CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Timing Attack on the RSA Cipher
Student:	Martin Andrýsek
Supervisor:	Ing. Jiří Buček
Study Programme:	Informatics
Study Branch:	Information Technology
Department:	Department of Computer Systems
Validity:	Until the end of winter semester 2018/19

Instructions

Review known timing side channel attacks on RSA decryption and signing operations. Create a demonstration application that will perform timing attack on RSA in order to determine the private key. The application will be used in courses on cryptology and computer security as a part of laboratory exercises. Consider an attack on a local computer or over the network and evaluate its time complexity.

References

Will be provided by the supervisor.

prof. Ing. Róbert Lórencz, CSc.
Head of Department

prof. Ing. Pavel Tvrdík, CSc.
Dean

Prague March 7, 2017

Czech Technical University in Prague
Faculty of Information Technology
Department of Computer Systems

Bachelor's thesis

Timing Attack on the RSA Cipher

Martin Andrýsek

Supervisor: Ing. Jiří Buček

15th May 2017

Acknowledgements

THANKS (remove entirely in case you do not with to thank anyone)

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 15th May 2017 . . . .

Citation of this thesis

Andrýsek, Martin. *Timing Attack on the RSA Cipher*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2017.

Abstrakt

Tato prace se zabyva utokem na sifru RSA casovym postrannim kanalem. Pomoci mereni casu podepisovani predgenerovanych zprav, je utocnik schopen postupne uhadnout kazdy bit soukromeho klice. Vysledkem prace je demonstrativni aplikace, ktera bude pouzita ve vyuce predmetu, zabyvajicimi se pocitacovou bezpecnosti.

Klíčová slova Replace with comma-separated list of keywords in Czech.

Abstract

This thesis is focused on replication of timing attack on RSA cipher, which is done by measuring time of square and multiply algorithm. Implementation should be used for education purposes, mainly in security courses.

Keywords RSA, cryptoanalysis, timing attack, side channel, square and multiply

Contents

List of Figures

Introduction

Chapter 1

State-of-the-art

Chapter 2

RSA

RSA is public-key cryptosystem which was invented by Ron Rivest, Adi Shamir and Leonard Adleman. The cryptosystem was published in the 1977.

2.1 Principle

The cipher is based on modular exponentiation. The whole process of crypting message is divided to four steps

2.1.1 Key generation

This is steps needed to generate public and private keypair

- Generate p and q, which have to be distinct prime numbers.
- Compute n, where $n = pq$
- Compute Euler's totient function $\phi(n)$. Because we know p and q it is simple to compute it.

$$\phi(n) = (p-1)(q-1)$$

- Generate e such as $\gcd(e, \phi(n)) = 1$
- Compute $d = e^{-1} \bmod \phi(n)$
- The pair (e, n) is released as public key
- The pair (d, n) is secret private key

2.1.2 Key distribution

Alice would like to send Bob secret message. Bob generates public key (e, n) and his private key (d, n). Bob sends Alice public key using reliable route (it has not to be secret route), Alice uses it to encrypt her message and sends it to Bob. Bob decrypts her message using his private key.

2.1.3 Encryption

Encryption is done by using public keypair (e, n):

$$c = |m^e|_n$$

where m is plaintext message and c is encrypted message which will be sent to receiver.

2.1.4 Decryption

Decryption is done similar thanks to relation $ed \equiv 1 \pmod{\phi(n)}$. We can simply power ciphertext to our private exponent d to obtain original message.

$$|c^d|_n = |(m^e)^d|_n = |m^{ed}|_n = |m^1|_n = m$$

2.2 Optimization

Because we generally use high value of modulus n the exponentiation of such high numbers is very time consuming so there are some algorithms to increase speed of computation

2.2.1 Chinese remainder theorem

By using CRT we can significantly speed up decryption of received messages. This method is not usable during encrypting phase because we need to know p and q factors of n. Assuming that $p > q$ we can precompute:

$$dP = e^{-1} \pmod{p-1}$$

$$dQ = e^{-1} \pmod{q-1}$$

$$qInv = q^{-1} \pmod{p}$$

After that, we compute message m with given c:

$$m_1 = c^{dP} \pmod p$$

$$m_2 = c^{dQ} \pmod q$$

$$h = qInv \cdot (m_1 - m_2) \pmod p$$

$$m = m_2 + hq$$

Finding modular exponentiation cost grows with cube of number of the bits in n, so it is still more efficient to do two exponentiation with half sized modulus

2.2.2 Montgomery Multiplication

Normal modular multiplication could be quite slow for large numbers, due to processor have to run several operations before it gets desired remainder. On the other hand P. L. Montgomery developed algorithm which assumes that processor do division by power of 2 really fast.

Montgomery presented algorithm, which transform numbers to Montgomery base and then compute modular multiplication efficiently. To transform number to Montgomery base we need to compute $\bar{a} = ar \pmod n$ where r is the next greater power of 2 than n. For example if $2^{63} < n < 2^{64}$ then desired r will be 2^{64}. The multiplication in Montgomery base is done by:

$$\bar{u} = \bar{a}\bar{b}r^{-1} \pmod n$$

where $r{-}1$ is modular inversion of r.

As we can see $\bar{u}$ is in Montgomery base of the corresponding $u = ab \pmod n$ since

$$\begin{aligned} \bar{u} &= \bar{a}\bar{b}r^{-1} \pmod n \\ &= (ar)(br)r^{-1} \pmod n \\ &= (ab)r \pmod n \end{aligned} \tag{2.1}$$

2.2.2.0.1 Montgomery reduction which gives us $\bar{u}$ is implemented this way:

Algorithm 1 Montgomery Reduction

function $\textsc{Mon\_Red}(\bar{a}, \bar{b}, N)$
  $t \leftarrow \bar{a} * \bar{b}$
  $m \leftarrow N^{-1} * t \pmod r$
  $\bar{u} \leftarrow (t + mN)/r$
  if $\bar{u} > N$ **then**
    $\bar{u} \leftarrow \bar{u} - N$
  end if
  return $\bar{u}$
end function

Its main advance is that it never performs division by the modulus n but we still need to find out u and precompute n^{-1} using the extended Euclidean algorithm. It is done by this algorithm:

Algorithm 2 Montgomery Multiplication

function $\textsc{Mon\_Mult}(a, b, n)$
  $r \leftarrow 2^{BitLen(n)}$
  Compute n^{-1} using the extended Euclidean algorithm
  $\bar{a} \leftarrow a * r \pmod n$
  $\bar{b} \leftarrow b * r \pmod n$
  $\bar{u} \leftarrow Mon_Red(\bar{a}, \bar{b})$
  $u \leftarrow Mon_Red(\bar{u}, 1)$
  return u
end function

2.2.3 Square and Multiply

This optimization uses bitwise representation of the exponent. The algorithm picks all byte from left (MSB) to right and despite their value, it determines which operation will be performed for each bit. For bits equal to 1 we perform squaring preset value c then we multiply it with the base of exponentiation m. For bits equal to 0 we just perform squaring part. Therefore we get data dependent operation, which will be used in our attack. For even faster implementation we use Montgomery multiplication instead of normal one. In some theses this Square and Multiply algorithm is called Montgomery exponentiation

Algorithm 3 Square & Multiply algorithm

```
function SQUARE_AND_MULTIPLY(m, e, n)
    c ← 1
    k ← BitLen(e)
    for i ← k − 1, 0 do
        c ← Mon_M ult(c, c)
        if e[i] == 1 then                    ▷ ith bit of exponent e
            c ← Mon_M ult(c, m)
        end if
    end for
    return c
end function
```

Chapter 3

Attacks

The basic idea of timing attacks was presented by Kocher in 1996. He specified theoretical attacks not only on RSA.

Both variant of attack are based on similar principle. They divide messages from set M to several subsets M_i due to response of some Oracle O. Then by measuring time of decrypting or signing and guessing bits of secret exponent by comparing times of each set.

3.1 Attack on multiply

First Kochers idea was to exploit multiply operation in Square and Multiply algorithm. Kocher mean to measure time of decryption (or signing) messages using the private key d and focus on conditional multiply step. We are attacking each bit of d with knowledge of $i-1$ bits we can guess the ith bit. Let $d = d_1, d_2, \ldots, d_k$ where k is bit length of d and d_1 is MSB. We can assume that $d_1 = 1$ so we can attack bit d_2.

We need oracle O which predict whether final Montgomery reduction happened during multiply step:

$$O(m) = \begin{cases} 1 & \text{if } m^2 * m \text{ is done with final reduction} \\ 0 & \text{if } m^2 * m \text{ is done without final reduction} \end{cases}$$

where m is message from set M. We can now divide messages to 2 subsets:

$$M_1 = \{m \in M : O(m) = 1\}$$

$$M_2 = \{m \in M : O(m) = 0\}$$

We can now measure time of these two subsets. We are expecting same times for doing square part, but in multiply part will be messages from M_1 higher, due to final Montgomery Reduction. We compare means of sets M_1 and M_2. If time of M_1 is significantly bigger then the final reduction was done therefore bit d_2 is 1. If the times of M_1 and M_2 are equal then bit d_2 is 0. .

Problem: We cannot be sure what is significant difference between time means. So our guesses cannot be precise.

3.2 Attack on square

Focusing on squaring operation will give us better results. The procedure is similar but we generate two oracles and four sets of messages. We similarly iterate through the bits of secret key d as in multiply attack. When we know $i-1$ bits and we are guessing ith bit we compute m_{temp} which has value before unknown possible multiplication step.

We first presume that bit d_i is 1. If the presumption is right then the following steps will be executed. m_{temp} will be multiplied by m, then the result of multiplication will be squared. We will execute the multiplication step and then we will check if in the square step is done with or without reduction. By this criterion we divide messages to subsets M_1 if the reduction was computed or M_2 if not. The oracle will be:

$$O_1(m) = \begin{cases} 1 & \text{if } (m_{temp} * m)^2 \text{ is done with final reduction} \\ 0 & \text{if } (m_{temp} * m)^2 \text{ is done without final reduction} \end{cases}$$

Secondly, we presume that bit d_i is 0. In that case only the square phase m_{temp}^2 will be executed so we similarly divide messages to subsets M_3 with reduction and M_4 without reduction. Oracle O_2:

$$O_2(m) = \begin{cases} 1 & \text{if } m_{temp}^2 \text{ is done with final reduction} \\ 0 & \text{if } m_{temp}^2 \text{ is done without final reduction} \end{cases}$$

We now get 4 subsets of M:

$$M_1 = \{m \in M : O_1(m) = 1\}$$
$$M_2 = \{m \in M : O_1(m) = 0\}$$
$$M_3 = \{m \in M : O_2(m) = 1\}$$
$$M_4 = \{m \in M : O_2(m) = 0\}$$

Let $T_i(M_i)$ be the mean time of computing messages from M_i.

Certainly, only one of oracles is giving us the right results. We can compare time difference between O_1 and O_2. That means if $T_1 - T_2$ is greater than $T_3 - T_4$ then we can be sure that bit d_i is 1, otherwise $d - i$ is 0. The problem from multiply attack is no more actual because one of the differences have to be higher than other.

CHAPTER 4

Defense

4.1 Additional reduction

The most obvious defense is to add dummy subtraction to Montgomery reduction algorithm which does not change any value but consume the same amount of time as if the real subtraction was performed. This should not significantly slow the computation but it totally eliminate this type of timing attack by making Montgomery reduction constant time function.

4.2 Masking

We can mask the ciphertext before computation of $c^d \pmod{n}$ so the attacker will not know which cipher text is decrypted. It is done simply by generating pair of masks before each exponentiation. We generate random mask m. Then we compute m':

$$m' = (m^{-1})^e \pmod{n}$$

where e is public exponent.

Before each exponentiation we multiply the ciphertext c with mask m' so we get masked x_m:

$$\begin{aligned} x_m &= (c * m')^d \pmod{n} \\ &= (c * (m^{-1})^e)^d \pmod{n} \\ &= c^d * m^{-1} \pmod{n} \end{aligned} \tag{4.1}$$

from where we can see that c^d is our desired message masked by m^{-1}. Then we simply recover x by multiplying by m:

$$\begin{aligned} x &= x_m * m \pmod{n} \\ &= x * m^{-1} * m \pmod{n} \\ &= x \pmod{n} \end{aligned} \tag{4.2}$$

To avoid situation when even generating of mask could become target of timing attack, there is simple workaround. To generate new mask, just square the mask pair:

$$m = m^2 \pmod n$$
$$m' = m'^2 \pmod n$$

Chapter 5

Realisation

Conclusion

Bibliography

Appendix A

Acronyms

MSB Most significant bit

LSB Least significant bit

CRT Chinese remainder theorem

Appendix B

Contents of enclosed CD

```
readme.txt ........................ the file with CD contents description
├── exe ............................. the directory with executables
├── src ............................. the directory of source codes
│   ├── wbdcm ....................... implementation sources
│   └── thesis ............ the directory of LaTeX source codes of the thesis
└── text ............................ the thesis text directory
    ├── thesis.pdf .................. the thesis text in PDF format
    └── thesis.ps ................... the thesis text in PS format
```