Lab on RSA Timing Attacks

RSA Timing Attacks

Brief Description
A timing attack is an attack which cleverly uses the fourth dimension, time. If an algorithm is not specifically designed to thwart this attack, then an attacker can observe the required amount of time for a calculation to be done and monitor the differences in calculation times. For example, the calculation of converting a “0” in plain text to cipher text versus converting a “1” in plain text to cipher text may require less time. This measured amount of time can be used to rebuild the key or figure out the plain text.

Lab Overview
RSA Encryption is complicated and also has protections against timing attacks, so we will be using a more simple example for this lab. We have performed two operations many, many times, specifically the add and multiply operation. The add is performed much faster than the multiply especially when scaled across many iterations. We will use this as our test case; a shorter operation will represent the processing of a zero and the long operation would represent the processing of a one. So given a stream of output times from a program which monitors these operations, you should be able to reconstruct a string ones and zeroes.

Different machines will require a different amount of time to process. So our implementation will take this into account by not using specific time values when processing the times from our “gathered” data. It may be a good approach to calculate an average time, and then compare each time value against this value to determine if it is a “1” or a “0”. After we have created a string of ones and zeroes, we will process these to generate our ASCII output (Google “ascii table”, if you are confused)

To Complete...
Using Visual Studio, open the provided “TimingAttackLab.sln”. This project was used to create the file “time_data”, which should be located in “....\TimingAttackLab\bin\Debug”. Have a look at the “time_data” file; it is simply the number of ticks used to calculate a 1 or a 0. We assume a 1 takes more time to calculate than a 0.

The project only has a few functions that are left to be implemented in order for you to decrypt the super secret message.

Turn in a lab report containing:

-Your implementation of the functions “public static String BinStr2ASCII(String BinStr)” and
The Timing attack is based on the fast-exponential algorithm. The algorithm uses only squaring if the corresponding bit in the private exponent $x$ is 0; it uses both squaring and multiplication if the corresponding bit is 1. In other words, the timing required to do each iteration is longer if the corresponding bit is 1. This timing difference allows Eve to find the value of bits in $x$, one by one.

The fast exponentiation algorithm can use both squaring and multiplication. The main idea behind this method is to treat the exponent as a binary number of $n_b$ bits ($x_0$ to $x_{n_b-1}$). For example, $x = 22 = (10110)_2$. The idea behind the square-and-multiply method is

$$y = a^{x_{n_b-1} \times 2^{n_b-1} + x_{n_b-2} \times 2^{n_b-2} + \ldots + x_1 \times 2^1 + x_0 \times 2^0}$$

in which $x_i$ is 0 or 1

$$y = \prod_{i=0}^{n_b-1} \begin{cases} a^{2^i} & \text{if } x_i = 1 \\ 1 & \text{if } x_i = 0 \end{cases}$$

Example:

$$y = a^9 = a^{10010} = a^8 \times 1 \times 1 \times a$$

Note that $y$ is the product of $n_b$ terms. Each term is either 1 (if the corresponding bit is 0) or $a^{2^i}$ is the bit is 1. In other words, the term $a^{2^i}$ is included in the multiplication if the bit is 1, it is not included if the bit is 0 (multiplication by 1 has no effect).

The following algorithm reflects these two observations. In each iteration, the algorithm checks the value of the corresponding bit. If the value of the bit is 1, it multiplies the current base with the previous value of the result. It then squares the base for the next iteration.
1. To encrypt bit pattern $P$ in RSA, compute

$$a = P^e \mod n$$

2. To decrypt received bit pattern $a$ in RSA, compute

$$P = a^x \mod n$$

Try to launch a timing attack against the decryption of RSA to figure out the private key $x$ if RSA uses above fast exponentiation algorithm. Hint: You can use a timer to record execution time of each iteration, use the average value of all iteration as threshold to determine each bit of $d$ is 0 (below threshold) or 1 (above threshold).

**What to submit:** The report includes the process of discover a key of RSA using timing attacks.