Timing Attack on RSA Signature Scheme

F. De Santis
desantis@tum.de

Sichere Implementierung kryptographischer Verfahren WS 2015-2016
Lehrstuhl für Sicherheit in der Informationstechnik
Technische Universität München

11.11.2015

Outline

Section 1

Timing Analysis

Timing Analysis

- The implementation of cryptographic algorithms often leads to non-constant execution times
  - ➥ **conditional instructions**
  - ➥ cache mechanisms
  - ➥ compiler level optimizations
  - ➥ ...
- These timing variations can be exploited by the means of statistical analysis to recover the secret key processed within the device

Section 2

Timing Attack on RSA Signature Generation

RSA Signature Scheme

Key Generation

- Given two primes p, q compute $n = pq$ and $\Phi(n) = (p-1)(q-1)$.
- Public key $k_{pub} = (n, e)$ s.t. $e \xleftarrow{\$}]1, \Phi(n)[$ and $\gcd(e, \Phi(n)) = 1$
- Private key $k_{pr} = (n, d)$ s.t. $ed = 1 \mod \Phi(n)$

Signature Generation and Verification

Signing Entity $\mathcal{A}$		Verifying Entity $\mathcal{B}$
$y = \mathrm{sig}^{\mathcal{A}}_{k_{pr}}(x) = x^d \bmod n$	$\xrightarrow{\text{send}(x,y)}$	
	$\xleftarrow{\text{accept or reject}}$	$\mathrm{ver}^{\mathcal{B}}_{k_{pub}}(x, y) = \begin{cases} \text{accept if} & y^e \bmod n = x \\ \text{reject if} & y^e \bmod n \neq x \end{cases}$

Lehrstuhl für Sicherheit in der Informationstechnik

Technische Universität München

Montgomery Multiplication

Algorithm 1 Montgomery Multiplication

Input: $a, b \in \mathbb{Z}, n \in \mathbb{Z}^+, z = 2^k$
Output: $abz^{-1} \mod n$

$n' \leftarrow (-n)^{-1} \mod z$ (precomputation)
$c \leftarrow ab$ (integer multiplication)
$d \leftarrow cn' \mod z$ (truncation to k bits being $z = 2^k$)
$e \leftarrow c + nd$ (integer multiplication and addition)
$f \leftarrow e/z$ (right shift by k bits being $z = 2^k$)
if $f \geq n$ **then**
    $f \leftarrow f - n$ (Extra-Reduction)
end if
return f

- The Extra-Reduction (ER) step leaks information about the inputs
- If the inputs depend on the secret key, a timing attack can be mounted to recover the secret information

Left-to-Right Square-and-Multiply

Algorithm 2 Left-to-Right Square-and-Multiply with Montgomery Multiplication

Input: $y, n, d = (d_{k-1}, d_{k-2}, \ldots, k_1, k_0)_2$
Output: $x = y^d \bmod n$

```
z ← 2^k                                  (precomputation)
z^2 ← zz (mod n)                         (precomputation)
n' ← (−n)^{−1} (mod z)                   (precomputation)
y' ← MM(y, z^2)                          (→ conversion to the Montgomery domain)
x' ← z                                   (→ conversion to the Montgomery domain)
for i = k − 1 downto 0 do                (scan through the key bits d_i from MSB to LSB)
    x' ← MM(x', x')                      (do the squaring always)
    if d_i = 1 then                      (if the key bit d_i is set)
        x' ← MM(x', y')                  (do the conditional multiplication)
    end if
end for
x ← MM(x', 1)                            (← conversion back to the integer domain)
return x
```

Timing Attack: Overture

- The execution times $t_0, \ldots, t_{N-1}$ of N signature generations on uniformly chosen inputs $x_0, \ldots x_{N-1}$ are measured

- Let x be an arbitrary integer of size k and $D_{i-1} = (d_{k-1}, \ldots, d_{k-i})_2$

- Let er be the function $(x, D_{i-1}) \mapsto \{0, 1\}$ which returns 1 if the ER step is performed on the LSB of D_{i-1} during the modular exponentiation of x or 0 otherwise:

$$\mathrm{er}(x, D_{i-1}) = \begin{cases} 1 & \text{if the reduction for } d_{k-i} \text{ is performed} \\ 0 & \text{otherwise} \end{cases}$$

Timing Attack: Procedure

Let assume $D_{i-1} = (d_{k-1}, \dots, d_{k-i})_2$ is known, then:

1. Make an hypothesis $\kappa \in \{0, 1\}$ on the key bit d_{k-i-1}:

$$D_i^\kappa = (d_{k-1}, \dots, d_{k-i}, \kappa)_2$$

2. Classify timings $t_0, \dots, t_{N-1}$ into two sets $\mathcal{T}_0^\kappa$ and $\mathcal{T}_1^\kappa$:

$$\mathcal{T}_0^\kappa = \{t_j : \mathrm{er}(x_j, D_i^\kappa) = 0\} \text{ and } \mathcal{T}_1^\kappa = \{t_j : \mathrm{er}(x_j, D_i^\kappa) = 1\}$$

3. The abs difference of means is used to verify the key hypothesis

$$|\tau_0^\kappa - \tau_1^\kappa|$$

where $\tau_0^\kappa = \frac{1}{|\mathcal{T}_0^\kappa|} \sum_j t_j$ with $t_j \in \mathcal{T}_0^\kappa$ and $\tau_1^\kappa = \frac{1}{|\mathcal{T}_1^\kappa|} \sum_j t_j$ with $t_j \in \mathcal{T}_1^\kappa$

4. The key hypothesis κ which lead to the largest value is chosen

Timing Attack: Procedure

- Alternatively, the sample Pearson's correlation coefficient r^κ can be used in place of 2 – 3:

$$r^\kappa = \frac{\sum_{j=0}^{N-1}(t_j - \frac{1}{N}\sum_{j=0}^{N-1} t_j)(e_j^\kappa - \frac{1}{N}\sum_{j=0}^{N-1} e_j^\kappa)}{\sqrt{\sum_{j=0}^{N-1}(t_j - \frac{1}{N}\sum_{j=0}^{N-1} t_j)^2 \sum_{j=0}^{N-1}(e_j^\kappa - \frac{1}{N}\sum_{j=0}^{N-1} e_j^\kappa)^2}},$$

  where $e_j^\kappa = er(x_j, D_i^\kappa)$.

- The last key bit must always be guessed, no look-ahead is possible for the last bit

Section 3

Task Description

Task Description

1. Implement the timing attack on RSA signature generation in Python 2.x
   - ➥ Left-to-right Exponentiation Algorithm
   - ➥ Montgomery Multiplication
   - ➥ Non-CRT Format
   - ➥ **Integers size is** 64**-bit**
   - ➥ **Sample Pearson's correlation coefficient**

2. Run the timing attack against given timings and recover the secret exponent

Section 4

Framework

Framework

- IDE: Ninja (Windows, Linux, MacOS)
  - ➥ http://ninja-ide.org/downloads/
- Skeleton files
  - ➥ `project.nja` is the project file for Ninja IDE
  - ➥ `main.py` has **NOT** to be modified
  - ➥ `student.py` must be modified to implement the timing attack in the `perform_timing_attack` function
- Timings at https://tueisec-sica.sec.ei.tum.de/rsa/
  - ➥ **Secret exponent *d* different for each student**
- A testing pair (message, signature) is provided to verify the correctness of the recovered secret exponent

Section 5

Submitting Results

Handing in Results

Hand in @ `https://tueisec-sica.sec.ei.tum.de/handin/`

- `student.py`
- `key.txt`
  - ➥ It is automatically generated by $ python main.py
  - ➥ **Submit `key.txt` without further modifications**

- Multiple submissions are possible
  - ➥ Only the last submission is considered (files are overwritten)

- Deadline for submission fixed in 3 weeks
  - ➥ **02.12.15 23:59:59 CET**

Final Remarks

- The assignment is passed if the `key.txt` is correct **and** the `student.py` works correctly on freshly generated new timings

- Reuse the information from previous computations to reduce the execution time of the timing attack and get rid of the noise

Stairway to Heaven

1. Download the framework
2. Download the timings
3. Implement the timing attack in Python
4. Run the timing attack on given timings
5. Submit the source code and the secret key **before 02.12.15 23:59:59 CET**