|  |
| --- |
| 2025 CICT high quality class  Group Project Report |
| Network Security |

| Project Title | Secure Communication using Symmetric and Asymmetric Cryptography | |
| --- | --- | --- |
| Project Area | Cryptography | |
| Students | ID: B2111933 | Name: Truong Dang Truc Lam |
| ID: B2111952 | Name: Le Xuan Thanh |
| ID: B2111984 | Name: Dang Hoang Hung |
| Reporting Date |  | |

**TABLE OF CONTENTS**

[**Ⅰ. Project Outline 2**](#_heading=h.6x6sbvx9ocr1)

[1. Purpose of Project 3](#_heading=h.9htkwlifwbh8)

[2. Project workflow 3](#_heading=h.fnr4cbcdrunh)

[**Ⅲ. Action Plan 4**](#_heading=h.ja4fsf1gv9kx)

[1. Select Environment/IDE 4](#_heading=h.sk1pauo9cnmg)

[2. Designing System Architecture XT 4](#_heading=h.hdibs3gcvgix)

[3. Role arrangements 5](#_heading=h.mtwsdkeai74w)

[4. Project Schedule 5](#_heading=h.lpku31cqrevd)

[**Ⅳ. Expected Benefit 6**](#_heading=h.h2s4gjkufoon)

[1. Performance Goals 6](#_heading=h.5htddd6bbbce)

[2. Benefit 6](#_heading=h.yy5p6h9h5m23)

[**Ⅴ. Practice Result 7**](#_heading=h.o35sy8jvest2)

[1. Preparations 7](#_heading=h.ku18iips3hpk)

[1.1. RSA Key Generation Script 7](#_heading=h.ag5m7pwkn8y3)

[1.2. Encryption Functions 9](#_heading=h.ckdehasawhei)

[2. Encrypt the Message (Sender) 11](#_heading=h.3wq1x2xcqcqm)

[3. Decrypt the Message (Receiver) 14](#_heading=h.ls8l6vhfob04)

[**Ⅵ. Problem and Solution 17**](#_heading=h.gfngs8o130uh)

[**Ⅶ. References 18**](#_heading=h.dlvwtfm3huwz)

# 

# 

# Ⅰ. Project Outline

Title: Secure Communication using Symmetric and Asymmetric Cryptography

Group Information:

| Team Name | | Group 1 | | | |
| --- | --- | --- | --- | --- | --- |
| Team Composition | | Name | Belong | Department | Position /year |
| Professor | | Nguyen Huu Hoa | CICT |  | Rector |
| Instructor | | Noh | CICT | IT Department | Advisor |
| Student | Team Leader | Truong Dang Truc Lam | CICT | IT Department | Student |
| Team Member 2 | Le Xuan Thanh | CICT | IT Department | Student |
| Team Member 3 | Dang Hoang Hung | CICT | IT Department | Student |
|  | | | | | |
|  | | | | | |

Ⅱ. Project Information

## 1. Purpose of Project

This project demonstrates the implementation of a secure communication system that ensures data protection over insecure networks by combining Symmetric Cryptography [1] and Asymmetric Cryptography [2].

The process begins with the generation of RSA public/private key pairs using OpenSSL, followed by the creation of a sample message. A random 128-bit AES key is then generated to encrypt the message using AES [3] in EAX mode, which provides both confidentiality and integrity through the use of a Nonce and an Authentication Tag. After encryption, the AES key is encrypted using the RSA Public Key [4], and all encrypted components (the ciphertext, nonce, tag, and RSA-encrypted AES key) are prepared for transmission. This hybrid encryption setup ensures that the actual message remains secure while the symmetric key used to encrypt it is also protected during exchange.

On the receiver's side, the RSA Private Key is used to decrypt the AES key, which is then used to decrypt the message content. The decrypted message is compared with the original to verify both correctness and integrity. This process highlights the strengths of combining AES and RSA. AES provides fast and secure data encryption , while RSA enables secure key distribution (overcome the limitations of symmetric and asymmetric encryption methods).

Designed primarily for educational purposes, this project simplifies complex cryptographic operations and provides a clear, hands-on demonstration of real-world secure communication. Possible improvements include adding digital signatures for sender authentication or incorporating key rotation mechanisms to strengthen long-term security.

## 2. Project workflow

Create RSA public/private keys using OpenSSL.

Save the message in message.txt.

Create a random 128-bit AES key.

Use AES (EAX mode) to encrypt the message. Save ciphertext, nonce, and tag.

Encrypt the AES key using the RSA public key.

Send encrypted AES key and AES-encrypted message components.

Use the RSA private key to decrypt the AES key.

Use the decrypted AES key to recover the original message.

Check that the decrypted message matches the original

Evaluate strengths of AES + RSA and suggest improvements.

# Ⅲ. Action Plan

## 1. Select Environment/IDE

| Kali Linux ⇒ Activity ⇒ Terminal ⇒ Python environment ⇒ Key generation with OpenSSL  Text Editor ⇒ Nano / Vim ⇒ Write and edit Python scripts  Web Browser ⇒ Used for downloading libraries, researching documentation  VS Code ⇒ Python Extension ⇒ PyCryptodome, Cryptography libraries for AES & RSA  PyCharm ⇒ Full-featured IDE ⇒ Virtual environment setup ⇒ Debugging encryption logic  OpenSSL CLI ⇒ Terminal-based ⇒ Generate RSA keys (private/public) |
| --- |

## 2. Designing System Architecture

| Items | Resource |  |  |  |
| --- | --- | --- | --- | --- |
| VM type | Oracle VirtualBox |  |  |  |
| OS | Kali Linux |  |  |  |
| IP/URL |  |  |  |  |
|  |  |  |  |
| Attacking type |  |  |  |  |
| Detecting type |  |  |  |  |
| Language & version | Python 3.x |  |  |  |
| Libraries | PyCryptodome, Cryptography |  |  |  |
| Library package | PyCryptodome, Cryptography, OpenSSL |  |  |  |
| Algorithm | AES, RSA |  |  |  |
| AI Technology type |  |  |  |  |
| Database type |  |  |  |  |
| Software tool | Terminal, Nano/Vim, Web Browser, VS Code, PyCharm, OpenSSL CLI |  |  |  |
| Source code reference |  |  |  |  |
| Dataset reference |  |  |  |  |

## 3. Role arrangements

| No. | Division | Role |
| --- | --- | --- |
| 1 | Plan and design workflow | Truong Dang Truc Lam B2111933 |
| 2 | Learn theory about Cryptography | Le Xuan Thanh B2111952  Dang Hoang Hung B2111984 |
| 3 | Research about Kali Linux and choose tools | Truong Dang Truc Lam B2111933  Le Xuan Thanh B2111952 |
| 4 | Practice on Cryptography | Truong Dang Truc Lam B2111933  Dang Hoang Hung B2111984 |

## 4. Project Schedule

| Division | Promotion contents | Schedule | | | | | | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| w9 | w10 | w11 | w12 | w13 | w14 | w15 |
| Plan and design workflow | Role sharing and analysis software installation | x | x | x |  |  |  |  |
| Learn theory about Cryptography | Cryptography technique option analysis |  | x | x | x |  |  |  |
| Research about Kali Linux and choose tools | Analysis using Kali Linux tools for Cryptography |  |  | x | x | x |  |  |
| Practice on Cryptography | Create exercise for Cryptography |  |  |  | x | x | x |  |
| Finish and prepare for presentation | Create result document through analysis |  |  |  |  | x | x | x |
| Offline meeting | Information sharing and progress confirmation of each other | x |  | x |  | x |  | x |

# 

# Ⅳ. Expected Benefit

## 1. Performance Goals

* **Efficient Message Encryption and Decryption:**The system should achieve fast and lightweight encryption/decryption operations using AES (a symmetric cipher known for its speed and low computational overhead). Even when processing large messages, the time taken for encryption and decryption should remain minimal, ensuring that the system can handle real-time or near-real-time communication scenarios.
* **Secure Symmetric Key Exchange via RSA:** The AES key must be securely transmitted to the recipient by encrypting it with RSA. The project aims to demonstrate that even if the transmission channel is insecure, interceptors cannot access the AES key without the corresponding RSA private key, protecting the confidentiality of the message.
* **Data Integrity and Authenticity Verification:** By using AES in EAX mode, the system not only encrypts the data but also produces an authentication tag. This tag is critical to ensure the integrity and authenticity of the message — meaning any modification or tampering during transmission will be detected immediately upon decryption.
* **Accurate Decryption and Message Recovery:** The decrypted output should exactly match the original plaintext message. Verification by matching the original and decrypted data ensures the correctness of the encryption-decryption process and demonstrates the robustness of the cryptographic system.
* **Resilience Against Common Attacks:** The hybrid system should offer protection against typical security threats such as man-in-the-middle attacks, key theft, ciphertext manipulation, and replay attacks, by ensuring that both the message and the encryption keys are securely handled and verified.
* **Modularity and Scalability:** The system should be built in a modular fashion, making it easy to extend in the future — for example, by adding digital signatures for sender authentication, rotating keys periodically, or supporting larger RSA key sizes (e.g., 4096 bits for higher security).

## 2. Benefit

* **In-Depth Practical Knowledge of Cryptography:** By implementing both symmetric and asymmetric encryption mechanisms manually, learners and developers gain firsthand experience with real-world cryptographic operations, understanding how they interact and the practical challenges involved.
* **Demonstrating the Strength of Hybrid Encryption Models:** This project showcases why most secure communication systems (like SSL/TLS, secure emails, and file encryption services) use a combination of symmetric and asymmetric cryptography — achieving both high performance and high security in a single system.
* **Enhanced Data Security Awareness:** Participants will understand the importance of encrypting not just the data but also the keys. They will learn that securing the key exchange is as important as encrypting the message itself, which is a critical concept for cybersecurity professionals.
* **Foundation for Building Secure Systems:** This project provides a solid baseline for implementing more complex security features later on, such as:
* Digital signatures to prove sender authenticity.
* Certificate-based authentication for validating public keys.
* Multi-party secure communication (group messaging).
* Regular key rotation policies for long-term protection.
* **Hands-On with Industry Tools:** By using OpenSSL for key generation, Python for encryption scripting, and libraries like PyCryptodome, participants gain experience with industry-standard tools and libraries that are commonly used in professional environments.
* **Portfolio Enhancement and Academic Growth:** Successfully completing this project can serve as a portfolio project to demonstrate skills in cryptography, cybersecurity, and secure software development. It also strengthens resumes and academic profiles when applying for internships, cybersecurity roles, or graduate studies in information security fields.
* **Critical Thinking and Problem-Solving Skills:** Debugging encryption issues, handling errors in decryption, and ensuring correct data handling build analytical and problem-solving abilities, which are critical in cybersecurity careers.

# 

# Ⅴ. Practice Result

## 1. Preparations

### 1.1. RSA Key Generation Script

Generate RSA public/private key pair using OpenSSL:

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -

#!/bin/bash

# Create RSA private key

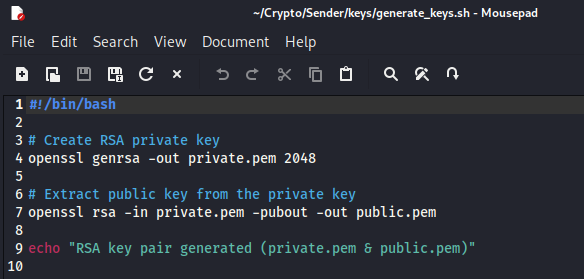
openssl genrsa -out private.pem 2048

# Extract public key from the private key

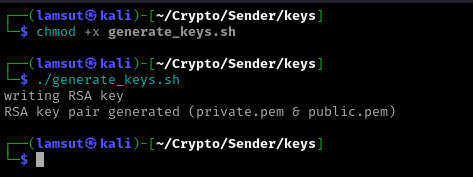
openssl rsa -in private.pem -pubout -out public.pem

echo "RSA key pair generated (private.pem & public.pem)"

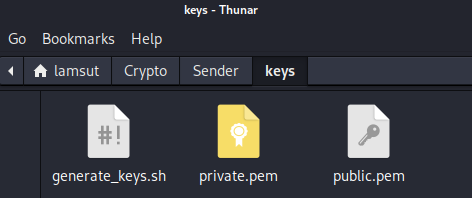
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -



The script for generating keys



RSA key pair generated



Check the result

### 1.2. Encryption Functions

Encrypt a plaintext message using:

* AES (for fast symmetric encryption):

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - -

from Crypto.Cipher import AES

from Crypto.Random import get\_random\_bytes

def encrypt\_aes(plaintext: str):

key = get\_random\_bytes(16) # AES-128

cipher = AES.new(key, AES.MODE\_EAX)

ciphertext, tag = cipher.encrypt\_and\_digest(plaintext.encode())

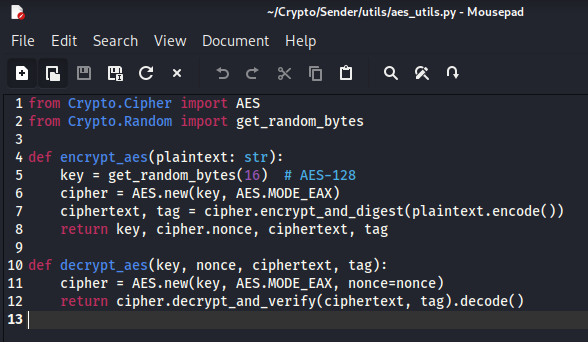
return key, cipher.nonce, ciphertext, tag

def decrypt\_aes(key, nonce, ciphertext, tag):

cipher = AES.new(key, AES.MODE\_EAX, nonce=nonce)

return cipher.decrypt\_and\_verify(ciphertext, tag).decode()

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - -



The code for AES encryption

* RSA (for secure key exchange or message encryption):

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - -

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

def encrypt\_rsa\_key(public\_key\_file: str, data: bytes) -> bytes:

pub\_key = RSA.import\_key(open(public\_key\_file).read())

cipher = PKCS1\_OAEP.new(pub\_key)

return cipher.encrypt(data)

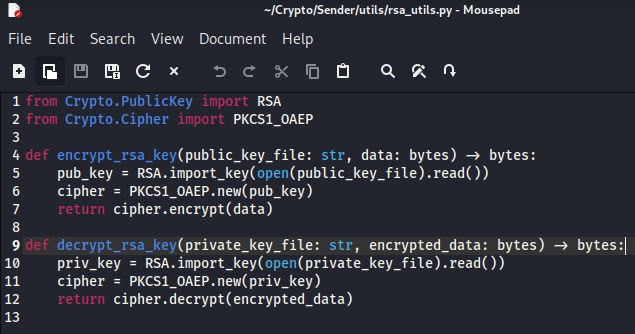
def decrypt\_rsa\_key(private\_key\_file: str, encrypted\_data: bytes) -> bytes:

priv\_key = RSA.import\_key(open(private\_key\_file).read())

cipher = PKCS1\_OAEP.new(priv\_key)

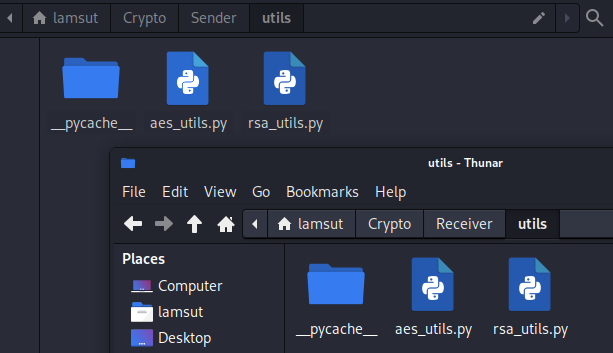
return cipher.decrypt(encrypted\_data)

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - -



The code for RSA encryption

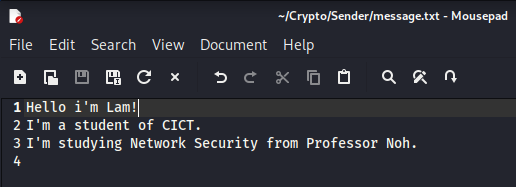
Those functions will be place in the utils folder:



Encryption functions

## 2. Encrypt the Message (Sender)

Before encrypting, we have to create a message in plaintext first:



The contents of the message in plain text

Demonstrate hybrid encryption (AES key encrypted using RSA):

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -

from utils.aes\_utils import encrypt\_aes

from utils.rsa\_utils import encrypt\_rsa\_key

import base64

# Load message

with open("message.txt", "r") as file:

message = file.read()

# Encrypt with AES

aes\_key, nonce, ciphertext, tag = encrypt\_aes(message)

# Encrypt AES key with RSA

encrypted\_key = encrypt\_rsa\_key("keys/public.pem", aes\_key)

# Save encrypted data to files

with open("../Receiver/encrypted\_mess/encrypted\_key.bin", "wb") as f: f.write(encrypted\_key)

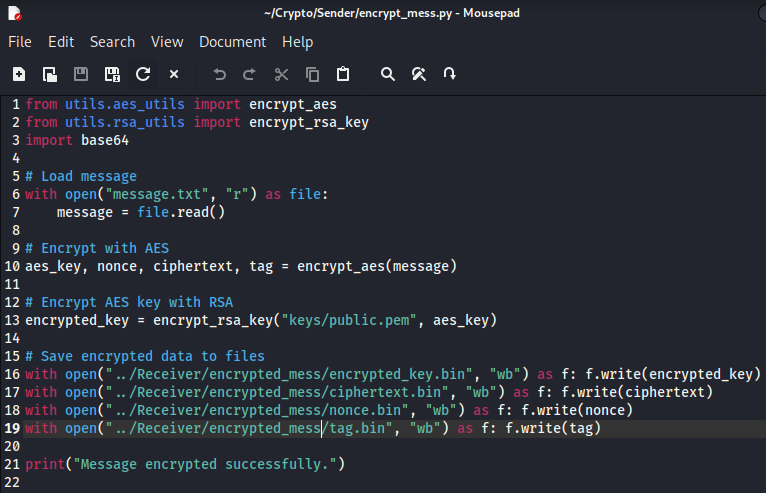
with open("../Receiver/encrypted\_mess/ciphertext.bin", "wb") as f: f.write(ciphertext)

with open("../Receiver/encrypted\_mess/nonce.bin", "wb") as f: f.write(nonce)

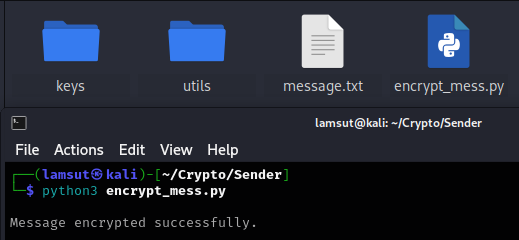
with open("../Receiver/encrypted\_mess/tag.bin", "wb") as f: f.write(tag)

print("Message encrypted successfully.")

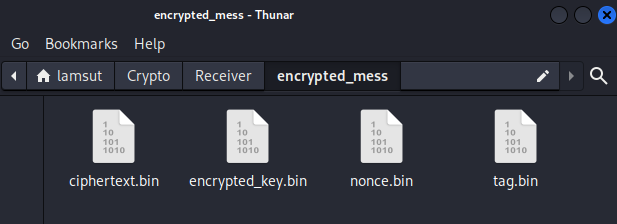
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -



The code for hybrid encryption



Encrypt the previous message



The encrypted message

## 3. Decrypt the Message (Receiver)

Decrypt encrypted message and verify correctness:

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -

from utils.aes\_utils import decrypt\_aes

from utils.rsa\_utils import decrypt\_rsa\_key

# Load encrypted data

with open("encrypted\_mess/encrypted\_key.bin", "rb") as f: encrypted\_key = f.read()

with open("encrypted\_mess/ciphertext.bin", "rb") as f: ciphertext = f.read()

with open("encrypted\_mess/nonce.bin", "rb") as f: nonce = f.read()

with open("encrypted\_mess/tag.bin", "rb") as f: tag = f.read()

# Decrypt AES key using RSA

aes\_key = decrypt\_rsa\_key("keys/private.pem", encrypted\_key)

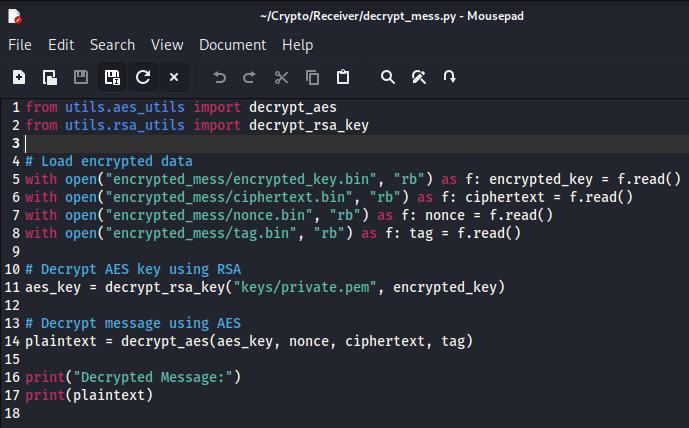
# Decrypt message using AES

plaintext = decrypt\_aes(aes\_key, nonce, ciphertext, tag)

print("Decrypted Message:")

print(plaintext)

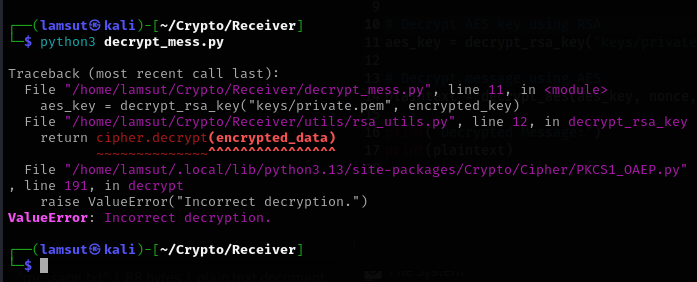
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -- - - - - - - - - - - - - - - - - - - -



The code for decrypting message

Up to now, the receiver does not have a suitable keys

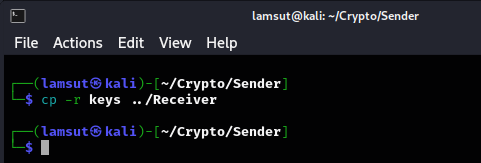
So it will send error message:



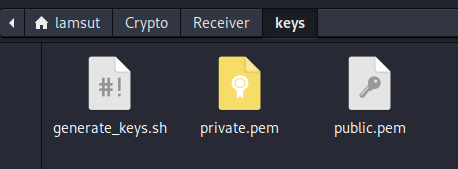
The result if the receiver does not have the correct keys

To enable decryption, the receiver must have the same keys as the sender:

So we will copy the keys to the receiver.

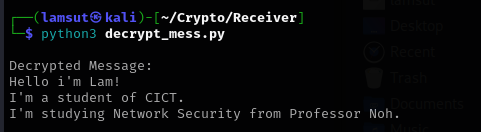


Give the keys to the receiver



The receiver now has the keys

Now we will try to decrypt again:



The decryption is successful now

# 

# Ⅵ. Problem and Solution

**1. Technical issues during project development**

* **Key Availability on Receiver Side:** The receiver must have access to the correct RSA private key and must also understand the decryption algorithm (AES in EAX mode) to successfully decrypt the message. Without these, decryption fails entirely.
* **File Handling and Format Compatibility:** Handling binary files (e.g., ciphertext, tag, nonce, and encrypted key) requires careful reading/writing to ensure no corruption. Mistakes in encoding (e.g., using text mode instead of binary) can lead to errors during decryption.
* **RSA Key Size and Encryption Limitations:** RSA encryption is only suitable for small data (like AES keys). Encrypting large messages directly with RSA is not feasible, so implementing and managing hybrid encryption properly was essential.
* **Error Handling and Debugging:** Incorrect decryption often results in unclear errors (like MAC check failure), making it difficult to determine whether the issue lies in key mismatches, file corruption, or algorithm misuse.
* **Key Distribution Security:** Sharing the private key with the receiver for testing purposes introduces a potential security concern. In real-world applications, securely exchanging keys without compromising them is a non-trivial problem.

**2. Solution**

* **Key and Algorithm Sharing:** The necessary keys (public and private RSA keys) and the decryption logic were clearly defined and securely transferred to the receiver. This ensured that the receiver could perform decryption without ambiguity.
* **Clear Folder Structure and File Separation:** A well-organized project directory was maintained to separate sender and receiver roles, and encryption components were stored clearly (e.g., encrypted\_key.bin, nonce.bin, etc.) to avoid confusion.
* **Hybrid Encryption Design:** RSA was only used to encrypt the AES key, while the message itself was encrypted using AES — ensuring both performance and security. This follows real-world cryptographic design best practices.
* **Robust File I/O and Encoding:** All binary files were handled using 'rb' and 'wb' modes to preserve integrity. Base64 encoding was avoided where unnecessary to keep the system efficient and clean.
* **Modular Utility Functions:** Encryption and decryption logic was separated into utility files (aes\_utils.py, rsa\_utils.py) to keep the code clean and maintainable.

# Ⅶ. References

[1] <https://en.wikipedia.org/wiki/Symmetric-key_algorithm>

[2] <https://en.wikipedia.org/wiki/Public-key_cryptography>

[3] <https://en.wikipedia.org/wiki/Advanced_Encryption_Standard>

[4] <https://en.wikipedia.org/wiki/RSA_cryptosystem>