Implementation of RSA Cryptography and Analysis of the Impact of Prime Number Size on Encryption Performance

Implementasi Kriptografi RSA dan Analisis Dampak Ukuran Bilangan Prima pada Kinerja Enkripsi

Ribka Kaylena Sanjaya - 13525045
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ribkakaylenas@gmail.com , 13525045@std.stei.itb.ac.id

***Abstract*—The application of cryptography is a widely adopted way to minimize the attacks on data transmission and storage. This study implements the RSA cryptographic algorithm and analyzes the impact of prime number size on encryption performance. RSA (Rivest-Shamir-Adleman) is a public-key cryptographic algorithm that relies on the mathematical properties of prime numbers and modular arithmetic to provide robust security in digital communication. In this study, RSA is implemented using various prime number sizes and a series of experiments is conducted to measure encryption and decryption execution times for messages of different lengths. These findings provide a deeper understanding of the practical trade-offs involved in RSA deployment and highlight the importance of selecting suitable key sizes when balancing security and performance requirements in different applications.**

***Keywords*— Cryptography, RSA, Algorithm Complexity, Encryption, Prime Number**

I. Introduction

Throughout history, people have communicated through various means, ranging from handwritten paper letters to modern electronic mail. As communication technologies evolved, ensuring the security of transmitted information became increasingly important. Cryptography is a core in protecting sensitive data during transmission. With cryptography, encrypted letters will be safe to be sent in any form of media or platforms. At first, cryptography was developed using the symmetric key algorithm. One of the earliest and most popular cryptography symmetric key algorithms is the Caesar Cipher algorithm.

As cryptographic research progressed, an asymmetric key algorithm was introduced to the network world to address several limitations of symmetric key systems, particularly the problem of secure key distribution. Asymmetric key cryptography utilizes a pair of mathematically linked pairs of keys: a public key for encryption and a private key for decryption. RSA (Rivest-Shamir-Adleman) is one of the most widely used public key cryptographic algorithms based on this principle. The linked pair of keys relation is fundamentally based on concepts from number theory, including prime numbers, modular arithmetic, Euler's totient function, and modular inverses. It shows that these mathematical concepts are vital to the security of RSA cryptography.

Since prime numbers play a big role in RSA, their size directly affects both the security and computational performance of the algorithm. Larger prime numbers generally result in stronger security because they produce larger key sizes that are harder to factor. However larger keys also require more computational resources during both encryption and decryption processes. Consequently a trade-off exists between security and performance.

Therefore, this study aims to implement the RSA algorithm and investigate how different prime number sizes affect encryption performance. To cover a wide range of scenarios, prime numbers with sizes ranging from 16-bit to 256-bit are generated during the key generation process. A series of experiments is then conducted using messages of different lengths to evaluate the behavior of RSA under varying conditions. Encryption execution time serves as the primary performance metric, while decryption time is also recorded to provide a more comprehensive evaluation of the implementation. In addition, the study examines the trade-off between security and computational efficiency associated with larger key sizes and compares the experimental results with

theoretical expectations derived from number theory and algorithm complexity.

II. Theoretical Basis

A. Cryptography

Cryptography is the practice of developing and applying algorithms to protect information by converting it into a secure form that can only be accessed by the authorized parties. Derived from the Greek word “kryptos”, meaning hidden, cryptography translates to “hidden writing”. In practice, cryptography is mainly used to transform messages into a ‘secret’ format (known as ciphertext) that can only be decrypted into a readable format (known as plain text) by the authorized intended recipient by using a specific secret key.

Encryption is the process to transform the messages into a ciphertext. Conversely, decryption is the process of converting back the ciphertext back into its real meaning, plaintext, so that the original information can be understood by the intended.

B. Number Theory

Number theory is the branch of pure mathematics devoted to the study of integers and integer-valued functions.

1.) Prime Numbers

Prime number is a positive integer greater than q that has exactly two positive divisors: 1 and itself. Since prime numbers must be greater than 1, the sequence of prime numbers begins with 2, followed by 3, 5, 7, 11, and so on. All prime numbers are odd except for 2, which is the only even prime number.

According to the Fundamental Theorem of Arithmetic, every positive integer greater than or equal to 2 can be expressed as a product of one or more prime numbers. This property makes prime numbers a fundamental concept in number theory and an essential component of many cryptographic systems, including RSA.

The Fermat’s Little Theorem shows that, for a prime number (denoted by p) and a is an integer such that gcd(a,p) = 1, then:

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

2.) Modular Arithmetic

Modular arithmetic is a method of arithmetic that solves problems involving integers. The idea of modular arithmetic is to look at the remainder division of two numbers.

For example, suppose (k) and m are integers with (m) > 0, then:

k mod m

gives the remainder of k divided by m.

3.) Euler’s Totient Function

For an integer n, the value of the Euler’s Totient Function denoted as φ (n). The function φ (n) represents the count of relatively prime positive integers less than or equal to n.

4.) Modular Inverses

The modular inverse of an integer (a) modulo (m) is an integer x such that

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

In other words, (x) is the multiplicative inverse of (a) under modulo (m). A modular inverse exists if and only if aaa and mmm are relatively prime, that is,

gcd(a,m)=1.\gcd(a,m)=1.gcd(a,m)=1.

C. RSA

RSA is one of the most widely used asymmetric cryptography algorithms developed by Ron Rivest, Adi Shamir, and Leonard Adleman at MIT. RSA utilizes a pair of keys: a public key for encryption and a private key for decryption. The security of RSA is based on several concepts from number theory, including prime numbers, modular arithmetic, Euler’s totient function, and modular inverses. The RSA algorithm consists of three main stages: public-private key generation, encryption, and decryption.

1.) Public-private key generation

The encryption and decryption in RSA require a pair of keys: a public key and a private key. Public key is allowed to be public, while private key must remain confidential.

The first step is selecting two large prime numbers, denoted by (p) and (q). These prime numbers are then multiplied to generate modulus (n):

$$n = p \cdot q$$

After the value of n is calculated, Euler’s totient function φ (n) is determined. Since (p) and (q) are prime numbers, φ (n) can be calculated using:

$$\varphi(n) = \varphi(p,q) = \varphi(p) \cdot \varphi(q)$$
$$\varphi(n) = (p-1).(q-1)$$

Euler’s totient function represents the number of positive integers less than (n) that are relatively prime to (n). The value of φ (n) plays an important role in determining both the public and private keys.

To generate the public key, an integer (e) is selected such that:

$$1 < e < \varphi(n)$$

and

$$\gcd(e, \varphi(n)) = 1$$

This expression ensures that (e) and $\varphi(n)$ are coprime. The value (e) serves as the public exponent used during the encryption process.

The next step is to get the private key exponent (d) calculated. The value of (d) is defined as the modular multiplicative inverse of (e) modulo $\varphi(n)$, satisfying the following equation:

$$d \cdot e = 1 \bmod \varphi(n)$$

The value of (d) is used during the decryption process and must be kept secret. Once (e) and (d) have been determined, the RSA key pair can be constructed. The public key consists of the pair (n,e), while the private key consists of the pair (n,d). Both keys share the same modulus (n), but use different exponents for encryption and decryption.

2.) Encryption

The encryption process begins after the public key (n,e) has been generated. In RSA, the encryption process transforms plaintext into ciphertext using the public exponent (e) and modulus (n).

$$C = M^e \bmod n$$

C is ciphertext, the encrypted letter or information. M represents the plaintext message and (n,e) is the public key.

Since RSA operates on numerical values, non-numeric data such as letters and symbols must be first converted into numbers before encryption. This conversion is performed using an encoding scheme.

The commonly used encoding scheme is ASCII, which assigns a unique numerical value to each character. After the plaintext has been converted into numerical form, the RSA encryption process can be done to produce the ciphertext.

3.) Decryption

Decryption is the process of recovering the original plaintext from the ciphertext. Unlike encryption, decryption requires the private key (n,d) which remains confidential. The RSA decryption operation is defined as follows:

$$M = C^d \bmod n$$

Where C is the ciphertext, M is the recovered plaintext, and (n,d) is the private key.

After the decryption process is completed, the resulting numerical values are converted back into their original character presentation using the same encoding scheme employed during encryption.

D. Algorithm Complexity

Algorithm complexity is a measure used to evaluate the efficiency of an algorithm in terms of the resources during the execution of the code. Two most common resources analyzed are execution time and memory usage (space). Time complexity describes how the execution time of an algorithm grows as the size of the input increases. It is commonly represented by the function T(n), where n denotes the input size.

In algorithm analysis, the running time of an algorithm is commonly represented by a function T(n), where n denotes the size of the input. As the input size becomes large, the exact running time becomes less important than the rate at which the running time grows. For this exact reason, asymptotic notation is used to describe the growth of an algorithm's complexity.

There are many forms of asymptotic, such as Big-O, Omega, and Theta. Big-O represents the upper bound of time complexity, Big-Omega represents the lower bound, and Big-Theta represents the tight bound of the time complexity. The most widely used asymptotic notation is Big-O notation. Big-O also represents the worst-case running time. For example:

$$T(n) = 2n^2 + 6n + 1 = O(n^2) = O(n^3)$$

As n becomes large, the quadratic term $2n^2$ dominates the growth of the function while the lower-order terms become more insignificant. Therefore, the time complexity is expressed as:

$$T(n) = O(n^2)$$

E. Prime Number Size in RSA

In the context of RSA cryptography, algorithm complexity is important because the encryption and decryption processes involve modular arithmetic operations on large integers. As the size of the prime numbers used in key generation increases, the size of the modulus also increases, resulting in more computationally expensive arithmetic operations. Consequently, larger key sizes are associated with increased execution times.

III. METHODOLOGY

This study adopts an experimental methodology to evaluate the performance of the RSA cryptographic algorithm under different key sizes. The research focuses on measuring the execution time required for encryption and decryption processes when varying the size of the prime numbers used during key generation. The implementation was developed in Python, and a series of controlled experiments were conducted to collect performance data. The obtained results were then

analyzed to identify the relationship between key size, computational efficiency, and security considerations.

A. Research Design

This study employs an experimental approach to investigate the impact of prime number size on the performance of the RSA cryptographic algorithm. The primary objective is to analyze how different prime sizes affect encryption and decryption execution times while also examining the trade-off between computational efficiency and security strength.

In addition, the study investigates whether message length significantly influences RSA performance compared to prime number size. To accomplish this, two different plaintext lengths are used throughout the experiments.

The independent variable is the size of the prime numbers used during RSA key generation, while the dependent variables are the encryption and decryption execution times. Message length is included as a secondary experimental factor.

B. Software and Hardware Environment

The RSA algorithm was implemented using Python language. Prime numbers were generated using the SymPy library, while modular arithmetic operations were performed using Python's built in functions.The implementation was developed and executed using Visual Studio Code integrated with the Windows Subsystem for Linux (WSL).

Execution times were measured using Python's time.perf_counter() function, which provides high-resolution timing suitable for performance evaluation.

C. RSA Implementation

1.) Generate two random prime numbers p and q according to the specified bit size

```python
from sympy import randprime
import math
import time

def generate_prime(bits):
    lower = 2 ** (bits - 1)
    upper = 2 ** bits - 1
    return randprime(lower, upper)
```

Documentation 3.1 Code Snippet of generate_prime() Function

2.) Generate key pair

The following function is used to compute the modulus (n), compute the Euler's totient function (φ(n)). Next, φ(n) and n are used to get the public exponent (e). Then the function returns the pair of keys, p, and q.

```python
# RSA Key generation
def generateKeys(bit):
    p = generate_prime(bit)
    q = generate_prime(bit)

    while p == q:
        q = generate_prime(bit)

    n = p * q
    phi = (p-1) * (q-1)
    e = 65537

    while math.gcd(e, phi) != 1:
        e += 2
    d = pow(e, -1, phi)
    # d.e = 1 mod phi

    return (n, e), (n,d), p, q
```

Documentation 3.2 Code Snippet of generateKeys() Function

3.) Encryption process

```python
def encrypt(message, public_key):
    n, e = public_key
    ciphertext = []
    #array kosong untuk diisi hasil encryption
    for char in message:
        m = ord(char)
        c = pow(m, e, n)
        # C = M^e mod n
        ciphertext.append(c)
    return ciphertext
```

Documentation 3.3 Code Snippet of encrypt() Function

4.) Decryption process

```python
def decrypt(ciphertext, priv_key):
    #ciphertetx stored as a list of integers
    n, d = priv_key
    plaintext = ""
    for c in ciphertext:
        m = pow(c, d, n)
        # M = C^d mod n
        plaintext += chr(m)
    return plaintext
```

Documentation 3.4 Code Snippet of decrypt() Function

5.) Calculate the run time

```python
def run_exp(bits, message):
    public_key, priv_key, p, q = generateKeys(bits)

    start = time.perf_counter()
    ciphertext = encrypt(message, public_key)
    encryption_time = time.perf_counter() - start

    start = time.perf_counter()
    plain_message = decrypt(ciphertext, priv_key)
    decryption_time = time.perf_counter()- start
```

```
    return {
        "bits" : bits,
        "p": p,
        "q": q,
        "encryption_time": encryption_time,
        "decryption_time": decryption_time,
        "ciphertext": ciphertext,
        "decrypted_message": plain_message
    }
```

Documentation 3.5 Code Snippet of run_exp() Function

6.) The main program

```
#main program

message = input("Ketik pesan di sini: ")

sizes = [16, 32, 64, 128, 256]

print("RSA Performance Experiment")

print("-" * 60)


results = []


for bits in sizes:

    total_enc = 0

    total_dec = 0

    for _ in range(20):

          result = run_exp(bits, message)

          total_enc += result["encryption_time"]

          total_dec += result["decryption_time"]

     results.append({

        "bits": bits,

        "avg_enc": total_enc / 20,

        "avg_dec": total_dec / 20

    })

for r in results:

    print(r)
```

Documentation 3.6 Code Snippet of the Main Program

D. Experimental Procedure

Two different plaintext messages were used during the experiments.

- Experiment A (Short message)

  "HELLO"

- Experiment B (Long Message)

  "HELLO I AM ANALYZING THE PERFORMANCE OF RSA CRYPTOGRAPHY WITH DIFFERENT PRIME NUMBERS FOR MY DISCRETE MATHEMATICS PROJECT"

Five prime number sizes were evaluated:

• 16-bit
• 32-bit
• 64-bit
• 128-bit
• 256-bit

For each prime size and message length, the following procedure was performed:

1. Generate a new RSA key pair.
2. Encrypt the plaintext.
3. Measure encryption execution time.
4. Decrypt the ciphertext.
5. Measure decryption execution time.
6. Repeat the experiment 20 times.
7. Calculate the average encryption time and average decryption time.

The average execution time is computed as:

Average Time = (Σ ti) / n

where ti denotes the execution time of trial i and n denotes the total number of trials.

IV. Results and Analysis

After the implementation and experiments were executed, analysis then can be done.

A. Experimental Results

Experiment A: Short Message ("HELLO")

Prime Size (bits)	Avg Encryption Time (s)
16	3.370459999132436e-05
32	3.145484997730818e-05
64	4.9679750054565376e-05
128	7.096950002960511e-05
256	0.0001306390001445834

Table 4.1 Avg Encryption Time of Exp A

Prime SIze (bits)	Avg Decryption Time (s)
16	9.797430006983632e-05
32	0.0001983235499665170 2
64	0.0005913303499255562
128	0.0016147127001204353
256	0.010256832099958046

Table 4.2 Avg Decryption Time of Exp A

Experiment B: Long Message

Message:

"HELLO I AM ANALYZING THE PERFORMANCE OF RSA CRYPTOGRAPHY WITH DIFFERENT PRIME NUMBERS FOR MY DISCRETE MATHEMATICS PROJECT"

Prime Size (bits)	Avg Encryption Time (s)
16	0.0005830319999859057
32	0.0005909271498239832
64	0.0008339526999407099
128	0.0011269988498952444
256	0.002812328149911991

Table 4.3 Avg Encryption Time of Exp B

Prime Size (bits)	Avg Decryption Time (s)
16	0.0016347405000942672
32	0.003655947150036809
64	0.010545915549982965
128	0.0306868738999583
256	0.1646004503000313

Table 4.4 Avg Decryption Time of Exp B

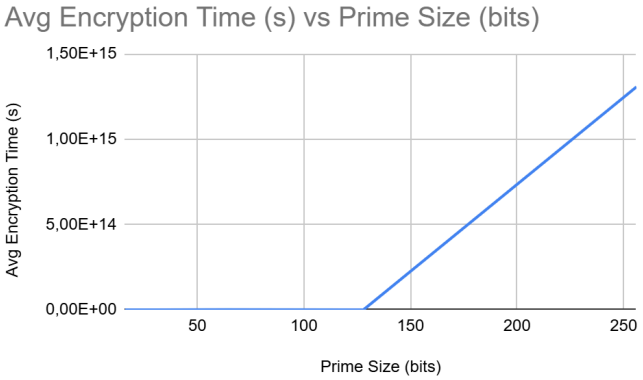
B. Impact of Prime Number Size on Encryption Performance

The experimental results indicate that prime number size affects RSA encryption performance. For both the short-message and long-message experiments, the average encryption time generally increased as larger prime numbers were used.

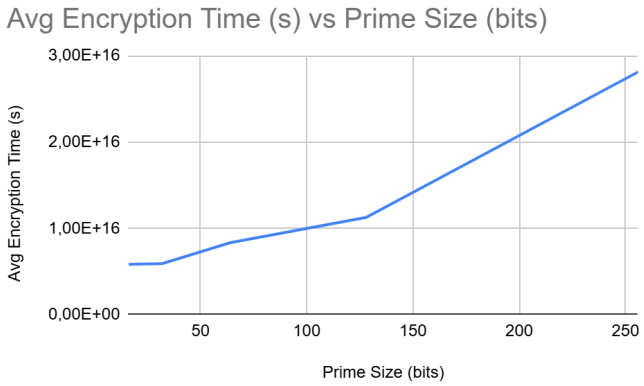
In Experiment A, the average encryption time increased from approximately 0.000034 seconds for 16-bit primes to 0.000131 seconds for 256-bit primes. Similarly, in Experiment B, the average encryption time increased from approximately 0.000583 seconds to 0.002812 seconds.

Although the increase is not perfectly linear, the overall trend shows that larger prime numbers require more computational effort during encryption. This occurs because larger primes produce larger RSA moduli, resulting in more expensive modular arithmetic operations.

Therefore, the experimental results suggest that increasing prime number size reduces encryption efficiency by increasing encryption execution time.



Graph 4.1 Line Graph between Bit Size and Encryption Time(A)



Graph 4.2 Line Graph between Bit Size and Encryption Time(B)

C. *Effect of Message Length on Encryption Performance*

A comparison between the two experiments shows that message length also influences encryption performance. The long message consistently required more encryption time than the short message for every prime size tested.

For example, using 256-bit primes, the average encryption time increased from approximately 0.000131 seconds for the "HELLO" message to 0.002812 seconds for the long message.

This behavior is expected because the implementation encrypts each character individually. As the number of characters increases, more encryption operations must be performed.

However, despite the effect of message length, the overall trend remains unchanged. In both experiments, larger prime sizes resulted in higher encryption times. This suggests that prime number size remains a significant factor affecting encryption performance regardless of message length.

D. *Additional Observation: Decryption Performance*

Although the primary focus of this study is encryption performance, decryption times were also recorded during the experiments.

The results show that decryption time increases much more rapidly than encryption time as prime size increases. This behavior is consistent with the RSA algorithm, where decryption typically involves a larger private exponent than the public exponent used during encryption.

While decryption performance is outside the main scope of this study, the observed results provide additional evidence that larger RSA key sizes introduce greater computational costs.

E. *Theoretical vs Experimental Analysis*

The theoretical foundation of RSA suggests that larger prime numbers lead to larger values of n and φ(n), resulting in more expensive modular exponentiation operations during encryption and decryption.

The experimental results support this theoretical expectation. In both experiments, execution times generally increased as prime size increased. The effect was also shown during decryption, where the increased execution time became substantially larger for higher prime sizes.

Furthermore, the experiments confirmed that message length affects overall execution time because more characters must be processed. Nevertheless, the relationship between prime size and execution time remained consistent across both short and long messages.

Therefore, the experimental findings validate the theoretical prediction that increasing prime number size improves security while simultaneously increasing computational cost.

V. Conclusion

The experimental results showed that encryption time generally increased as larger prime numbers were used. In addition, longer messages required more processing time than shorter messages because more characters had to be encrypted. However, regardless of message length, larger prime sizes consistently resulted in higher encryption times, indicating that prime number size is a significant factor affecting RSA encryption performance.

The results also support the theoretical principles of RSA cryptography, where larger prime numbers produce larger RSA moduli and increase the computational cost of modular arithmetic operations. Although larger prime sizes reduce encryption efficiency, they provide stronger cryptographic security. Therefore, selecting an appropriate prime size requires balancing performance requirements and security needs according to the intended application.

Acknowledgment

The author would like to express gratitude to God Almighty for His blessings and guidance throughout the completion of this paper. The author also sincerely thanks Prof. Dr. Ir. Rinaldi, M.T, lecturer of the Discrete Mathematics course, for his guidance and for providing the knowledge that served as the foundation of this work.

This assignment has provided valuable insights into Discrete Mathematics and valuable experience in scientific writing and research.

References

[1] A. B. Nugroho, I. M. Putri, and, A. A. S. Myriano "Implementation of RSA Cryptography for Web-Based Scholarship Application Data Security," *International Journal of Engineering Computing Advanced Research (IJECAR)*, Vol. 2, No. 1, October 2025, pp. 40~47, Oct 2025

[2] A. ginting, R. Isnanto, I. Windasari "Implementasi Algoritma Kriptografi RSA untuk Enkripsi dan Dekripsi Email" *Jurnal Teknologi dan Sistem Komputer*, Vol.3, No.2, April 2015 (e-ISSN: 2338-0403)

[3] https://www.ibm.com/think/topics/cryptography#17494626

[4] https://www.geeksforgeeks.org/dsa/eulers-totient-function/

[5] https://sagi.io/crypto-classics-wieners-rsa-attack/, diakses pada 14 Juni 2026.

[6]https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/15-Teori-Bilangan-Bagian1-2026.pdf

[7]https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/16-Teori-Bilangan-Bagian2-2026.pdf

[8]https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/17-Teori-Bilangan-Bagian3-2026.pdf

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, 19 Juni 2025

Ribka Kaylena Sanjaya, 13525045