Remote Timing Attacks: Exposing RSA Private Keys

Ioan Ionescu, Radu Tomescu, Virgil Turcu
Computer Science Department, University of Bucharest, Romania

Abstract

1 Introduction

2 Technical description of the problem

3 Possible solution

4 Experiments and Results

The experiments conducted in the paper aim to clarify the proper requirements for the attack to work and to test the overall efficiency of the attack in various environments.

The first experiment, which is the most relevant, is split into two sections: the first one is to illustrate that increasing the sample size for each bit guessing attempt provides a better chance to expose the private key; the second one aims to prove that the size of the neighborhood over which we query each guess is directly proportional to the zero-one bit gap.

The first part is quite intuitive. Its result shows, in simple terms, how trying to guess a bit of the q multiple times will yield a higher chance of actually guessing that bit right. This is a direct consequence of the Law of Large Numbers, which states that the sample mean tends to the expected value of the distribution that models a trial as the sample size approaches infinity.

The second part of the experiment is based on the measurement T_g defined in the previous section and shows how increasing the neighborhood over which we query increases the difference in decryption time that indicates whether the current bit of q is 1 or 0. This is, without a doubt, the most relevant conclusion of this article, because it provides a solution

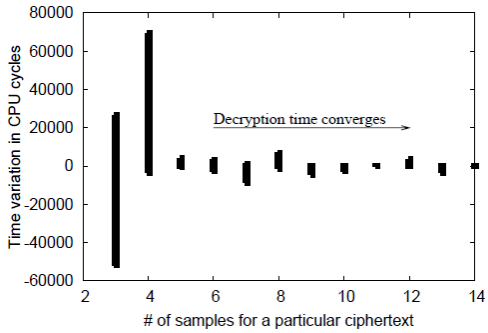

Figure 1: Variance evolution as sample size increases

to bypass "Sliding window multiplication".

To further solidify this outcome we provide our own simulation of this experiment, realized in R. The following statements are our statistical assumptions:

- $DecryptTime(g)$ can be modelled by an exponential random variable $Exp(\lambda)$ where $\lambda = 1/ExpectedDecryptionTime$.
- Increasing the $g + i$ sequence translates to increasing the sample size for the exponential random variable.
- All trials over a neighborhood are independent.

Having taken into consideration the previous assumptions, and the result that states that the sum of identical and independent exponential random variables is a Γ distribution, we obtain the following:

$$T_g = \left(\sum_{i=0}^{n} DecryptTime(g+i) \right) \sim \Gamma(n, \lambda)$$

```r
maxNeighbourhoodSize <- 1000
bits <- 512
time <- 5*(10^6)

#simulate Tg for neighbourhood of size n
Tg <- function (sampleSize, n, expectedTime) {
  return (rgamma(sampleSize, n, 1/expectedTime))
}

#generates |Tg-Tghi| with a neighbourhood of size n for a certain number of bits
generateDiffs <- function (n, numberOfBits, expectedTime) {
  return (abs(diff(Tg (numberOfBits, n, expectedTime))))
}

input <- 1:maxNeighbourhoodSize
meansMax <- c()
meansMin <- c()

for (i in input) {
  temp <- generateDiffs(i, bits, time)

  medianTemp<-median(temp)

  meansMax<-c(meansMax, mean(temp[temp >= medianTemp]))
  meansMin<-c(meansMin, mean(temp[temp < medianTemp]))
}

plot(input, meansMax, type ='l', xlab="Neighbourhood size", ylab="Time difference")
lines(input, meansMin)
```

Listing 1: R simulation code

In Listing 1 we have the R code used for generating the plot in Figure 3.

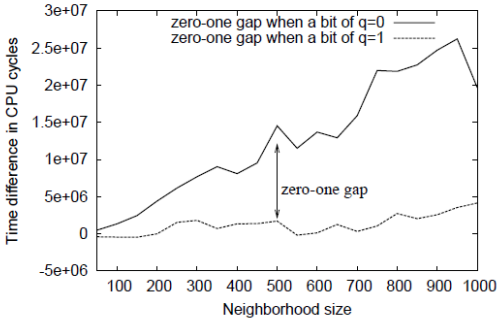

Figure 2: Measured Evolution of T_g difference

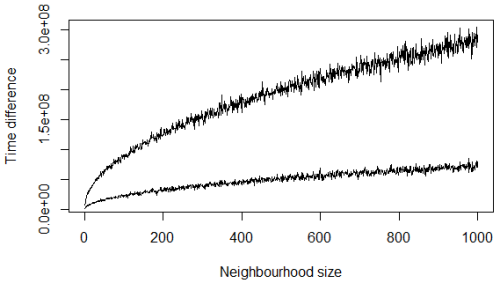

Figure 3: Simulated Evolution of T_g difference

When looking at Figure 2 and Figure 3 we notice that increasing the neighborhood size of n produces a more noticeable gap between the "high" and "low" decryption times. In turn, this means that we can predict more accurately whether a bit of q is 0 or 1.

One significant difference between the two figures is that in the simulation the "low" time differences are also slightly increasing, probably due to the assumptions we previously made. However, the general idea is still the same, as the gap between low and high values of T_g increases indefinitely.

Some other experiments presented in the paper involve: testing the efficiency of the algorithm against randomly generated 1024-bit keys, analyzing the difference between inter-process and local network attacks, or testing the attack against different source-based optimizations.