# ICR 2014-2015

# Practical Work #3

# *Fault Attacks against RSA-CRT*

## Rules of the Game

This practical work can be performed by *groups of two*, or *individually*. A written report must be delivered, that will contain your answers to the questions, all the code you wrote (that must be commented), an introduction, a conclusion, and if necessary, screenshots. The report can be written in French, German, or English, and it must be delivered as a PDF file named

`lab03_report_Lastname(s).pdf`.

The code must be delivered as a ZIP file named

`lab03_code_Lastname(s).zip`.

The two files must be sent by electronic mail to

`pascal.junod@heig-vd.ch`

before

**Monday January 12th, 2015, 18h00 CET.**

Being up to one day late will cost you one grade point, from one day to two days will cost two grade points, etc. Please do not forget to cite all your sources in a clear and precise manner!

# 1 Preliminaries

A way to accelerate the RSA signature procedure consists in exploiting the fact that one knows the two primes $p$ and $q$, as it is a private-key operation, and to use the Chinese Remainder Theorem (CRT). The goal of this practical work consists in implementing a fast RSA signature procedure that exploits the CRT and to study the security of such an implementation at the light of *fault attacks*. This practical work can be implemented in any programming language. However, safe choices are C/C++, since numerous libraries implementing big-numbers arithmetic are freely available (such as GMP[1]), or Python v3, as it natively supports the handling of large numbers.

# 2 RSA-CRT

This part is dedicated to the implementation of a RSA-CRT fast signing procedure. One can assume that 1024-bit RSA keys are used and that the digest formatting operation is performed elsewhere.

**Task 1.**

1. Implement an RSA key generation routine.
2. Implement standard RSA signature and verification routines.
3. Implement a fast RSA signature procedure relying on the Chinese Remainder Theorem.

**Question 1.**

1. How have you tested that your routines are properly working?
2. What is the gain in terms of speed that you obtain when using RSA-CRT with respect to a standard RSA signature generation procedure?
3. What are the values that one could pre-compute and store besides $n$ and $d$, in order to speed up as much as possible the signature generation procedure?

# 3 The Boneh-DeMillo-Lipton Attack

In 1997, Boneh, DeMillo and Lipton have demonstrated that if a fault is induced during one of the two partial signature computation steps, that erroneous signature can be exploited in order to factor the public modulus.

[1] https://www.gmplib.org

**Task 2.**

Describe in mathematical terms how the Boneh-DeMillo-Lipton fault attack against RSA-CRT is working.

**Question 2.**

1. In practice, how is it possible to induce faults in cryptographic implementations?
2. Is this attack working on a non-deterministic padding scheme?

**Task 3.**

Write a program simulating Boneh-DeMillo-Lipton attack that allows to factor $n = pq$ in a very efficient way.

## 4 Implementing Shamir's Trick

Several countermeasures have been proposed to defend against Boneh-DeMillo-Lipton attack. In this part, we will study and implement the one that is known as *Shamir's trick*. This technique essentially works as follows: the partial signatures are computed modulo $rp$ and $rq$, where $r$ is a small (i.e., 32-bit) random integer, instead of working modulo $p$ and $q$, respectively.

**Task 4.**

Describe in mathematical terms how Shamir's trick works.

**Task 5.**

Implement an RSA-CRT routine protected against Boneh-DeMillo-Lipton attack thanks to Shamir's trick.