You can use any document but communicating devices are strictly forbidden. Please number the different pages of your paper and indicate on each page your first and last names. You can write your answers in French or in English, as you wish. Precede your answers with the question’s number. If some information or hypotheses are missing to answer a question, add them. If you consider a question as absurd and thus decide to not answer, explain why. If you do not have time to answer a question but know how to, briefly explain your ideas. Note: copying verbatim the slides of the lectures or any other provided material is not considered as a valid answer. Advice: quickly go through the document and answer the easy parts first.

The 5 questions are worth 2 points each. The problem is worth 10 points.

1 Questions

1.1. Timing attacks. A RSA implementation uses the Chinese Remainder Theorem. What does it change with respect to its sensitivity to timing attacks?

1.2. Power attacks. Table 1 lists the notations used in this question.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>signature</td>
</tr>
<tr>
<td>$m$</td>
<td>message to sign</td>
</tr>
<tr>
<td>$n$</td>
<td>public modulus</td>
</tr>
<tr>
<td>$d$</td>
<td>secret exponent</td>
</tr>
<tr>
<td>$w$</td>
<td>bit-width of $d$</td>
</tr>
<tr>
<td>$d_0$</td>
<td>Least Significant Bit (LSB) of $d$</td>
</tr>
<tr>
<td>$d_k$</td>
<td>$k^{th}$ bit of $d$</td>
</tr>
<tr>
<td>$d_{w-1}$</td>
<td>Most Significant Bit (MSB) of $d$</td>
</tr>
<tr>
<td>$d_{tmp}$</td>
<td>temporary variable</td>
</tr>
</tbody>
</table>

Table 1: Notations

Algorithm is the pseudo-code of a software implementation of RSA signature ($s = m^d \mod n$) that uses square and multiply and consumes one exponent bit per iteration, from LSB to MSB.

Assume you are in charge of mounting a power attack against a smart card running this software implementation. The micro-processor of the smart card (the CPU) is a simple one that executes one instruction per clock cycle (single core, no pipe-line, no super-scalar, no parallelism). You know the software implementation, it is part of an open source cryptographic library. Your goal is to retrieve the secret exponent. You can send chosen messages to the smart card and you...
Algorithm 1 RSA signature

1. \( s \leftarrow 1 \) \hfill \triangleright \text{ initial value of } s
2. \( \text{tmp} \leftarrow m \) \hfill \triangleright \text{ initial value of tmp}
3. for \( k \leftarrow 0 \ldots w - 1 \) do \hfill \triangleright \text{ loop from LSB to MSB of } d
4. \quad \text{if } d_k = 1 \text{ then}
5. \quad \quad \textbf{s} \leftarrow (s \times \text{tmp}) \pmod{n} \hfill \triangleright \text{ modular multiplication}
6. \quad \textbf{end if}
7. \quad \textbf{tmp} \leftarrow \text{tmp}^2 \pmod{n} \hfill \triangleright \text{ modular square}
8. \quad \textbf{end for}
9. \textbf{return } s

have access to the computed signatures. During each signature computation you can record the power consumption with a reasonably fast digital oscilloscope. Unfortunately its resolution is too low to distinguish the CPU instructions executed by the smart card; they all look exactly the same in the power traces. The only information you can retrieve from each power trace is thus the total number of executed CPU instructions during the signature computation, nothing more.

Is the attack possible? If no explain why. If yes explain what your attack will be.

1.3. Fault attacks. Give an example of a situation where a faulty result can be exploited by an attacker to retrieve a valuable secret. Explain what information is available in the faulty result that is not in the correct result.

1.4. Countermeasures. In order to protect a RSA software implementation against attacks, a security engineer decides to design his own implementation, instead of using publicly available software libraries, and to keep it secret. How efficient do you think this countermeasure is? What are its advantages and its drawbacks?

1.5. Side channel attacks. Among the different leak sources usually designated by the term side channel we find the computation time, the power consumption and the electromagnetic emissions. Explain the advantages and drawbacks of each of these 3 side channels from an attacker’s point of view.

2 Problem: Fault attack against a DES hardware implementation

To solve this problem you will need a global understanding of the DES algorithm. Figures 1, 2, and 3 should be sufficient. \( IP \) and \( FP \) are 64 to 64 bits permutations, inverse one of the other. \( \oplus \) is the bitwise exclusive of two bit strings. \( E \) is a 32 to 48 bits expansion-permutation. \( P \) is a 32 to 32 bits permutation. \( S_1, S_2, \ldots, S_8 \) are 8 different 6 to 4 bits non linear substitution functions (SBoxes). \( PC_1 \) is a 64 to 56 bits selection-permutation. \( PC_2 \) is a 56 to 48 bits selection-permutation. \( LS \) is a 28 to 28 bits rotation by one or two positions to the left, depending on the round index. All these primitive functions are perfectly defined in the DES standard.

Assume you are in charge of attacking the DES hardware implementation represented on figure 4. Each 64 bits block encryption takes 16 clock cycles. During the first clock cycle the input block is passed through \( IP \) and the result (denoted \( L_0R_0 \) in the DES standard) is stored in the \( L, R \) registers. During the same first clock cycle
Figure 1: DES encryption
Figure 2: DES Feistel function

Figure 3: DES key schedule
the 64 bits secret key is passed through $PC_1$ and $LS$ and the result (denoted $C_1D_1$ in the DES standard) is stored in the $C, D$ registers. The $L, R$ registers consecutively store $L_0R_0, \ldots, L_{15}R_{15}$ and the $C, D$ registers consecutively store $C_1D_1, \ldots, C_{16}D_{16}$. Note that $L_{16}R_{16}$ and $C_0D_0$ are never stored in the registers.

You can send chosen plaintexts and the DES hardware implementation will return you the corresponding ciphertexts. You can send the same plaintext several times. Your goal is to retrieve the unknown but constant secret key. During each encryption you can inject a fault or not. If you inject a fault it will cause an error but you cannot select when and where. All you know is that your fault will cause a single bit flip in one of the $C, D$ registers used by the key schedule. The fault may thus occur on any of the 56 bits of the registers and in any of the 16 successive values they store ($C_1D_1, \ldots, C_{16}D_{16}$), that is, 896 possibilities with a uniform probability.

- In your opinion, among these 896 possible faults which ones would be the easiest to exploit?
- Why? What can you deduce from such a fault? How?
- What is the probability that such a fault occurs?
- Explain what your attack will be. Carefully define your notations. In your notations use the same bit numbering as in the DES standard: the leftmost bit of a
$n$-bits bit string is numbered 1 and the rightmost bit is numbered $n$.

- What amount of information can you extract?
- Estimate the cost of your attack in terms of number of injected faults, number of DES encryptions, amount of storage, other computations . . .
- Do you think it is practical?
- If yes propose a countermeasure and discuss its efficiency and its drawbacks.