# Horizontal and Vertical Side-Channel Attacks against Secure RSA Implementations

Aurélie Bauer Éliane Jaulmes Emmanuel Prouff
Justine Wild

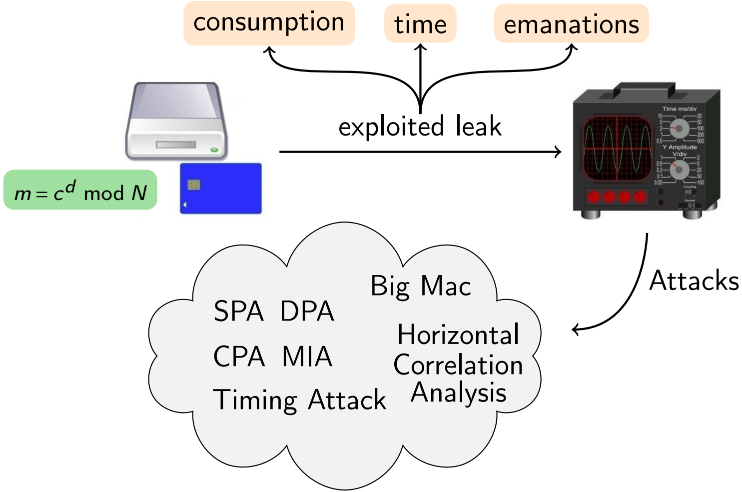
ANSSI

Session ID: CRYP-T18
Session Classification: Advanced

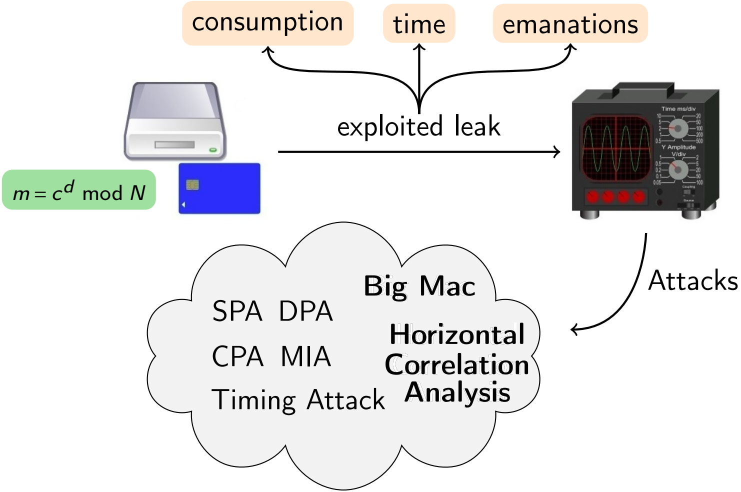
1. Introduction
2. Clavier *et al.*'s Paper
   - Attack: Horizontal Correlation Analysis
   - Countermeasures against Horizontal Attacks
3. This Paper
   - Attacks on Clavier et al. Countermeasures
   - New Countermeasure against Horizontal Attacks
   - Simulation Results of Our Attacks
4. Conclusion

1. Introduction
2. Clavier *et al.*'s Paper
   - Attack: Horizontal Correlation Analysis
   - Countermeasures against Horizontal Attacks
3. This Paper
   - Attacks on Clavier et al. Countermeasures
   - New Countermeasure against Horizontal Attacks
   - Simulation Results of Our Attacks
4. Conclusion

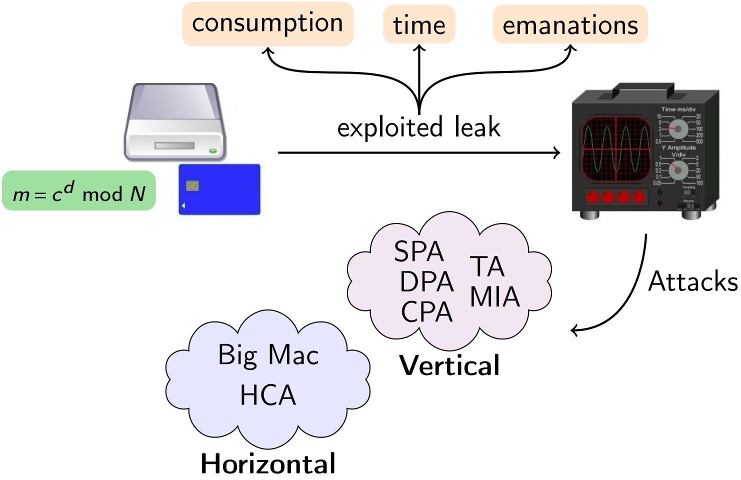
## Side-Channel Analysis On RSA



## Side-Channel Analysis On RSA



## Side-Channel Analysis On RSA



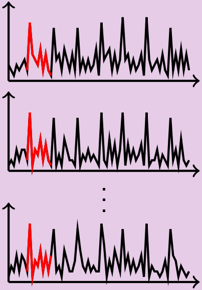
## A Unified Framework [This paper]

### Vertical Analysis

$1^{st}$ execution

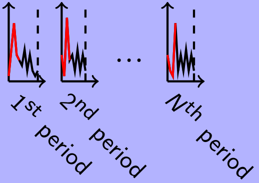
$2^{nd}$ execution

$N^{th}$ execution



- **several** acquisitions
- traces treatment

### Horizontal Analysis




- a **single** acquisition
- sub-traces treatment

SPA, DPA, CPA, Big Mac, MIA, Template, Timing, HCA,...

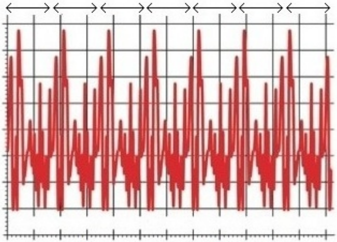
**Exponentiation**: $c = m^d \bmod N$, secret $d = (1, d_{\ell-2}, \ldots, d_0)_2$

## Square & Multiply **Atomic**

$R_0 = 1,\ R_1 = m,\ i = l-1,\ k = 0$
`while` $i \leq 0$ `do`
  $\boldsymbol{R_0 = R_0 \times R_k}$
  $k = k \oplus d_i$
  $i = i - \neg k$
Return $R_0$

## Example: $d = 1011$

- $\boldsymbol{R_0 = 1 \times 1}$, $d_3 = 1$, $k = 1$, $i = 3$



Regular Square & Multiply

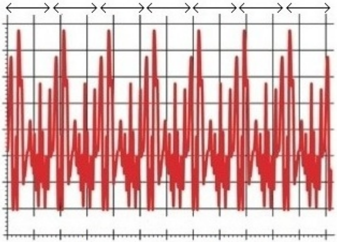
**Exponentiation**: $c = m^d \bmod N$, secret $d = (1, d_{\ell-2}, \ldots, d_0)_2$

### Square & Multiply **Atomic**

$R_0 = 1,\ R_1 = m,\ i = l - 1,\ k = 0$
while $i \leq 0$ do
  $\boldsymbol{R_0 = R_0 \times R_k}$
  $k = k \oplus d_i$
  $i = i - \neg k$
Return $R_0$

### Example: $d = 1011$

- $\boldsymbol{R_0 = 1 \times 1}$, $d_3 = 1$, $k = 1$, $i = 3$
- $\boldsymbol{R_0 = 1 \times m}$, $d_3 = 1$, $k = 0$, $i = 2$



Regular Square & Multiply

**Exponentiation**: $c = m^d \bmod N$, secret $d = (1, d_{\ell-2}, \ldots, d_0)_2$

## Square & Multiply **Atomic**

$R_0 = 1,\ R_1 = m,\ i = l - 1,\ k = 0$

`while` $i \leq 0$ `do`

$\boldsymbol{R_0} = \boldsymbol{R_0} \times \boldsymbol{R_k}$

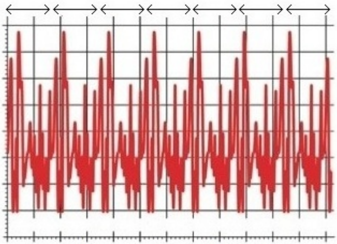
$k = k \oplus d_i$

$i = i - \neg k$

Return $R_0$

## Example: $d = 1011$

- $\boldsymbol{R_0} = \mathbf{1} \times \mathbf{1},\ d_3 = 1,\ k = 1,\ i = 3$
- $\boldsymbol{R_0} = \mathbf{1} \times \boldsymbol{m},\ d_3 = 1,\ k = 0,\ i = 2$
- $\boldsymbol{R_0} = \boldsymbol{m} \times \boldsymbol{m},\ d_2 = 0,\ k = 0,\ i = 1$



Regular Square & Multiply

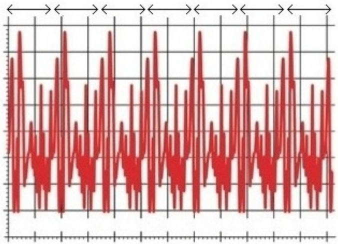
**Exponentiation**: $c = m^d \bmod N$, secret $d = (1, d_{\ell-2}, \ldots, d_0)_2$

## Square & Multiply **Atomic**

$R_0 = 1,\ R_1 = m,\ i = l-1,\ k = 0$
`while` $i \leq 0$ `do`
  $\boldsymbol{R_0 = R_0 \times R_k}$
  $k = k \oplus d_i$
  $i = i - \neg k$
Return $R_0$

## Example: $d = 1011$

- $\boldsymbol{R_0 = 1 \times 1}$, $d_3 = 1,\ k = 1,\ i = 3$
- $\boldsymbol{R_0 = 1 \times m}$, $d_3 = 1,\ k = 0,\ i = 2$
- $\boldsymbol{R_0 = m \times m}$, $d_2 = 0,\ k = 0,\ i = 1$
- $\boldsymbol{R_0 = m^2 \times m^2}$, $d_1 = 1,\ k = 1,\ i = 1$



Regular Square & Multiply

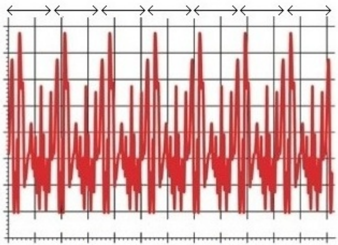
## Exponentiation: $c = m^d \bmod N$, secret $d = (1, d_{\ell-2}, \ldots, d_0)_2$

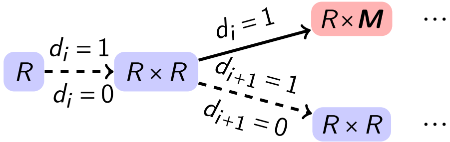
### Square & Multiply **Atomic**

$R_0 = 1,\ R_1 = m,\ i = l - 1,\ k = 0$
`while` $i \leq 0$ `do`
  $\boldsymbol{R_0 = R_0 \times R_k}$
  $k = k \oplus d_i$
  $i = i - \neg k$
Return $R_0$

### Example: $d = 1011$

- $\boldsymbol{R_0 = 1 \times 1}$, $d_3 = 1,\ k = 1,\ i = 3$
- $\boldsymbol{R_0 = 1 \times m}$, $d_3 = 1,\ k = 0,\ i = 2$
- $\boldsymbol{R_0 = m \times m}$, $d_2 = 0,\ k = 0,\ i = 1$
- $\boldsymbol{R_0 = m^2 \times m^2}$, $d_1 = 1,\ k = 1,\ i = 1$
- $\boldsymbol{R_0 = m^4 \times m}$, …


Regular Square & Multiply

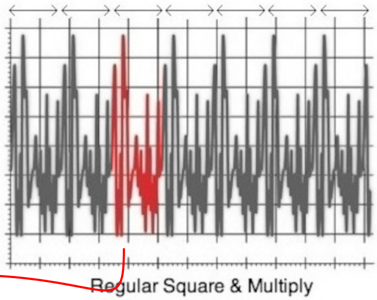


1. Introduction
2. Clavier *et al.*'s Paper
   - Attack: Horizontal Correlation Analysis
   - Countermeasures against Horizontal Attacks
3. This Paper
   - Attacks on Clavier et al. Countermeasures
   - New Countermeasure against Horizontal Attacks
   - Simulation Results of Our Attacks
4. Conclusion

**Horizontal Side-Channel Analysis** [Clavier *et al.*, *Horizontal Correlation Analysis on Exponentiation*, (ICICS 2010)]

**Square & Multiply Atomic**

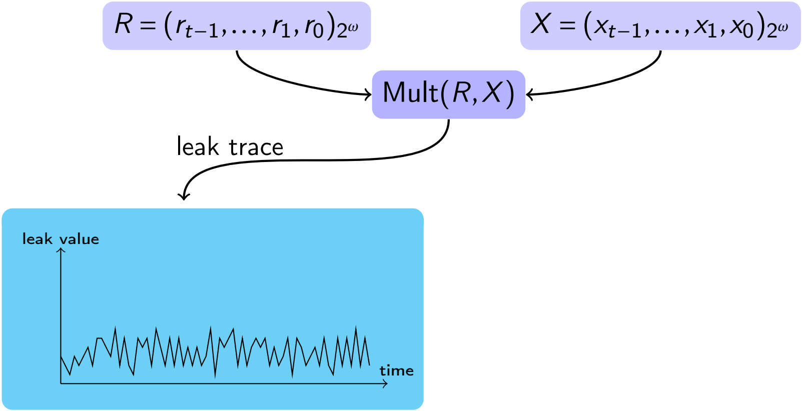
Only multiplications:
$R \leftarrow R \times \boldsymbol{R}$
or $R \leftarrow R \times \boldsymbol{M}$ *(bit 1)*



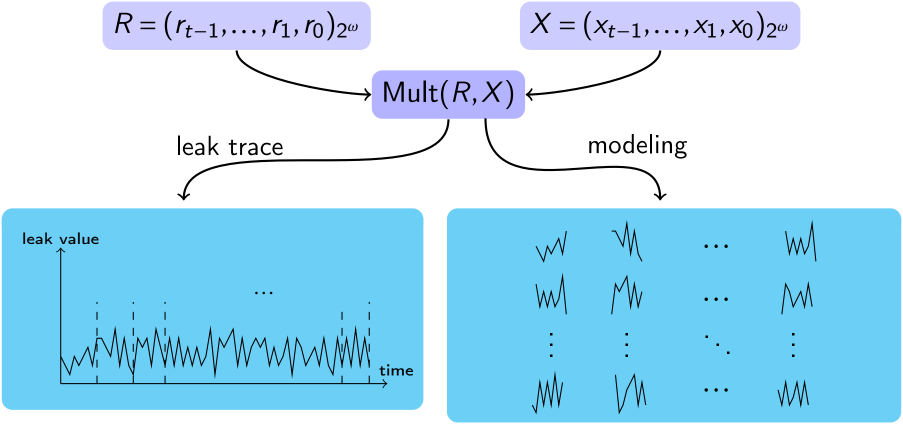

1 Multiplication

- Do we multiply **by the message** or not ?
- **Horizontal core idea**: distinguish $R \times R$ from $R \times M$ with a single trace

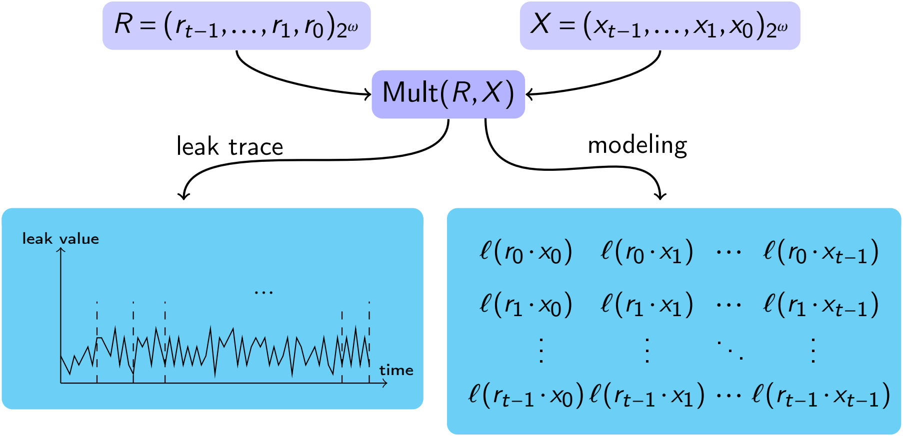
## Zoom on the Long Integer Multiplication



## Zoom on the Long Integer Multiplication



# Zoom on the Long Integer Multiplication



## Horizontal Correlation Analysis

- $M$ **known**
- **Hypothesis**: $X = M$

$$\begin{cases} X = M & \Rightarrow \quad \text{bit} = 1 \\ X \neq M & \Rightarrow \quad \text{bit} = 0 \end{cases}$$

**Simulation**

$$\begin{matrix} \ell(r_0 \cdot x_0) & \ell(r_0 \cdot x_1) & \cdots & \ell(r_0 \cdot x_{t-1}) \\ \ell(r_1 \cdot x_0) & \ell(r_1 \cdot x_1) & \cdots & \ell(r_1 \cdot x_{t-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \ell(r_{t-1} \cdot x_0) & \ell(r_{t-1} \cdot x_1) & \cdots & \ell(r_{t-1} \cdot x_{t-1}) \end{matrix}$$

**Observation**

$$\begin{matrix} HW(r_0 \cdot m_0) & HW(r_0 \cdot m_1) & \cdots & HW(r_0 \cdot m_{t-1}) \\ HW(r_1 \cdot m_0) & HW(r_1 \cdot m_1) & \cdots & HW(r_1 \cdot m_{t-1}) \\ \vdots & \vdots & \ddots & \vdots \\ HW(r_{t-1} \cdot m_0) & HW(r_{t-1} \cdot m_1) & \cdots & HW(r_{t-1} \cdot m_{t-1}) \end{matrix}$$

$$\rho\left( \ell(r_i \cdot x_j), HW(r_i \cdot m_j) \right)$$

- **Actual** Countermeasures Analysis
  - **Single curve** $\Rightarrow$ Exposant/Message randomisation **ineffective**
- **Clavier *et al.*'s countermeasures**

**Blind the $r_i \cdot x_j$**

Replace $r_i \cdot x_j$ by $(r_i - a_1)(x_j - a_2)$

**Blind the $x_j$, permute the $r_i$**

Replace $r_i \cdot x_j$ by $r_{\alpha[i]} \cdot (x_j - a_2)$

**Permute the $r_i$ and the $x_j$**

Replace $r_i \cdot x_j$ by $r_{\alpha[i]} \cdot x_{\beta[j]}$

$$\begin{matrix} \ell(\widetilde{r}_0 \cdot \widetilde{x}_0) & \dots & \ell(\widetilde{r}_0 \cdot \widetilde{x}_{t-1}) \\ \ell(\widetilde{r}_1 \cdot \widetilde{x}_0) & \dots & \ell(\widetilde{r}_1 \cdot \widetilde{x}_{t-1}) \\ \vdots & \ddots & \vdots \\ \ell(\widetilde{r}_{t-1} \cdot \widetilde{x}_0) & \dots & \ell(\widetilde{r}_{t-1} \cdot \widetilde{x}_{t-1}) \end{matrix}$$

- **Actual** Countermeasures Analysis
  - **Single curve** ⇒ Exposant/Message randomisation **ineffective**
- **Clavier *et al.*'s countermeasures**

**Blind the $r_i \cdot x_j$**

Replace $r_i \cdot x_j$ by $(r_i - a_1)(x_j - a_2)$

**Blind the $x_j$, permute the $r_i$**

Replace $r_i \cdot x_j$ by $s_{\alpha[i]} \cdot (x_j - a_2)$

**Permute the $r_i$ and the $x_j$**

Replace $r_i \cdot x_j$ by $r_{\alpha[i]} \cdot x_{\beta[j]}$

$$\begin{matrix} \ell(r_{\alpha[0]} \cdot \widetilde{x}_0) & \dots & \ell(r_{\alpha[0]} \cdot \widetilde{x}_{t-1}) \\ \ell(r_{\alpha[1]} \cdot \widetilde{x}_0) & \dots & \ell(r_{\alpha[1]} \cdot \widetilde{x}_{t-1}) \\ \vdots & \ddots & \vdots \\ \ell(r_{\alpha[t-1]} \cdot \widetilde{x}_0) & \dots & \ell(r_{\alpha[t-1]} \cdot \widetilde{x}_{t-1}) \end{matrix}$$

- **Actual** Countermeasures Analysis
  - **Single curve** ⇒ Exposant/Message randomisation **ineffective**
- **Clavier *et al.*'s countermeasures**

**Blind the $r_i \cdot x_j$**

Replace $r_i \cdot x_j$ by $(r_i - a_1)(x_j - a_2)$

**Blind the $x_j$, permute the $r_i$**

Replace $r_i \cdot x_j$ by $s_{\alpha[i]} \cdot (x_j - a_2)$

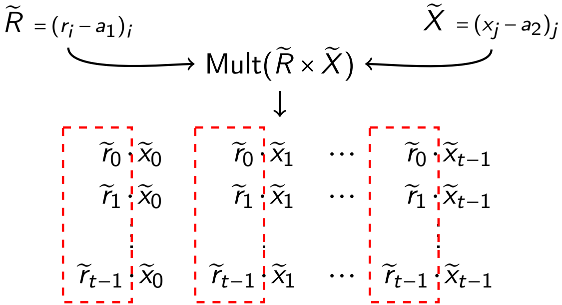
**Permute the $r_i$ and the $x_j$**

Replace $r_i \cdot x_j$ by $r_{\alpha[i]} \cdot x_{\beta[j]}$

$$\begin{matrix} \ell(r_{\alpha[0]} \cdot x_{\beta[0]}) & \cdots & \ell(r_{\alpha[0]} \cdot x_{\beta[t-1]}) \\ \ell(r_{\alpha[1]} \cdot x_{\beta[0]}) & \cdots & \ell(r_{\alpha[1]} \cdot x_{\beta[t-1]}) \\ \vdots & \ddots & \vdots \\ \ell(r_{\alpha[t-1]} \cdot x_{\beta[0]}) & \cdots & \ell(r_{\alpha[t-1]} \cdot x_{\beta[t-1]}) \end{matrix}$$

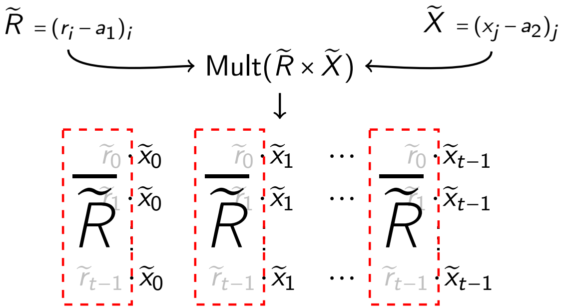
# Attacks on the Clavier *et al.*'s countermeasures

**Blind the $r_i \cdot x_j$** (Replace $r_i \cdot x_j$ by $(r_i - a_1)(x_j - a_2)$)



## Attacks on the Clavier *et al.*'s countermeasures

**Blind the $r_i \cdot x_j$** (Replace $r_i \cdot x_j$ by $(r_i - a_1)(x_j - a_2)$)




- **Correlation** between the $\overline{\widetilde{r}_i} \cdot \widetilde{x}_j$ and the $\overline{r_i} \cdot m_j$
- When $t$ **increases**, $\overline{R} = \overline{\widetilde{R}}$

## Attacks on the Clavier *et al.*'s countermeasures

**Blind the $x_j$ and permute the $r_i$**

$$\begin{array}{ccc} r_{\alpha[0]} \cdot \widetilde{x}_0 & \cdots & r_{\alpha[0]} \cdot \widetilde{x}_{t-1} \\ r_{\alpha[1]} \cdot \widetilde{x}_0 & \cdots & r_{\alpha[1]} \cdot \widetilde{x}_{t-1} \\ \vdots & & \vdots \\ r_{\alpha[t-1]} \cdot \widetilde{x}_0 & \cdots & r_{\alpha[t-1]} \cdot \widetilde{x}_{t-1} \end{array}$$

**Permute the $r_i$ and the $x_j$**

$$\begin{array}{ccc} r_{\alpha[0]} \cdot x_{\beta[0]} & \cdots & r_{\alpha[0]} \cdot x_{\beta[t-1]} \\ r_{\alpha[1]} \cdot x_{\beta[0]} & \cdots & r_{\alpha[1]} \cdot x_{\beta[t-1]} \\ \vdots & & \vdots \\ r_{\alpha[t-1]} \cdot x_{\beta[0]} & \cdots & r_{\alpha[t-1]} \cdot x_{\beta[t-1]} \end{array}$$

- Exhaustive research on $\beta$
- $t!$ possibilities

- **Weakness:** $\alpha$ and $\beta$ are **independent**

## New Countermeasure

**Permute simultaneously the $r_i$ and the $x_j$**

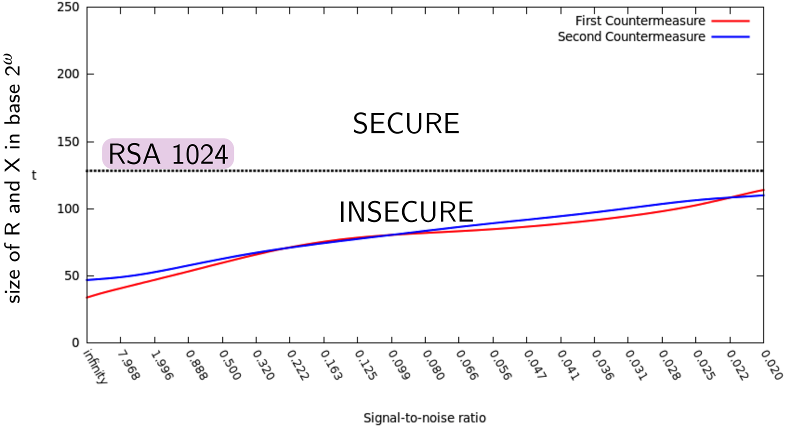
Use a $\boldsymbol{t^2}$**-size permutation** in order to randomize **simultaneously** the $r_i$ and the $x_j$

- Leak modelisation:

$$\begin{matrix} \ell(r_1 \cdot x_2) & \ell(r_1 \cdot x_0) & \ell(r_2 \cdot x_0) \\ \ell(r_0 \cdot x_2) & \ell(r_2 \cdot x_2) & \ell(r_1 \cdot x_1) \\ \ell(r_2 \cdot x_1) & \ell(r_0 \cdot x_1) & \ell(r_0 \cdot x_0) \end{matrix}$$

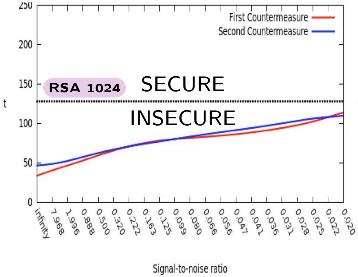
- Find the permutation: $\boldsymbol{t^2}$**! possibilities**
- Third countermeasure of Clavier *et al.*: $\boldsymbol{t}$**!** possibilities

## Attack on Architecture 8 bits

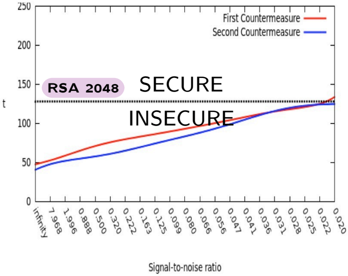


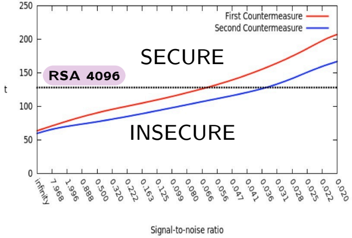
## Simulation Results of Our Attacks

Architecture 8 bits



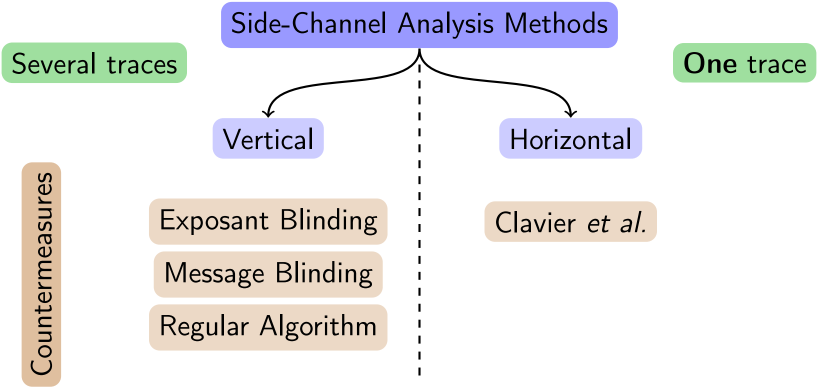

Architecture 16 bits

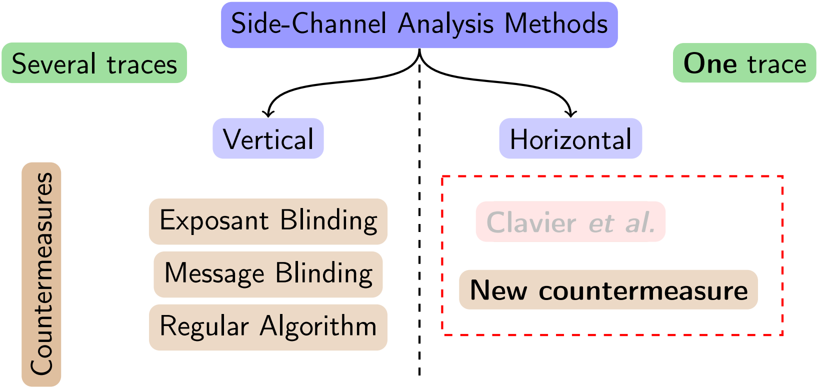



Architecture 32 bits

- $t$: size of R and X in base $2^{\omega}$
- Works also on 16 and 32 bits
- When $\omega \nearrow$, security level $\nearrow$


Side-Channel Analysis Methods
Several traces
One trace
Vertical
Horizontal
Countermeasures
Exposant Blinding
Message Blinding
Regular Algorithm
Clavier et al.


Side-Channel Analysis Methods
Several traces
One trace
Vertical
Horizontal
Countermeasures
Exposant Blinding
Message Blinding
Regular Algorithm
Clavier et al.
New countermeasure

## And more in our paper ...

- **Framework:** model both Horizontal and Vertical Attacks
- Attacks on Square and Multiply **Always**
- More Simulations:
    - Attacks on **variant** Clavier *et al.* first countermeasure
    - Variant of our attack (no average)
    - Test the **robustness** of our countermeasure

*Thank you for your attention. Questions ?*

# Timing Attack against protected RSA-CRT implementation used in PolarSSL

Cyril Arnaud and Pierre-Alain Fouque

Defense Ministry and Rennes 1 University

February 26, 2013

# Overview

# Overview

1. Detecting a Timing Bias on RSA implementation of POLARSSL
   - Introduction
   - Finding a bias
   - is the set of extra bit observable ?
2. Our timing attack
   - Cryptographic Analysis
   - Statistical Tools
   - Results against PolarSSL 1.1.4
3. Countermeasure
   - State of the Art
   - Alternatives to blinding
4. Conclusion

# State of the Art

## Related Work

- 1996 : Timing Attacks on Implementations of Diffie-Hellman,RSA, DSS, and Other Systems [Kocher] at CRYPTO '96
- 2000 : Timing attack on RSA-CRT [Schindler] at CHES '00
- 2003 : Remote timing attacks are practical [Brumley et Boneh] at Usenix '03
- 2005 : Improving Brumley and Boneh timing attack on unprotected SSL implementation [O. Aciiçmez, *et al.*] at CCS '05

## Attacks

- Countermeasures of OPENSSL avoided
- Exploit timing bias induced by RSA optimizations
- Old monocore processors

# State of the Art

## Related Work

- 1996 : Timing Attacks on Implementations of Diffie-Hellman,RSA, DSS, and Other Systems [Kocher] at CRYPTO '96
- 2000 : Timing attack on RSA-CRT [Schindler] at CHES '00
- 2003 : Remote timing attacks are practical [Brumley et Boneh] at Usenix '03
- 2005 : Improving Brumley and Boneh timing attack on unprotected SSL implementation [O. Aciiçmez, *et al.*] at CCS '05

## Attacks

- Countermeasures of OPENSSL avoided
- Exploit timing bias induced by RSA optimizations
- Old monocore processors

# Why using a Timing Attack ?

## Side-Channel Cryptanalysis

- Electromagnetic emanation and power consumption hard to apply on a computer
- Computation Time :
  - cannot be detected
  - allows to factor RSA modulus by measuring the time to decrypt
  - possible on network
  - Timing measurement : 2 instructions

# Measurement of computation time

## Time Stamp Counter (TSC)

- 64-bit counter that records cycle of the FSB bus
- Common to each CPU core
- Use of *RDTSC* instruction : do not use any privilege

## Performance Counter (PMC)

- Counter used in the CPU microarchitecture
- Allow to measure each tick of a core
- Reading using the *RDMSR* instruction : require privilege
- Require a specific kernel module

# Target choice

## RSA of POLARSSL 1.1.4

- Opensource library developed in C
- Operational version (Adobe flash player, ...)
- Use countermeasures suggested by Boneh and Brumley and Schindler (One subroutine for multiplication and a dummy substraction)
- RSA decryption in constant time
- *Protected* against timing attacks

# Goals and Results

## Goals

- Evaluate the countermeasure
- Mount an attack
- Determining efficient countermeasures
- Work on recent processors (Intel Core 2 Duo and Core i7).

## Results

- Detection of an unknown bias
- Attack verified on a chosen ciphertext attack for RSA 512, 1024 and 2048 bits
- Propose two countermeasures avoiding this attack

# Goals and Results

## Goals

- Evaluate the countermeasure
- Mount an attack
- Determining efficient countermeasures
- Work on recent processors (Intel Core 2 Duo and Core i7).

## Results

- Detection of an unknown bias
- Attack verified on a chosen ciphertext attack for RSA 512, 1024 and 2048 bits
- Propose two countermeasures avoiding this attack

# RSA Implementation of POLARSSL

## RSA Decryption

To decrypt $c \in (\mathbb{Z}/n\mathbb{Z})$ : modular exponentiation using the private exponent :

$$m = c^d \ mod \ n$$

## RSA decryption optimizations of POLARSSL

- Chinese Remainder Theorem using Garner recombination
- Montgomery Multiplication using one countermeasure
- Modular Exponentiation using the *sliding window* method

# Multiprecision Montgomery Modular Multiplication

## Number representations

$A = \sum_{i=0}^{i=s-1} a_i r^i$,

- size of words is denoted by $w$
- $s$ represents the number of required words of size $w$
- $r = 2^w$
- $R = r^s$

```
function MULTIMONTMUL(A, B, P)
    Z = (z_s, ..., z_0)_r ← 0
    for i = 0 to s − 1 do
        u ← ((z_0 + a_i × b_0) × μ_0) mod r
        Z ← (Z + a_i ⊗_w B)
        Z ← (Z + u ⊗_w P) div r
    if Z ≥ P then Z ← Z − P
    return Z (= ABR^{-1} mod P)
```

## $\otimes_w$

$w$-bit multiplication to multiply a word with a large integer

# Schindler's observation

## Extra reduction - Time variance in MULTIMONTMUL

- Montgomery representation $\bar{A} = AR \; mod \; P$
- Suppose $B$ is uniformly distributed in $\mathbb{Z}_P$
- $P$ (extra-reduction in MULTIMONTMUL$(\bar{X}, B, P)) = \frac{\bar{X} \; mod \; P}{2R}$
- $P$ (extra-reduction in MULTIMONTMUL$(B, B, P)) = \frac{P}{3R}$

# PolarSSL's Multiprecision Montgomery multiplication

function MULTIMONTMUL$(A, B, P)$
$Z = (z_s, \ldots, z_0)_r \leftarrow 0$
for $i = 0$ to $s - 1$ do
$u \leftarrow ((z_0 + a_i \times b_0) \times \mu_0) \bmod r$
$Z \leftarrow (Z + a_i \otimes_w B)$
$Z \leftarrow (Z + u \otimes_w P)\ div\ r$
$(ABR^{-1} \leq Z < P + ABR^{-1})$
if $Z \geq P$ then $Z \leftarrow Z - P$
else dummy subtraction
return $Z$ $(= ABR^{-1} \bmod P)$

Running times to execute the branch condition is identical for all inputs

Dummy subtraction used in MULTIMONTMUL$(A, B, P)$ makes the time to perform the multiplication independent on the $A$ and $B$

# Extra bit

- The condition of MULTIMONTMUL is
$$ABR^{-1} \leq Z < P + ABR^{-1}$$
- We suppose that $\frac{R}{2} < P < R$ (key lengths multiple of world size)
- Then $Z < 2R$.
- Hence, if $Z > R$ then $z_s = 1$, this bit is called extra bit

## Extra bit in source code

- in the $s^{th}$ loop of MULTIMONTMUL, if $Z > R$ then the extra bit is set by a carry propagation up to the top most significant word of $Z$.
- is extra bit imply a timing variance ?

# An attacker can observe a timing difference

## Methodology

We generate :

- Random numbers $A$, $B$ with known size, converted in Montgomery representation
- a prime number $Q$ where $\frac{R}{2} < Q < R$.

We sort :

- the time in CPU's clock ticks to perform MULTIMONTMUL$(A, B, Q)$ according to the size numbers
- if an extra bit, an extra-reduction without extra bit or neither of them is carried out

# Results

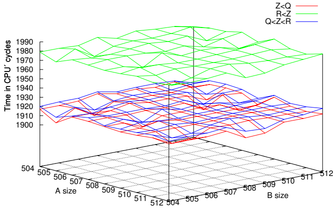



FIGURE: $|Q| = 512$

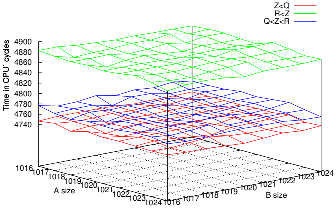



FIGURE: $|Q| = 1024$

- A timing difference is observable when $Z > R$
- Extra-reduction ($Q \leq Z < R$) : masked by dummy subtraction
- Bias is proportional to the bitsize of Q

# Results of timing attacks against POLARSSL

**Kocher Attack (1996)**

does not apply to RSA-CRT

**Schindler's Attack (2000)**

works only with the square-&-multiply algorithm

**Schindler's Attack (2005)**

does not work due to the window in the sliding windows exponentiation

**Brumley-Boneh Attacks (2003)**

can work

# Overview

# Probability of an extra bit

**MULTIMONTMUL(A,B,P), $0 \leq A, B < P$**

The probability of an extra bit is not null **iff** $P > \frac{\sqrt{5}-1}{2} \times R$

**Probability of an extra bit**

B is uniformly distributed and C is a fixed value :

- $P_{\text{MULTIMONTMUL}(B,B,P)} = \frac{P}{3R} + \frac{2(R-P)\sqrt{(R-P)R}}{3P^2} - \frac{(R-P)}{R}$
- $P_C = P_{\text{MULTIMONTMUL}(C,B,P)} = \frac{C}{2R} + \frac{(R-P)^2 R}{2CP^2} - \frac{(R-P)}{R}$

For $P > \frac{\sqrt{5}-1}{2} \times R$ and $X, Y \in \left(\frac{(R-P)R}{P}, P\right)$, if $X > Y$ then $P_X > P_Y$

## Attack Characteristics

- Allows to recover the $\frac{|p|}{2}$ most significant bits of $p$ or $q$
- Search each bit gradually using an approximation of $p$ or $q$
- Use Coppersmith algorithm to complete the attack

## Recovering a 1024 key size with RDMSR instruction in inter-process

- Around ten minutes.
- 215200 queries.

## Attack Characteristics

- Allows to recover the $\frac{|p|}{2}$ most significant bits of $p$ or $q$
- Search each bit gradually using an approximation of $p$ or $q$
- Use Coppersmith algorithm to complete the attack

## Recovering a 1024 key size with RDMSR instruction in inter-process

- Around ten minutes.
- 215200 queries.

# Searching the $p_k$ bit

Assume the adversary knows the $k$ most significant bits of $p$. He can generate the integers $c_1$ and $c_2$ :

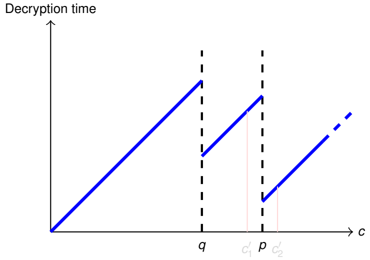
- $c'_1 = (p_0, p_1, .., p_{k-1}, 0, 0, ..., 0)_2$
- $c'_2 = (p_0, p_1, .., p_{k-1}, 1, 0, ..., 0)_2$

# Searching the $p_k$ bit

Assume the adversary knows the $k$ most significant bits of $p$. He can generate the integers $c_1$ and $c_2$ :

- $c'_1 = (p_0, p_1, .., p_{k-1}, 0, 0, ..., 0)_2$
- $c'_2 = (p_0, p_1, .., p_{k-1}, 1, 0, ..., 0)_2$

$p_k = 0$



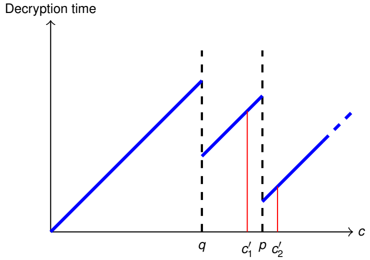

$p_k = 1$

# Searching the $p_k$ bit

Assume the adversary knows the $k$ most significant bits of $p$. He can generate the integers $c_1$ and $c_2$ :

- $c'_1 = (p_0, p_1, .., p_{k-1}, 0, 0, ..., 0)_2$
- $c'_2 = (p_0, p_1, .., p_{k-1}, 1, 0, ..., 0)_2$

$p_k = 0$

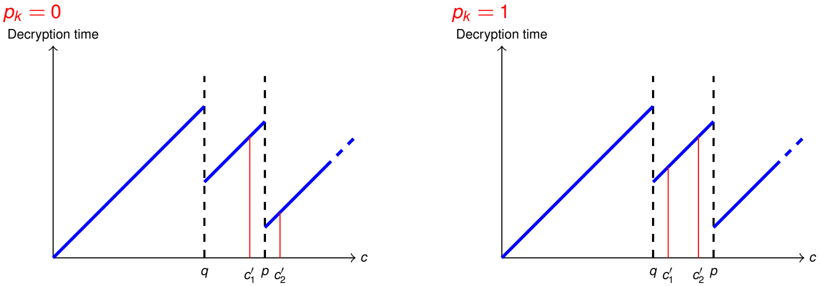



$p_k = 1$

# Searching the $p_k$ bit

Assume the adversary knows the $k$ most significant bits of $p$. He can generate the integers $c_1$ and $c_2$ :

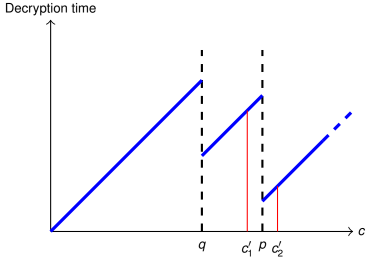
- $c'_1 = (p_0, p_1, .., p_{k-1}, 0, 0, ..., 0)_2$
- $c'_2 = (p_0, p_1, .., p_{k-1}, 1, 0, ..., 0)_2$

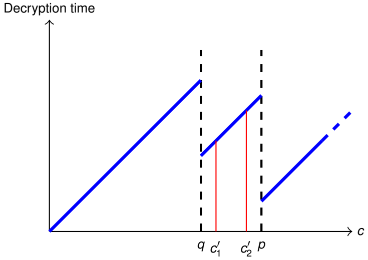


# Searching the $p_k$ bit

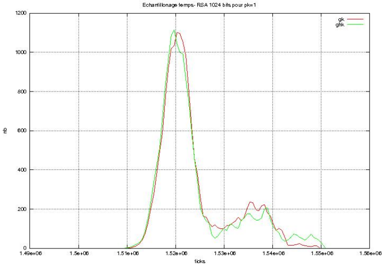
Assume the adversary knows the $k$ most significant bits of $p$. He can generate the integers $c_1$ and $c_2$ :

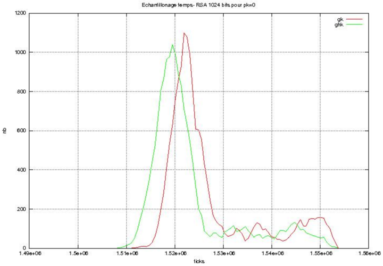
- $c_1' = (p_0, p_1, .., p_{k-1}, 0, 0, ..., 0)_2$
- $c_2' = (p_0, p_1, .., p_{k-1}, 1, 0, ..., 0)_2$



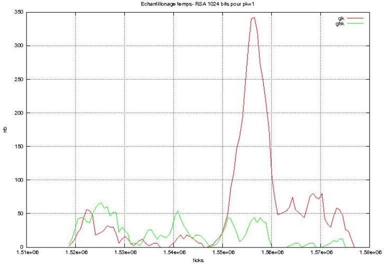



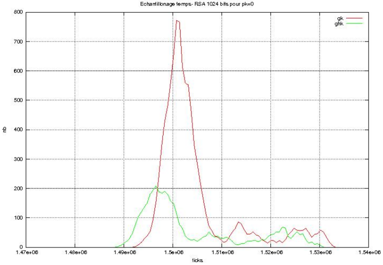

Two timing samples for values close to $c_1 + \varepsilon_1$ and $c_2 + \varepsilon_2$ : $\zeta_{c_1}$ and $\zeta_{c_2}$




How can we quantify
the difference between the two samples $\zeta_{c_1}$ and $\zeta_{c_2}$ ?




How can we check that the measure of the two samples $\zeta_{c_1}$ and $\zeta_{c_2}$ is correct ?

# Statistical Tests

## T-test

- Allow to compare the means of two samples generated by 2 populations with equal variance
- If $t_{\text{observed}} > t_{\text{threshold}}$, $p_k = 0$ otherwise $p_k = 1$.

## Fisher-Snedecor Test

- Allows to compare the variances of two samples generated from 2 populations
- If $F_{\text{observed}} > F_{\text{threshold}}$ : replay otherwise the value of $t_{\text{observed}}$ allows to determine $p_k$

## In practice

Search the intervalle $I$ or $F_{\text{observed}}$ is maximal then compute t-test on $I$

# Same process

| Modulus size | #query/bit | ratio of replay | # query |
|---|---|---|---|
| 512 bits | 600 | 2% | 78600 |
| 1024 bits | 800 | 18% | 241600 |
| 2048 bits | 1000 | 50% | 768000 |

TABLE: RDTSC instruction

| Modulus size | #query/bit | ratio of replay | # query |
|---|---|---|---|
| 512 bits | 600 | 2% | 78600 |
| 1024 bits | 800 | 10% | 225600 |
| 2048 bits | 1000 | 15% | 589000 |

TABLE: RDMSR instruction

# Inter-process via TCP IP

| Modulus size | #query/bit | ratio of replay | # query |
|---|---|---|---|
| 512 bits | 1000 | 2% | 131000 |
| 1024 bits | 1100 | 21% | 341000 |
| 2048 bits | 1200 | 55% | 952800 |

TABLE: RDTSC instruction

| Modulus size | #query/bit | ratio of replay | # query |
|---|---|---|---|
| 512 bits | 1000 | 0% | 128000 |
| 1024 bits | 1100 | 5% | 295900 |
| 2048 bits | 1200 | 10% | 675600 |

TABLE: RDMSR instruction

# Amplifying the bias by repetiting

The bias in our case is very small and is consequently hard to detect

## Sliding Exponentiation

PolarSSL uses a CLNW (Constant Length Non-Zero Window) method whereas OpenSSL uses a VLNW method (Variable)

## Precomputation phase

- As in the CCS '05 paper, we use the precomputation phase
- many multiplications are used with the same value $c_1$ or $c_2$ to compute the precomputation table
- 31 multiplications with the same value if the window length is 5

# Distribution of modulus

We generated randomly keys with PolarSSL's key generation routine ($p > q$)

| | $]0.5R; \frac{\sqrt{5}-1}{2}R]$ | $]\frac{\sqrt{5}-1}{2}R, 0.7R]$ | $]0.7R; 0.8R]$ | $]0.8R; R]$ |
|---|---|---|---|---|
| Distribution of p size | 0% | 0.011% | 13.33% | 86.66% |
| Distribution of q size | 18.78% | 24.46% | 31.32% | 25.44% |

TABLE: Distribution of modulus.

Our attack is always feasible in practice

# Overview

# OPENSSL Countermeasure

## Blinding

- Use a random before each decryption
- Efficient for all parameter size
- Slow down performance by a factor between 10 to 25%

# Cancelling out extra bit

## Use particular modulus size

if $|Q| \neq kw$, extra-bit is cancelling out

## Modify PolarSSL's key generation routine

Generate keys where prime factors are less than $\frac{\sqrt{5}-1}{2} \times R$

# Overview

# Conclusion

## Conclusions

- Constant time cryptographic implementations are hard to achieve
- Known attacks exploit the bias of the extra-reduction due to an extra bit

## Our contributions

- Practical Timing Attack against a protected implementation
- Use statistical tests to reduce the number of chosen ciphertexts
- Introduce a new bias
- Propose 2 countermeasures for specific key sizes