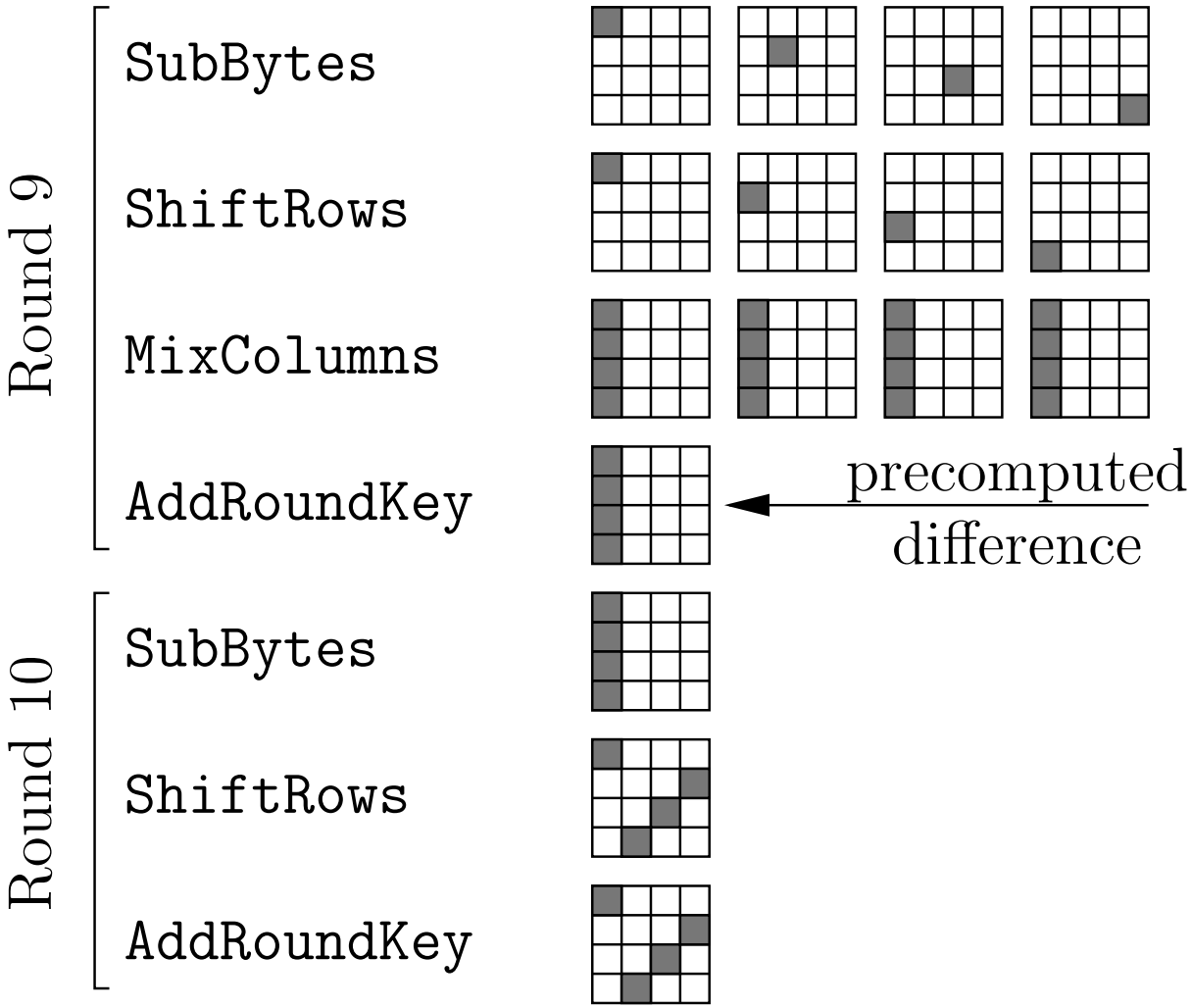
What Has Happened ?

Bit $b \in [1, 32]$ of R_{15} is flipped.

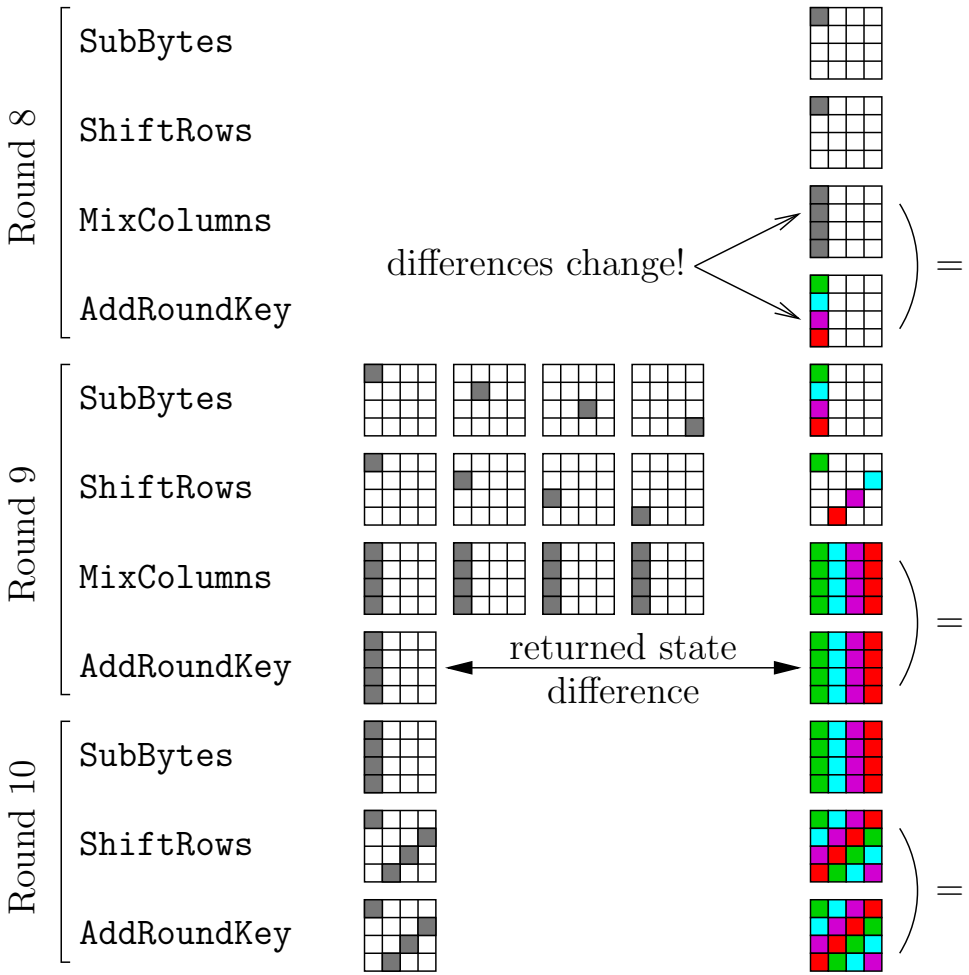
Attack Scenario

- Find b by looking in L_{16}.
- Deduce which S_i, $i \in [1, 8]$, (there can be two of them) has output a wrong value.
- Solve the equation couple :
$$\begin{cases} R_{16} = L_{15} \oplus S_i(K_{16} \oplus R_{15}), \\ \widetilde{R_{16}} = L_{15} \oplus S_i(K_{16} \oplus \widetilde{R_{15}}). \end{cases}$$
- It has $\approx$ four 6-bit solutions.

G. Piret & J.-J. Quisquater in 2003



G. Piret & J.-J. Quisquater in 2003

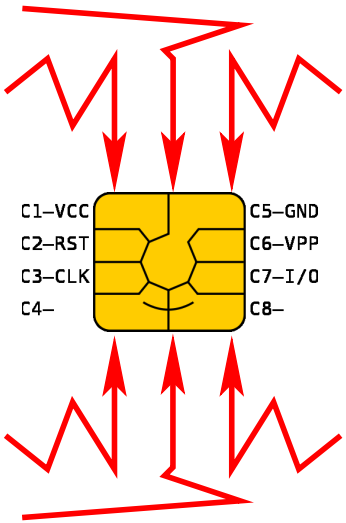


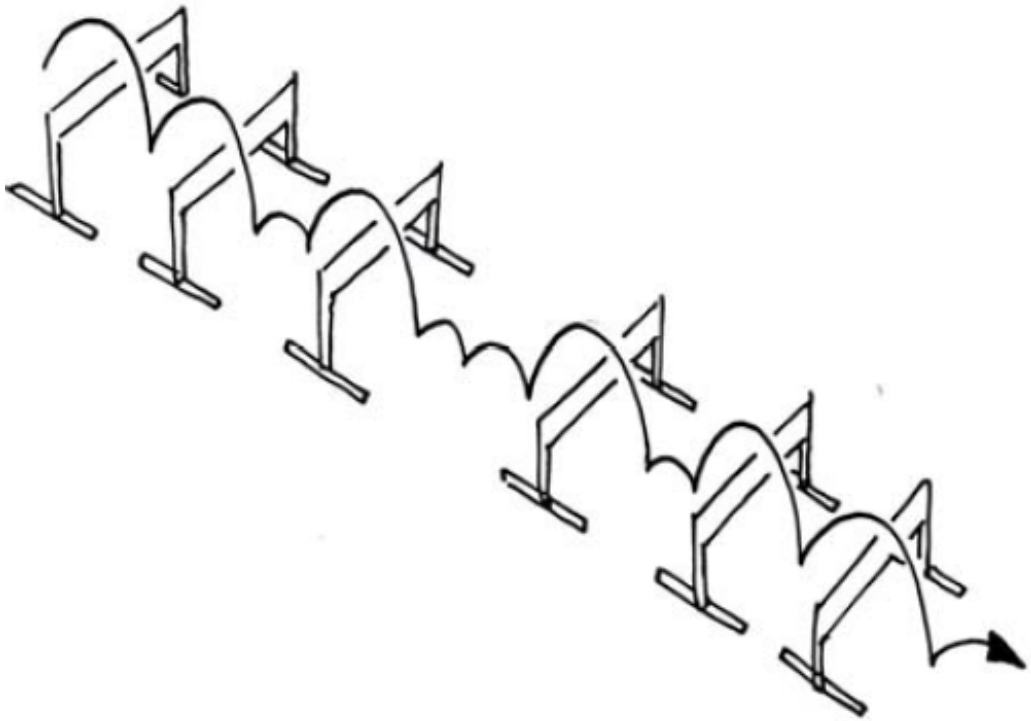
Kill 4 birds with one stone

A fault at round 8 yields 4 faults at round 9 ! This is optimal. . .

Presentation Outline

- **Defense** : against all attacks
- **Attack** : one is enough !





Generic protections against SCA + FIA

Against SCA

- Randomize
  - Data : with masks
  - Control : with shuffling
- Balance
- Tolerate : resilience

Against FIA

- Verification
  - Data : with codes
  - Control : with check-points
- Tolerate :
  - denial of exploitation
  - infective countermeasures

Asymmetric

- Arrange for the system not to contain sensitive data
  - *e.g.* only signatures and certificates

Symmetric

- Whitebox crypto
  - Save the PIN hashed instead of encrypted
  - Save the block cipher as a codebook (*theoretical*)
- Leakage resilient and/or fault injection resilient
  - See next slides

Observation attacks are easily thwarted by masking :
$\forall r_1, r_2, r_3 \neq 0$,
$\left((M + r_1 \times N)^{d + r_2 \times \phi(N)} \mod r_3 \times N\right) \mod N = M^d \mod N$,
hence multiple degrees of freedom to mask cryptographic parameters [Joy03].

Perturbation attacks are fought thanks to similar properties :

- Randomness can also be injected within the algorithm, so as to enable verifications afterwards [BHT09].

Or use $\texttt{Verify} \circ \texttt{Sign} = \texttt{Id}$.

This paper by Jean-Sébastien CORON (@ AsiaCrypt 2009) [CM09] proves that RSA with PSS is provably secure against random fault injection attacks in the random oracle model, and side-channel attacks. **But... [FGL+13]**.

Observation attacks are easily thwarted by masking :

$\forall r_1, r_2, r_3 \neq 0$,

$\left(\left(M \times {r_1}^{-e} \right)^{d + r_2 \times \phi(N)} \times r_1 \mod r_3 \times N \right) \mod N = M^d \mod N$, (see [Koc96, Sec. 10])

hence multiple degrees of freedom to mask cryptographic parameters [Joy03].

Perturbation attacks are fought thanks to similar properties :

- Randomness can also be injected within the algorithm, so as to enable verifications afterwards [BHT09].

Or use $\texttt{Verify} \circ \texttt{Sign} = \texttt{Id}$.

This paper by Jean-Sébastien CORON (@ AsiaCrypt 2009) [CM09] proves that RSA with PSS is provably secure against random fault injection attacks in the random oracle model, and side-channel attacks. **But... [FGL+13]**.

Algorithm 1: RSA implementation (unprotected)

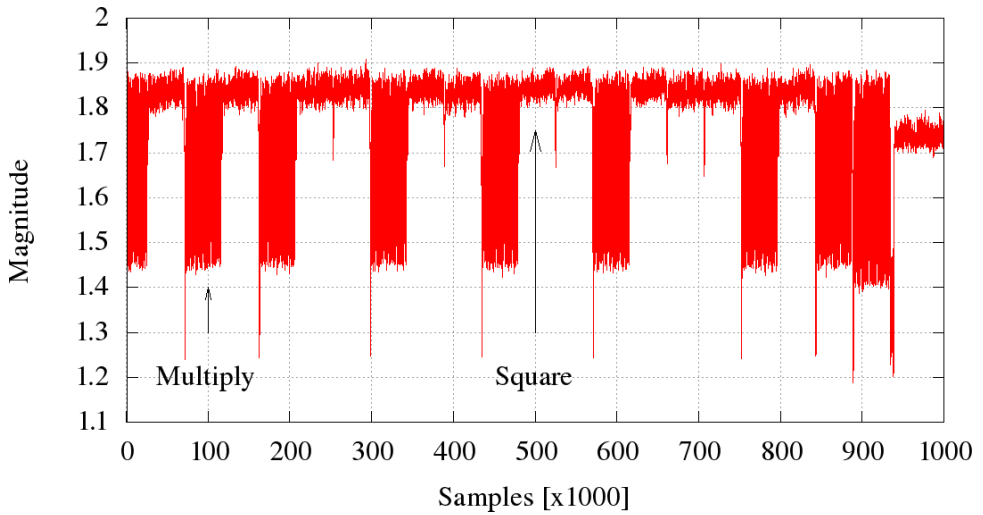
Input : $M \in \mathbb{Z}_N, d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$

1 $R[1] \leftarrow 1$
2 $R[2] \leftarrow M$
3 **for** $i \in [\![0, n-1]\!]$ **do**
4 | **if** $d_i = 1$ **then**
5 | | $R[1] \leftarrow R[2] \cdot R[1]$
6 | **end**
7 | $R[2] \leftarrow R[2]^2$
8 **end**
9 **return** $R[1]$

Algorithm 1: RSA implementation (unprotected)

Input : $M \in \mathbb{Z}_N, d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$

1 $R[1] \leftarrow 1$
2 $R[2] \leftarrow M$
3 **for** $i \in [\![0, n-1]\!]$ **do**
4 | **if** $d_i = 1$ **then**
5 | | $R[1] \leftarrow R[2] \cdot R[1]$
6 | **end**
7 | $R[2] \leftarrow R[2]^2$
8 **end**
9 **return** $R[1]$



Algorithm 2: RSA implementation protected against SCA and FIA.

Input : $M \in \mathbb{Z}_N, d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$ or "Error"

1 Generate a random $r \in \mathbb{Z}_N^*$
2 $R[0] \leftarrow r$
3 $R[1] \leftarrow r^{-1}$
4 $R[2] \leftarrow M$
5 **for** $i \in [\![0, n-1]\!]$ **do**
6 $\quad R[d_i] \leftarrow R[d_i] \cdot R[2]$
7 $\quad R[2] \leftarrow R[2]^2$
8 **end**
9 **if** $R[0] \cdot R[1] \cdot M = R[2]$ **then**
10 $\quad$ **return** $r \cdot R[1]$
11 **else**
12 $\quad$ **return** "Error"
13 **end**

Algorithm 2: RSA implementation protected against SCA and FIA.

Input : $M \in \mathbb{Z}_N, d = (d_{n-1}, \cdots, d_0)_2$
Output: $M^d \in \mathbb{Z}_N$ or "Error"

1 Generate a random $r \in \mathbb{Z}_N^*$
2 $R[0] \leftarrow r$
3 $R[1] \leftarrow r^{-1}$
4 $R[2] \leftarrow M$
5 **for** $i \in [\![0, n-1]\!]$ **do**
6 $\quad R[d_i] \leftarrow R[d_i] \cdot R[2]$
7 $\quad R[2] \leftarrow R[2]^2$
8 **end**
9 **if** $R[0] \cdot R[1] \cdot M = R[2]$ **then**
10 $\quad$ **return** $r \cdot R[1]$
11 **else**
12 $\quad$ **return** "Error"
13 **end**

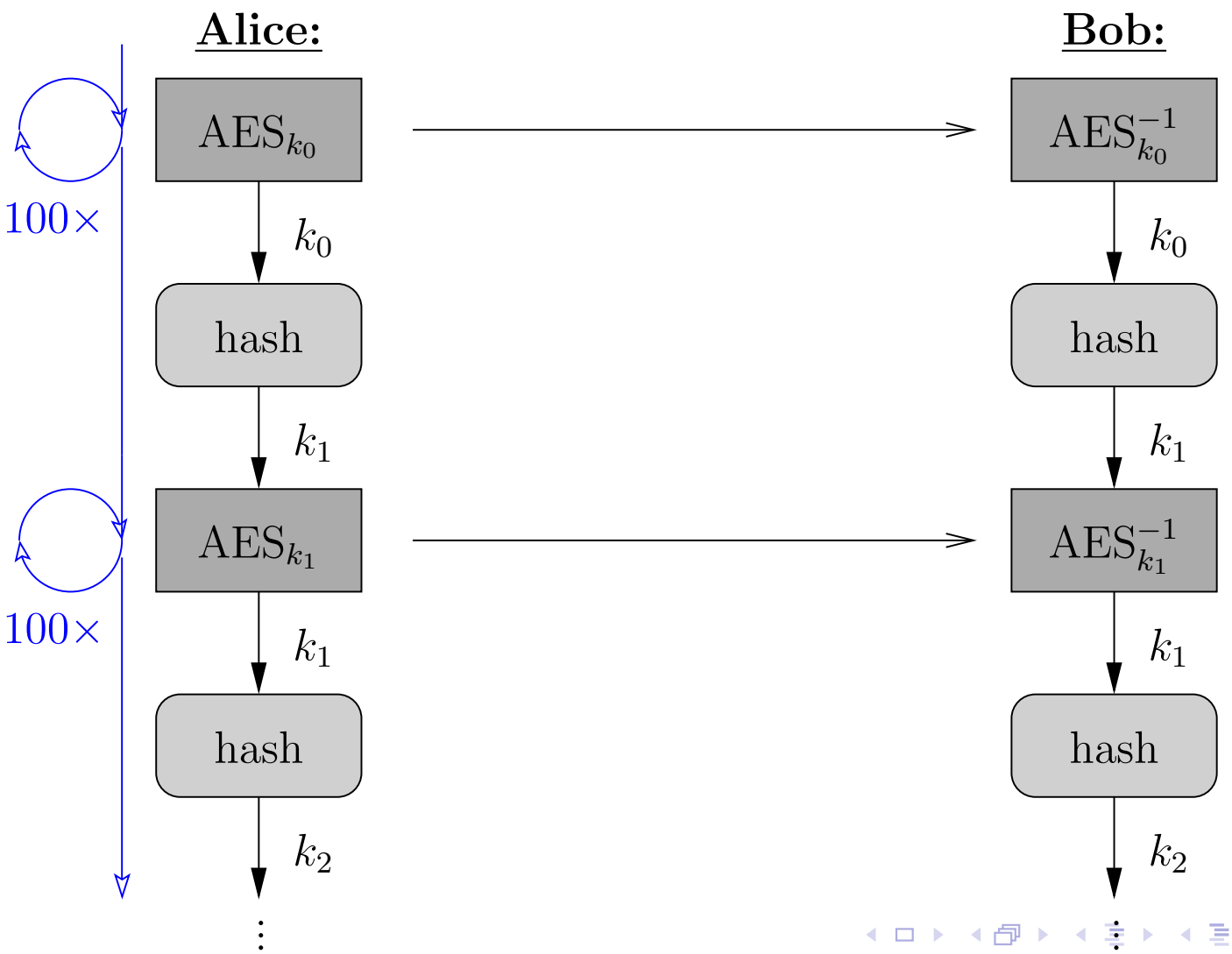
Final condition :

$$R[2] = ((M^2)^2) \cdots) = M^{(2^n)}$$
$$R[0] = r \cdot M^{\overline{d}}$$
$$R[1] = r^{-1} \cdot M^d$$

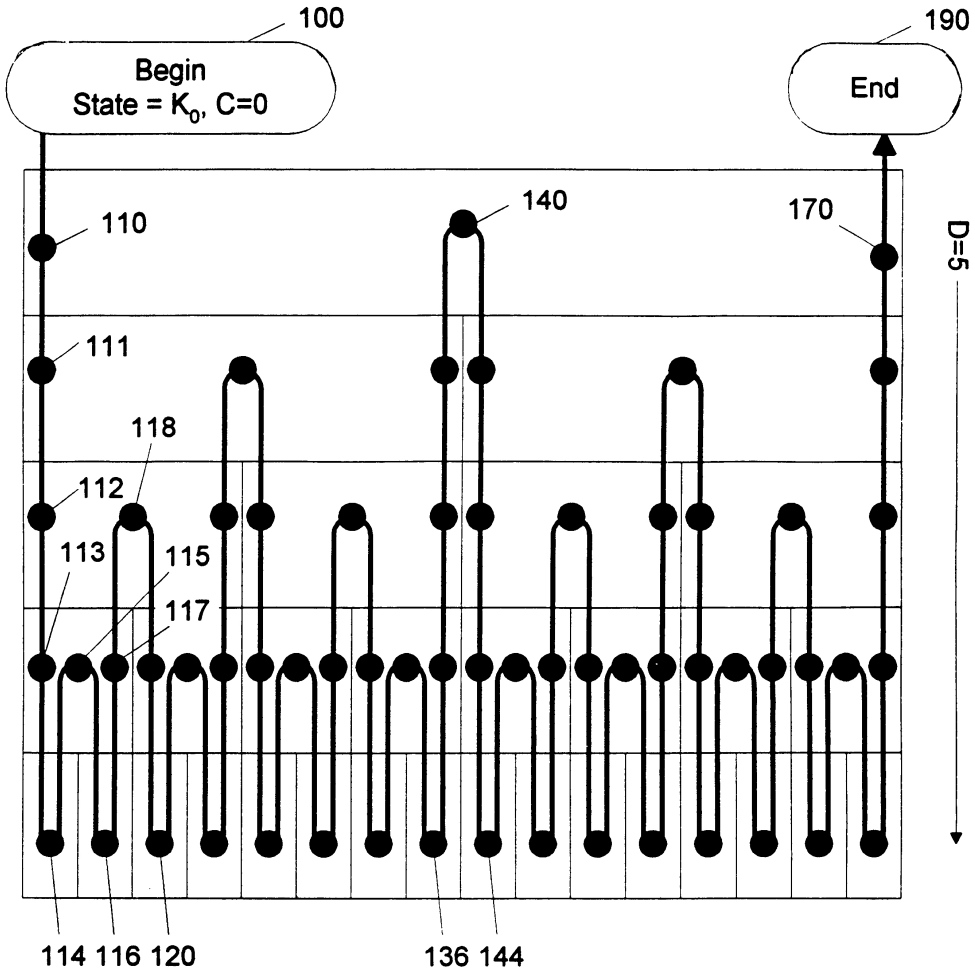
Resilience against passive attacks

- **Ephemeral keys** is the solution :
  1. Indexed key update : [Koc03, Koc05]
  2. Fresh rekeying : [MSGR10]
- (1) is applicable to scenarios, such as **bitstream decryption** of an FPGA
  - *Indeed, attacks are out : [MBKP11, MKP11, MKP12]*
- **Limits** : key scheduling is expensive

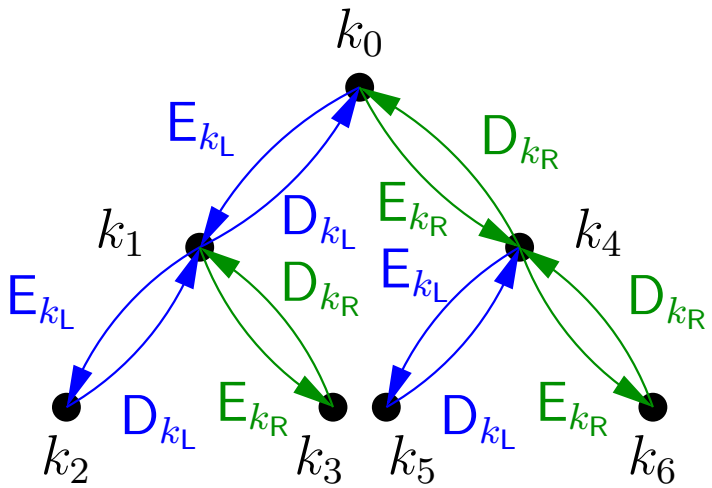
Protocol level [Koc05, §4] : if ≈ 1 bit is leaked per 100 encryptions...



Logarithmic key access speed with indices in a binary table [Koc03]



Logarithmic key access speed with indices in a binary table [Koc03]



Caption:

E: Symmetric block cipher
D: Symmetric inverse block cipher

k_L: Public key for left accesses (↙, ↗)
k_R: Public key for right accesses (↘, ↖)

k_0: Private root key
$k_{i>0}$: Secondary private session keys

$\Rightarrow$ also protects against related keys attacks [BK09] !

Resilience against active attacks

- Recall AES can be broken with one $(C, C^\star)$ pair [TMA11]
- Against cryptography (DFA) : [GSDS10]
  - **Do not encrypt twice the same message**
    - Randomize the input
  - **Do not output directly the result**
    - Hash it. But beware of *safe error* attacks
- Against the rest (control, that can be altered by skipping instructions) :
  - **Chain at protocol-level**

DPA can be thwarted easily (at protocol-level)

$\Rightarrow$ We focus on fault injection attacks

Case-study : secure messaging

Assumptions

- The reader is easily protected
- The smartcard is hard to protect
- Symmetric cryptography (E)
- Shared keys :
  - k_1 : external authenticate
  - k_2 : internal authenticate
  - k_3 : secure canal encryption
- Evaluating the resilience to fault attacks only

Example on Smartcards Protocols

The attacker monitors faulty outputs here

Reader

Smartcard

The attacker injects faults on this side

envelop e_3

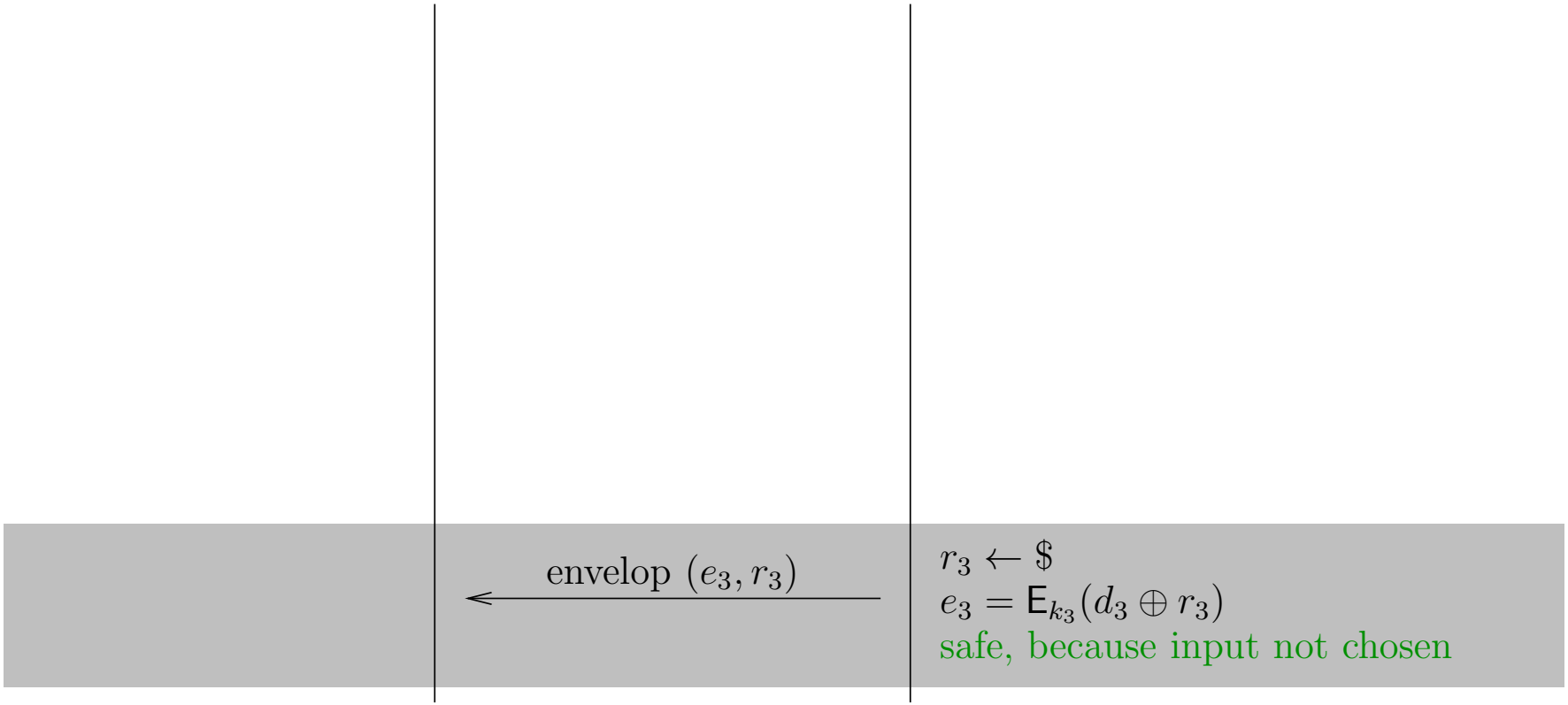
$e_3 = \mathsf{E}_{k_3}(d_3)$

unsafe, because input is chosen

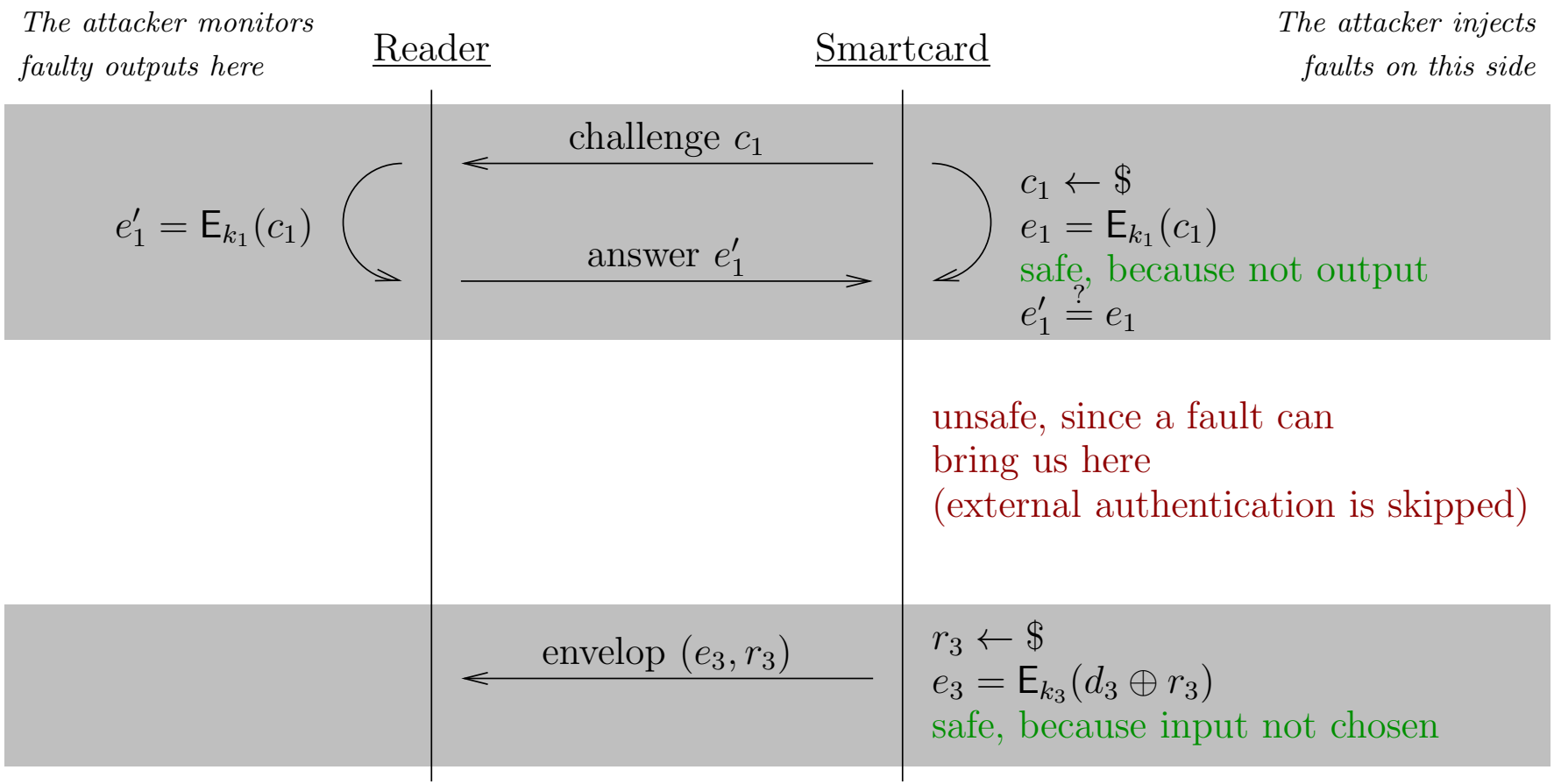
Example on Smartcards Protocols

The attacker monitors faulty outputs here

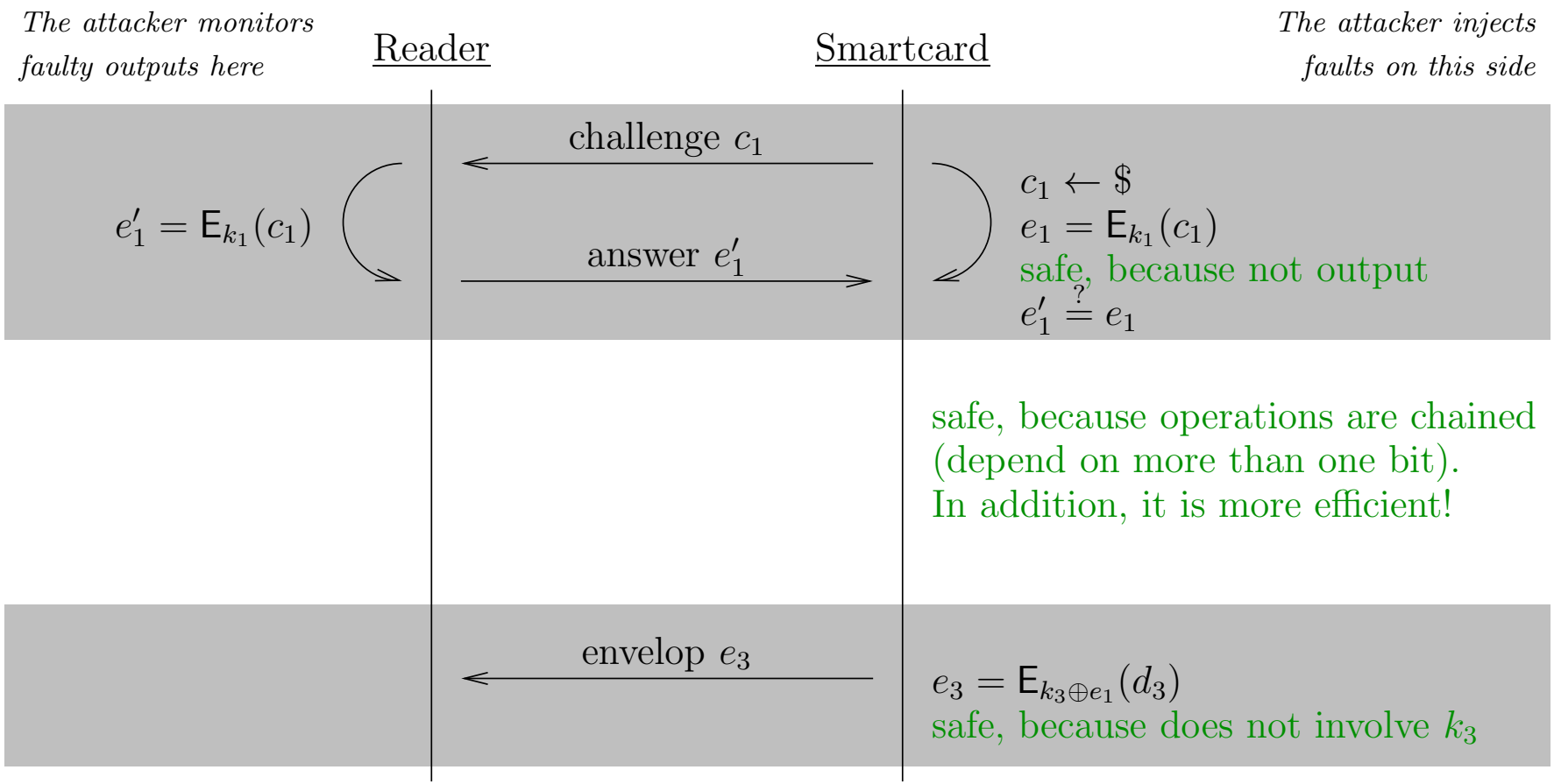
The attacker injects faults on this side



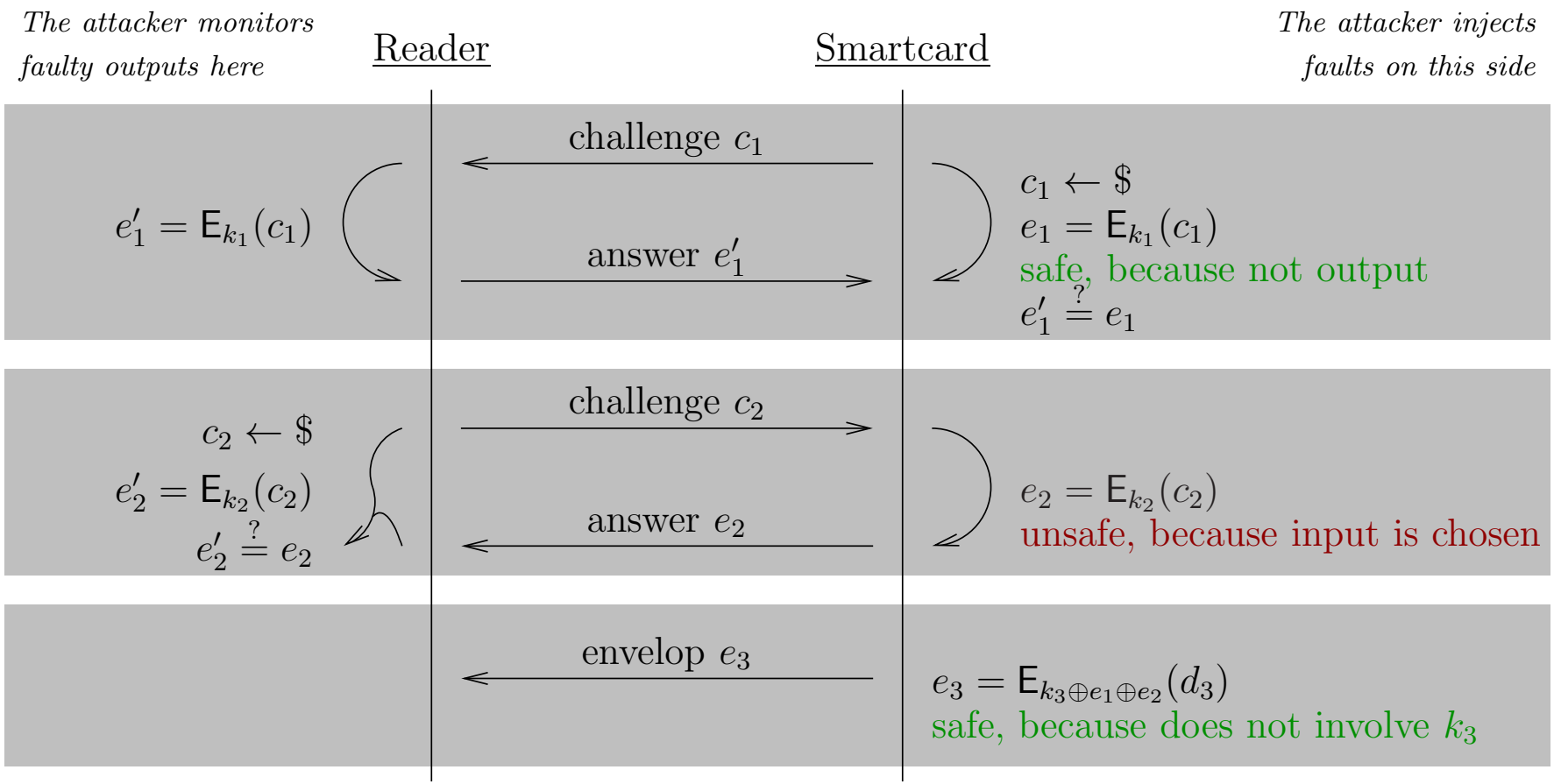
Example on Smartcards Protocols



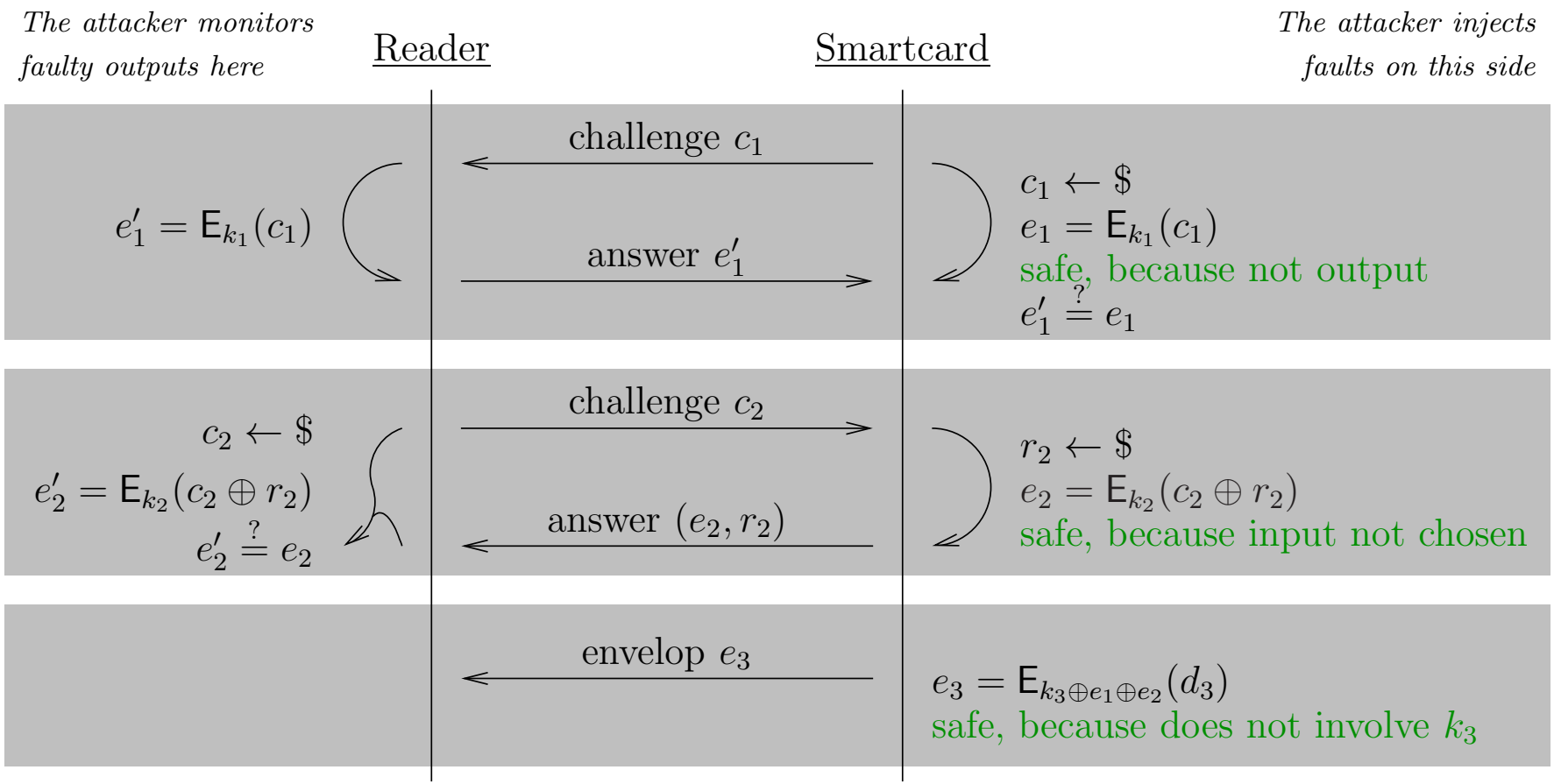
Example on Smartcards Protocols



Example on Smartcards Protocols



Example on Smartcards Protocols

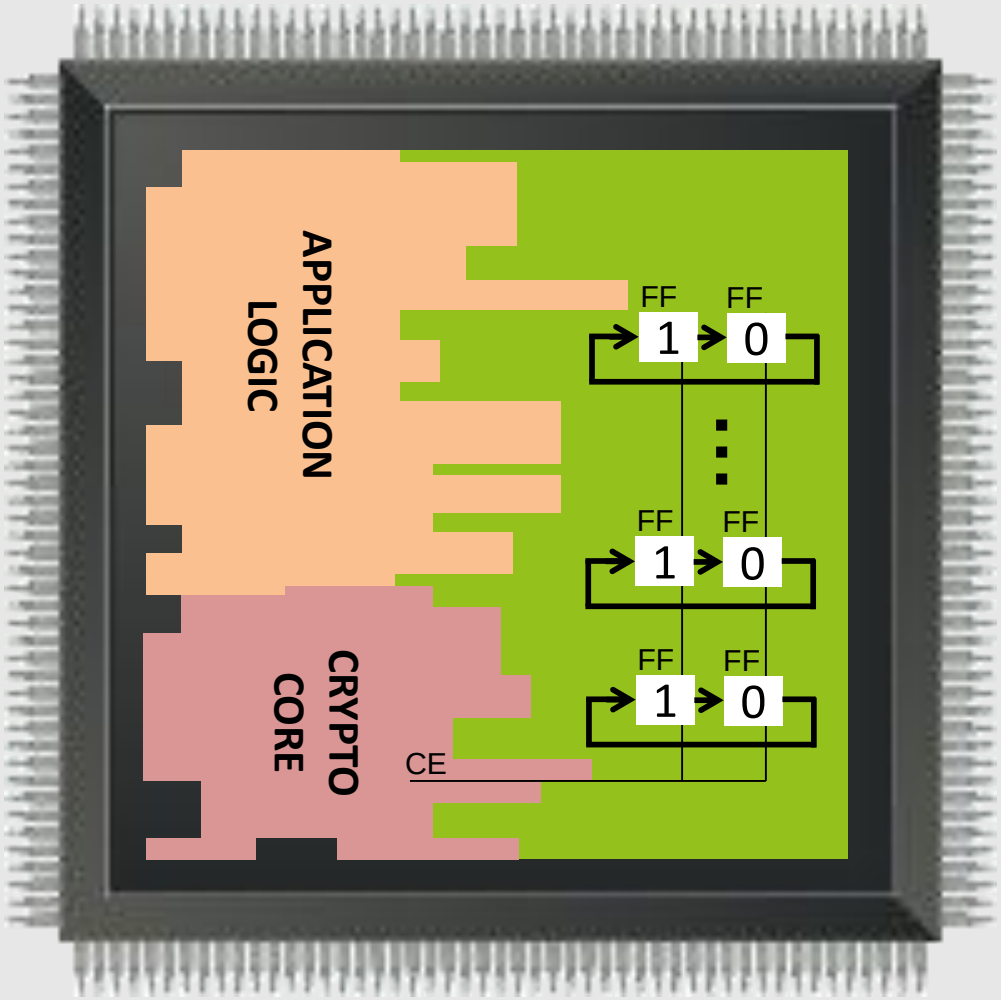


This paper [GM11] presents some heuristic countermeasures

- Complementary with other countermeasures !
- Especially suitable for FPGAs if some resources remain available.
- Next slides courtesy of the authors (Tim Güneysu and Amir Moradi)

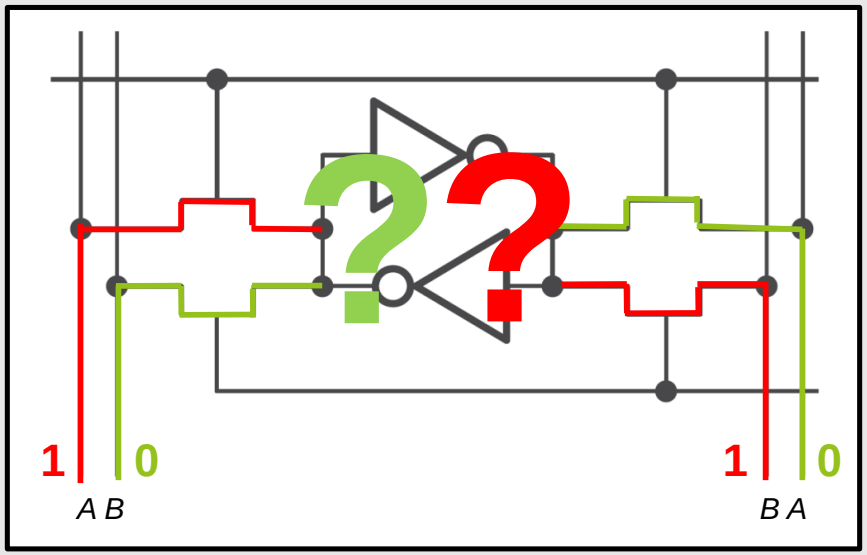
Implementing Noise Generators in FPGAs

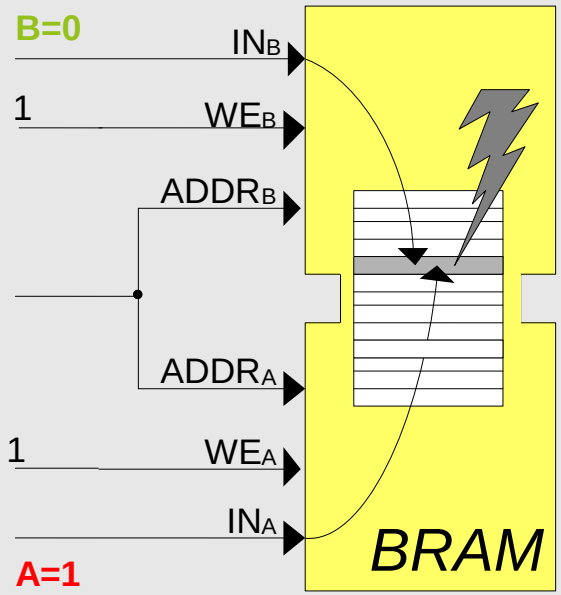
- **Common design**: application including cryptographic core
- **Noise generation strategy**
  - Configure remaining, routable slices (flip-flops) as cyclic shift registers
  - Preload sequence „01“ into shift registers
  - Run noise generator in synch with crypto core



Proposal #2: Write Collisions in BRAMs

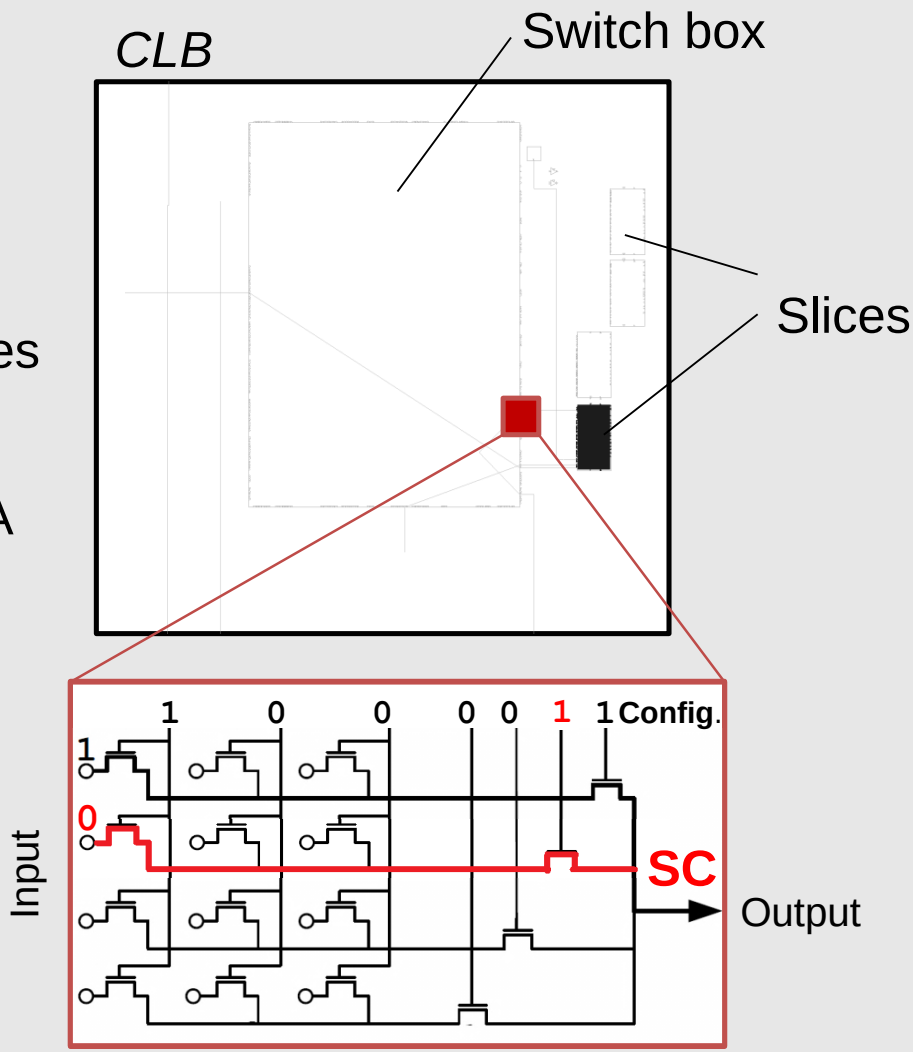
- Write collision when concurrently writing data to the same address of dual-ported memories (BRAM)
- Opposite driving directions in inverter pair result in uncertain outcome [GP09,G10]





Proposal #3: Short Circuits in FPGAs

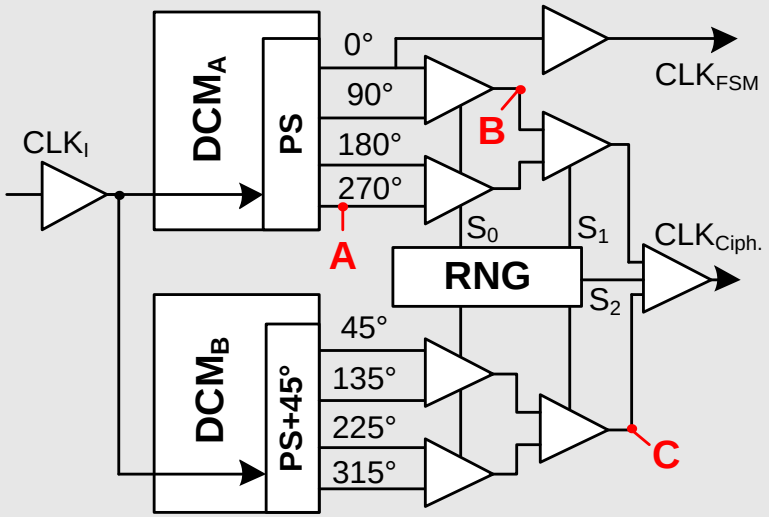
- Short circuits (SC) can be created in the FPGA‘s routing network [BKT10]
- SCs in output multiplexers of switch boxes
- Power restriction limits currents < 100 µA
- Establishing controlled SCs requires manual routing (via XDL)



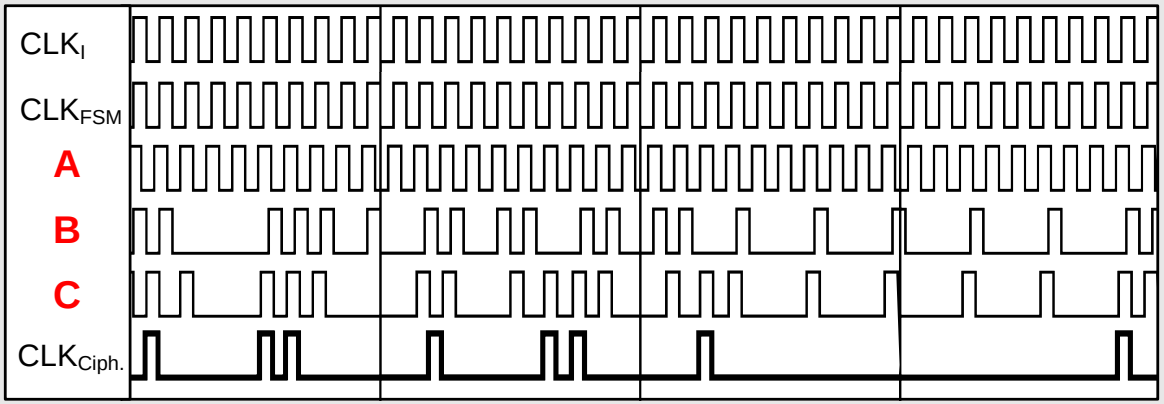
Output multiplexer

Proposal #4: Clock Disalignment using DCMs

- Digital Clock Managers (DCM) support concurrent phase-shift channels
- Clock buffers can be configured as glitch-free clock multiplexers
- Cascading clock muxes result in a randomly delayed, phase-shifted clock

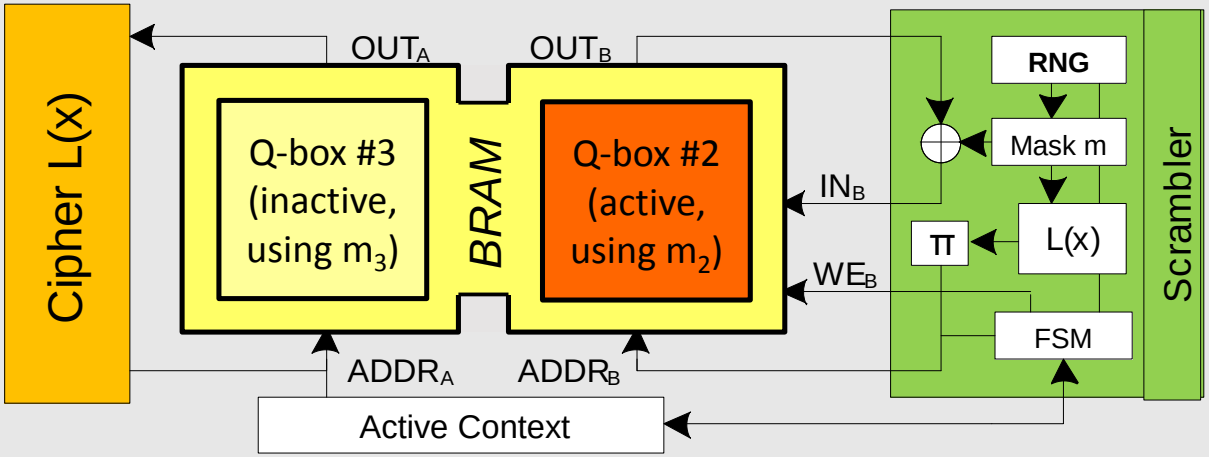


Clock Output Waveform



Proposal #5: Data Masking with BRAMs

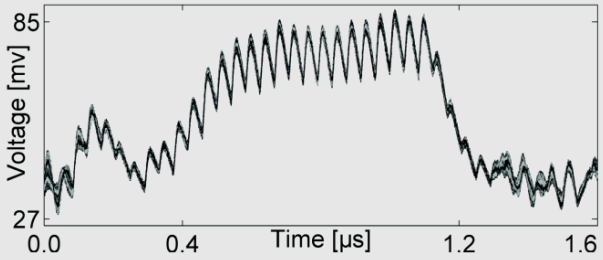
- Dual-ported BRAM allow simultaneous access and mask update in Q-box
  - Active context (Q-box #1) used by cipher operation
  - Inactive context (Q-box #2) updates mask by concurrent process
  - Context switch after update and cipher process are finished

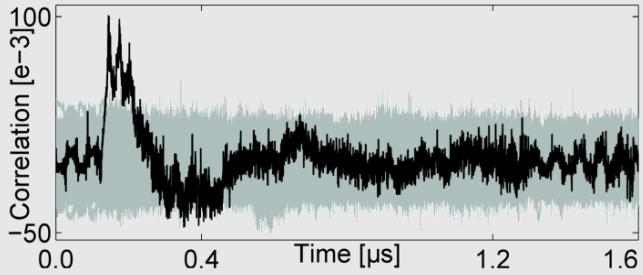


Current Mask m:
F439AD0B8C…

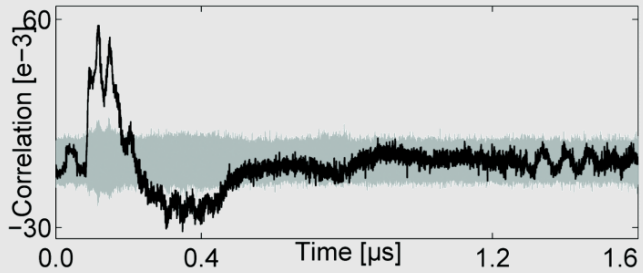
Evaluations: CPA on individual CMs

Plain AES-128@24Mh:
10^4 measurements
→ 3,000 traces req.

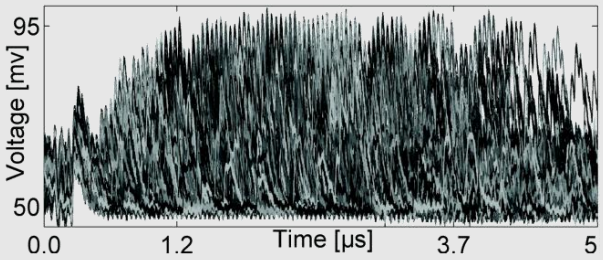


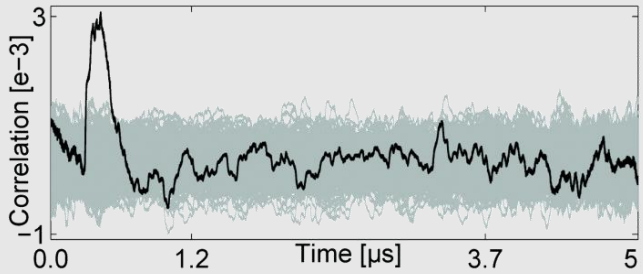


Individual/all noise generators combined:
Parameters used: *r=16 (instances), s=36 (width)*
5x10^4 measurements
→ 8,000 traces req.

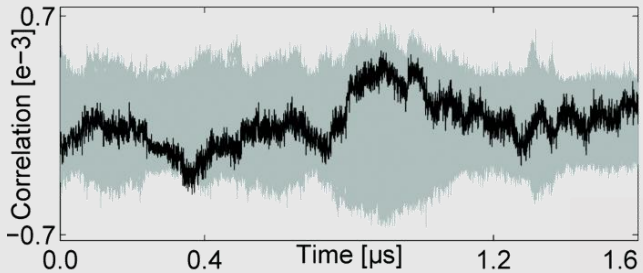


Clock disalignment
8 phase shift steps
10^7 measurements
→ 3,000,000 traces req.



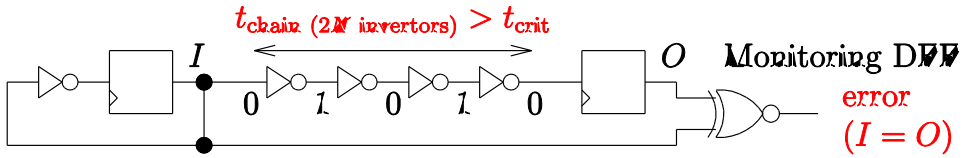


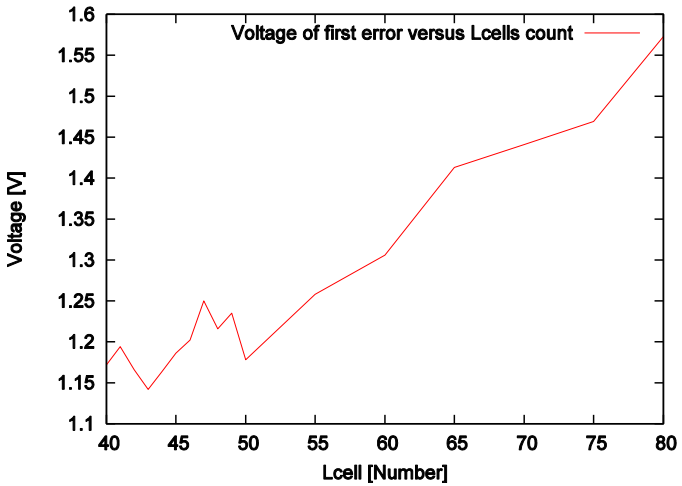
Memory masking with dual-ported BRAMs
10^8 measurements
→ Not successful (using first-order attack)

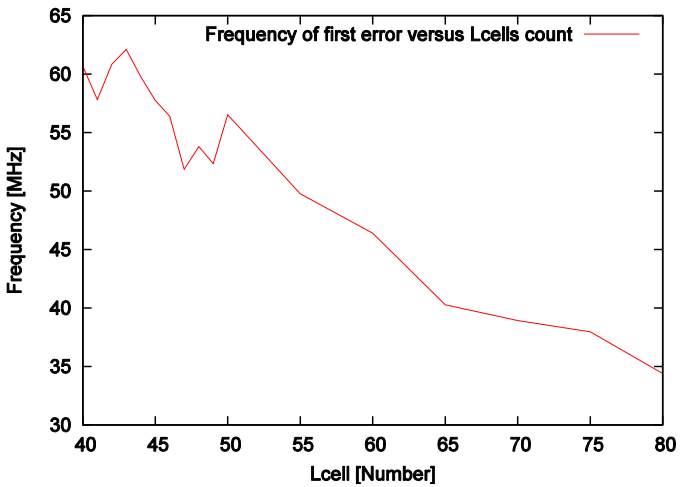


1. Introduction and reminder

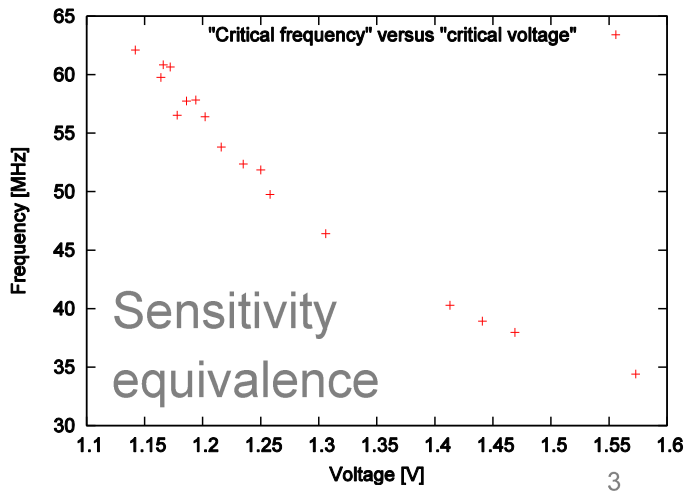
Digital attack sensor







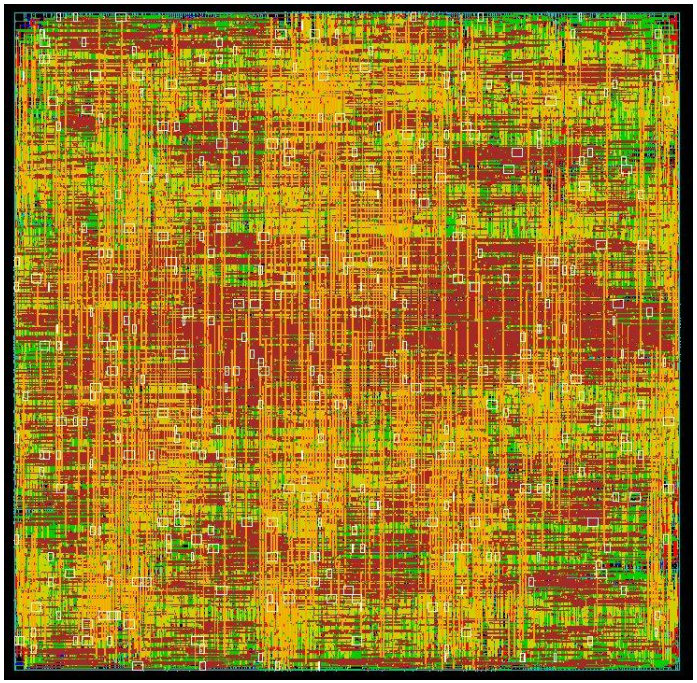
Digital, hence portable & low-cost
- can be spread over the circuit
- difficult to spot by an attacker
- responds to any kind of stress (clock / power glitches, heating, overclocking, laser spot)



IET Inf. Secur., 2011, Vol. 5, Iss. 4, pp. 181–190
doi: 10.1049/iet-ifs.2010.0238

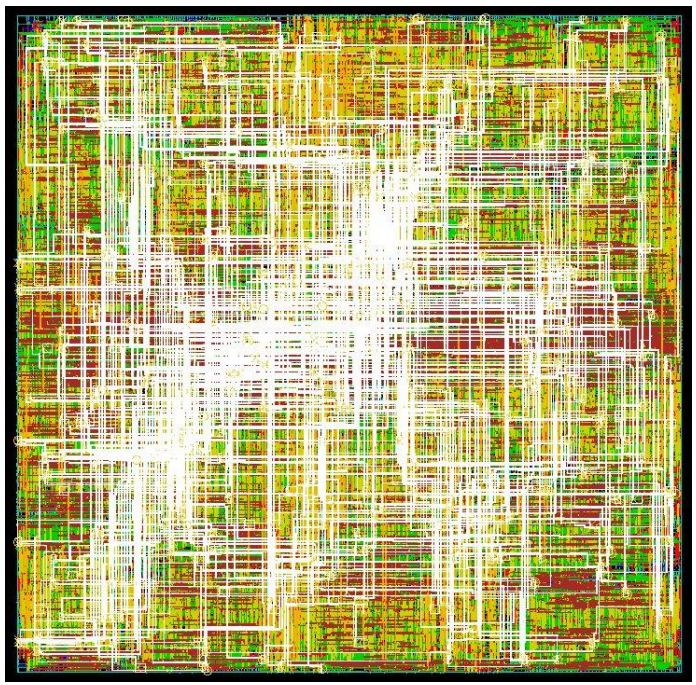
2. Main technical characteristics

- Digital, hence:
  - **Simple API**
  - **Stable**
  - **Small**
  - **Discreet, more difficult to recognize**

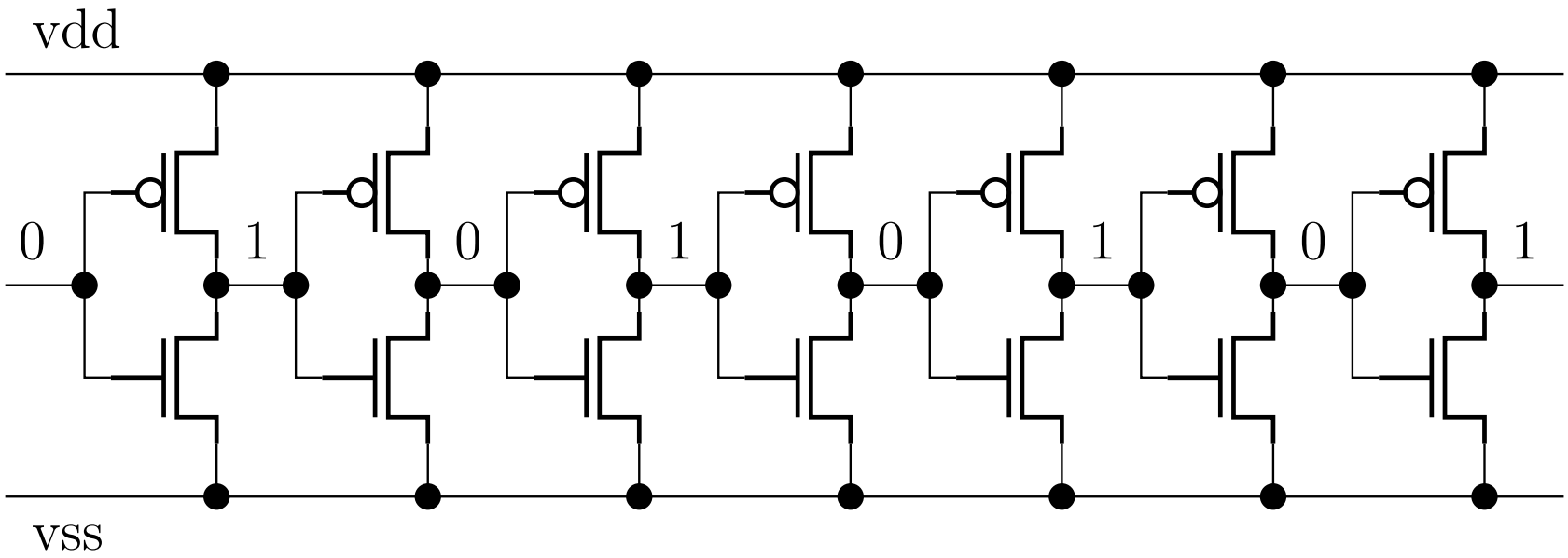


2. Main technical characteristics

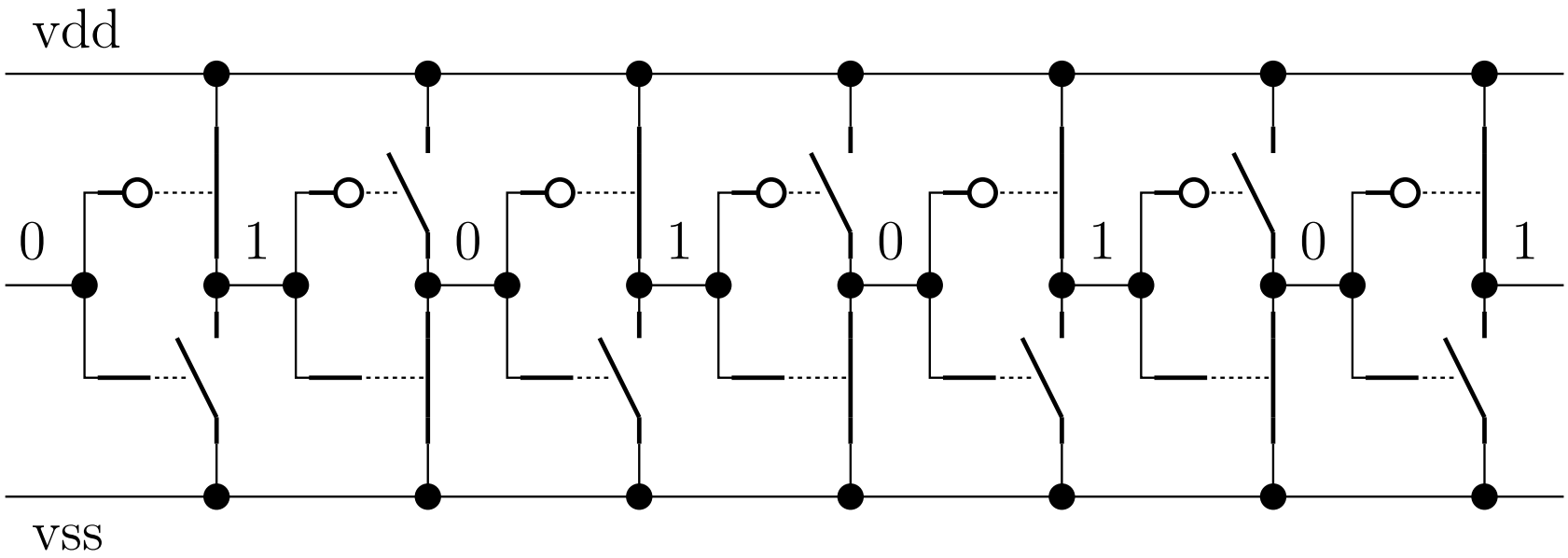
- Digital, hence:
  - **Simple API**
  - **Stable**
  - **Small**
  - **Discreet, more difficult to recognize**
  - **Melted within the rest of the SoC, more difficult to by-pass**



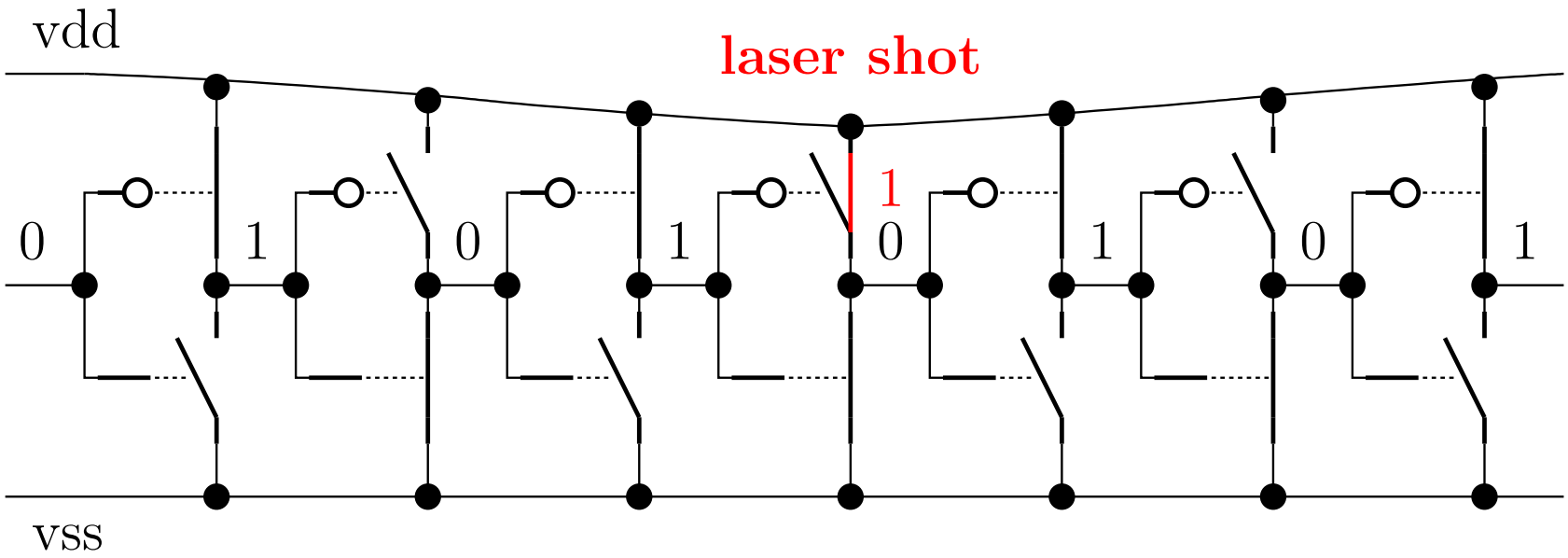
Effect of laser spot on CMOS cells



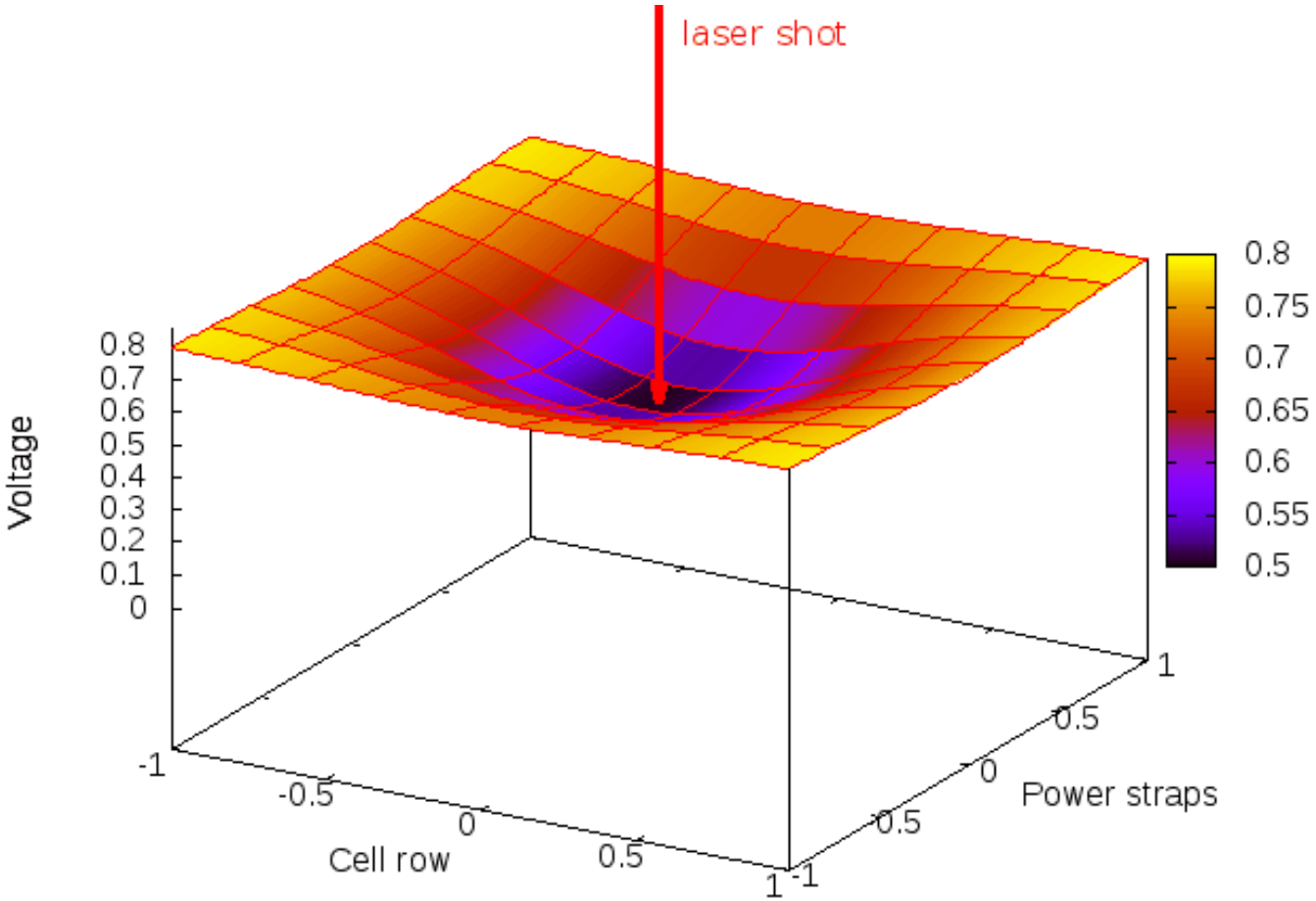
Effect of laser spot on CMOS cells



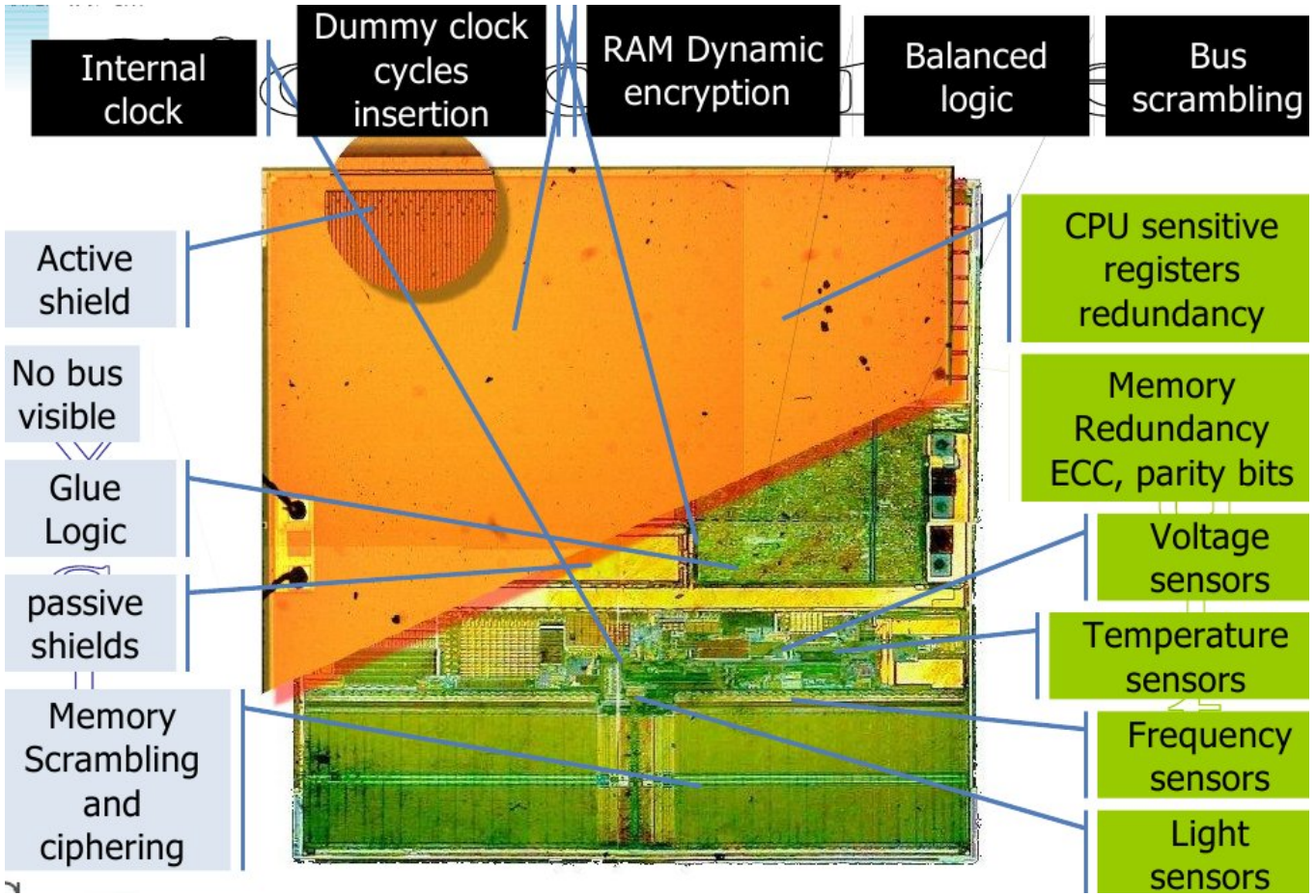
Effect of laser spot on CMOS cells



Effect of laser spot on CMOS cells

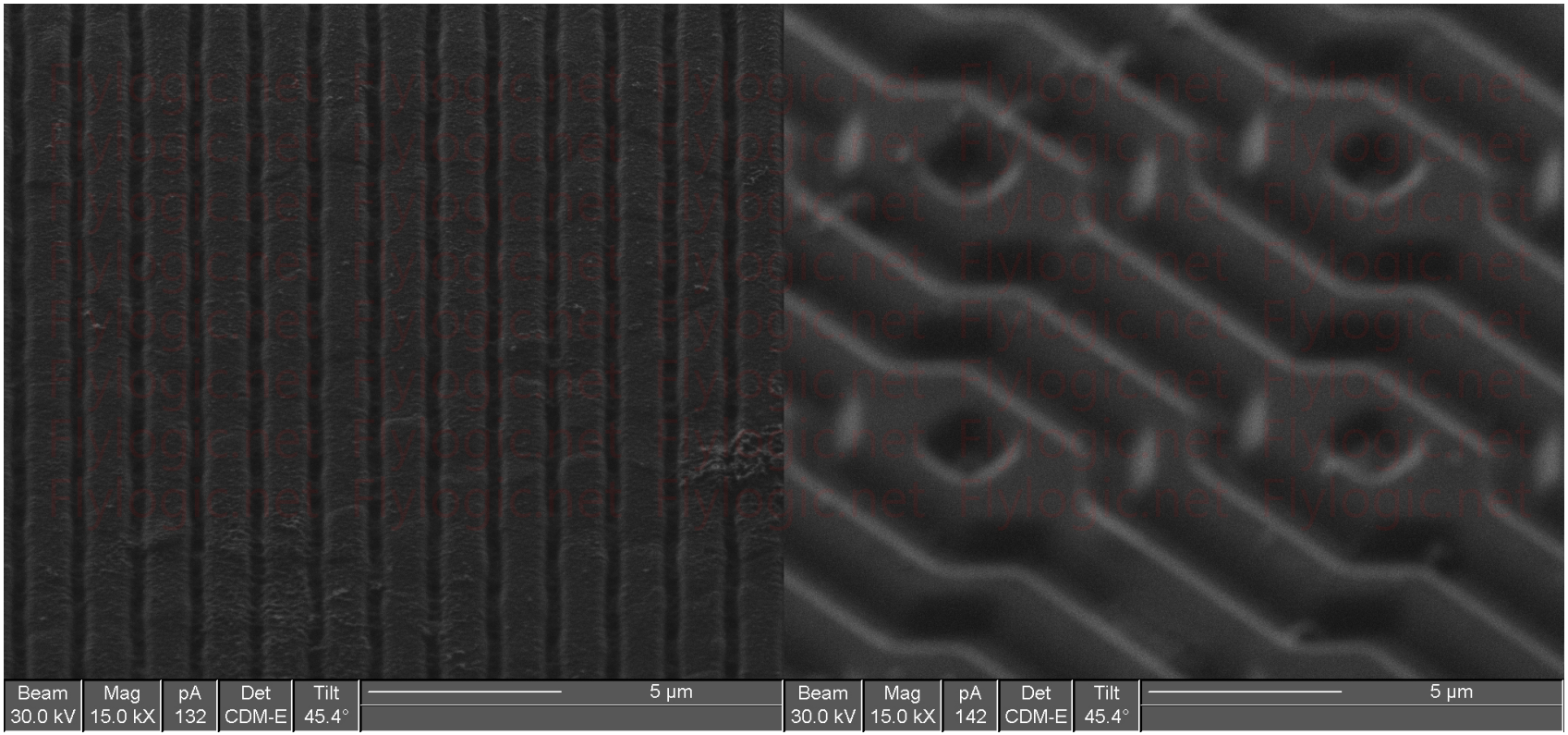


A summary of the countermeasures embedded in a smartcard

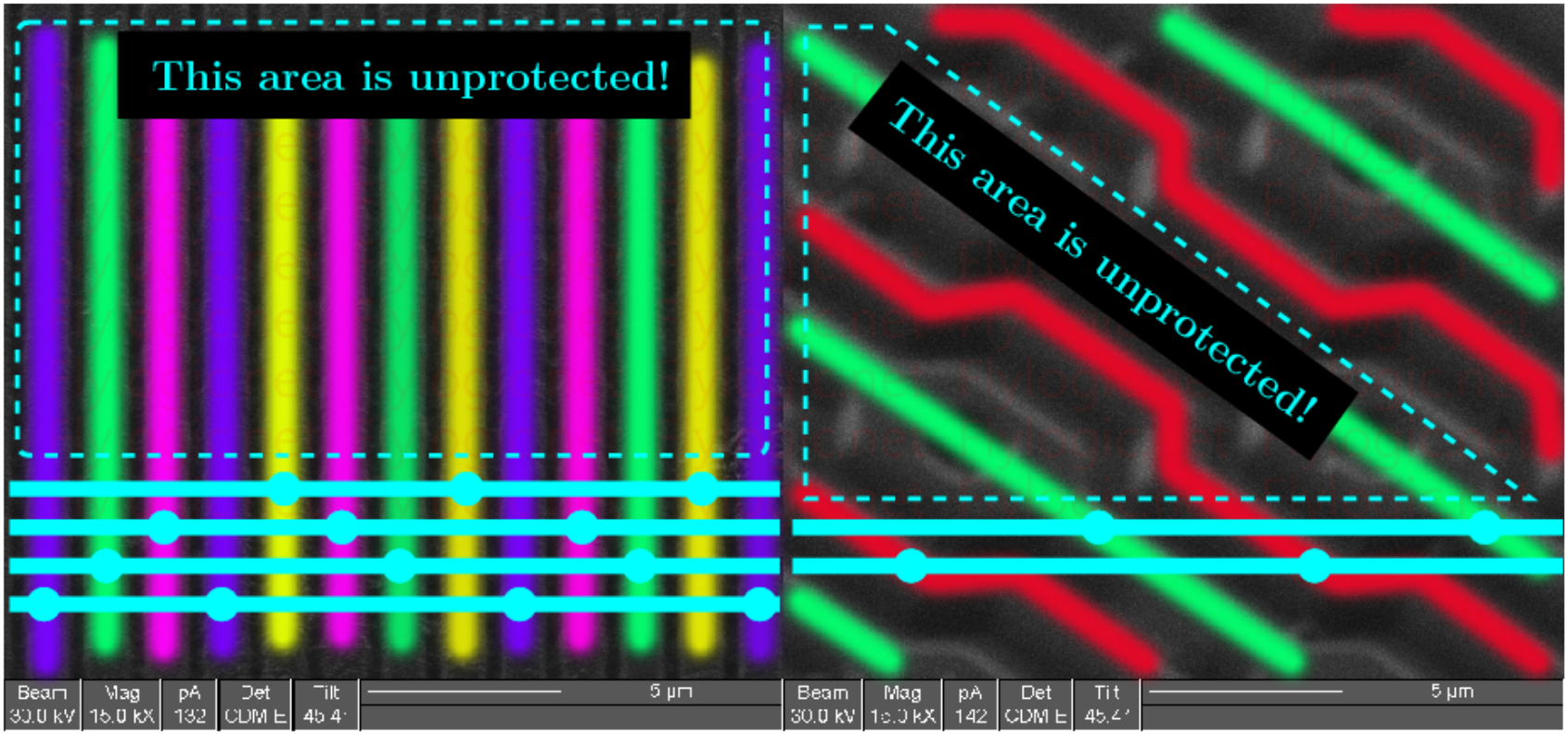


Courtesy of Assia Tria, CEA/LETI.

Active shield



Active shield



Masking is "Secret Sharing"

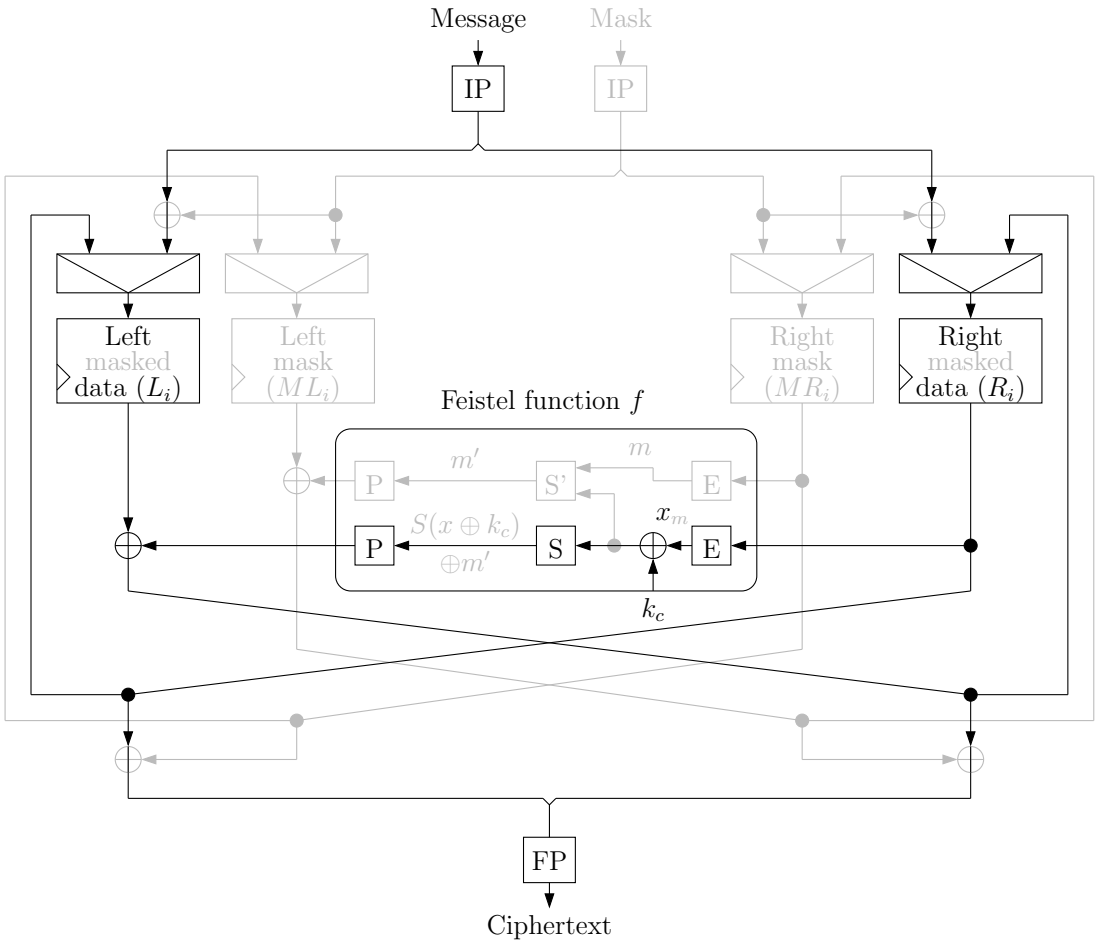
Principle

- Every variable s, potentially sensible, is represented as a set of shares $\{s_0, s_1, \cdots, s_d\}$.
- To reconstruct s, all the s_i are required.
- Example : $d = 1$, $\boxed{s \doteq s_0 \oplus s_1}$.
  - $\Rightarrow$ Boolean [GP99], multiplicative [AG01], affine [FMPR10], homographic [PR10], *etc.*

- Leakage resistant since variables are never used plain.
- Attractive but works only fine for registers.
- Efforts done to protect also the combinational logic.

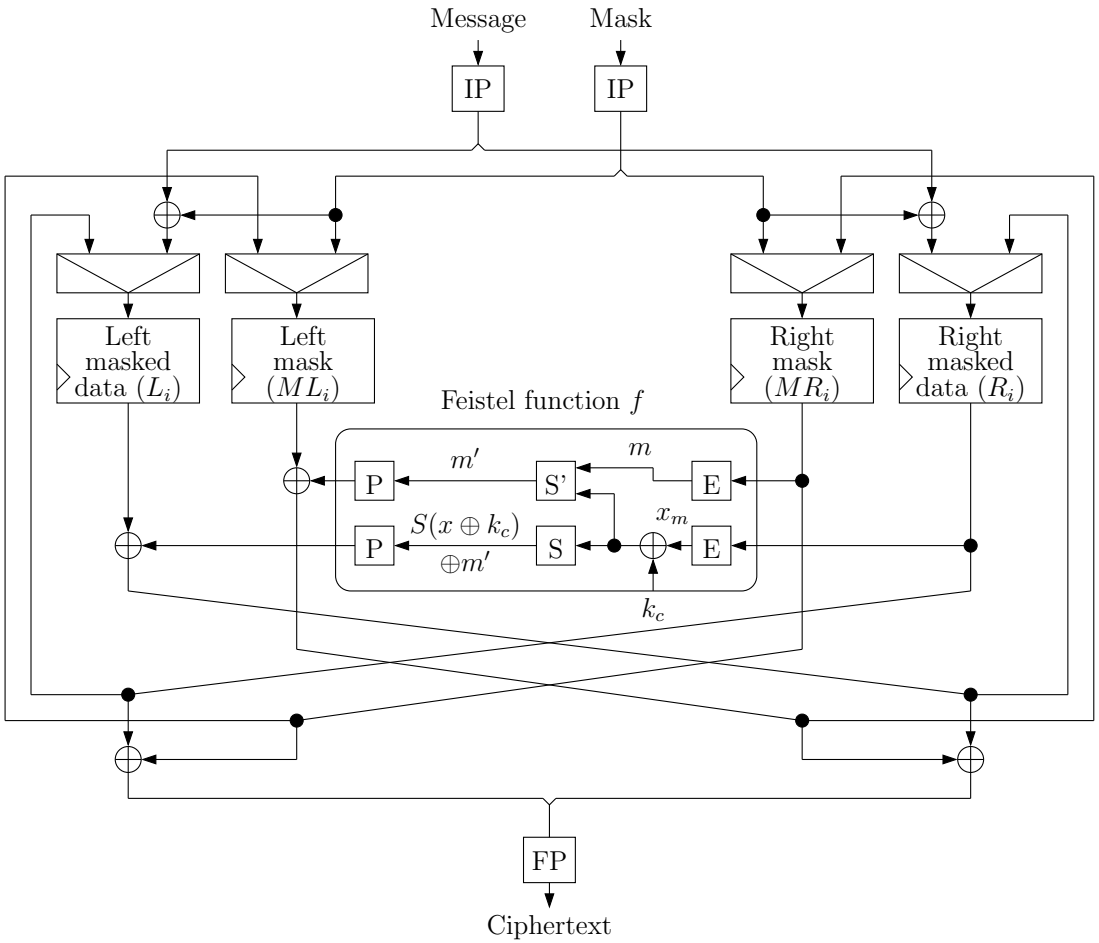
Computing Masked DES [PS08]

Everything is linear, but the sboxes.



Computing Masked DES [PS08]

Everything is linear, but the sboxes.



Computing Masked AES

Everything is linear, but the sboxes (*sic*). $a \mapsto a^{-1}$ mod $X^8 + X^4 + X + 1$, and $a^{-1} = a^{254} = (((a^2) \times a)^4) \times ...$ mod $X^8 + X^4 + X + 1$.

- Squaring is linear
- We miss a secure AND

Next slides courtesy of the authors of [RP10].

Ishai-Sahai-Wagner (ISW) Scheme

Principle

- AND gates encoding:
  - Input: $(a_i)_i$, $(b_i)_i$ s.t. $\bigoplus_i a_i = a$, $\bigoplus_i b_i = b$
  - Output: $(c_i)_i$ s.t. $\bigoplus_i c_i = ab$

$$\bigoplus_i c_i = \left(\bigoplus_i a_i\right)\left(\bigoplus_i b_i\right) = \bigoplus_{i,j} a_i b_j$$

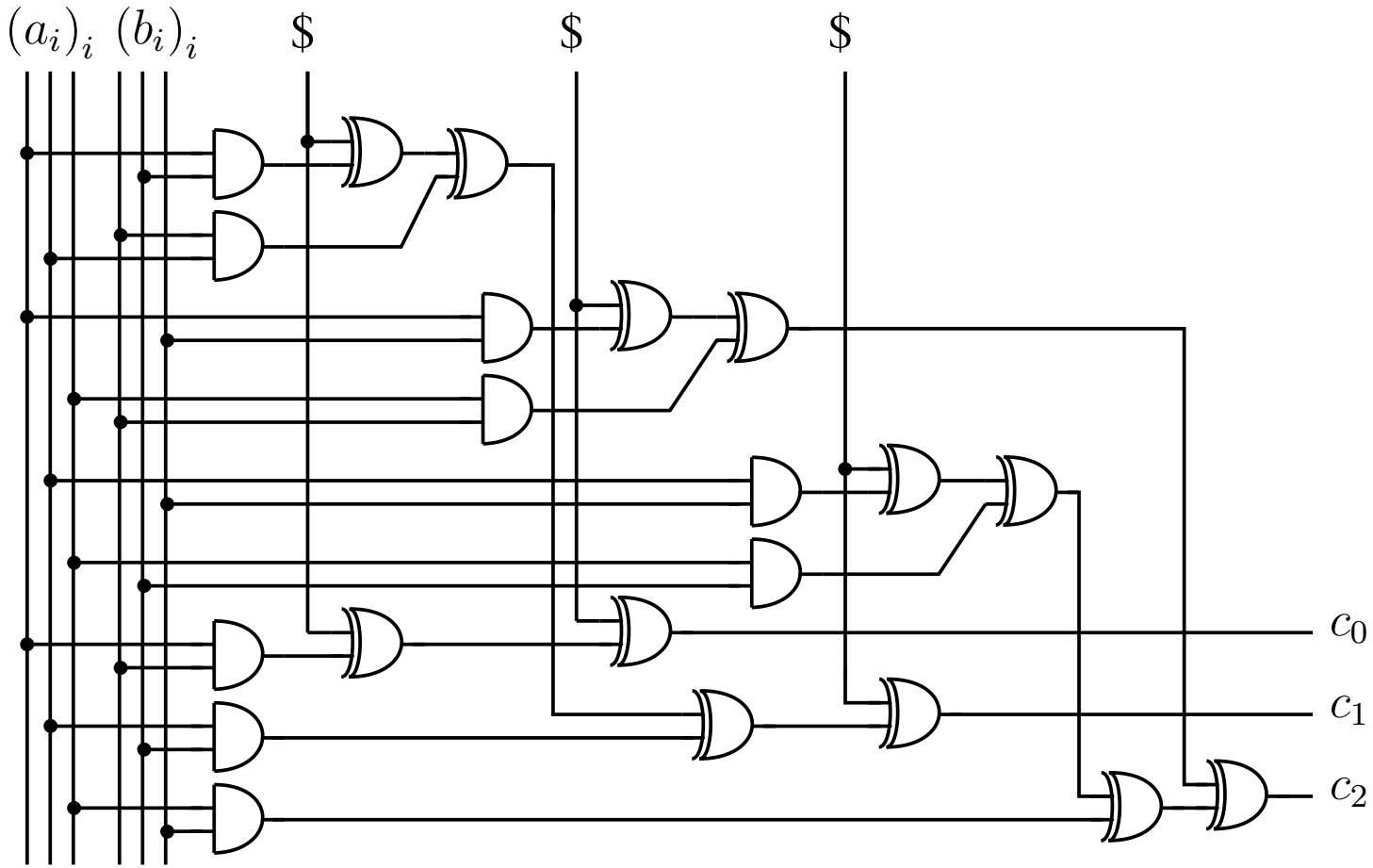
- Example $(d = 2)$:

$$\begin{pmatrix} a_0b_0 & (a_0b_1 \oplus r_{1,2}) \oplus a_1b_0 & (a_0b_2 \oplus r_{1,3}) \oplus a_2b_0 \\ r_{1,2} & a_1b_1 & (a_1b_2 \oplus r_{2,3}) \oplus a_2b_1 \\ r_{1,3} & r_{2,3} & a_2b_2 \\ c_1 & c_2 & c_3 \end{pmatrix}$$

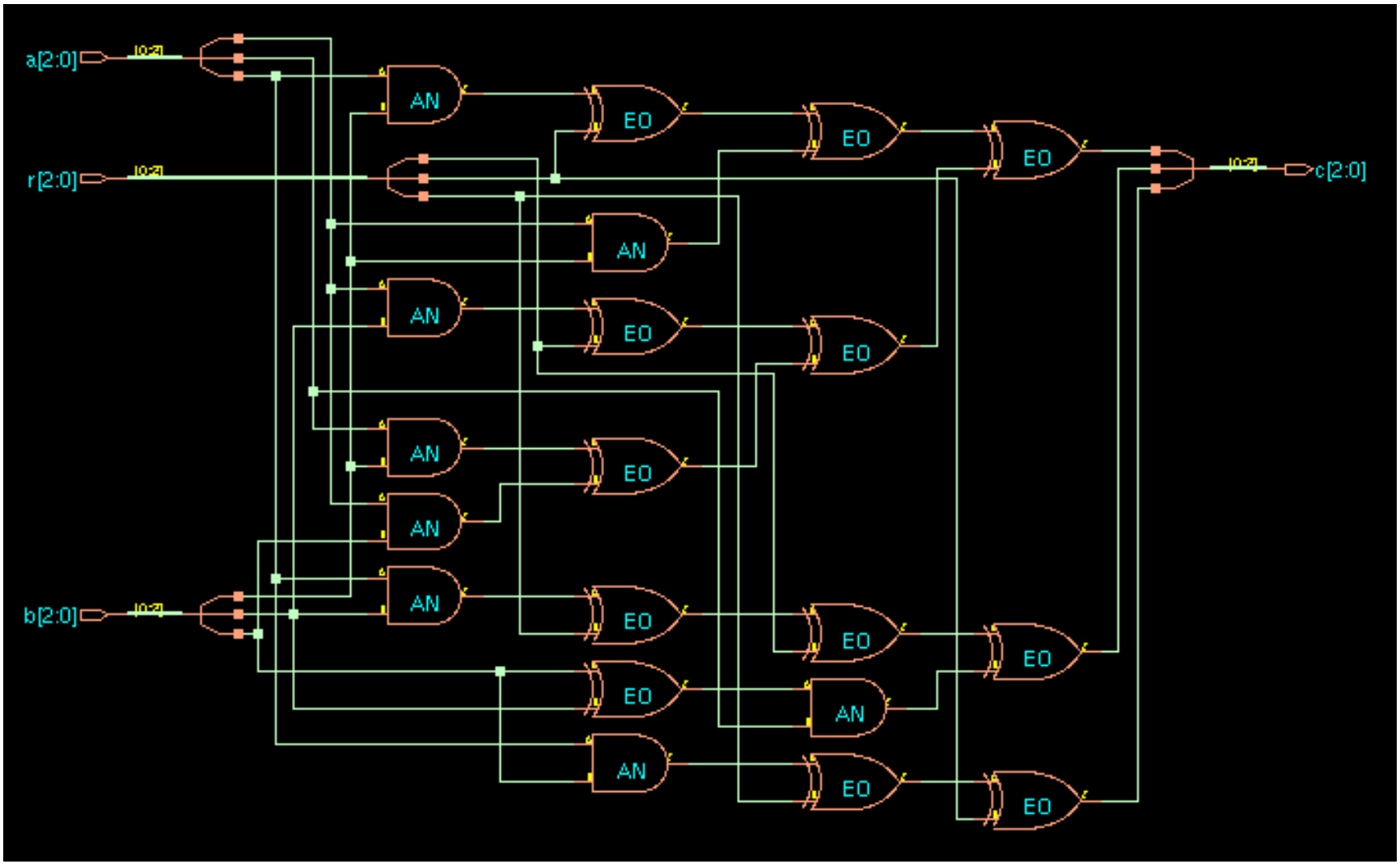
- Ishai *et al.* prove $(d/2)$th-order security
  - We prove dth-order security

Ishai-Sahai-Wagner (ISW) Scheme

Example: AND gate for $d = 2$



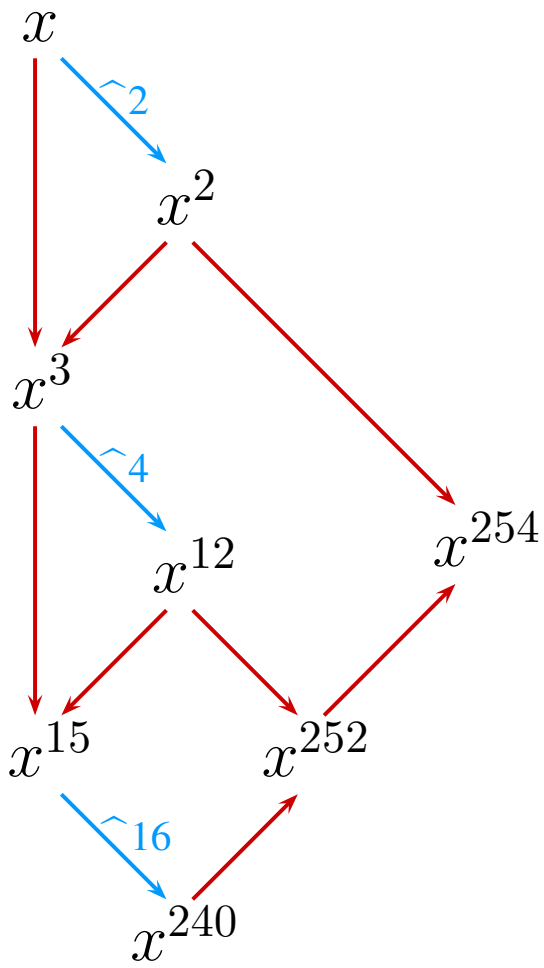
Warning for optimizations (here under Cadence) ! [RBG+15]



Caption : AN = `and`, EO = `xor`.

Masking the S-box

The proposed addition chain:



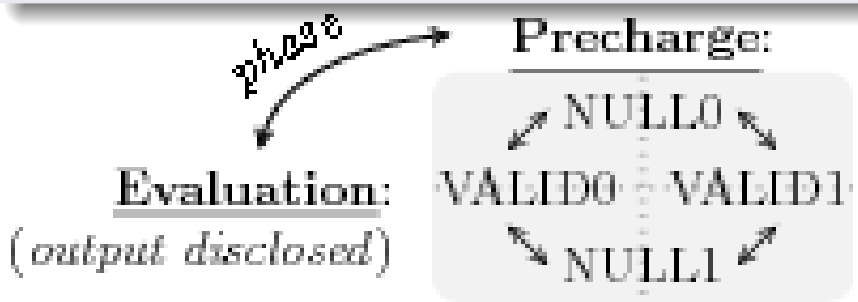
- one square
- one mult
- one ^4 (two squares)
- one mult
- one ^16 (four squares)
- one mult
- one mult
- Total: 4 mult and 7 squares
- Memory: 3 registers
- LUT for ^2, ^4 and ^16

Implementation Results (8051)

Method	K cycles	ms (31MHz)	RAM (bytes)	ROM (bytes)
Unprotected Implementation				
Na.	3	0.1	32	1150
First-Order Masking				
[Messerges FSE'00]	10	0.3	256+35	1553
[Oswald+ FSE'05]	77	2.5	42	3195
Our scheme (d=1)	129	4	73	3153
Second-Order Masking				
[Schramm+ CT-RSA'06]	594	19	512+90	2336
[Rivain+ FSE'08]	672	22	256+86	2215
Our scheme (d=2)	271	9	79	3845
Third-Order Masking				
Our scheme (d=3)	470	15	103	4648

$a \leftrightarrow (a_f, a_t)$ DPL representation :

- a is **VALID** if $\boxed{a_f \oplus a_t = 1}$.
  VALID $\doteq \{\text{VALID0}, \text{VALID1}\}$ or
  VALID $\doteq \{(1,0),(0,1)\}$.
- a is **NULL** if $\boxed{a_f \oplus a_t = 0}$.
  NULL $\doteq \{\text{NULL0}, \text{NULL1}\}$ or
  NULL $\doteq \{(0,0),(1,1)\}$.



Flavors of DPL – gate g style :

- **DPL w/ EE** :
  $\exists a$ VALID, $g(a, \text{NULL}) = \text{VALID}$.
- **DPL w/o EE** :
  $\forall a$ VALID, $g(a, \text{NULL}) = \text{NULL}$.

Faults typology on DPL :

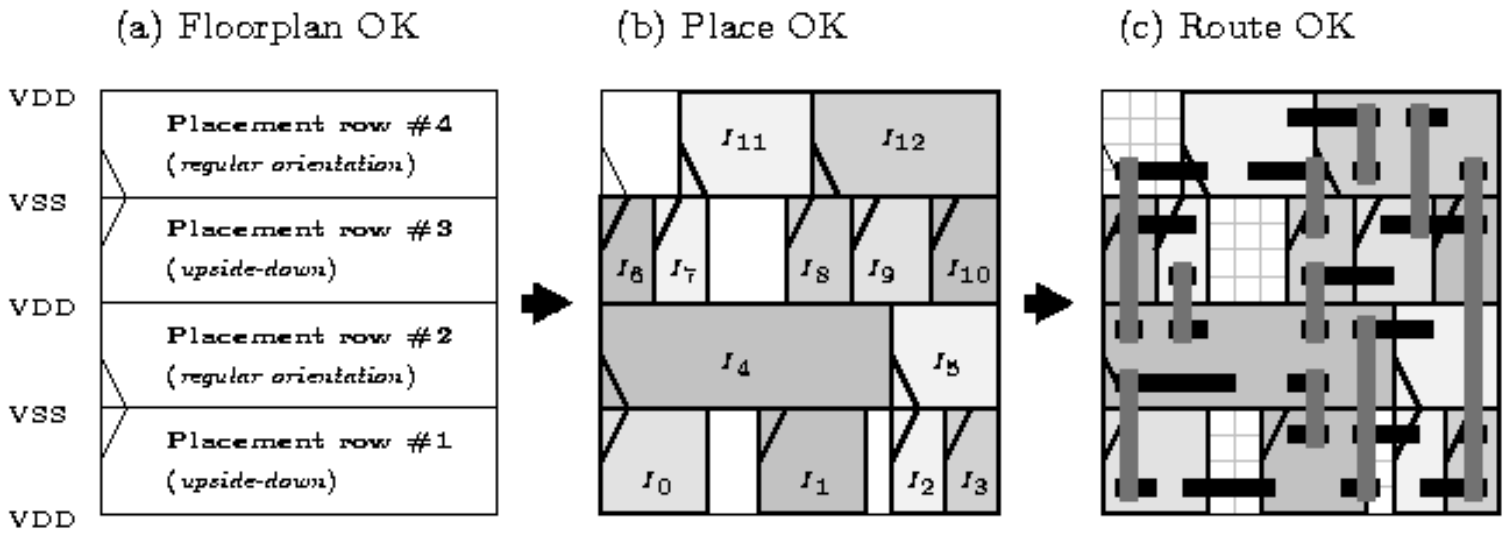
- Only results on *evaluation* are observable.
- **Asymmetric faults** :
  {VALID0, VALID1} $\xrightarrow{\downarrow}$ NULL0, caused by **global** perturbations (*e.g.* VC glitch, overclocking, under-powering).
- **Symmetric faults** :
  {VALID0, VALID1} $\xrightarrow{\downarrow \text{ or } \uparrow}$ {NULL0, NULL1}, caused by **local** perturbations (*e.g.* laser or EM injection).

Regular Backend Flow in ASIC Design

a) **Floorplan split into rows**

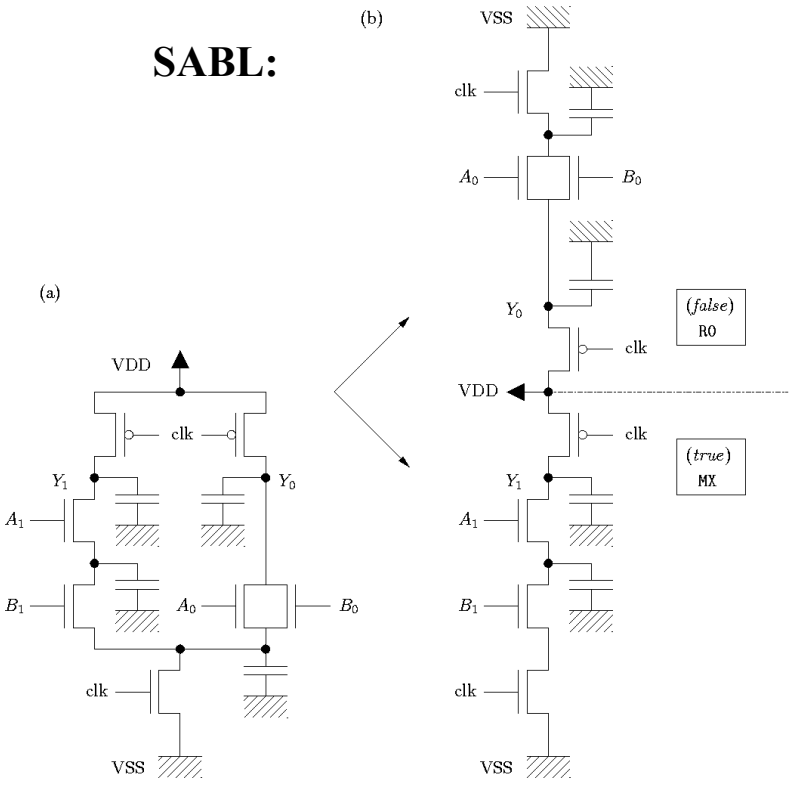
b) **Instances I_x of the netlist are dispatched into the placement rows**
The cells share the supply (power or VDD / ground or VSS) lines

c) **Routes are created over the cells**
E.g. in HCMOS9GP, cell pins are in M1, thus M2 – M6 is devoted to interconnection (M1 can be used to route side-by-side cells.)

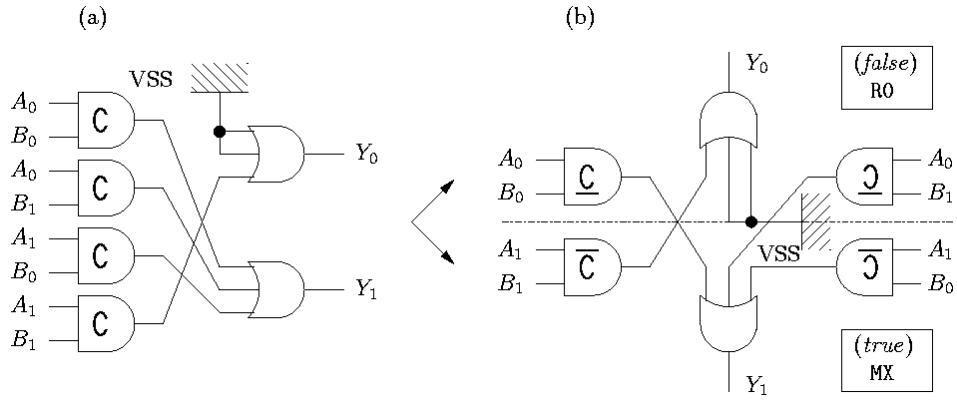


Secured Cells Come in Pairs: SABL & DI gates

SABL:



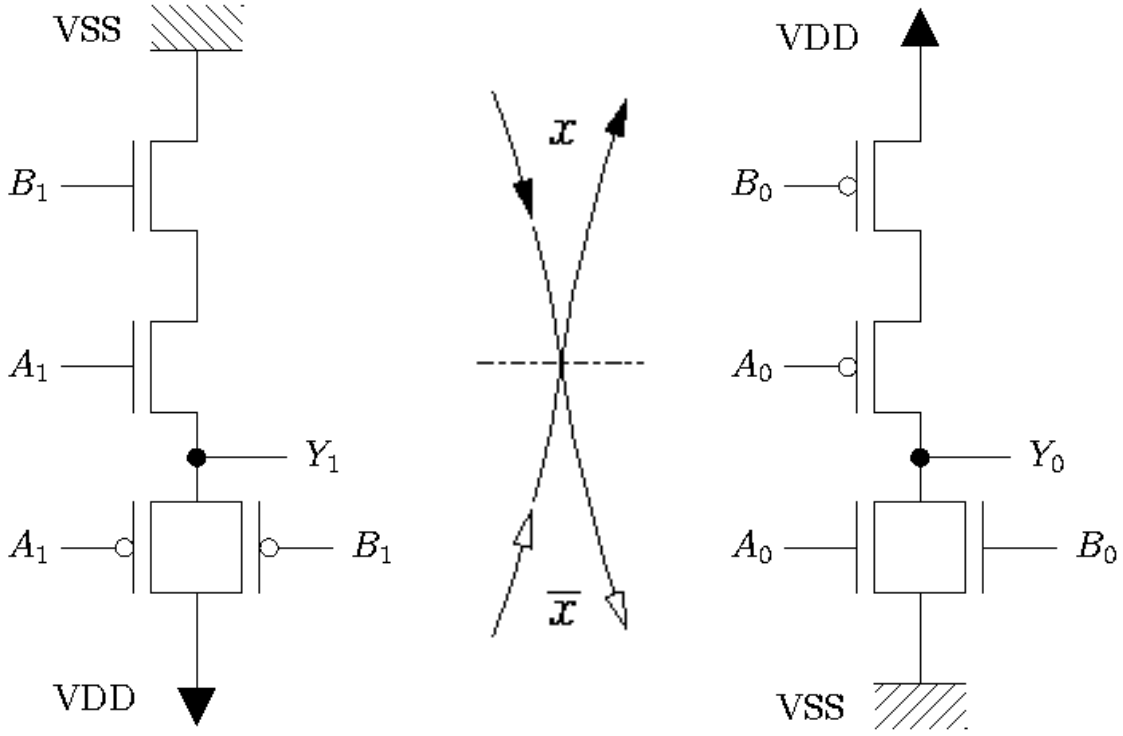
DI:



WDDL example: placement strategy

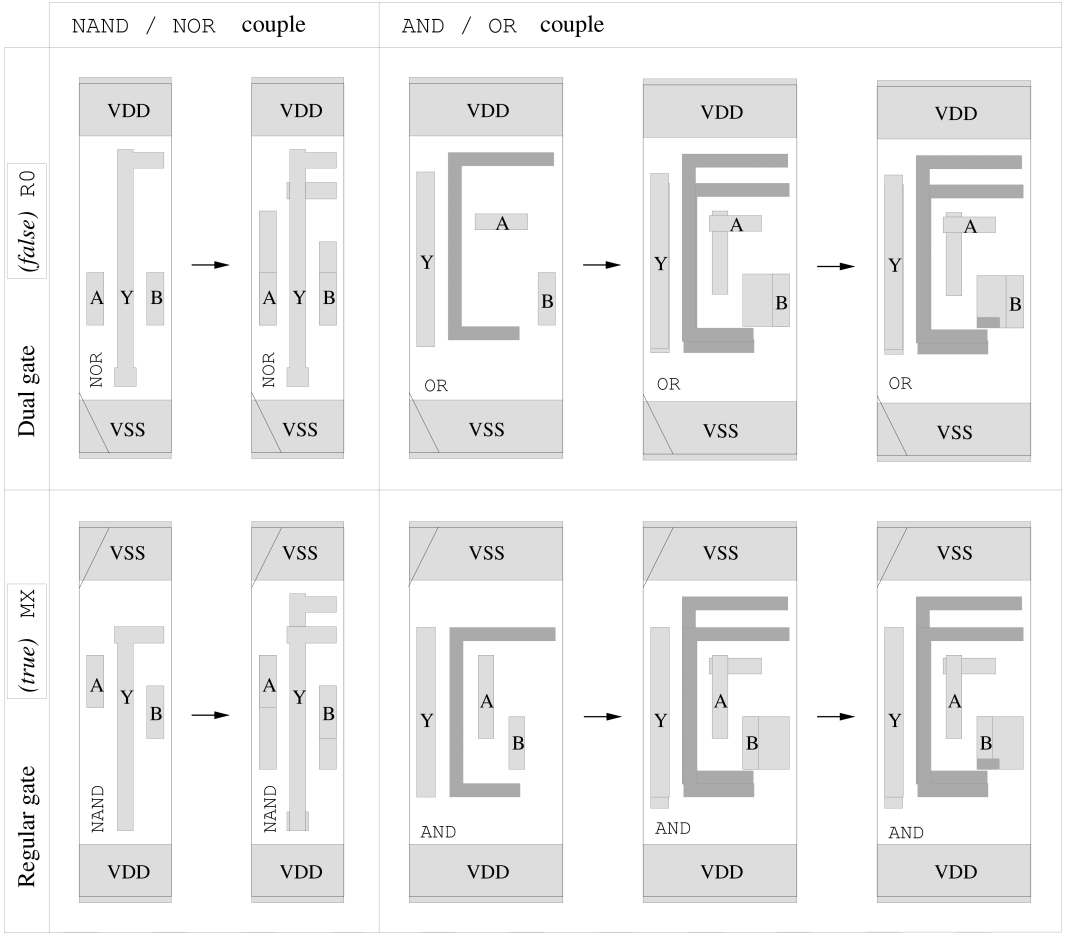
NAND (R0)
placed into row *i*

NOR (MX)
Placed into row *i+1*



After they are flipped `R0` and `MX`,
dual gates are much alike!

Making Standard Cells Compliant with WDDL



Each dual pair must have a compatible interface.

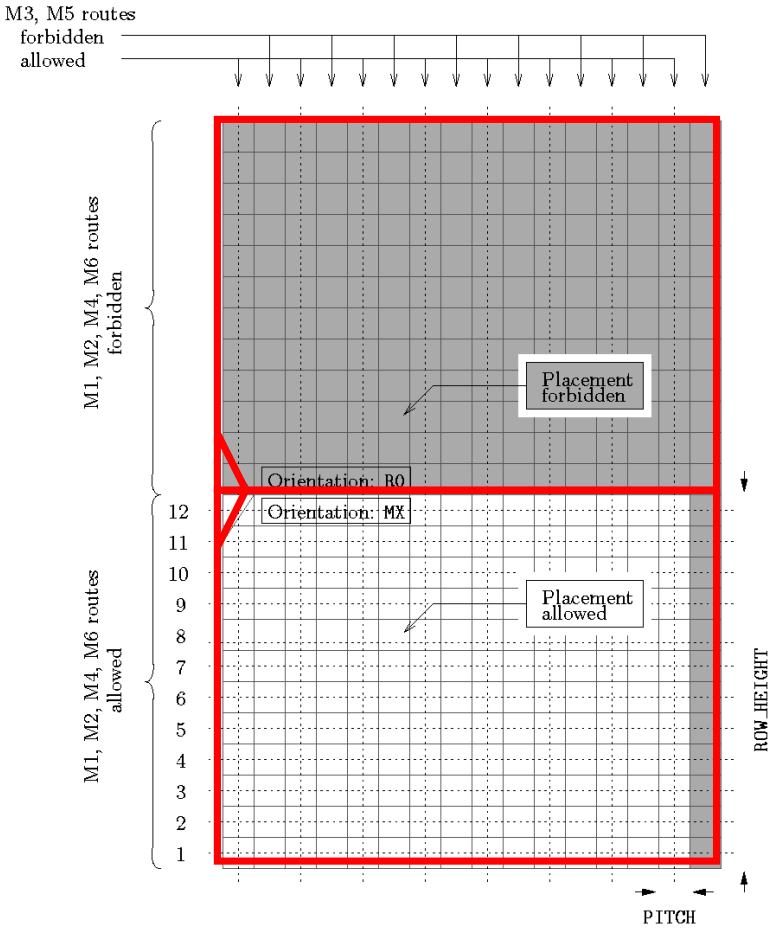
The transformation done on the abstracts (LEF description) + pins metal consists in:

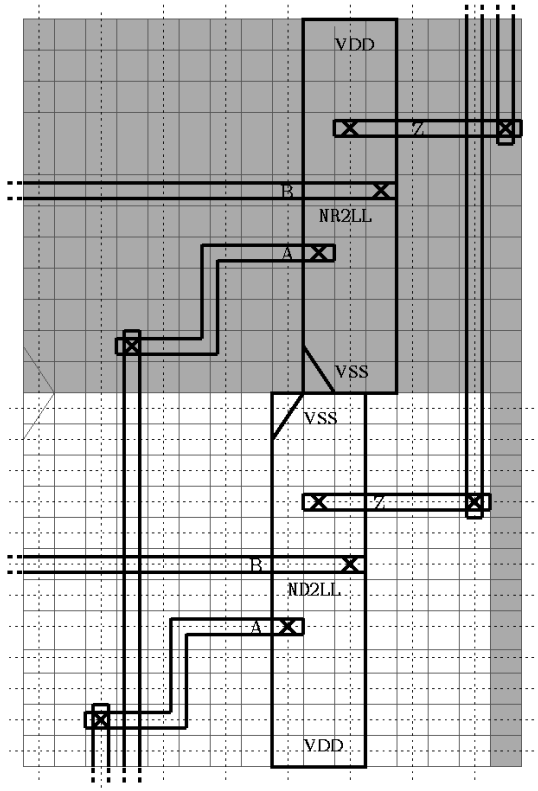
1. **Reorder pins**
2. **Enlarge pins for overlap**
3. **Keep pins intersection**

At that point:

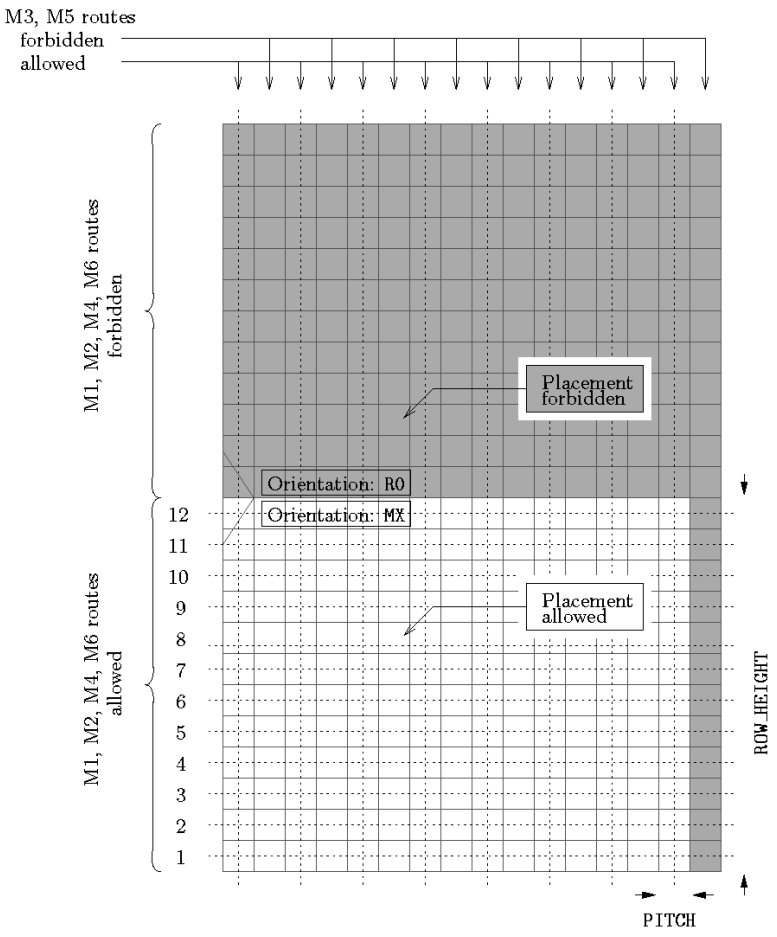
- **dual cells have similar layout in transistor**
- **the port position allow for a differential routing**

« Backend-duplication » overview

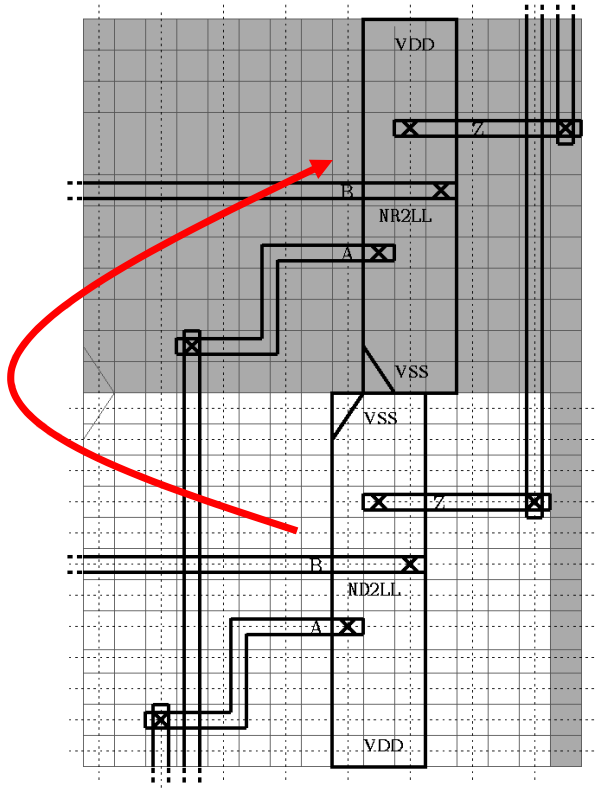




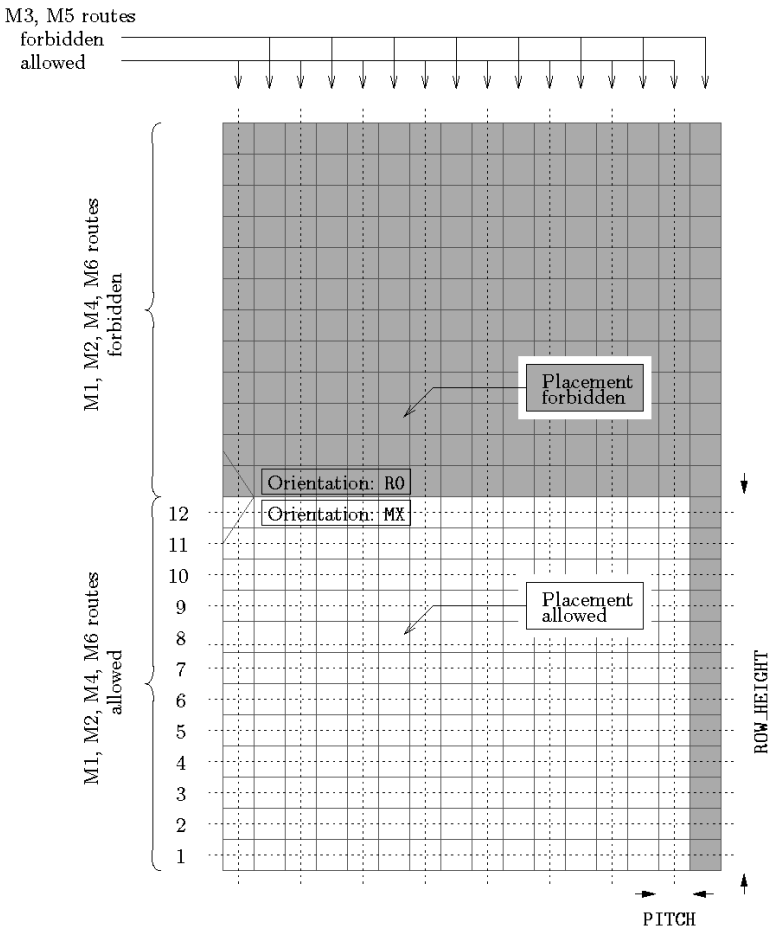
« Backend-duplication »: placement



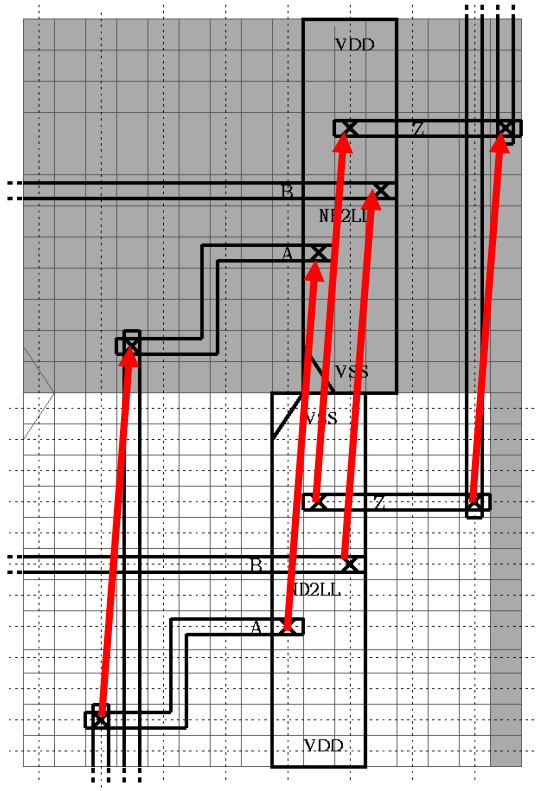
Flip Placement



« Backend-duplication »: routing

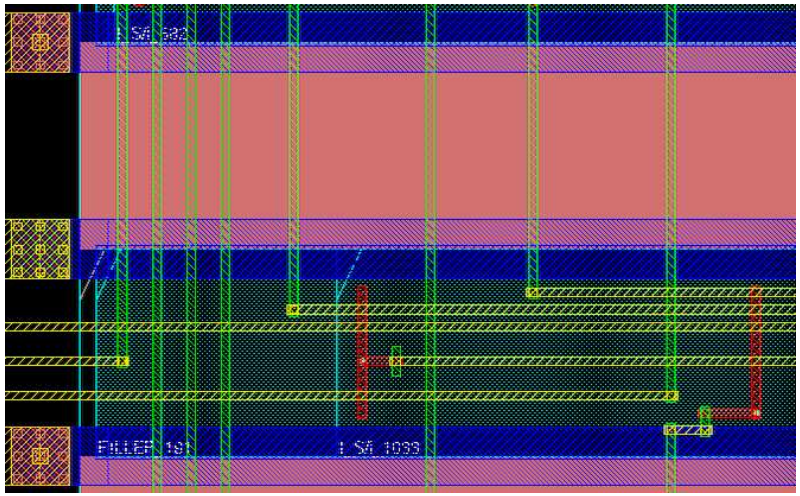


Translate routing



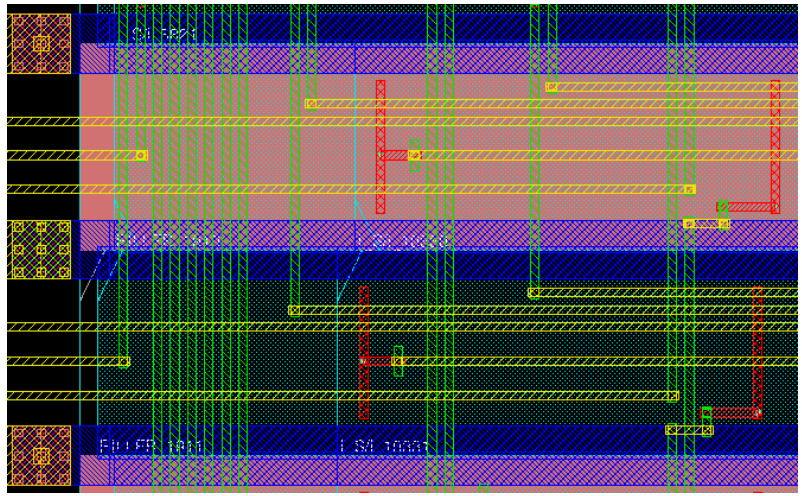
« Backend-duplication » Realization

Before duplication



- Half of the placement rows are obstructed
- Half of the routing channel are obstructed

After duplication



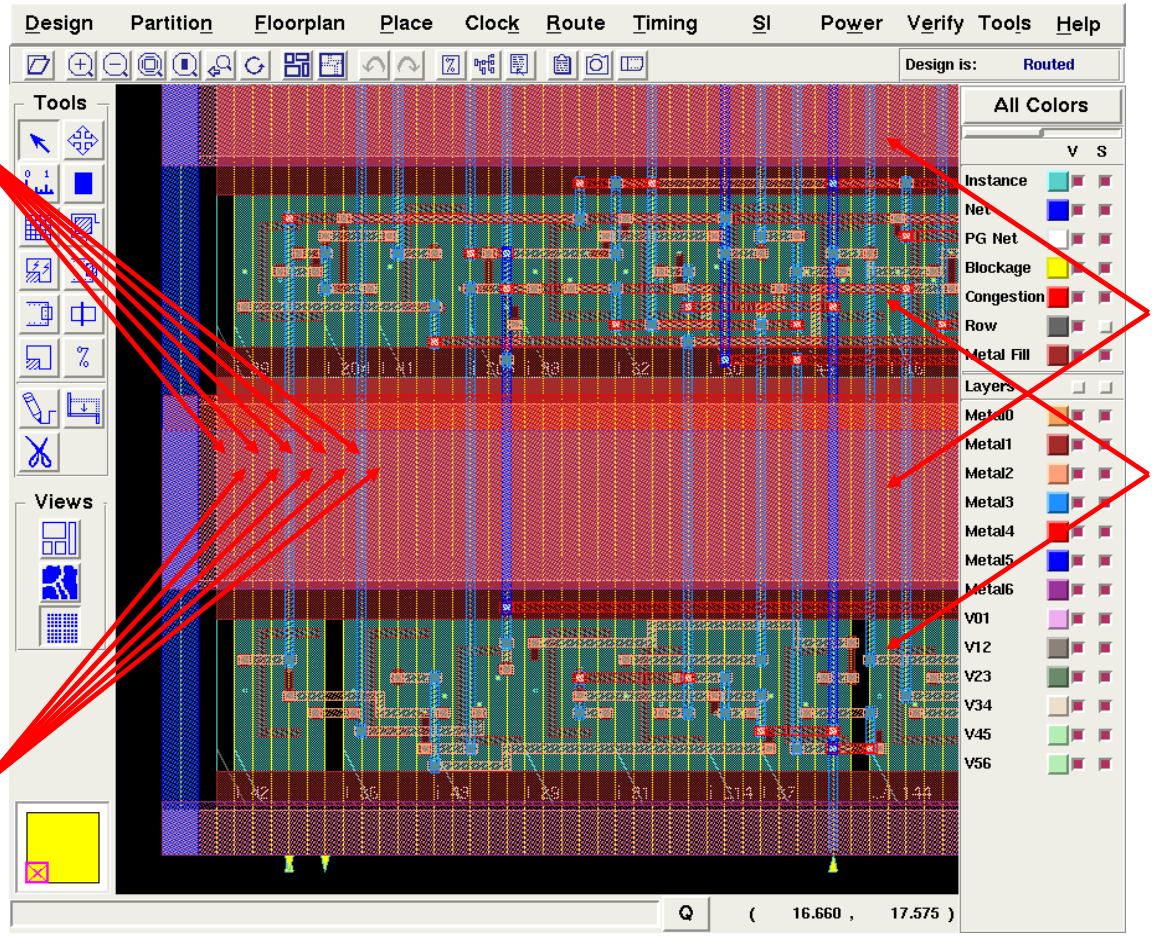
- Cells are duplicated by vertical flip (`R0`→ `MX`)
- Routing is translated by: (`PITCH`, `ROW_HEIGHT`)

The method fully relies on the setting of appropriate constraints

WDDL example: constraints

No vertical routing

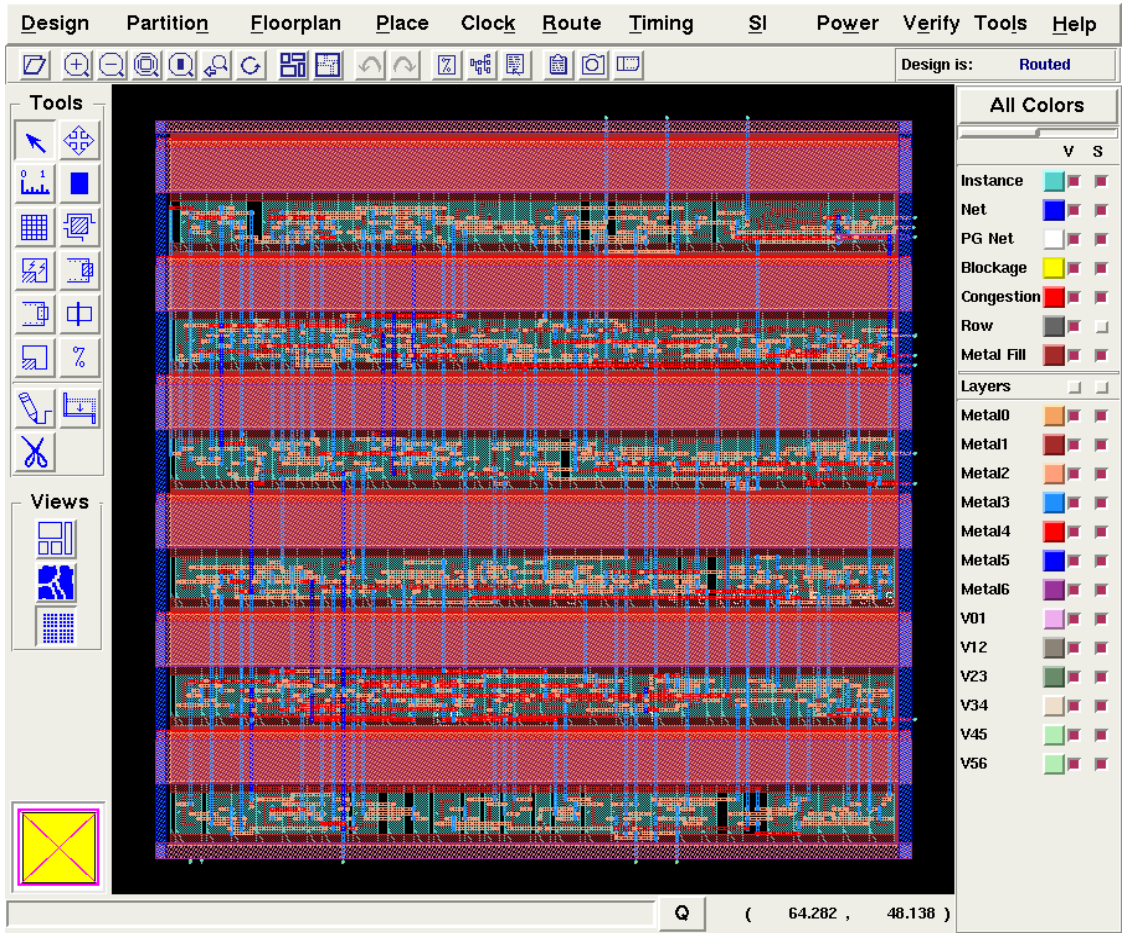
Vertical routing OK



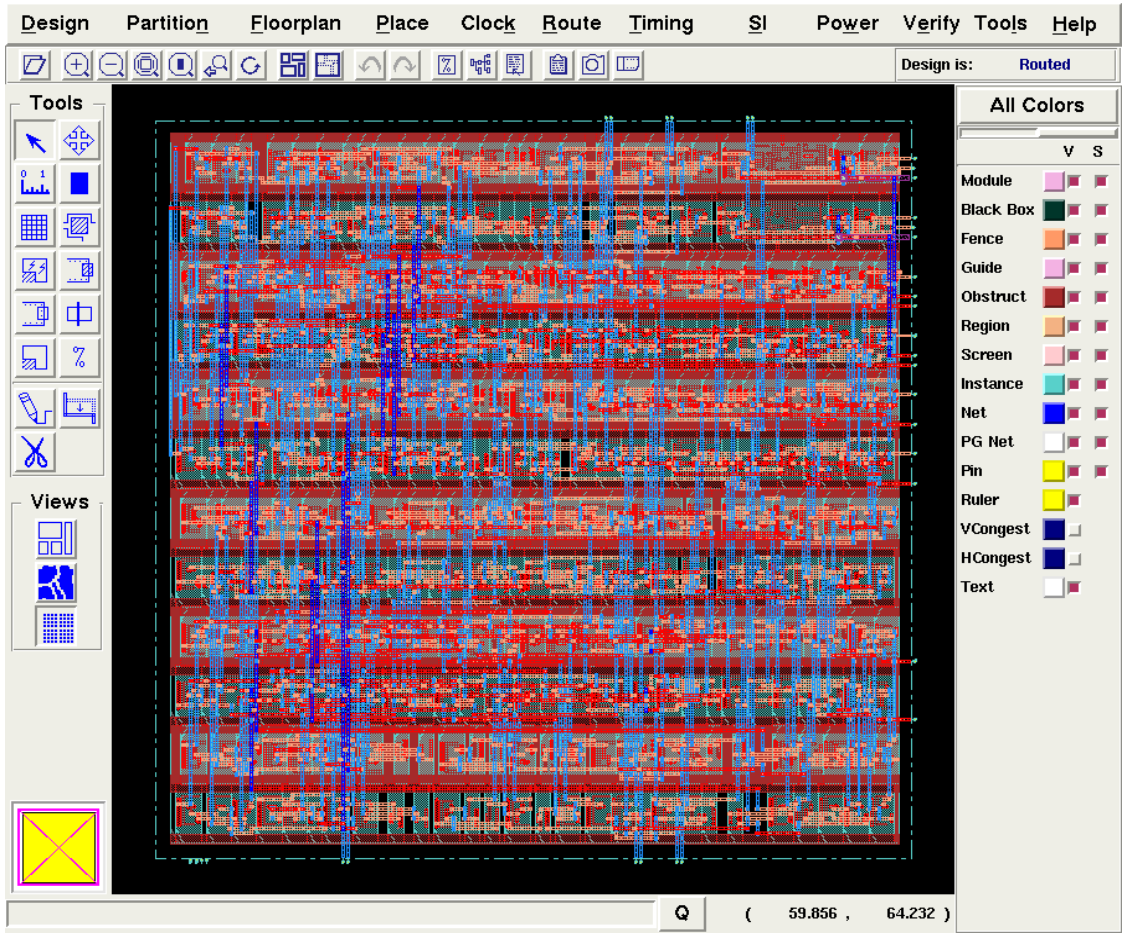
No placement
No routing

Placement OK
Horizontal routing OK

WDDL example: before duplication



WDDL example: after duplication

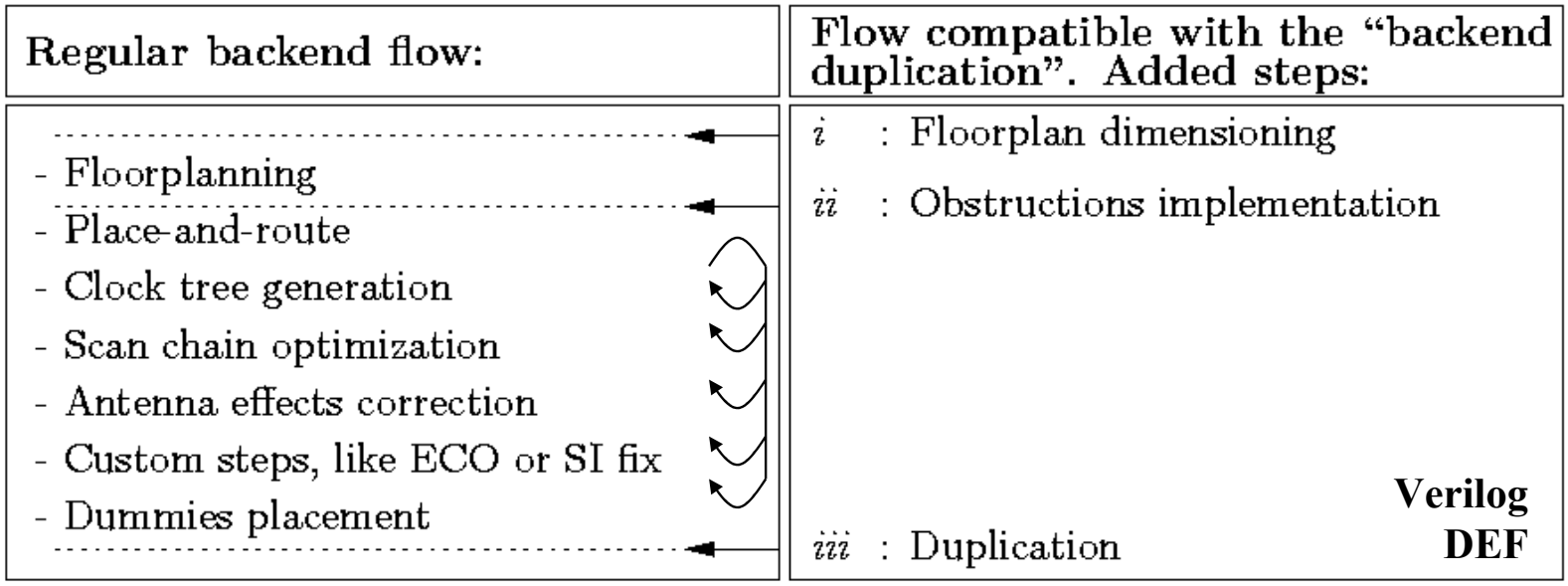


Note:

Results can be visualized in a backend tool without rewritting (*error-prone*) nor reloading (*not interactive*) design rules.

Implementation

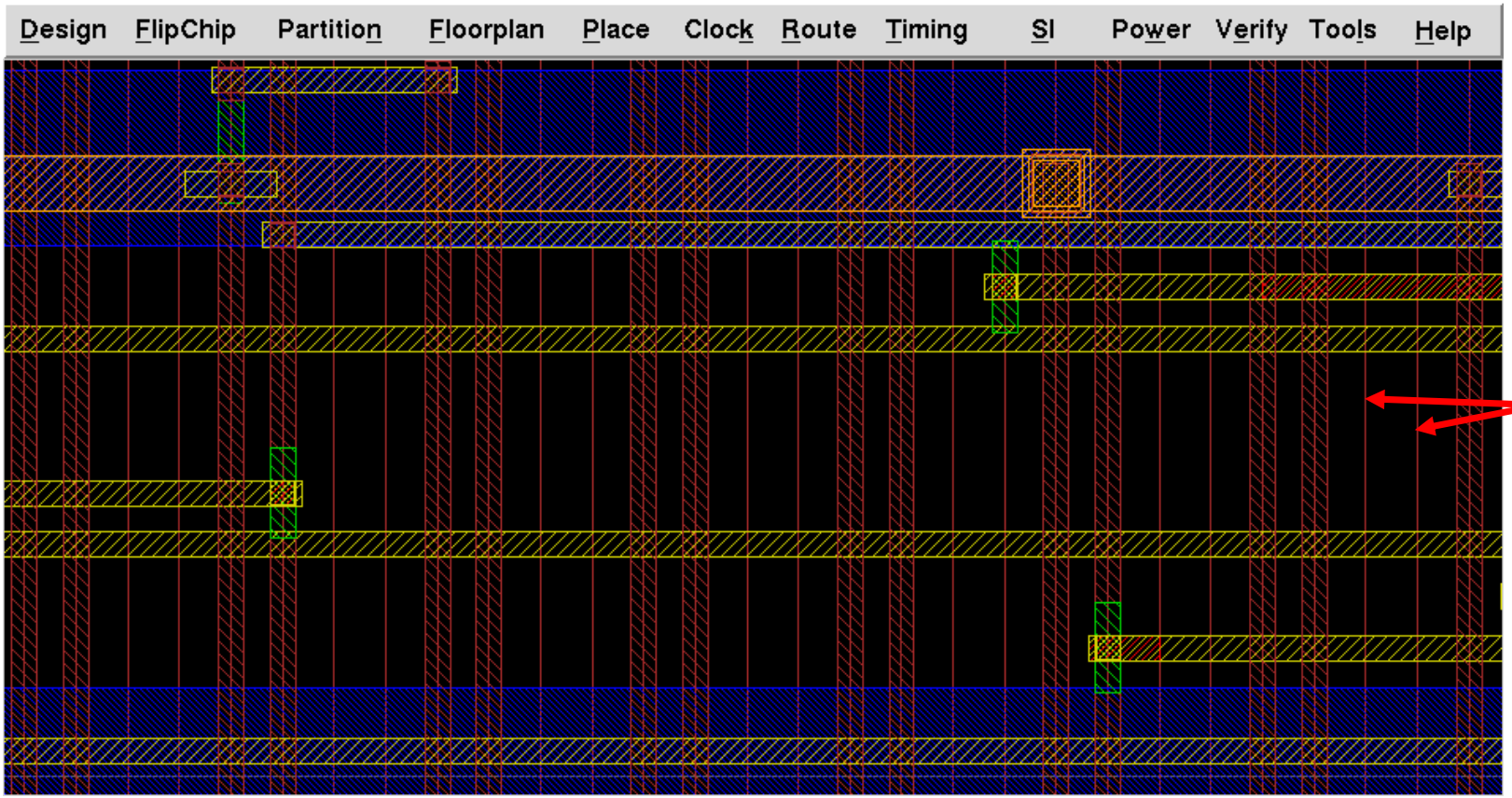
+3 lines added in the Makefile:



Execution time in the example of DES:

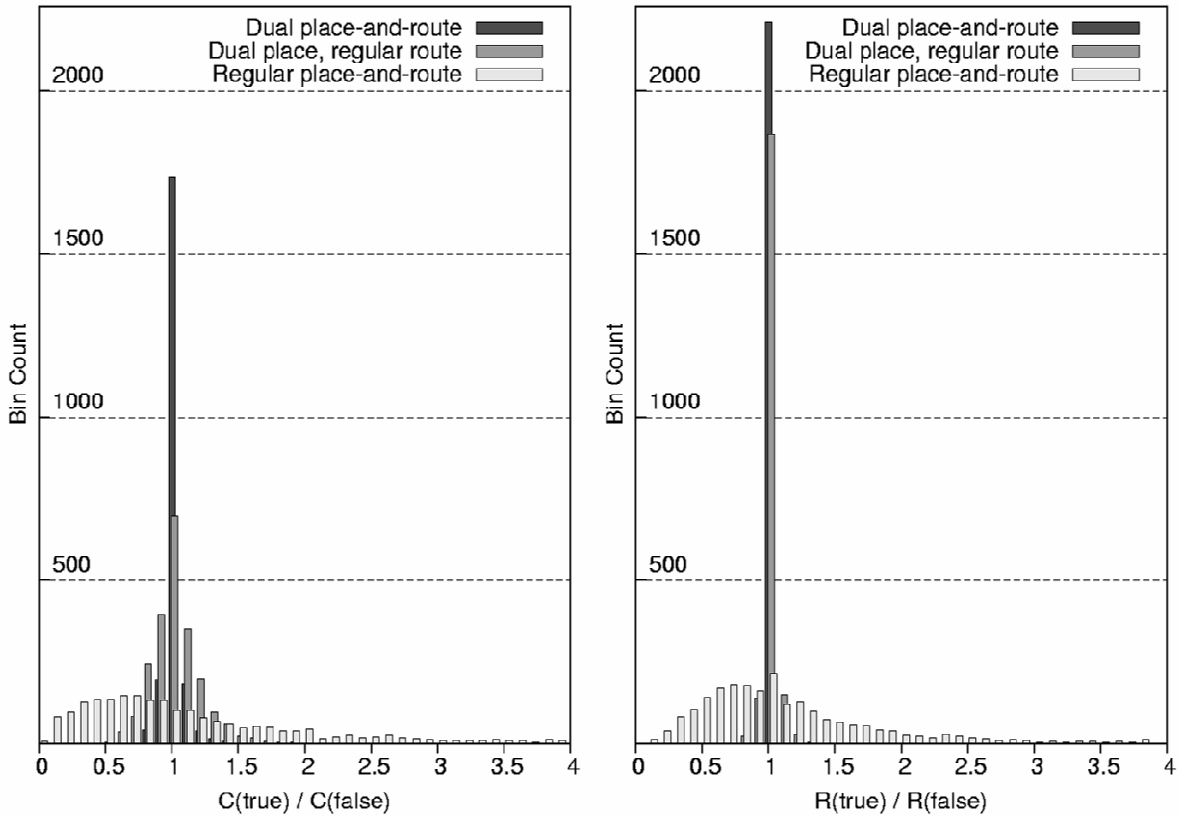
	Regular	Backend-duplicated
Place	1.9 s	6.2 s
Route	39.0 s	80.0 s
Duplication	-	77.5 s

Reducing the cross-coupling: routing constraints



Routing forbidden: tracks obstrusted = shield

« Backend Duplication » Efficiency Assessment



In FPGAs ; example for Altera [GCS+08].

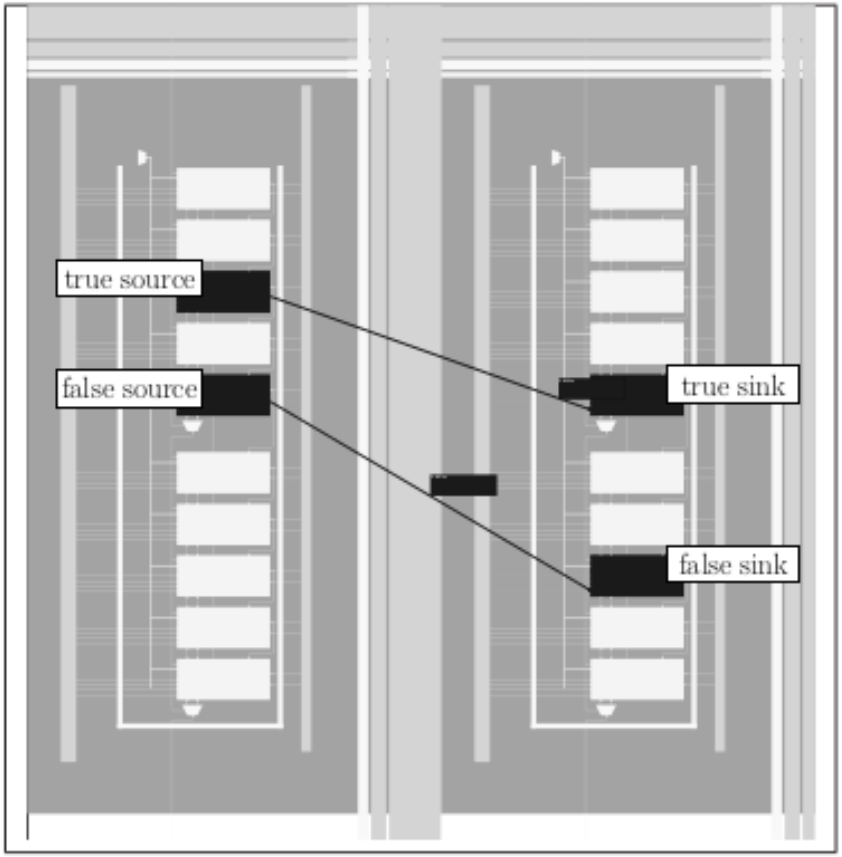
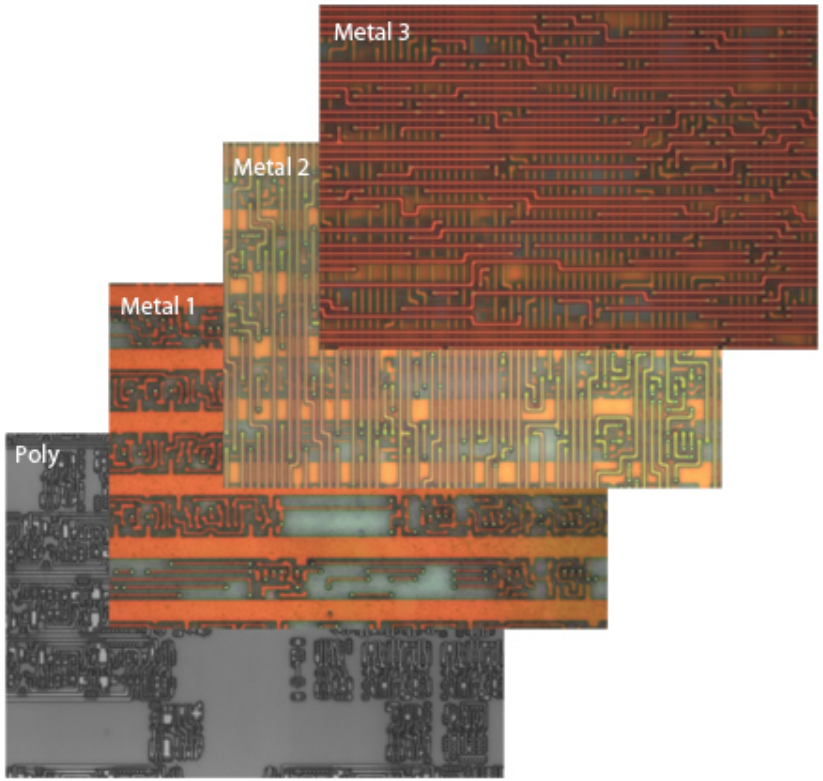


Figure 1. Constrained dual-placed DES sbox # 5 in a Stratix (zoom on two adjacent LABs).

Against delayering



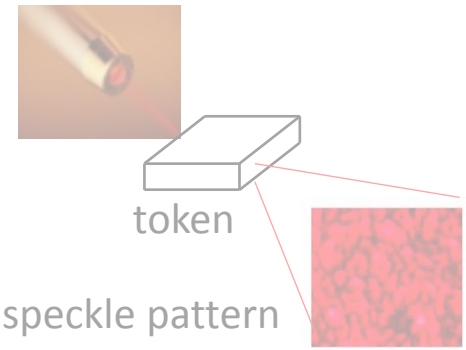
- PUF : Physically Unclonable Functions
  - Optical PUF [Pap01]
  - Coating PUF [TSK07]
  - SRAM PUF [HBF09]
  - Glitch PUF [SS10]
  - Arbiter PUF [GCvDD02]
  - Loop PUF [CDGB12]
  - Memory contention PUF [Gün12]
- NVM : Non-Volatile Memory

Overview

- Background
  - Management issues on Critical Security Parameter (CSP) storage on crypto chips.
    - Key generation – generated keys are stored
    - Random number generation – a random seed is stored for PRNG
  - Stored values may be taken out and copied by using reverse-engineering techniques.
- PUF – Physically (Physical) Unclonable Function
  - Every semiconductor chip has intrinsic subtle variations in its physical properties.
  - These variations are unique to each chip and very hard to clone.
  - A PUF is based on such variations and considered an object's fingerprint.
  - PUFs take the same input but respond with different outputs.
  - Enables non-stored, internally-generated CSP management.
  - Building structures and evaluation metrics have been studied.
  - Some technical considerations:
    - A typical use of PUF involves a challenge (input) and response (output) scheme.
    - An error correction scheme should be combined in use because a PUF does not generate the exactly same output every time (it contains a partial errors due to the physical variations).
    - Not designed to be a TRNG, but may be utilized to make a TRNG.
  - Applications of PUFs fall into two categories:
    - Anti-counterfeiting with product authentication (goes to other standardization groups: ISO TC247, ISO/IEC SC31, SEMI, etc.)
    - Information security : Non-stored CSP generation (above-mentioned)
  - Related businesses are emerging.
- Purpose of standardization
  - Before non-interoperable/interchangeable or low-reliability PUF applications are widely distributed, a well-considered standard must be established.
    - For a higher usability and reliability
    - For building a wider market

Examples of PUFs

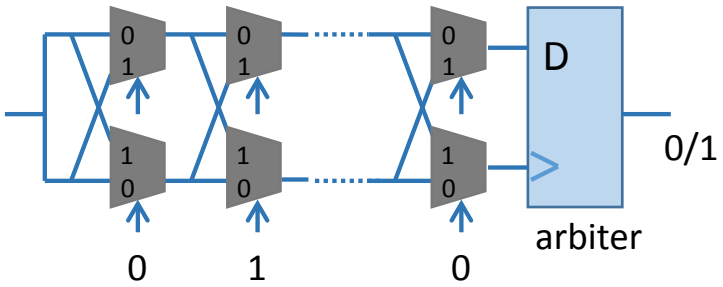
Optical PUF



Uses speckle pattern of transmitted laser

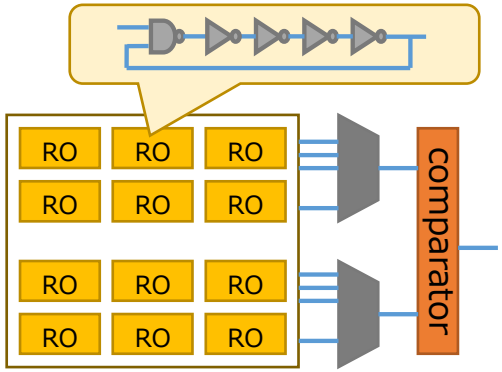
Pappu Srinivasa Ravikanth, "Physical One-Way Functions," PhD Thesis, MIT, 2001.

Arbiter PUF



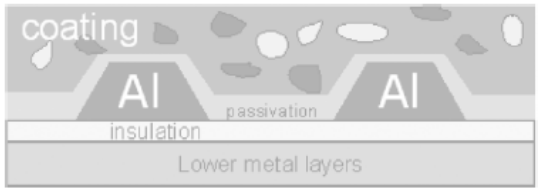
Uses the delay difference of two signals

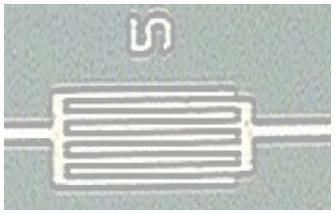
Ring-Oscillator PUF



Uses the difference of oscillating frequencies

Coating PUF



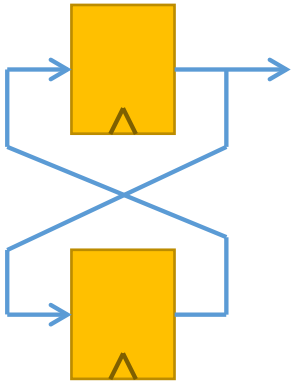


capacitive sensor

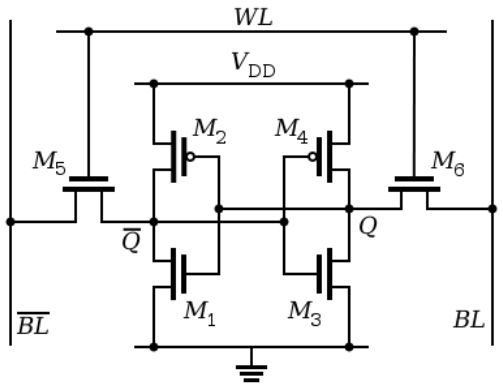
Uses the difference of capacitive load

Skoric, B., et al. "Experimental hardware for coating PUFs and optical PUFs," *Security with Noisy Data*. Springer London, 2007. 255-268.

Butterfly PUF



SRAM PUF

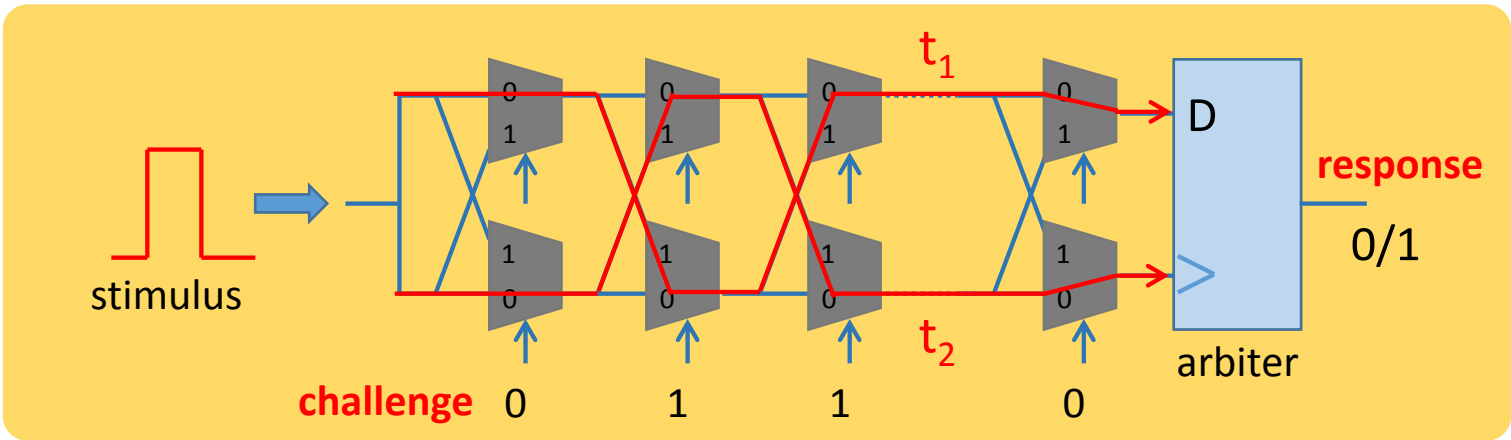


http://commons.wikimedia.org/

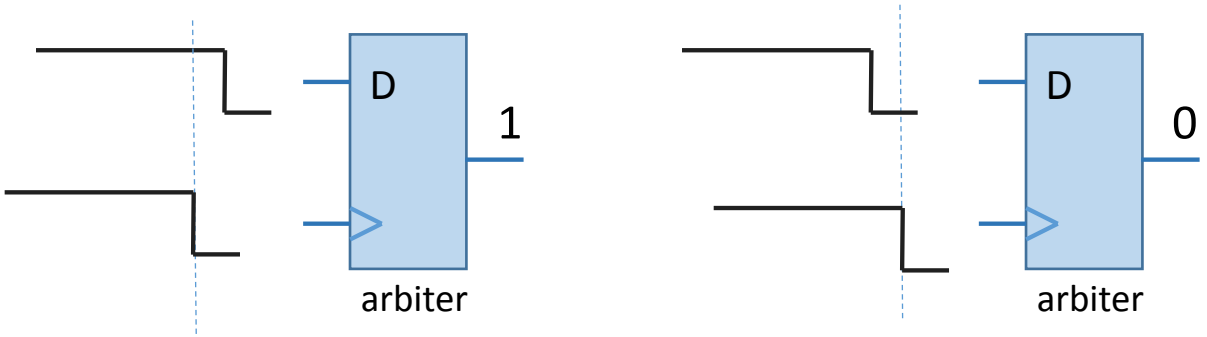
Uses the difference of the initial state of memory

Arbiter PUF

- Utilizes the delay difference of two selector chains

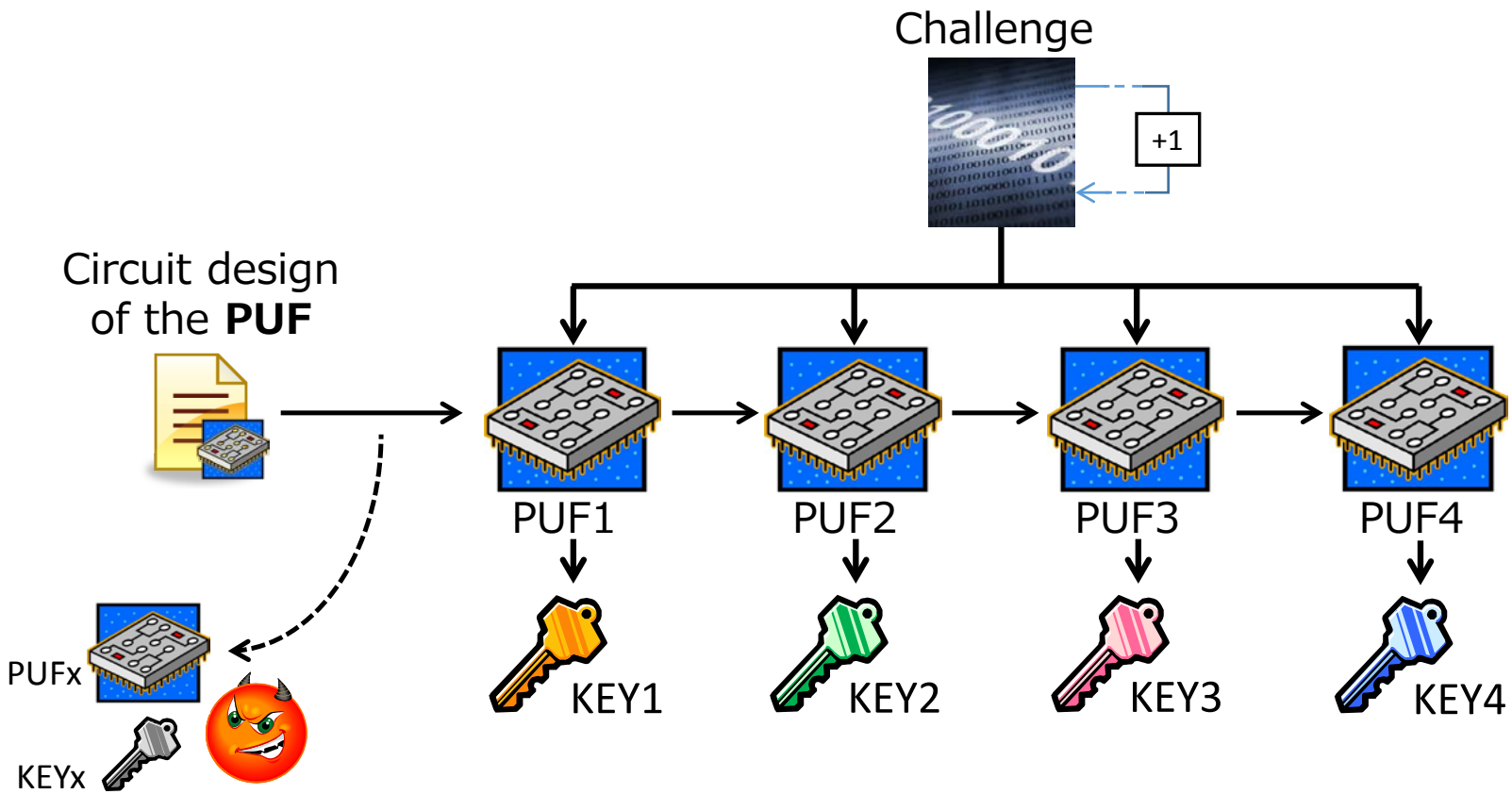


- Arbiter implemented using D-FF



Key Generation Using PUF

- The **same** circuit design is used for all the chips to configure PUFs
- The **same** challenge is given to each PUF
- The PUFs generate **different** keys



Presentation Outline

Recommendations for symmetric algorithms

No timing constraints

- Against SCA : 1st order masking and/or shuffling
- Against DFA : check by decryption

Otherwise use Akashi Satoh's trick [SSHA08]

- Against SCA : 1st order masking and/or shuffling
- Against DFA : check encryption with decryption hardware

For hardware

- Against SCA and DFA : Use masked dual-rail

Recommendations for asymmetric algorithms

RSA : use PKCS

- Secure against all attacks (the input is formatted randomly, and thus unknown to the attacker)
- Just protect against SPA, by key blinding or side-channel atomicity [CMCJ04]

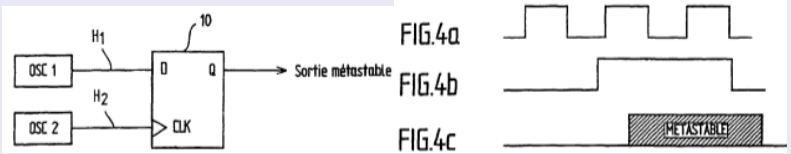
Plain RSA

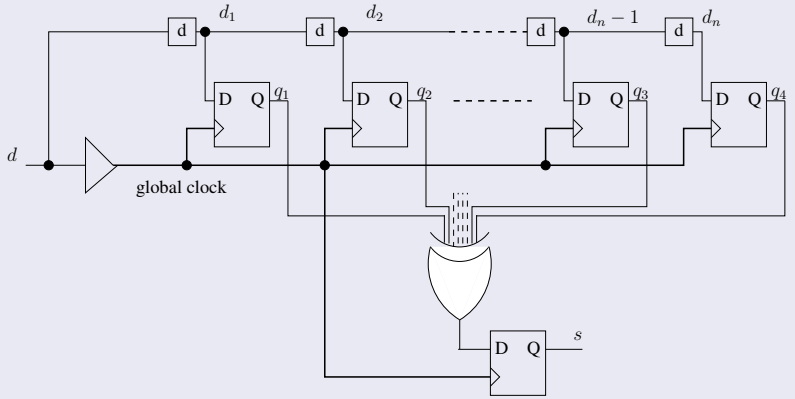
- Exponent splitting (doubles the cost) if SPA secure.

Recommendations for misc auxiliary functions

TRNG

- Avoid ring-based structures
- Open-loop solutions exist : [Qué03], [DGH09]





PUF

- Avoid delay-PUF or hash the output to prevent modeling attacks [RSS+10].

[AG01] Mehdi-Laurent Akkar and Christophe Giraud.
An Implementation of DES and AES Secure against Some Attacks.
In LNCS, editor, *Proceedings of CHES'01*, volume 2162 of *LNCS*, pages 309–318. Springer, May 2001.
Paris, France.

[BHT09] Arnaud Boscher, Helena Handschuh, and Elena Trichina.
Blinded Fault Resistant Exponentiation Revisited.
In *FDTC*, pages 3–9. IEEE Computer Society, September 6 2009.
Lausanne, Switzerland.

[BK09] Alex Biryukov and Dmitry Khovratovich.
Related-Key Cryptanalysis of the Full AES-192 and AES-256.
In Mitsuru Matsui, editor, *ASIACRYPT*, volume 5912 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2009.

[CDGB12] Zouha Cherif, Jean-Luc Danger, Sylvain Guilley, and Lilian Bossuet.
An Easy-to-Design PUF based on a single oscillator : the Loop PUF.
In *DSD*, September 5-8 2012.
Çeşme, Izmir, Turkey ; (Online PDF).

[CM09] Jean-Sébastien Coron and Avradip Mandal.
PSS Is Secure against Random Fault Attacks.
In *ASIACRYPT*, volume 5912 of *LNCS*, pages 653–666. Springer, December 6-10 2009.
Tōkyō, Japan.

[CMCJ04] Benoît Chevallier-Mames, Mathieu Ciet, and Marc Joye.
Low-Cost Solutions for Preventing Simple Side-Channel Analysis : Side-Channel Atomicity.
IEEE Trans. Computers, 53(6) :760–768, 2004.

[DGH09] Jean-Luc Danger, Sylvain Guilley, and Philippe Hoogvorst.
High Speed True Random Number Generator based on Open Loop Structures in FPGAs.
Microelectronics Journal, 40(11) :1650–1656, November 2009.
DOI : 10.1016/j.mejo.2009.02.004.

[FD01] Viktor Fischer and Miloš Drutarovský.
Two Methods of Rijndael Implementation in Reconfigurable Hardware.
In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *CHES*, volume 2162 of *Lecture Notes in Computer Science*, pages 77–92. Springer, 2001.

[FGL+13] Pierre-Alain Fouque, Nicolas Guillermin, Delphine Leresteux, Mehdi Tibouchi, and Jean-Christophe Zapalowicz.
Attacking RSA-CRT signatures with faults on Montgomery multiplication.
J. Cryptographic Engineering, 3(1) :59–72, 2013.

[FMPR10] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain.
Affine Masking against Higher-Order Side Channel Analysis.
In Alex Biryukov, Guang Gong, and Douglas R. Stinson, editors, *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

[GCS+08] Sylvain Guilley, Sumanta Chaudhuri, Laurent Sauvage, Tarik Graba, Jean-Luc Danger, Philippe Hoogvorst, Vinh-Nga Vong, and Maxime Nassar.
Place-and-Route Impact on the Security of DPL Designs in FPGAs.
In *HOST (Hardware Oriented Security and Trust), IEEE*, pages 29–35, Anaheim, CA, USA, jun 2008.

[GCvDD02] Blaise Gassend, Dwaine E. Clarke, Marten van Dijk, and Srinivas Devadas.
Silicon physical random functions.
In Vijayalakshmi Atluri, editor, *Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, Washington, DC, USA, November 18-22, 2002*, pages 148–160. ACM, 2002.

[GM11] Tim Güneysu and Amir Moradi.
Generic side-channel countermeasures for reconfigurable devices.
In Bart Preneel and Tsuyoshi Takagi, editors, *CHES*, volume 6917 of *LNCS*, pages 33–48. Springer, 2011.

[GP99] Louis Goubin and Jacques Patarin.
DES and Differential Power Analysis. The "Duplication" Method.
In *CHES*, LNCS, pages 158–172. Springer, Aug 1999.
Worcester, MA, USA.

[GSD+08] Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger, Tarik Graba, and Yves Mathieu.
Evaluation of Power-Constant Dual-Rail Logic as a Protection of Cryptographic Applications in FPGAs.
In *SSIRI*, pages 16–23, Yokohama, Japan, jul 2008. IEEE Computer Society.
DOI : 10.1109/SSIRI.2008.31, http://hal.archives-ouvertes.fr/hal-00259153/en/.

[GSDS10] Sylvain Guilley, Laurent Sauvage, Jean-Luc Danger, and Nidhal Selmane.
Fault Injection Resilience.
In *FDTC*, pages 51–65. IEEE Computer Society, August 21 2010.
Santa Barbara, CA, USA. DOI : 10.1109/FDTC.2010.15 ; Complete version : http://hal.archives-ouvertes.fr/hal-00482194/en/.

[Gün12] Tim Güneysu.
Using Data Contention in Dual-ported Memories for Security Applications.
Signal Processing Systems, 67(1) :15–29, 2012.

[GW08] Catherine H. Gebotys and Brian A. White.
EM analysis of a wireless Java-based PDA.
ACM Trans. Embedded Comput. Syst., 7(4) :44 :1–44 :28, 2008.

[HBF09] D.E. Holcomb, W.P. Burleson, and K. Fu.
Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers.
Computers, IEEE Transactions on, 58(9) :1198 –1210, sept. 2009.

[Joy03] Marc Joye.
Ingénierie cryptographique : Aspects sécuritaires et algorithmiques, September 2003.
Toulouse, France. http://joye.site88.net/theses/Joye_HDR.pdf.

[KDK+14] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji-Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu.
Flipping bits in memory without accessing them : An experimental study of DRAM disturbance errors.
In *ACM/IEEE 41st International Symposium on Computer Architecture, ISCA 2014, Minneapolis, MN, USA, June 14-18, 2014*, pages 361–372. IEEE Computer Society, 2014.

[Koc96] Paul C. Kocher.
Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.
In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[Koc03] Paul C. Kocher.
Leak-resistant cryptographic indexed key update, March 25 2003.
United States Patent 6,539,092 filed on July 2nd, 1999 at San Francisco, CA, USA.

[Koc05] Paul C. Kocher.
Design and Validation Strategies for Obtaining Assurance in Countermeasures to Power Analysis and Related Attacks, September 26-29 2005.
Honolulu, Hawai, USA ; NIST's Physical Security Testing Workshop. Website :
http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-3/physec/physecdoc.html.

[MBKP11] Amir Moradi, Alessandro Barenghi, Timo Kasper, and Christof Paar.
On the vulnerability of FPGA bitstream encryption against power analysis attacks : extracting keys from Xilinx Virtex-II FPGAs.
In Yan Chen, George Danezis, and Vitaly Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 111–124. ACM, 2011.

[MBO+05] Elke De Mulder, Pieter Buysschaert, Sıddıka Berna Örs, Peter Delmotte, Bart Preneel, Guy Vandenbosch, and Ingrid Verbauwhede.
Electromagnetic Analysis Attack on an FPGA Implementation of an Elliptic Curve Cryptosystem.
In *IEEE International Conference on Computer as a tool (EUROCON)*, pages 1879–1882, November 2005.
Belgrade, Serbia & Montenegro. DOI : 10.1109/EURCON.2005.1630348,
http://www.sps.ele.tue.nl/members/m.j.bastiaans/spc/demulder.pdf.

[MKP11] Amir Moradi, Markus Kasper, and Christof Paar.
On the Portability of Side-Channel Attacks — An Analysis of the Xilinx Virtex 4 and Virtex 5 Bitstream Encryption Mechanism.
Cryptology ePrint Archive, Report 2011/391, 2011.
http://eprint.iacr.org/2011/391/.

[MKP12] Amir Moradi, Markus Kasper, and Christof Paar.
Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures - An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism.
In Orr Dunkelman, editor, *CT-RSA*, volume 7178 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2012.

[MSGR10] Marcel Medwed, François-Xavier Standaert, Johann Großschädl, and Francesco Regazzoni.
Fresh Re-Keying : Security against Side-Channel and Fault Attacks for Low-Cost Devices.
In *AFRICACRYPT*, volume 6055 of *LNCS*, pages 279–296. Springer, May 03-06 2010.
Stellenbosch, South Africa. DOI : 10.1007/978-3-642-12678-9_17.

[Pap01] Ravikanth S. Pappu.
Physical One-Way Functions.
PhD thesis, Massachusetts Institute of Technology, March 2001.

[PR10] Emmanuel Prouff and Thomas Roche.
Attack on a Higher-Order Masking of the AES Based on Homographic Functions.
In Guang Gong and Kishan Chand Gupta, editors, *INDOCRYPT*, volume 6498 of *Lecture Notes in Computer Science*, pages 262–281. Springer, 2010.

[PS08] Gilles Piret and François-Xavier Standaert.
Security Analysis of Higher-Order Boolean Masking Schemes for Block Ciphers (with Conditions of Perfect Masking).
IET Information Security, 2(1) :1–11, 2008.
DOI : 10.1049/iet-ifs :20070066.

[Qué03] Patrick Le Quéré.
High Rate Random Number Generator, November 19 2003.
Patent EP1159673, (also WO0146797), http://www.freepatentsonline.com/EP1159673.html.

[RBG+15] Debapriya Basu Roy, Shivam Bhasin, Sylvain Guilley, Jean-Luc Danger, and Debdeep Mukhopadhyay.
From Theory to Practice of Private Circuit : A Cautionary Note.
In *The 33rd IEEE International Conference on Computer Design (ICCD '15)*, pages 296–303, October 18-21 2015.
New York City, USA. DOI : 10.1109/ICCD.2015.7357117.

[RP10] Matthieu Rivain and Emmanuel Prouff.
Provably Secure Higher-Order Masking of AES.
In Stefan Mangard and François-Xavier Standaert, editors, *CHES*, volume 6225 of *LNCS*, pages 413–427. Springer, 2010.

[RSS+10] Ulrich Rührmair, Frank Sehnke, Jan Sölter, Gideon Dror, Srinivas Devadas, and Jürgen Schmidhuber.
Modeling attacks on physical unclonable functions.
In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 237–249, New York, NY, USA, 2010. ACM.

[SGD+09] Laurent Sauvage, Sylvain Guilley, Jean-Luc Danger, Yves Mathieu, and Maxime Nassar.
Successful Attack on an FPGA-based Automatically Placed and Routed WDDL+ Crypto Processor.
In *DATE, track A4 (Secure embedded implementations)*, April 20–24 2009.
Nice, France. Electronic version : http://hal.archives-ouvertes.fr/hal-00325417/en/.

[SGM09] Laurent Sauvage, Sylvain Guilley, and Yves Mathieu.
ElectroMagnetic Radiations of FPGAs : High Spatial Resolution Cartography and Attack of a Cryptographic Module.
ACM Trans. Reconfigurable Technol. Syst., 2(1) :1–24, March 2009.
Full text in http://hal.archives-ouvertes.fr/hal-00319164/en/.

[SS10] Daisuke Suzuki and Koichi Shimizu.
The Glitch PUF : A New Delay-PUF Architecture Exploiting Glitch Shapes.
In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 366–382. Springer, August 17-20 2010.
Santa Barbara, CA, USA.

[SSHA08] Akashi Satoh, Takeshi Sugawara, Naofumi Homma, and Takafumi Aoki.
High-Performance Concurrent Error Detection Scheme for AES Hardware.
In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 100–112. Springer, 2008.

[TMA11] Michael Tunstall, Debdeep Mukhopadhyay, and Subidh Ali.
Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault.
In Claudio Agostino Ardagna and Jianying Zhou, editors, *WISTP*, volume 6633 of *Lecture Notes in Computer Science*, pages 224–233. Springer, 2011.

[TSK07] Pim Tuyls, Boris Skoric, and Tom Kevenaar.
Security with Noisy Data : Private Biometrics, Secure Key Storage and Anti-Counterfeiting.
Springer-Verlag New York, Inc., Secaucus, NJ, USA, December 2007.
1st Edition, ISBN 978-1-84628-983-5.