Prime Modulo RSA Analysis With Euler's Criterion

Implementation of Divide and Conquer Algorithm

Frankie Huang - 13521092
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521092@std.stei.itb.ac.id

***Abstract*—RSA is one of the most used encryption algorithm commonly used in data transmission. One of the criteria for building an RSA cryptosystem is to use two prime numbers to calculate the modulo. If, however the modulo is a prime number, there exists different ways to crack the entire cryptosystem.**

Keywords—Cryptography, RSA, Divide and Conquer, Security

I. Introduction

RSA (Rivest-Shamir-Adleman) is a widely used cryptographic algorithm for secure communication and data encryption. Its security relies on the computational difficulty of factoring large composite numbers into their prime factors. However, the effectiveness of RSA can be comprimised if the underlying prime factors are not chosen carefully.

This paper focuses on the analysis and implementation of prime modulo RSA encryption systems, employing Euler's criterion to determine the quadratic residues and non-residues modulo a prime. Specifically, we will delve into the utilization of Euler's Criterion in conjunction with a Divide and Conquer algorithm to enhance the efficiency of prime modulo RSA computations.

The main objective of this paper is twofold. Firstly, we will try to understand the underlying theories used to reverse engineer the RSA cryptosystem. By understanding the theories, we hope to understand why the implementation of prime modulo RSA is a bad one. Secondly, we aim to implement the algorithms used to crack the prime modulo RSA using a divide and conquer algorithm to expand the theories to the n-th power.

The remainder of this paper is organized as follows: Section II provides an introduction to the theories used in the following sections. Section III will introduce the concept of *modular square root* and the various algorithms to compute the value. Section IV aims to explain and implement the algorithms defined before to the n-th power. Finally, in Section V we will discuss why a prime modulo RSA is a bad cryptosystem.

II. Theoritical Framework

A. Divisibility, Factors, and Prime Numbers

If *a* and *b* are integers with $a \neq 0$, we say that *a divides b* if there is an integer *c* such that $b = ac$, or equivalently, if $\frac{b}{a}$ is an integer. When *a divides b*, we say that *a* is a *factor* or *divisor* of *b*, and that *b* is a multiple of *a*. The notation $a \mid b$ denotes that *a* divides *b* and the notation $a \nmid b$ denotes that *a* does not divide *b* [1].

If *a* does not divides *b*, then when *b* is divided by *a*, there is a quotient *q* and a remainder *r*, as shown below.

$$b = qa + r$$

The factors of a number *n* are defined as a set of numbers $f_1, f_2, ..., f_k$, where $1 \leq f_i \leq n$ and $f_i \mid n$; or formally as

$$F = \{f_i : 1 \leq f_i \leq n, f_i \mid n\}$$

An integer *p* is called a *prime* if the only factors of *p* are 1 and *p*. Typically, the sieve of erastosthenes is used to find all primes not exceeding a specified integer.

B. Modular Arithmetic

Modular arithmetic, also known as clock arithmetic, is a branch of arithmetic that deals with the remainder of a division of an integer by a divisor commonly known as a modulo. Modular arithmetic uses the notation

$$x \equiv a \ mod \ p$$

The triple equal sign denotes *congruance*, and for all *a* and *x*, we say *a* is congruent to *x* if and only if there exist an integer *n* where

$$x = a + kn, \ n \in \mathbb{Z}$$

Similar to a regular arithmetics, there exists an inverse modulo a^{-1} of *a* modulo *p*, such that

$$aa^{-1} \equiv 1 \ mod \ p$$

However, it can be proven that not all numbers *a* where $a \neq 0$ has an inverse modulo *p* if *p* is not a prime number. This is due to the fact that an inverse modulo only exist if and only if

$$gcd(a, p) = 1$$

Consequently, it can be proven that for a prime modulo *p*, all numbers $a = \{1, 2, ..., p-1\}$ will have a modulo inverse.

C. Greatest Common Divisor, Euclidean Algorithm, and Linear Congruance

The largest integer that divides both of two integers is called the *greatest common divisor*[1] of these integers. Let *a* and *b* be positive integers, there exists such integer *d*, where $d \mid a$ and $d \mid b$, such that no integer larger than *d* can divide both *a* and *b*. The greatest common divisor of *a* and *b* is denoted by $gcd(a, b)$.

Computing the *gcd* of two integers by searching the largest prime factorizations if inneficient, because it is time consuming to find the prime factorizations. Hence, the *Euclidean Algorithm* is used to efficiently find the *gcd* of two numbers.

$$gcd(a, b) = gcd(b, r)$$

A congruance in the form

$$ax \equiv b \ mod \ m$$

where *m* is a positive integer, *a* and *b* are integers, and *x* is a variable, is called a *linear congruance*. There are two ways to solve the linear congruance, the first method is to find the inverse modulo a^{-1}, such that $aa^{-1} \equiv 1 \ mod \ m$. Then, we can multiply the congruance above to

$$aa^{-1}x \equiv ba^{-1} \ mod \ m$$
$$x \equiv ba^{-1} \ mod \ m$$

The second method involves bruteforcing the value of *x* by rearranging the congruance above to

$$ax = km + b$$
$$x = \frac{km+b}{a}$$

D. System of Linear Congruences and Chinese Remainder Theorem

A system of linear congruances [1] is a system consisting of more than one linear congruance in the form

$$\begin{aligned} x &\equiv a_1 \ mod \ m_1 \\ x &\equiv a_2 \ mod \ m_2 \\ &\cdots \\ x &\equiv a_n \ mod \ m_n \end{aligned}$$

The system above is solvable using the *Chinese Remainder Theorem* [1], that states that when the moduli of problems involving linear congruances are pairwise relatively prime, there is a unique solution of the system modulo the product of the moduli.

Let $m_1, m_2, ..., m_n$ be pairwise relatively prime positive integers greater than one and $a_1, a_2, ..., a_n$ be arbitrary integers. The system

$$\begin{aligned} x &\equiv a_1 \ mod \ m_1 \\ x &\equiv a_2 \ mod \ m_2 \\ &\cdots \\ x &\equiv a_n \ mod \ m_n \end{aligned}$$

has a unique solution modulo $m = m_1 m_2 ... m_n$. (That is, there is a solution x with $0 \leq x < m$, and all other solutions are congruent modulo m to this solution.

Generally, the solution of a linear congruance system is defined as

$$x = a_1 M_1 y_1 + a_2 M_2 y_2 + ... + a_n M_n y_n$$

where

$$M_i = \frac{m}{m_i}$$
$$y_i \equiv M_i \ mod \ m_i$$

E. Group Theory

A set *G* [4] is a group if the following requirements hold:

- **Closure**: for all $a, b \in G : a \cdot b \in G$
- **Associativity**: for all $a, b, c \in G : (ab) \cdot c = a \cdot (b \cdot c)$
- **Identity**: There exists an element $e \in G$ such that $a \cdot e = e \cdot a = a$ for all $a \in G$
- **Inverse**: For all elements $a \in G$, there exists some $b \in G$ such that $b \cdot a = a \cdot b = e$

In the set *G* where $a, n \in \mathbb{Z}$, a^n means $a \cdot a ... \cdot a$ *n* amount of times when $n > 0$, $(a^{-n})^{-1}$ when $n < 0$, and $a^n = e$ when $n = 0$.

Furthermore, if $ab = ba$, then the operation $\cdot$ is commutative and the group is called **abelian**. Usually $\cdot$ is denoted by the operation $+$ and we typically use na instead of a^n.

An example of a group is all integers modulo *n* (a remainder) under modular addition, where $(\mathbb{Z}/n\mathbb{Z}, +)$, where $\mathbb{Z}/n\mathbb{Z} = \{0, 1, ..., n-1\}$. We will now prove that $\mathbb{Z}/n\mathbb{Z}$ is a group.

- $\forall a, b \in \mathbb{Z}/n\mathbb{Z}$, let $c \equiv a + b \ mod \ n$. Since $c < n$, $c \in \mathbb{Z}/n\mathbb{Z}$.
- $\forall a, b \in \mathbb{Z}/n\mathbb{Z}$, modular addition is associative, that is $(a + b) + c \ mod \ n \equiv a + (b + c) \ mod \ n$.

- The identity requirements also hold, i.e. $0 + a \ mod \ n \equiv a + 0 \ mod \ n$.
- $\forall a \in \mathbb{Z}/n\mathbb{Z}$, the modulo inverse of a is $a - n$, since $a + n - a \mod n \equiv n \ mod \ n \equiv 0 \ mod \ n$.

F. Fermat's Little Theorem

Fermat's Little Theorem [3] states that if *p* is a prime and *a* is an integer not divisible by *p*, then

$$a^{p-1} \equiv 1 \ mod \ p$$

Furthermore, for every *a*, we have

$$a^p \equiv a \ mod \ p$$

We can use *Fermat's Little Theorem* to quickly compute the modular inverse of *a*.

$$\begin{gathered} a^{p-1} \equiv 1 \ mod \ p \\ a^{p-1}a^{-1} \equiv a^{-1} \ mod \ p \\ a^{p-2} \equiv a^{-1} \ mod \ p \end{gathered}$$

However, there exist composite numbers *n*, such that the numbers *n* satisfies *Fermat's Little Theorem.* Such numbers are commonly called *pseudoprimes*.

G. RSA Algorithm

The *RSA Algorithm* [6] is an example of asymmetric encryption, also known as public key encryption, where a public key is used to encrypt data and only a secret, private key can be used to decrypt the data.

Commonly used terminologies of the *RSA Algorithm* includes:

1. The message ***m***;
2. The encrypted message ***c***;
3. The modulo ***n***, which is made by multiplying two prime numbers ***p*** and ***q***;
4. The public exponent ***e***;
5. The private exponent ***d***;
6. The totient value $\phi(n)$;
7. The public key pair ***(n, e)***; and
8. The private key pair ***(n, d)***.

To generate the encryption and decryption keys, first compute *n* as the value of two primes *p* and *q*.

$$n = pq$$

These primes should be large, "random" primes. Although the value *n* will be public, the factors *p* and *q* will be effectively hidden from everyone due to the enormous difficulty of factoring *n*. This same concept also hides the way *d* can be derived from *e*.

Then, we calculate the totient $\phi(n)$ to generate the value *d*.

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

Then, we pick the public exponent *e*, most commonly used is the prime 65537. The private exponent *d* is finally computed by finding the modular inverse of *e* modulo $\phi(n)$.

$$ed \equiv 1 \ mod \ \phi(n)$$

To encrypt the message *M* using the public key pair ***(n, e)***, first represent the message into an integer *m* between 0 and *n-1*. Then, encrypt the message by raising it to the *e*-th power modulo *n*. That is, the result (the ciphertext *C*) is the remainder when m^e is divided by *n*.

$$C \equiv M^e \ mod \ n$$

To decrypt the ciphertext, we raise it to the *d*-th power modulo *n*.

$$M \equiv C^d \ mod \ n$$

III. Modular Square Root

A. Modular Square Root

Similar to classical arithmetic, an integer *a* modulo an integer *m* greater than 1 is said to have a **modular square root** if there exists an integer *r* such that

$$r^2 \equiv a \ mod \ m$$

For each integers $r \ mod \ m, (m - r) \ mod \ m$ is also a valid modular square root, since

$$\begin{gathered} (m-r)^2 \equiv a \ mod \ m \\ m^2 - 2rm + r^2 \equiv a \ mod \ m \\ r^2 \equiv a \ mod \ m \end{gathered}$$

However, not all integers r in $\mathbb{Z}/n\mathbb{Z}$ have modular square root. Formally, if there exists an integer $0 < r < p$ such that the congruance above has a solution, then *a* is said to be a quadratic residue [5] modulo *p*. If the congruance above does not have a solution, then *a* is said to be a quadratic non-residue modulo *p*.

B. Euler's Criterion

Euler's Criterion [6] is a formula used to determine whether an integer *a* is a quadratic residue modulo a prime *p*. Precisely, if *p* is an odd prime and *a* is an integer where $p \nmid a$, then we can find whether the quadratic congruance has solutions by calculating

$$a^{\frac{p-1}{2}} \equiv \begin{cases} 1 \ mod \ p & if \ quadratic \ residue \ exists \\ -1 \ mod \ p & if \ quadratic \ residue \ does \ not \ exist \end{cases}$$

C. Lagendre's Symbol

Lagendre's Symbol is a number theoritic function (in the form of $\frac{a}{p}$) which is defined to be equal to -1, 0, or 1; depending on whether *a* is a quadratic residue modulo *p*.

$$(\tfrac{a}{p}) = (a \mid p) \equiv \begin{cases} 0, & \text{if } p \mid a \\ 1, & \text{if } a \text{ is a quadratic residue modulo } p \\ -1, & \text{if } a \text{ is a quadratic nonresidue modulo } p \end{cases}$$

The lagendre's symbol obeys the identity

$$\left(\frac{ab}{p}\right) = \left(\frac{a}{p}\right)\left(\frac{b}{p}\right)$$

Other particular identities include

$$\left(\frac{-1}{p}\right) = (-1)^{\frac{p-1}{2}}$$
$$\left(\frac{2}{p}\right) = (-1)^{\frac{p^2-1}{8}}$$
$$\left(\frac{-3}{p}\right) = \begin{cases} 1, & if p \equiv 1 \ mod \ 6 \\ -1, & if p \equiv 5 \ mod \ 6 \end{cases}$$
$$\left(\frac{5}{p}\right) = \begin{cases} 1, & if p \equiv 1, 9 \ mod \ 10 \\ -1, & if p \equiv 3, 7 \ mod \ 10 \end{cases}$$
$$\left(\frac{q}{p}\right) = \left(\frac{p}{q}\right)(-1)^{(\frac{p-1}{2})(\frac{q-1}{2})}$$

In general, if *p* is an odd prime, then

$$\left(\frac{a}{p}\right) = a^{\frac{(p-1)}{2}} \ mod \ p$$

D. Calculating the Modular Square Root

In this part, we will only consider the case when the modulo *m* is a prime number since the modular square root of coprime modulo RSA are hard to calculate, as we will see in the following parts.

1) Modulus equal to 2

If the modulus *p* is equal to 2, then the value of *r* is congruent to the value of *a*, since the square root of an odd number is an odd number and the square root of an even number is an even number.

$$r^2 \equiv \begin{cases} 1, & \text{if } a \ mod \ 2 \equiv 1 \\ 0, & \text{otherwise} \end{cases}$$

2) Modulus congruent to 3 mod 4

In the case when $p \equiv 3 \ mod \ 4$, $p + 1$ is divisible by 4. We can expand Euler's criterion to get the modular square root if the square root exists.

$$a^{\frac{m-1}{2}} \ mod \ m \equiv 1 \ mod \ m$$

Multiplying both sides with *a*

$$a^{\frac{m+1}{2}} \ mod \ p \equiv a \ mod \ p$$

Let *r* be the modulo square root, we can simplify the congruance

$$r^2 \equiv a \ mod \ p$$

into

$$r^2 \equiv a^{\frac{m+1}{2}} \ mod \ m$$

using the equivalence above. Then, replacing the value *m* with $3i + 4$, we get

$$r^2 \equiv a^{(4i+3)+1} 2 \ mod \ m$$
$$r^2 \equiv a^{2i+2} \ mod \ m$$

Taking the square root of both sides, we get

$$r \equiv \pm a^{i+1}$$

Lastly, substitute the value *m* back into the equation, where $i \equiv \frac{p-3}{4}$.

$$r \equiv \pm a^{\frac{p+1}{4}}$$

Since the value of the modular square root can be directly calculated, the complexity of this algorithm is $O(1)$.

3) Modulus Congruent to 1 mod 4

In the case where the prime *p* has congruance equivalent to $p \equiv 1 \ mod \ 4$, the modular square root is harder to calculate than the other case. There exists many algorithms to calculate the modular square root in this case, but now we will only look at how *Tonelli-Shanks* [7] algorithm calculate the modular square root. The steps are as follows:

- Set $m = \frac{p-1}{2}$. If $a^m = -1$, then *a* is not a square modulo *p* according to Euler's criterion.
- Otherwise, we have $a^m = 1$. Next, set $k = 1$ so that $m = \frac{p-1}{2^k}$.
- While m is even and $a^m = 1$, set $k = k + 1$ and update the value of m.
- If $a^m = 1$, then the square root of a is $a^{\frac{m+1}{2}}$, since m is odd.
- Otherwise, $a^m = -1$ for $m = \frac{p-1}{2^k}$, where $k \geq 2$. Choose an arbitrary non-square $b \in (\mathbb{Z}/p\mathbb{Z}) * Q_P$ by repeatedly picking $b \in \mathbb{Z}/p\mathbb{Z}$ uniformly at random until $\left(\frac{b}{p}\right) = b^{\frac{p-1}{2}} = -1$
- Run the algorithm recursively from the 3rd step with the variable $a' = ab^{2k-1}$. This establishes the invariant of step 3, because

$$1 = a^m \cdot b^{\frac{p-1}{2}} = a^{\frac{p-1}{2^k}} \cdot b^{\frac{p-1}{2}}$$

where the lagendre symbol of each equivalence equals to $-1 - 1 = 1$. Expanding the equation above further, we get

$$a^{\frac{p-1}{2^k}} \cdot (b^{2k-1})^{\frac{p-1}{2^k}}$$
$$(a \cdot b^{2k-1})^{\frac{p-1}{2^k}} = (a \cdot b^{2k-1})^m = a'$$

- Let $\sqrt{a'}$ be the square root of a' obtained from the recursive call. We will compute the value $\sqrt{a}$ such that $(\sqrt{a})^2 = a$. Observe that

$$(\sqrt{a'})^2 = a' = a \cdot b^{2k-1} = (\sqrt{a})^2 \cdot b^{2k-1}$$

- Bringing $(\sqrt{a})^2$ to the left side yields

$$(\sqrt{a})^2 = (\sqrt{a'})^2 \cdot b^{-2^{k-1}} = (\sqrt{a} \cdot b^{-2^{k-2}})^2$$

- Since we use the value $k \geq 2$, it follows that $\sqrt{a'} \cdot b^{-2^{k-2}}$ is a square root of a modulo p.

Now, we will argue why this algorithm always terminates and is correct. Since the value of m is changed throughout the algorithm in a way that ensures $a^m \in \{1, -1\}$. This is the case, because $1, -1$ are the only square roots of 1 modulo p and in every step we divide the exponent m only by 2. The case $a^m = -1$ is reduced to the case $a^m = 1$ but for a different value of a, such that the square root for the original a can be recovered from the new value of a. Lastly, the algorithm will always terminates, because $a^m = 1$ holds for an odd m, or k gets increased by at one, hence m gets divided by a power of two greater than one. This guarantees termination, because iterative division of m by two will yield an odd number whereupon the algorithm will terminate.

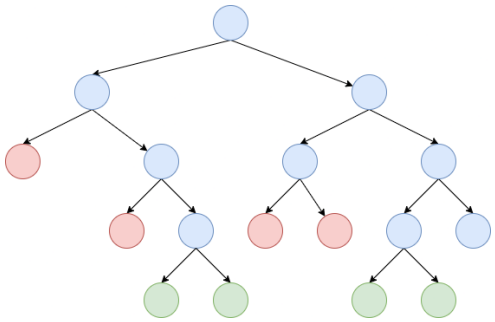
This algorithm is expected to run in $O(log(p)^2)$ time, if we assume that the multiplication modulo p runs in a constant time.

IV. Modular n^th Root

Now, we will discuss the topic to see why RSA does not use a prime modulo in its implementation. Firstly, we need to expand the topic before for the n-th root. Since we only cover the topic of modular square root, we will only cover the topic where $n = 2^k;\ k \in \mathbb{Z}$.

Since n is an integer of power of two, we can recursively compute the value of the n-th root by dividing the value of n by two for each iteration. For each successful calculations (that is, the modular square root exists), we will get two integers, each are valid modular square root of a modulo p.

Since we are finding a valid message m from a ciphertext c, we can use the ***divide and conquer*** algorithm to compute the message *m* quickly. Additively, since not all integers computed from the algorithm are a modular square root on itself, not all nodes will be expanded from the search.



Graph 1. Divide and Conquer Example

To illustrate, the graph above shows one possible cases for the algorithm used. The blue areas indicate that the modular square root exists, but $n \neq 2$ yet, continuing the computation for both integers found. The red areas indicate that the modular square root does not exists, stopping the computation, and the green areas indicate that the modular square root exists and $n = 2$.

At the end of the computation, we just need to linearly check each of the possible values computed before and see which value fits the criteria.

I've provided the code implementing the algorithms above in the following link [8].

V. Prime Modulo RSA Analysis

We have seen above that for any RSA cryptosystem with a prime modulo, it is very easy to crack the entire cryptosystem using the algorithm described above. Calculating the time complexity of the worse case, each "node" requires $O(log(p)^2)$ time to calculate each it's modular square root. For an integer $n = 2^k$, the total worse case complexity of the algorithm used is

$$O(log(n) \cdot log(p)^2)$$

Since there are $log\ n$ amount of checks needed, and each checks requires $log(p)^2$ amount of computation. Finally, the last check of the algorithm needs $O(t)$ amount of linear checks to determine the correct prime chosen.

Acknowledgment

First of all, the author would like to thank God Almighty for giving us the patience and perseverance for completing this work successfully. The author would also like to thank the lecturer of class II Algorithm Strategies, Mrs. Nur Ulfa

Maulidevi of Bandung Institute of Technology. Next, the author would like to thank the head lecturer of Algorithm Strategies, Mr. Rinaldi Munir, who provides the topics taught in class, of which is used in this paper. The author will also take this opportunity to express his sincere thanks and deep gratitude to his friends and seniors who may have helped in one way or another for completing this paper successfully.

REFERENCES

[1] K. H. Rosen, *Discrete mathematics and its applications*. New York, Ny Mcgraw-Hill Education, 2019.

[2] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 26, no. 1, pp. 96–99, Jan. 1983, doi: 10.1145/357980.358017.

[3] "Rings, Fields and Euler's Totient Function - Crypto," *Gitbook.io*, 2022. https://ir0nstone.gitbook.io/crypto/fundamentals/rings-fields-and-eulers-totient-function (accessed May 21, 2023).

[4] "Groups," Groups - CryptoBook, https://cryptohack.gitbook.io/cryptobook/abstract-algebra/groups (accessed May 22, 2023).

[5] E. W. Weisstein, "Quadratic Residue," *mathworld.wolfram.com*. https://mathworld.wolfram.com/QuadraticResidue.html

[6] "Euler's Criterion - Mathonline," *mathonline.wikidot.com*. http://mathonline.wikidot.com/euler-s-criterion (accessed May 22, 2023).

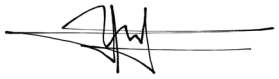
[7] zerobone, "Computing square roots modulo a prime with the Tonelli-Shanks algorithm," *ZeroBone*, Sep. 15, 2022. https://zerobone.net/blog/math/tonelli-shanks/ (accessed May 22, 2023).

[8] F. Huang, *Github*, May 22, 2023. https://github.com/frankiehuangg/prime-modulo-RSA (accessed May 22, 2023).

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 22 Mei 2022



Frankie Huang, 13521092