Circuits électroniques de cryptographie
attaques et protections

Sylvain GUILLEY.

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

ENS L3, “systèmes numériques”
Presentation Outline

1. Cryptography
2. Attacks
3. Protection
4. Conclusions
AddRoundKey

```
| a_{0,0} | a_{0,1} | a_{0,2} | a_{0,3} |
| a_{1,0} | a_{1,1} | a_{1,2} | a_{1,3} |
| a_{2,0} | a_{2,1} | a_{2,2} | a_{2,3} |
| a_{3,0} | a_{3,1} | a_{3,2} | a_{3,3} |
```

```
| b_{0,0} | b_{0,1} | b_{0,2} | b_{0,3} |
| b_{1,0} | b_{1,1} | b_{1,2} | b_{1,3} |
| b_{2,0} | b_{2,1} | b_{2,2} | b_{2,3} |
| b_{3,0} | b_{3,1} | b_{3,2} | b_{3,3} |
```

AddRoundKey

```
| k_{0,0} | k_{0,1} | k_{0,2} | k_{0,3} |
| k_{1,0} | k_{1,1} | k_{1,2} | k_{1,3} |
| k_{2,0} | k_{2,1} | k_{2,2} | k_{2,3} |
| k_{3,0} | k_{3,1} | k_{3,2} | k_{3,3} |
```

Bit-wise XOR

```
| a_{2,2} |
| b_{2,2} |

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
SubBytes

Understand “Substitute Bytes”

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

Data independent circular shift

\[
\begin{array}{cccc}
  a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\
  a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\
  a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\
  a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\
\end{array}
\rightarrow
\begin{array}{cccc}
  a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\
  a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\
  a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\
  a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \\
\end{array}
\]
MixColumns

\[
\begin{align*}
\begin{array}{ccc}
 a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\
 a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\
 a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\
 a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \\
\end{array}
\end{align*}
\]

\[
\begin{align*}
\begin{array}{ccc}
 b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\
 b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\
 b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\
 b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \\
\end{array}
\end{align*}
\]

\[\times c(x)\]

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

\[
\begin{align*}
\text{MixColumns} : & \quad a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\} , \\
\text{InvMixColumns} : & \quad a^{-1}(x) = \{0b\}x^3 + \{0d\}x^2 + \{09\}x + \{0e\}.
\end{align*}
\]
Encryption

- **AddRoundKey** // Isolated

- **SubBytes**
  - **ShiftRows**
  - **MixColumns**
  - **AddRoundKey** // \( n \) times ...

- **SubBytes**
  - **ShiftRows**
  - **AddRoundKey** // Last round

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

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
## 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...

---

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
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**:
\[
\text{InvMixColumns}(\text{state} \oplus K) = \text{InvMixColumns}(\text{state}) \oplus \text{InvMixColumns}(K).
\]

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
C language:

```c
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:

```c++
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:

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

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:

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

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:

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

Computation of column \( 0 \leq j \leq 3 \) in one round:

\[
\begin{pmatrix}
  e_{0,j} \\
  e_{1,j} \\
  e_{2,j} \\
  e_{3,j}
\end{pmatrix} = \begin{pmatrix}
  k_{0,j} \\
  k_{1,j} \\
  k_{2,j} \\
  k_{3,j}
\end{pmatrix} \oplus \begin{pmatrix}
  02 \\
  01 \\
  01 \\
  03
\end{pmatrix} \oplus \begin{pmatrix}
  01 \\
  02 \\
  03 \\
  01
\end{pmatrix} \oplus \begin{pmatrix}
  01 \\
  03 \\
  02 \\
  01
\end{pmatrix} \oplus \begin{pmatrix}
  01 \\
  01 \\
  03 \\
  02
\end{pmatrix}
\]
Optimization on 32-bit Machines (see [FD01, GW08])

Notations:

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

Computation of column \(0 \leq j \leq 3\) in one round:

\[
\begin{pmatrix}
e_{0,j} \\
e_{1,j} \\
e_{2,j} \\
e_{3,j}
\end{pmatrix} = \begin{pmatrix}
k_{0,j} \\
k_{1,j} \\
k_{2,j} \\
k_{3,j}
\end{pmatrix} \oplus \begin{pmatrix}
02 \\
01 \\
01 \\
03
\end{pmatrix} b_{0,j} \oplus \begin{pmatrix}
03 \\
02 \\
01 \\
01
\end{pmatrix} b_{1,j+1} \oplus \begin{pmatrix}
01 \\
01 \\
03 \\
02
\end{pmatrix} b_{2,j+2} \oplus \begin{pmatrix}
01 \\
01 \\
03 \\
02
\end{pmatrix} b_{3,j+3}
\]
Optimization on 32-bit Machines (see [FD01, GW08])

Notations:

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

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

\[
\begin{pmatrix}
  e_{0,j} \\
  e_{1,j} \\
  e_{2,j} \\
  e_{3,j}
\end{pmatrix} = \begin{pmatrix}
  k_{0,j} \\
  k_{1,j} \\
  k_{2,j} \\
  k_{3,j}
\end{pmatrix} \oplus \bigoplus_{k=0}^{3} (MC_k) b_{k,j+k}
\]
Optimization on 32-bit Machines (see [FD01, GW08])

Notations:

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

Computation of column \(0 \leq j \leq 3\) in one round:

\[
\begin{pmatrix}
  e_{0,j} \\
  e_{1,j} \\
  e_{2,j} \\
  e_{3,j}
\end{pmatrix} = \begin{pmatrix}
  k_{0,j} \\
  k_{1,j} \\
  k_{2,j} \\
  k_{3,j}
\end{pmatrix} \oplus \bigoplus_{k=0}^{3} (MC_k) S(a_{k,j+k})
\]
Optimization on 32-bit Machines (see [FD01, GW08])

Notations:

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

Computation of column \(0 \leq j \leq 3\) in one round:

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

\(\text{Te}_k\) are four memories of 256 words of 32 bits.
 OpenSSL AES code

```c
/* 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**

- $S^e \equiv H(m) \mod N$
Presentation Outline

1. Cryptography
2. Attacks
3. Protection
4. Conclusions
## Overview of attacks

<table>
<thead>
<tr>
<th></th>
<th>Side-channel attack</th>
<th>Fault attack</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSA</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong>AES</strong></td>
<td>100</td>
<td>2</td>
</tr>
</tbody>
</table>

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
Altera Excalibur evaluation board “customized for DPA”

Resistor:
\[ u = Ri, \text{ where } R \]

\[ u \]

\[ i \]

6.8 \( \Omega \) resistor

Voltage regulator

Power pins of the FPGA unsoldered

Coaxial cable for the power acquisition

Ground

External clock input

External 5.0 V power supply

External 1.8 V power supply

S. Guilley, <sylvain.guilley@telecom-paristech.fr>
Parallax ALTERA Stratix board “customized for DPA” [GSD+08]

Inductor:

\[ u = L \frac{\partial i}{\partial t} \]

where \( L \) is the inductance.

S. Guilley, <sylvain.guilley@telecom-paristech.fr>
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 \([\text{SGM09, SGD}^+09]\)
XILINX Virtex-5 evaluation board “customized for EMA”

XC5VLX50 evaluation board

FPGA chip

Metallic cover

FF324 (18^2 pins) socket
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**: $M, d$
- $R = 1$
- \[
\text{for } i = |d| - 1; \ i \geq 0; \ i -- \ do
\]
  \[
  R = R^2
  \]
  \[
  \text{/* Unbalanced branching */}
\]
  \[
  \text{if } d_i == 1 \ \text{then}
  \]
  \[
  R = R \times M
  \]
  \[
  \text{end if}
\]
- \[
\text{end for}
\]
- \[
\text{Return } R = M^d;
\]

**Montgomery ladder exponentiation**

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

---

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
SCA Example on the Last Round of AES

(Observation $O$, ciphertext $C'$)

1:
ab 20 4f 0c
e7 31 c2 66
89 07 a0 fd
5a 59 8d 73
09 11 8d 32
d6 91 dd 84
54 e4 89 e2
07 aa 31 05
2c 83 82 29
10 83 91 00
df 84 64 ec
a2 91 c6 76

Model $M$: $\forall k_{0,0} \in \{00, 01, \cdots, ff\}$, $\text{ISB}(c_{0,0} \oplus k_{0,0}) \oplus c_{0,0}$

<table>
<thead>
<tr>
<th>$\text{ISB}(ab \oplus 00) \oplus ab = a5$</th>
<th>$\text{ISB}(ab \oplus 01) \oplus ab = c9$</th>
<th>$\text{ISB}(ab \oplus ff) \oplus ab = 56$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Rightarrow M = \text{HW}(a5) = 4$</td>
<td>$\Rightarrow M = \text{HW}(c9) = 4$</td>
<td>$\Rightarrow M = \text{HW}(56) = 4$</td>
</tr>
<tr>
<td>$\text{ISB}(09 \oplus 00) \oplus 09 = 49$</td>
<td>$\text{ISB}(09 \oplus 01) \oplus 09 = b6$</td>
<td>$\text{ISB}(09 \oplus ff) \oplus 09 = df$</td>
</tr>
<tr>
<td>$\Rightarrow M = \text{HW}(49) = 3$</td>
<td>$\Rightarrow M = \text{HW}(b6) = 5$</td>
<td>$\Rightarrow M = \text{HW}(df) = 7$</td>
</tr>
<tr>
<td>$\text{ISB}(2c \oplus 00) \oplus 2c = 6e$</td>
<td>$\text{ISB}(2c \oplus 01) \oplus 2c = d6$</td>
<td>$\text{ISB}(2c \oplus ff) \oplus 2c = 85$</td>
</tr>
<tr>
<td>$\Rightarrow M = \text{HW}(6e) = 5$</td>
<td>$\Rightarrow M = \text{HW}(d6) = 5$</td>
<td>$\Rightarrow M = \text{HW}(85) = 3$</td>
</tr>
</tbody>
</table>

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

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

© SecureIC 2014 All rights reserved | Confidential document, property of Secure-IC S.A.S.
Laser station
Laser station
Cryptography
Attacks
Protection
Conclusions

Side-channel attacks
Fault injection attacks

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

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
Cryptography
Attacks
Protection
Conclusions

Side-channel attacks
Fault-injection attacks

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


S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
Bellcore attack against RSA signatures with CRT

- Signature $S$ of message $x$ : $S = x^d \mod N$, 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 :
  \[
  \begin{align*}
  a &= 0 \mod q \\
  a &= 1 \mod p
  \end{align*}
  \quad \text{and} \quad
  \begin{align*}
  b &= 1 \mod q \\
  b &= 0 \mod p
  \end{align*}
\]
- $S = a \cdot S_1 + b \cdot S_2 \mod N$.
- Now, if $S_1$ happens to be faulty : $S_1 \rightarrow \hat{S}_1$ for whatever reason,
- $\gcd(S - \hat{S}, N) = \gcd(a \cdot (S_1 - \hat{S}_1), N) = q$. 

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
**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 $f \in \{\text{Id}, \text{P}, \text{E}, \text{FP}\}$.
- $L_{16} = R_{15}$.  

---

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
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:
  \[
  \begin{cases}
  R_{16} = L_{15} \oplus S_i(K_{16} \oplus R_{15}), \\
  \overline{R_{16}} = L_{15} \oplus S_i(K_{16} \oplus \overline{R_{15}}).
  \end{cases}
  \]
- It has $\approx$ four 6-bit solutions.

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
G. Piret & J.-J. Quisquater in 2003

Cryptology
Attacks
Protection
Conclusions

Side-channel attacks
Fault injection attacks

Round 9
SubBytes
ShiftRows
MixColumns
AddRoundKey

Round 10
SubBytes
ShiftRows
AddRoundKey

precomputed difference

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
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

1. Cryptography
2. Attacks
3. Protection
4. Conclusions
- **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 \( \text{Verify} \circ \text{Sign} = \text{Id} \).

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:
\[
\forall r_1, r_2, r_3 \neq 0,
\left( (M \times r_1 - e)^{d + r_2 \times \phi(N)} \times r_1 \mod r_3 \times N \right) \mod N = M^d \mod N,
\]
(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 \( \text{Verify} \circ \text{Sign} = \text{Id} \).

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
   6. end
4. end
9. return $R[1]$
Algorithm 1: RSA implementation (unprotected)

Input : $M \in \mathbb{Z}_N, d = (d_{n-1}, \ldots, d_0)_2$
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
   6.   end
4. 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. $R[d_i] \leftarrow R[d_i] \cdot R[2]$
5. end
6. if $R[0] \cdot R[1] \cdot M = R[2]$ then
   7. return $r \cdot R[1]$
6. else
   8. return “Error”
6. 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. \hspace{1em} $R[d_i] \leftarrow R[d_i] \cdot R[2]$
8. end
9. if $R[0] \cdot R[1] \cdot M = R[2]$ then
10. \hspace{1em} return $r \cdot R[1]$
11. else
12. \hspace{1em} return “Error”
13. end

Final condition:
$R[2] = ((M^2)^2) \cdots = M^{2^n}$
$R[0] = r \cdot M^d$
$R[1] = r^{-1} \cdot M^d$
### Resilience against passive attacks

- **Ephemeral keys** is the solution:
  - Indexed key update: [Koc03, Koc05]
  - 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 \([\text{Koc05, §4}]\) :

If \(\approx 1\) bit is leaked per 100 encryptions...

\[
\begin{align*}
\text{Alice:} & \quad \text{AES}_{k_0} \\
& \quad k_0 \\
& \quad \text{hash} \\
& \quad k_1 \\
& \quad \text{AES}_{k_1} \\
& \quad k_1 \\
& \quad \text{hash} \\
& \quad k_2 \\
& \quad \vdots
\end{align*}
\]

\[
\begin{align*}
\text{Bob:} & \quad \text{AES}_{k_0}^{-1} \\
& \quad k_0 \\
& \quad \text{hash} \\
& \quad k_1 \\
& \quad \text{AES}_{k_1}^{-1} \\
& \quad k_1 \\
& \quad \text{hash} \\
& \quad k_2 \\
& \quad \vdots
\end{align*}
\]
Logarithmic key access speed with indices in a binary table [Koc03]
Logarithmic key access speed with indices in a binary table [Koc03]

\[ \begin{align*}
&k_0: \text{Private root key} \\
&k_{i>0}: \text{Secondary private session keys} \\
&k_R: \text{Public key for right accesses} (ց, տ) \\
&D: \text{Symmetric inverse block cipher} \\
&E: \text{Symmetric block cipher} \\
&k_L: \text{Public key for left accesses} (ւ, ր) \\
\end{align*} \]

\[ \Rightarrow \text{also protects against related keys attacks [BK09]}! \]
Resilience against active attacks

- Recall AES can be broken with one \((C, C^*)\) 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)

⇒ 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

\[ e_3 = E_k(d_3) \]

unsafe, because input is chosen

\[ e_3 = E_{k_3}(d_3) \]

The attacker injects faults on this side

\[\text{envelop } e_3\]
Example on Smartcards Protocols

The attacker monitors faulty outputs here

\[ e_3 = E_k^3 (d_3 \oplus r_3) \]
\[ r_3 \leftarrow \text{envelop}(e_3, r_3) \]

The attacker injects faults on this side

\[ e_3 = E_k^3 (d_3 \oplus r_3) \]
safe, because input not chosen

S. Guilley, <sylvain.guille@telecom-paristech.fr>
Example on Smartcards Protocols

The attacker monitors faulty outputs here

**Reader**

\[ e'_1 = E_{k_1}(c_1) \]

\[ \text{challenge } c_1 \]

\[ \text{answer } e'_1 \]

**Smartcard**

\[ c_1 \leftarrow $\]

\[ e_1 = E_{k_1}(c_1) \]

safe, because not output

\[ e'_1 \neq e_1 \]

unsafe, since a fault can bring us here
(external authentication is skipped)

\[ r_3 \leftarrow $\]

\[ e_3 = E_{k_3}(d_3 \oplus r_3) \]

safe, because input not chosen

The attacker injects faults on this side
Example on Smartcards Protocols

The attacker monitors
faulty outputs here

$e'_1 = E_{k_1}(c_1)$

Reader

challenge $c_1$

$e'_1 = E_{k_1}(c_1)$

answer $e'_1$

Smartcard

$c_1 \leftarrow \$$

$e_1 = E_{k_1}(c_1)$

safe, because not output $e'_1 = e_1$

The attacker injects
faults on this side

safe, because operations are chained (depend on more than one bit).
In addition, it is more efficient!

envelop $e_3$

$e_3 = E_{k_3 \oplus e_1}(d_3)$

safe, because does not involve $k_3$
Example on Smartcards Protocols

The attacker monitors faulty outputs here

Reader

Smartcard

The attacker injects faults on this side

\[ e'_1 = E_{k_1}(c_1) \]

challenge \( c_1 \)

\[ e'_1 = E_{k_1}(c_1) \]

answer \( e'_1 \)

\[ e'_1 \]

\[ c_1 \leftarrow \$
\]

\[ e_1 = E_{k_1}(c_1) \]

safe, because not output \( e'_1 \) = \( e_1 \)

\[ c_2 \leftarrow \$
\]

\[ e'_2 = E_{k_2}(c_2) \]

challenge \( c_2 \)

\[ e'_2 = E_{k_2}(c_2) \]

answer \( e_2 \)

\[ e'_2 \] = \( e_2 \)

\[ e_2 = E_{k_2}(c_2) \]

unsafe, because input is chosen

\[ e_3 = E_{k_3} \oplus e_1 \oplus e_2 (d_3) \]

safe, because does not involve \( k_3 \)
Example on Smartcards Protocols

The attacker monitors faulty outputs here

Reader

e_1' = E_{k_1}(c_1)

challenge c_1

answer e_1'

Smartcard

c_1 ← $ e_1 = E_{k_1}(c_1)

safe, because not output e_1' = e_1

r_2 ← $

c_2 = E_{k_2}(c_2 \oplus r_2)

safe, because input not chosen

\text{envelop } e_3

\overset{?}{e_2} = e_2

\overset{?}{e_2} = e_2

(\overset{?}{e_3} = e_3 = E_{k_3 \oplus e_1 \oplus e_2}(d_3)

safe, because does not involve k_3

The attacker injects faults on this side

S. Guilley, < sylvain.guilley@TELECOM-ParisTech.fr >
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)
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

<table>
<thead>
<tr>
<th>Signal</th>
<th>Waveform</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{CLK}_i)</td>
<td></td>
</tr>
<tr>
<td>(\text{CLK}_{FSM})</td>
<td></td>
</tr>
<tr>
<td>(A)</td>
<td></td>
</tr>
<tr>
<td>(B)</td>
<td></td>
</tr>
<tr>
<td>(C)</td>
<td></td>
</tr>
<tr>
<td>(\text{CLK}_{\text{Ciph.}})</td>
<td></td>
</tr>
</tbody>
</table>
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)
5\( \times \)10^4 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)
2. Main technical characteristics

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

\[ \text{vdd} \]

\[ \text{vss} \]

S. Guilley, < sylvain.guilley@TELECOM-ParisTech.fr >
Effect of laser spot on CMOS cells
Effect of laser spot on CMOS cells

S. Guilley, <sylvain.guille@TELECOM-ParisTech.fr>
A summary of the countermeasures embedded in a smartcard.

- Internal clock
- Dummy clock cycles insertion
- RAM Dynamic encryption
- Balanced logic
- Bus scrambling
- CPU sensitive registers redundancy
- Memory Redundancy ECC, parity bits
- Voltage sensors
- Temperature sensors
- Frequency sensors
- Light sensors

 Courtesy of Assia Tria, CEA/LETI.

S. Guilley, sylvain.guilley@TELECOM-ParisTech.fr
## Active shield

<table>
<thead>
<tr>
<th>Beam</th>
<th>Mag</th>
<th>pA</th>
<th>Det</th>
<th>Tilt</th>
<th>5 μm</th>
<th>Beam</th>
<th>Mag</th>
<th>pA</th>
<th>Det</th>
<th>Tilt</th>
<th>5 μm</th>
</tr>
</thead>
<tbody>
<tr>
<td>30.0 kV</td>
<td>15.0 kX</td>
<td>132</td>
<td>pA-M</td>
<td>45.4°</td>
<td>5 μm</td>
<td>30.0 kV</td>
<td>15.0 kX</td>
<td>142</td>
<td>pA-M</td>
<td>45.4°</td>
<td>5 μm</td>
</tr>
</tbody>
</table>

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
This area is unprotected!
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: $d = 1$, $s = s_0 \oplus s_1$.
  \[ \Rightarrow \] 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.

S. Guilley, <sylvain.guilley@TELECOM-ParisTech.fr>
Computing Masked DES

Everything is linear, but the sboxes.

![Diagram of masked DES computation](image)
Everything is linear, but the sboxes.
Everything is linear, but the sboxes (sic). \( a \mapsto a^{-1} \mod X^8 + X^4 + X + 1 \), and \( a^{-1} = a^{254} = (((a^2) \times a)^4) \times \ldots \mod X^8 + X^4 + X + 1 \).

- 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. \(\bigoplus_i c_i = ab\)

\[
\bigoplus_i c_i = (\bigoplus_i a_i)(\bigoplus_i b_i) = \bigoplus_{i,j} a_ib_j
\]

- **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 \(d\)th-order security

CHES 2010 – Provably Secure Higher-Order Masking of AES
Ishai-Sahai-Wagner (ISW) Scheme

Example: AND gate for $d = 2$
Caption: AN = and, EO = xor.
Masking the S-box

The proposed addition chain:

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

CHES 2010 – Provably Secure Higher-Order Masking of AES
## Implementation Results (8051)

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

Interpolation:
- \( d=4 \) : 730 Kc / 24 ms
- \( d=5 \) : 1050 Kc / 34 ms

CHES 2010 – Provably Secure Higher-Order Masking of AES
\( a \leftrightarrow (a_f, a_t) \) DPL representation:

- **a is VALID** if \( a_f \oplus a_t = 1 \).
  
  \[
  \text{VALID} \doteq \{\text{VALID0, VALID1}\} \text{ or } \{1, 0\}, \{(0, 1)\}.
  \]

- **a is NULL** if \( a_f \oplus a_t = 0 \).
  
  \[
  \text{NULL} \doteq \{\text{NULL0, NULL1}\} \text{ or } \{(0, 0), (1, 1)\}.
  \]

Flavors of DPL – gate \( g \) style:

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

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

Faults typology on DPL:

- **Only results on evaluation are observable.**

- **Asymmetric faults**: \( \{\text{VALID0, VALID1}\} \downarrow \rightarrow \{\text{NULL0, NULL1}\} \) or \( \{0, 0\}, \{(1, 1)\} \), caused by **global** perturbations (e.g., VC glitch, overclocking, under-powering).

- **Symmetric faults**: \( \{\text{VALID0, VALID1}\} \downarrow \text{ or } \uparrow \rightarrow \{\text{NULL0, NULL1}\} \), 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.)

(a) Floorplan OK
(b) Place OK
(c) Route OK
Secured Cells Come in Pairs: SABL & DI gates
WDDL example: placement strategy

After they are flipped $R_0$ and $M_X$, 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
« Backend-duplication »: routing
« Backend-duplication » Realization

Before duplication

After duplication

- Half of the placement rows are obstructed
- Half of the routing channel are obstructed
- Cells are duplicated by vertical flip ($R0 \rightarrow MX$)
- 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 rewriting (error-prone) nor reloading (not interactive) design rules.
Implementation

+3 lines added in the Makefile:

<table>
<thead>
<tr>
<th>Regular backend flow:</th>
<th>Flow compatible with the “backend duplication”. Added steps:</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Floorplanning</td>
<td>i : Floorplan dimensioning</td>
</tr>
<tr>
<td>- Place-and-route</td>
<td>ii : Obstructions implementation</td>
</tr>
<tr>
<td>- Clock tree generation</td>
<td>iii : Duplication</td>
</tr>
<tr>
<td>- Scan chain optimization</td>
<td></td>
</tr>
<tr>
<td>- Antenna effects correction</td>
<td></td>
</tr>
<tr>
<td>- Custom steps, like ECO or SI fix</td>
<td></td>
</tr>
</tbody>
</table>
| - Dummies placement   | Verilog LoC:

<table>
<thead>
<tr>
<th></th>
<th>Regular</th>
<th>Backend-duplicated</th>
</tr>
</thead>
<tbody>
<tr>
<td>Place</td>
<td>1.9 s</td>
<td>6.2 s</td>
</tr>
<tr>
<td>Route</td>
<td>39.0 s</td>
<td>80.0 s</td>
</tr>
<tr>
<td>Duplication</td>
<td>-</td>
<td>77.5 s</td>
</tr>
</tbody>
</table>

Place 1.9 s 6.2 s Route 39.0 s 80.0 s Duplication - 77.5 s

Execution time in the example of DES:
Reducing the cross-coupling: routing constraints

Routing forbidden: tracks obstructed = 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).
PUF: Physically Unclonable Functions

- Optical PUF \([\text{Pap01}]\)
- Coating PUF \([\text{TSK07}]\)
- SRAM PUF \([\text{HBF09}]\)
- Glitch PUF \([\text{SS10}]\)
- Arbiter PUF \([\text{GCvDD02}]\)
- Loop PUF \([\text{CDGB12}]\)
- Memory contention PUF \([\text{Gün12}]\)

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)
  • 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
  - http://commons.wikimedia.org/

- **Arbiter PUF**
  - Uses the delay difference of two signals

- **Coating PUF**
  - Uses the difference of capacitive load

- **Butterfly PUF**
  - Uses the difference of the initial state of memory

- **Ring-Oscillator PUF**
  - Uses the difference of oscillating frequencies

- **SRAM PUF**

---


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

1. Cryptography
2. Attacks
3. Protection
4. Conclusions
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].
Nothing to display.


Generic side-channel countermeasures for reconfigurable devices. 

[GP99] Louis Goubin and Jacques Patarin. 
DES and Differential Power Analysis. The “Duplication” Method. 
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. 

Fault Injection Resilience. 
Santa Barbara, CA, USA. DOI : 10.1109/FDTC.2010.15 ; Complete version : 
http://hal.archives-ouvertes.fr/hal-00482194/en/.

Using Data Contention in Dual-ported Memories for Security Applications. 

[GW08] Catherine H. Gebotys and Brian A. White. 
EM analysis of a wireless Java-based PDA. 
Power-Up SRAM State as an Identifying Fingerprint and Source of True Random Numbers.


Flipping bits in memory without accessing them: An experimental study of DRAM disturbance errors.

Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.

[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.
On the vulnerability of FPGA bitstream encryption against power analysis attacks: extracting keys from Xilinx Virtex-II FPGAs. 

[MBO+05] Elke De Mulder, Pieter Buysschaert, Siddika 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. 

On the Portability of Side-Channel Attacks — An Analysis of the Xilinx Virtex 4 and Virtex 5 Bitstream Encryption Mechanism. 

Black-Box Side-Channel Attacks Highlight the Importance of Countermeasures - An Analysis of the Xilinx Virtex-4 and Virtex-5 Bitstream Encryption Mechanism. 

Stellenbosch, South Africa. DOI: 10.1007/978-3-642-12678-9_17.
[Pap01] Ravikanth S. Pappu. 
Physical One-Way Functions. 

[PR10] Emmanuel Prouff and Thomas Roche. 
Attack on a Higher-Order Masking of the AES Based on Homographic Functions. 

Security Analysis of Higher-Order Boolean Masking Schemes for Block Ciphers (with Conditions of Perfect Masking). 
DOI : 10.1049/iet-ifs :20070066. 

[Qué03] Patrick Le Quéré. 
High Rate Random Number Generator, November 19 2003. 

From Theory to Practice of Private Circuit : A Cautionary Note. 
New York City, USA. DOI : 10.1109/ICCD.2015.7357117. 

Provably Secure Higher-Order Masking of AES. 


[TSK07] Pim Tuyls, Boris Skoric, and Tom Kevenaar.  
Security with Noisy Data: Private Biometrics, Secure Key Storage and Anti-Counterfeiting.  