TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

**Logo, company name

Description automatically generated**

**DISCRETE STRUCTURES ESSAY**

**Implementing and Analyzing the RSA Cryptosystem using Modular Arithmetic through Programming**

*Người hướng dẫn*: **Ths NGUYỄN QUỐC BÌNH**

*Người thực hiện*: **TRẦN HỮU NHÂN – 521H0507**

Lớp **: 21H50302**

Khoá  **: 25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

****

**DISCRETE STRUCTURES ESSAY**

**Implementing and Analyzing the RSA Cryptosystem using Modular Arithmetic through Programming**

Người hướng dẫn: **Ths NGUYỄN QUỐC BÌNH**

Người thực hiện: **TRẦN HỮU NHÂN**

Lớp **: 21H50302**

Khoá  **: 25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

LỜI CẢM ƠN

I would like to extend my heartfelt appreciation to Dr. Nguyen Quoc Binh for his invaluable guidance and support throughout the preparation and completion of this report. Dr. Binh's expertise, professionalism, and encouragement have helped me to gain a deep understanding of Discrete Structures and develop my analytical and problem-solving skills. Thank you, Dr. Nguyen Quoc Binh, for your unwavering commitment to your students and your dedication to teaching and research.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH

TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của ThS Nguyễn Quốc Bình. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong bài luận còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 10 tháng 4 năm 2023*

*Tác giả*

*Text, letter

Description automatically generated (ký tên và ghi rõ họ tên)*

*Trần Hữu Nhân*

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

TÓM TẮT

The extended Euclidean algorithm and the RSA encryption algorithm are two important cryptographic tools for secure communication. The extended Euclidean algorithm is used to find the greatest common divisor between two numbers and their respective coefficients. It is a critical component in RSA key generation, which is based on the difficulty of factoring large integers. The RSA algorithm provides secure communication by using public-key cryptography, where the sender and receiver use different keys for encryption and decryption. RSA has four essential requirements for secure communication: confidentiality, integrity, authentication, and non-repudiation. To ensure its security, RSA requires large keys, careful implementation, and regular key updates. Additionally, using block cipher algorithms like AES to encrypt data and then using RSA to encrypt the AES key can further enhance RSA's security. In this essay I research and implement these algorithms in programming and have overview and utilizing the extended Euclidean algorithm and the RSA encryption algorithm can provide a powerful tool for secure communication in various real-world applications. I also give me comment and recommendation about rsa.

TABLE OF CONTENTS

LỜI CẢM ƠN i

ĐỒ ÁN ĐƯỢC HOÀN THÀNH ii

TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG ii

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii

TÓM TẮT iv

CHAPTER 1: FINDING AN INVERSE MODULO N 1

1.1 Theorical research 1

1.1.1 Inverse modulo 1

1.1.2 The Extended Euclidean algorithm 1

1.2 Implement extended Euclidean algorithm 2

1.3 Test 3

CHAPTER 2: RSA CRYPTOSYSTEM 5

2.1 Theory research 5

2.2 Implementation 6

2.3 Test 7

2.4 Analysis and discussion 8

2.4.1 Application of RSA 8

2.4.2 Security Analysis of RSA 9

2.4.3 Pros and Cons 10

2.4.4 Comment 12

2.5 Recommendations 13

REFERENCES 14

CHAPTER 1: FINDING AN INVERSE MODULO N

1.1 Theorical research

1.1.1 Inverse modulo

In modular arithmetic, the inverse modulo is a crucial concept that allows us to perform various mathematical operations efficiently. Given a number a and a modulo n, the inverse modulo of a modulo n is another number x that satisfies the equation ax ≡ 1 (mod n). In other words, x is the number that we can multiply a by to get 1 modulo n.

The inverse modulo only exists if a and n are coprime, that is, if their greatest common divisor is 1. If a and n are not coprime, then there is no inverse modulo. The inverse modulo is an important concept in number theory and has various applications in cryptography, computer science, and other fields.

Finding the inverse modulo n using the extended Euclidean algorithm is an important concept in number theory that has various applications in cryptography, computer science, and other fields. In this context, the extended Euclidean algorithm is used to calculate the greatest common divisor (GCD) of two numbers a and b, as well as to find two integers x and y such that ax + by = GCD(a, b).

1.1.2 The Extended Euclidean algorithm

To find the inverse modulo n using the extended Euclidean algorithm, we need to find two integers x and y such that ax + by = 1, where a is the number whose inverse we are trying to find and b is the modulo. Once we have found x, it will be the inverse of a modulo b.

For example, let's say we want to find the inverse of 17 modulo 43. We can use the extended Euclidean algorithm to find x and y such that 17x + 43y = 1. Starting with a = 43 and b = 17, we can use the following steps:

Divide 43 by 17 to get a quotient of 2 and a remainder of 9.

This means 43 = 2\*17+9.

Divide 17 by 9 to get a quotient of 1 and a remainder of 8. This means 17 = 1(9)+8.

Divide 9 by 8 to get a quotient of 1 and a remainder of 1. This means 9 = 1(8) + 1.

Now we can work backwards to express 1 as a linear combination of 17 and 43:

1 = 9 - 1(8) = 9 - 1(17 - 1(9)) = 2(9) - 1(17) = 2(43 - 2(17)) - 1(17) = 2(43) - 5(17)

So x = 2 and y = -5, which means the inverse of 17 modulo 43 is 2.

1.2 Implement extended Euclidean algorithm

To implement the extended Euclidean algorithm in Python, we can define a function that takes two integers a and b as arguments and returns the greatest common divisor (GCD) of a and b, as well as x and y such that ax + by = GCD(a, b). We can then define another function that takes two integers a and n as arguments and uses the extended Euclidean algorithm to find the inverse of a modulo.

Here is the Python code to implement the extended Euclidean algorithm to find the inverse modulo:

def extended\_euclid(a, b):

    if a == 0:

        return (b, 0, 1)

    else:

        gcd, x, y = extended\_euclid(b % a, a)

        return (gcd, y - (b // a) \* x, x)

def inverse\_modulo(a, b):

    gcd, x, y = extended\_euclid(a, b)

    if gcd != 1:

        return "{} does not have a modular inverse modulo {}".format(a,b)

    else:

        return "inverse\_modulo({}, {}) is : {}".format(a,b, x % b)

**Explain:**

The function **'extended\_euclid(a, b)**' implements the extended Euclidean algorithm to calculate the greatest common divisor (GCD) of two numbers 'a' and 'b', as well as to find two integers 'x' and 'y' such that ax + by = GCD(a, b).

Then, the function **'inverse\_modulo(a, b)**' uses the **'extended\_euclid()**' function to find the GCD and two integers 'x' and 'y' such that ax + by = GCD(a, b). If the GCD is not equal to 1, it means that 'a' does not have a modular inverse modulo 'b'. Otherwise, the function returns the value of 'x' modulo 'b', which is the inverse modulo of 'a' modulo b.

1.3 Test

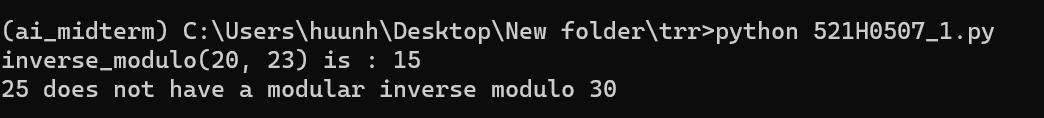
**Here are two test case:**

# test

print(inverse\_modulo(20, 23))

print(inverse\_modulo(25, 30))

**Result:**



**Explain:**

**Example 1: Find inverse\_modulo(20, 23)**

First, I find the greatest common divisor (GCD) of 20 and 23 using the Euclidean algorithm:

23 = 1 \* 20 + 3

20 = 6 \* 3 + 2

3 = 1 \* 2 + 1

The last non-zero remainder is 1, so the GCD(20, 23) = 1. Since the GCD is 1, we know that 20 has a modular inverse modulo 23.

Next, we work backwards to express the GCD as a linear combination of 20 and 23:

1 = 3 - 1 \* 2

1 = 3 - 1 \* (20 - 6 \* 3)

1 = -1 \* 20 + 7 \* 3

1 = -1 \* 20 + 7 \* (23 - 1 \* 20)

1 = -8 \* 20 + 7 \* 23

From this, we can see that -8 is a solution to the equation 20x ≡ 1 (mod 23). To find the smallest positive solution, we can add 23 to -8 until we get a positive number: -8 + 23 = 15. So, 15 is the modular inverse of 20 modulo 23.

Therefore, **inverse\_modulo(20, 23) = 15**

**Example 2: Find inverse\_modulo(25, 30)**

First, we need to find the greatest common divisor (gcd) of 25 and 30 using the Euclidean algorithm:

30 = 1 \* 25 + 5

25 = 5 \* 5 + 0

The last non-zero remainder is 5, so gcd(25,30) = 5. Since the gcd is not equal to 1, we know that 25 does not have a modular inverse modulo 30.

Therefore, inverse\_modulo(25,30) does not exist.

CHAPTER 2: RSA CRYPTOSYSTEM

2.1 Theory research

**RSA** is a public-key cryptosystem that is widely used for secure data transmission. It was developed by **Ron Rivest**, **Adi Shamir**, and **Leonard Adleman** in 1977, and is named after their initials. The security of RSA is based on the difficulty of factoring large integers into their prime factors, which is currently believed to be an intractable problem for classical computers.

The following mathematical ideas form the foundation of the RSA cryptosystem:

**Generation of keys** To begin, we randomly choose two large prime numbers of the same length: p, q, n =p\*q, t=(p-1)\*(1-1). The next number we choose is e, which has a range greater than 0 and smaller than t. We often utilize the value of e as (2\*k+1) for numbers like 3, 17, 19, 65537, and e must satisfy the requirement that d\*e%t == 1 to determine the value of d. The result is that we have the four numbers n, t, e, and d; e is the encryption key, d is the decryption key, and n and d are open to the public while p, q, and e are kept secret from anybody.

Even though the transformations involved in encryption and decryption are the opposite, the public key system treats data as a number that performs mathematical operations, while as the private key system treats it as a bit. The cipher text cannot be decoded using easy techniques since what is said can be true from one side yet be very difficult from the other.

**Encryption algorithm:** Given a known plaintext x (x<n), divide x into character blocks. The length of each plaintext block xi must satisfy 0<xi<k (where k is the length of n). Then calculate the ciphertext C = x^e (mod n).

**Decryption algorithm:** Given a known ciphertext c and the private key (n, d), the plaintext x can be calculated as x = c^d (mod n).

**Security of the RSA algorithm:** The RSA algorithm’s system structure is based on number theory. It is one of the most secure systems among key systems. The security of the RSA algorithm relies on the difficulty of factoring large numbers. To break the encryption, one would need to factorize a large number, meaning it is difficult to obtain the private key through factoring from a public key.

2.2 Implementation

We can either implement manually or use Python libraries to support cryptography. In this case, we have alots options: the Crypto library (**cryptography**, **cryptodome**) and the **rsa** module. For this tutorial, I will be using the rsa library because I just need rsa algorithm and it result return big int number while cryptography or cryptodome return base64.

To run this code you need install rsa library (python version 3.10.6):

In your terminal, type command: pip install rsa

import rsa # import library

# Generation public key and private key

(public\_key, private\_key) = rsa.newkeys(2048)

#print public key

print("public key: {}\n".format(public\_key))

print("private key: {}\n".format(private\_key))

# plain text

message = "I have waited for this opportunity for more than half a century, to repeat to you once again my vow of eternal fidelity and everlasting love."

#encrypt

encrypted\_message = rsa.encrypt(message.encode(), public\_key)

print("encrypt text: {}\n".format(encrypted\_message))

#decrypt

decrypted\_message = rsa.decrypt(encrypted\_message, private\_key).decode()

# print decrypted message

print("decrypt message: {}".format(decrypted\_message))

**Explain:**

**public\_key:** The variable public\_key stores information related to the public key of an RSA key pair, including:

**n:** The modulus, which is the product of two large prime numbers (p and q) used to generate the public and private keys.

**e**: The public exponent, which is a small prime number that is part of the public key.

These values are used to encrypt messages using the RSA algorithm.

**private\_key:** The variable private\_key stores information related to the private key of an RSA key pair, including:

**n:** The modulus, which is the product of two large prime numbers (p and q) used to generate the public and private keys.

**e:** The public exponent, which is a small prime number that is part of the public key.

**d:** The private exponent, which is a number derived from the public exponent and the prime factors of the modulus and is used to decrypt messages.

**p:** The first large prime factor of the modulus, used to generate the private key.

**q:** The second large prime factor of the modulus, used to generate the private key.

2.3 Test

# length of key 2048 bit

(public\_key, private\_key) = rsa.newkeys(2048)

# plain text

message = "I have waited for this opportunity for more than half a century, to repeat to you once again my vow of eternal fidelity and everlasting love."

#encrypt

encrypted\_message = rsa.encrypt(message.encode(), public\_key)

print("encrypt text: {}\n".format(encrypted\_message))

#decrypt

decrypted\_message = rsa.decrypt(encrypted\_message, private\_key).decode()

# print decrypted message

print("decrypt message: {}".format(decrypted\_message))

print('plain text = decrypt text: {}'.format(message == decrypted\_message))

**Result:**

**Text

Description automatically generated with medium confidence**

The console show that decrypt message equal original message. The public key and private key store value I mention above. The last line I compare plain text and decrypted text, it returns true meaning successful decrypt.

2.4 Analysis and discussion

2.4.1 Application of RSA

RSA encryption is often used in combination with other encryption schemes and for digital signatures that can verify the authenticity and integrity of a message. It is usually not used to encrypt entire messages or files because it is less efficient and more resource-intensive than symmetric key encryption.

To make things more efficient, a file is usually encrypted using a symmetric key algorithm. Then the symmetric key is encrypted using RSA encryption. In this process, only the person with access to the RSA private key can decrypt the symmetric key.

If the symmetric key cannot be accessed, then the original file cannot be decrypted. This method can be used to secure messages and files without taking up too much time and resources.

RSA encryption can be used in various systems. It can operate in OpenSSL, wolfCrypt, cryptlib, and other cryptographic libraries.

Traditionally, it has been used in TLS and was also the initial algorithm used in PGP encryption. RSA is still seen in a variety of web browsers, email, VPNs, chat, and other communication channels.

RSA is also commonly used to create secure connections between VPN clients and VPN servers. In protocols such as OpenVPN, TLS can use the RSA algorithm to exchange keys and establish a secure channel.

2.4.2 Security Analysis of RSA

Secure communication is when two entities are communicating and do not want a third party to listen in. For this to be the case, the entities need to communicate in a way that is unsusceptible to eavesdropping or interception. There are several requirements for secure communication which rsa have, including:

**Confidentiality:** The content of the communication should be kept secret from anyone who is not authorized to access it. (Has been encrypt)

**Integrity:** The content of the communication should not be altered in transit without being detected. (Encrypt message equal original message)

**Authentication:** The identity of the sender and receiver should be verified to ensure that they are who they claim to be. (Has private key)

**Non-repudiation:** The sender should not be able to deny sending the message, and the receiver should not be able to deny receiving it. (Ensures that the sender of a message cannot deny sending it and the receiver cannot deny receiving it. In the RSA (Rivest-Shamir-Adleman) algorithm, non-repudiation is achieved using digital signatures. A digital signature is a mathematical scheme that allows a sender to prove the authenticity and integrity of a message to a receiver. When the sender signs a message with their private key, the receiver can verify the signature using the sender's public key. If the verification process is successful, the receiver can be assured that the message was indeed sent by the sender and has not been tampered with in transit. This makes it difficult for either party to deny their involvement in the communication.

The security of RSA is based on the difficulty of factoring large integers. To ensure its security, security experts and researchers conduct various analyses to identify potential vulnerabilities and improve the encryption algorithm. RSA's security analysis involves analyzing the complexity of computational operations, mathematical analysis, encryption and decryption analysis, vulnerability testing, and detecting potential attacks.

Mathematical analysis of RSA involves analyzing the properties of prime numbers, mathematical functions used in the algorithm, and the properties of modular arithmetic. Encryption and decryption analysis of RSA involves analyzing the encryption and decryption processes and the properties of the keys used. Vulnerability testing of RSA involves testing the algorithm for potential vulnerabilities and weaknesses. Detecting potential attacks involves analyzing the different methods attackers could use to exploit RSA's vulnerabilities.

These analyses provide insights that help security experts to improve and enhance the security of RSA, ensuring the safety of information encryption in real-world applications.

2.4.3 Pros and Cons

Pros:

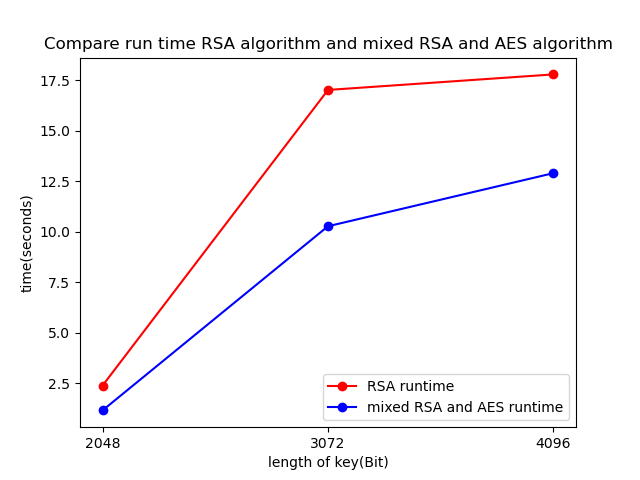
* Security: RSA is considered one of the most secure public-key cryptography algorithms available today. It provides strong encryption, which makes it difficult for attackers to break into.
* Versatility: RSA is versatile and can be used for a wide range of applications, including secure email, secure file transfer, and secure online transactions.
* Compatibility: RSA is widely supported by software and hardware, making it easy to implement and use in a variety of environments.
* Digital signatures: RSA supports digital signatures, which provide a way for users to verify the authenticity and integrity of messages and documents.

**Cons**

* Performance: RSA encryption and decryption operations can be slow, especially when dealing with large amounts of data. This can be a problem in real-time applications where speed is critical.

Time limits: I conducted an experiment (10 times per case) by measuring the execution time of the RSA algorithm and the combined RSA and AES algorithm on the same piece of text and obtained the key bit sizes as shown in the table below. I then used the matplotlib library to graph the results as shown in table and the figure below: (Note that the running speed may depend on your computer hardware, Python version, and the randomly generated key.)

|  |  |  |  |
| --- | --- | --- | --- |
| Number Bits of key | 2048 | 3072 | 4096 |
| RSA time | 2.38 | 17.2 | 17.79 |
| RSA & AES time | 1.15 | 10.27 | 12.99 |



2.4.4 Comment

After conduct experiment, I see that RSA runtime longer than RSA mixed AES ((Advanced Encryption Standard) is a symmetric encryption algorithm, which means that the same key is used for both encryption and decryption. It was introduced as a replacement for the outdated DES (Data Encryption Standard) algorithm).

Time complete depend on number of bit and length of plain text. Key size: RSA requires large key sizes to ensure security, which can be a problem in certain applications where the available storage space is limited.

This experiment does not conclude that the RSA algorithm is inferior to other algorithms in terms of speed and security, but rather shows that the RSA algorithm can increase operational efficiency if combined with other algorithms.

2.5 Recommendations

RSA is still widely used and considered a secure cryptosystem when used with sufficiently large keys. To enhance the security of RSA, it is crucial to use large keys, implement the algorithm carefully, and update the keys regularly. Additionally, using block cipher algorithms such as AES to encrypt data and then using RSA to encrypt the AES key can further enhance the security of RSA, making it more robust against potential vulnerabilities and attacks.

In conclusion, analyzing the security and limitations of the RSA algorithm is essential to ensure the safety of information encryption in real-world applications. Security experts and researchers should continue to update, research, and develop new methods to enhance the security of RSA against potential vulnerabilities and attacks. By continuously improving the security of RSA, we can ensure that it remains a reliable and secure method for data encryption.

REFERENCES

1. S. Azad and A. K. Pathan, Practical Cryptography. Auerbach Publications, 2014. ISBN: 9781482228908, [Chapter 8: The RSA Algorithm](https://learning.oreilly.com/library/view/practical-cryptography/9781482228892/ch08.html?_gl=1*c6n57h*_ga*MTAwNzI2NDgxNC4xNjgxMzc2MDc2*_ga_092EL089CH*MTY4MTM3NjA3NS4xLjEuMTY4MTM3NjMwMC42MC4wLjA.)
2. NaQi, Wei Wei, Jing Zhang, Wei Wang, Jinwei Zhao, Junhuai Li, Peiyi Shen, Xiaoyan Yin, Xiangrong Xiao and Jie Hu, 2013. Analysis and Research of the RSA Algorithm. Information Technology Journal, 12: 1818-1824.
3. Nisha, Shireen & Farik, Mohammed. (2017). RSA Public Key Cryptography Algorithm – A Review. International Journal of Scientific & Technology Research. 6. 187-191.