Implementation of security mechanisms using AES and RSA Hybrid Algorithm with Digital Signatures

Team Details

1. Shaista Firdous(20eg105442)
2. M. Sathvika(20eg105428)
3. K. Shiva Sai(19H61A05L6)

Project Supervisor
Dr. K. Madhuri
Associate Professor

Introduction

- It is a secured hybrid algorithm used to protect the safe transmission of data in the Network communication using AES and RSA algorithm. To secure the text as well as key.
- There are different parameters that are needed, those are AES Encryption/Decryption, RSA Encryption/Decryption, Hybrid Encryption, Secure Key Management, Random Number Generation, Secure E-mail transmission, Digital Signatures.

APPLICATIONS:

- **Secure Email Communication:** Implementing AES for symmetric encryption and RSA for asymmetric encryption can ensure that emails are securely transmitted and only accessible to authorized recipients. This is crucial for protecting sensitive information in message exchanges, such as personal data, financial information, or confidential business communications.
- **Digital Signatures:** RSA can be used for digital signatures, which verify the authenticity and integrity of email messages. A sender can sign their message with their private key, and the recipient can verify the signature.

Problem Statement

AES, a symmetric key encryption algorithm, offers efficiency but demands secure key sharing and management. It's not ideal for key exchange. RSA, an asymmetric algorithm, excels in key exchange but is relatively slow for encrypting large data. Key management is critical in hybrid encryption, ensuring secure key exchange between parties. RSA's slower speed may introduce computational overhead. Padding in RSA can increase data size.

The objective of implementing an AES and RSA hybrid algorithm is to develop a secure and efficient system that balances the strengths of these encryption methods. This project aims to enhance data security by leveraging AES for efficient encryption, RSA for secure key exchange, and protection of sensitive information. Key management is a critical focus, ensuring secure key distribution. The system will enable secure communication over untrusted channels, protect data integrity, and maintain practicality for real-world applications while being adaptable and scalable for various platforms and user needs. Rigorous testing and documentation will validate its security and functionality, ensuring a reliable solution for data confidentiality and integrity.

Proposed Method

1)Our proposed idea is we are using AES algorithm to encrypt the message using a key k, and we are using RSA algorithm to encrypt the key k using receiver's public key PU, that cipher key is considered as "ck".

2)The encrypted key and email are transferred to the receiver, then the receiver uses his private key to decrypt the "ck" then we get k, using k he will decrypt the cipher text.

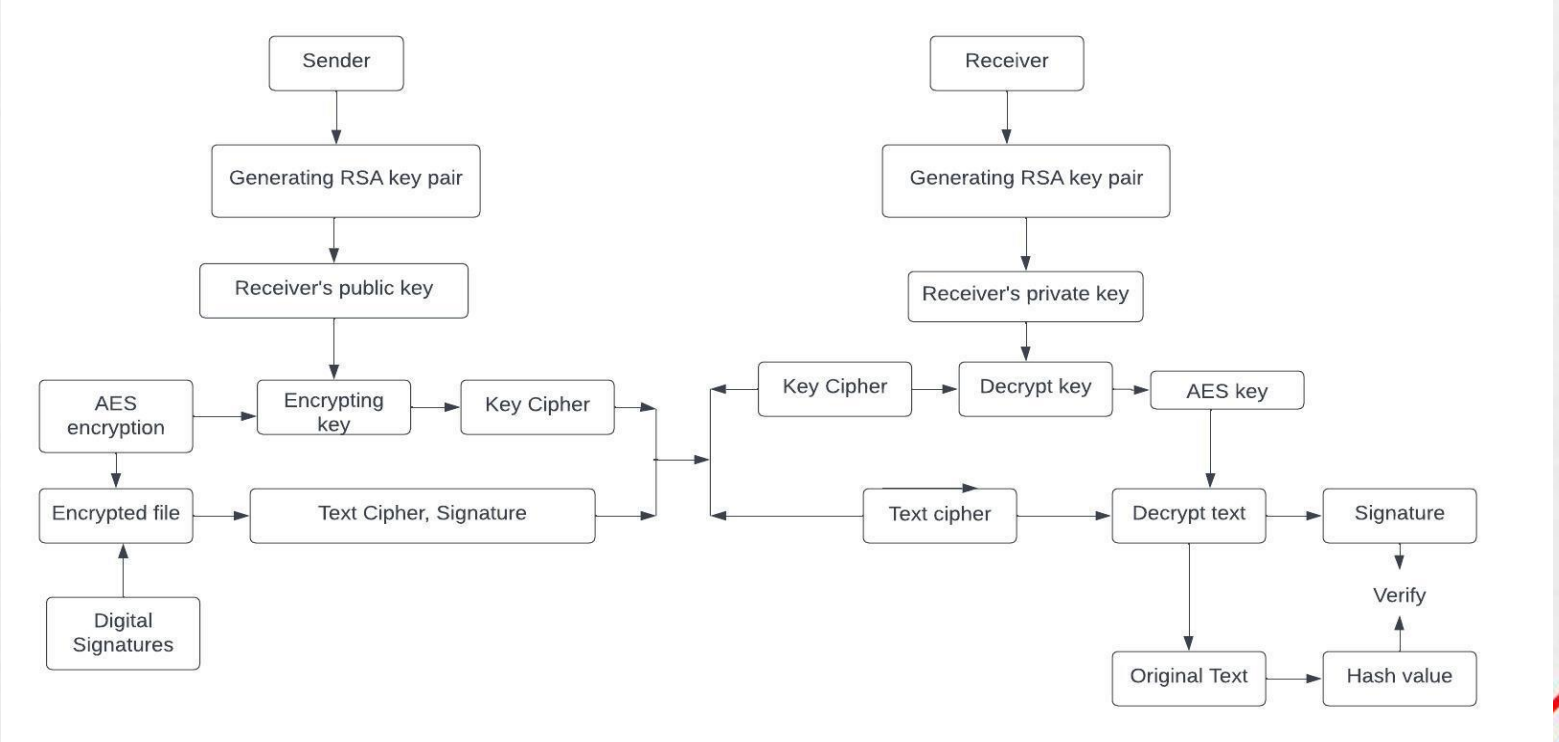
3)Using above Hybrid algorithm we will ensure the confidentiality of the email, but in the next step we will check whether the sender is authorized or not.

4)For that we are using Digital Signature to authenticate the user, we are using RSA approach to encrypt the Digital Signature.

5)Using cryptographic hash function we generate a fixed-size hash value, which is a digital signature and using sender's private key we encrypt that key and send that to the receiver.

6)Then, receiver will decrypt the digital signature using the sender's public key, the using the same hash function the receiver will generate a hash value and compare it with the sender's digital signature.

Proposed Method



Experiment Environment

- To complete this project, for execution I have used python idle 3.11(64-bit)
- Before using the python idle, I have installed packages in command prompt. Those are:
  1)Pycryptodome
  2)Cryptography
- We have used different packages and libraries. Those are:
  1) hashlib: In the code, hashlib is used to generate a SHA-256 hash value from the input message.
  2) Crypto.PublicKey.RSA: The RSA module is used to perform RSA encryption and decryption.
  3) Crypto.Cipher.AES: This module provides AES (Advanced Encryption Standard) encryption and decryption functionalities.The AES module is used to encrypt and decrypt messages using AES in CBC mode.
  4)Crypto.Random: The get_random_bytes function from this module is used to generate a random initialization vector (IV) for AES encryption.
  5)Crypto.Protocol.KDF.PBKDF2: This module provides the PBKDF2 key derivation function, which is used to derive a key from the user's password.
  6)base64: This module provides functions for encoding binary data to ASCII and decoding it back. In the code, it is used to base64 encode the IV and ciphertext before sending it to the user.
  7)math: The math module is used to perform mathematical operations. In the code, it is used to calculate the greatest common divisor (gcd) of two numbers.
  8)time: The time module is used to measure the execution time of certain parts of the code.

Experiment Screenshots

ANURAG UNIVERSITY

Libraries used in this project

```
import hashlib
from Crypto.PublicKey import RSA
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Protocol.KDF import PBKDF2
import base64
import math
import time
import matplotlib.pyplot as plt
```

Padding and unpadding the text

```
print("Email Security System")
print("..............AES ALGORITHM.............")
password = input("Enter the AES key: ")
message = input("Enter the E-mail content: ")
def pad(text):
    return text + (16 - len(text) % 16) * chr(16 - len(text) % 16)
def unpad(text):
    return text[:-ord(text[-1])]
```

Experiment Screenshots

Encryption and Decryption of Email content: Here we are first padding random bytes to the text because in AES plaintext size is 16bytes, later we are encrypting. While decrypting we are removing the extra bytes we added and showing the plaintext we entered.

```
def encrypt(key, plaintext):
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    padded_plaintext = pad(plaintext)
    padded_plaintext_bytes = padded_plaintext.encode('utf-8')
    ciphertext = cipher.encrypt(padded_plaintext_bytes)
    return base64.b64encode(iv + ciphertext).decode('utf-8')

def decrypt(key, ciphertext):
    ciphertext = base64.b64decode(ciphertext)
    iv = ciphertext[:16]
    ciphertext = ciphertext[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    decrypted_text = cipher.decrypt(ciphertext)
    decrypted_text = unpad(decrypted_text.decode('utf-8'))

    return decrypted_text
```

Experiment Screenshots

We are declaring values of p and q, which are prime numbers. Using this we are generating the public and private keys in RSA Algorithm.

```
print("................RSA ALGORITHM...................")
def is_prime(num):
    if num < 2:
        return False
    for i in range(2, int(num ** 0.5) + 1):
        if num % i == 0:
            return False
    return True

def get_user_input_prime():
    while True:
        try:
            prime = int(input("Enter a prime number required to generate public and private keys: "))
            if is_prime(prime):
                return prime
            else:
                print("Please enter a valid prime number.")
        except ValueError:
            print("Please enter a valid integer.")

# Get user input for p and q
p = get_user_input_prime()
q = get_user_input_prime()
```

Experiment Screenshots

Using RSA mathematical calculations we are generating public and private keys.

```
def find_coprime(num):
    value = 2
    while True:
        if math.gcd(num, value) == 1:
            return value
        value += 1
def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1
def generate_keypair(p, q):
    n = p * q
    phi_n = (p - 1) * (q - 1)
    e = find_coprime(phi_n)
    d = mod_inverse(e, phi_n)
    return (e, n),(d, n)
public_key, private_key = generate_keypair(p, q)
```

Experiment Screenshots

We are encrypting the AES key using RSA public key and decrypting it using RSA private key

```
def encrypt1(public_key, plaintext):
    e,n=public_key
    return pow(plaintext, e, n)

def decrypt1(private_key, ciphertext):
    d,n=private_key
    return pow(ciphertext, d, n)
```

Experiment Screenshots

ANURAG UNIVERSITY

We are using RSA approach in digital signature to authenticate the user. We are generating hash value using SHA256 and considering it as digital signature.

```
def sign(message, private_key):
    hash_value = hashlib.sha256(message.encode()).digest()
    print("The generated hash value from the message:",hash_value)
    integer_value=int.from_bytes(hash_value, 'big')
    integer_value1=integer_value%private_key[1]
    print("The hash value which should be encrypted is:",integer_value1)
    signature1=encrypt1(private_key,integer_value1)
    print("The encrypted signature:",signature1)
    return signature1

def verify(message, signature, public_key):
    hash_value = hashlib.sha256(message.encode()).digest()
    integer_value=int.from_bytes(hash_value, 'big')
    integer_value1=integer_value%private_key[1]
    computed_hash=decrypt1(public_key,signature)
    print("The decrypted signature:",computed_hash)
    return integer_value1 == computed_hash
```

Experiment Screenshots

```
def encryption_time(encryption_time_aes,encryption_time_rsa):
    time=encryption_time_aes+encryption_time_rsa
    return time
def decryption_time(decryption_time_aes,decryption_time_rsa):
    time=decryption_time_aes+decryption_time_rsa
    return time
def plot_line_chart(labels,aes_times,rsa_times,hybrid_times,title):
    plt.plot(labels, aes_times,marker="o", label='AES')
    plt.plot(labels, rsa_times, marker="o",label='RSA')
    plt.plot(labels, hybrid_times,marker="o",label='Hybrid (AES+RSA)')
    plt.xlabel('Message Length (bytes)')
    plt.ylabel('Time(s)')
    plt.title(title)
    plt.legend()
    plt.grid()
    plt.show()
```

Experiment Screenshots

ANURAG UNIVERSITY

```
def main():
    def convert_to_int(password):
        if password.isdigit():
            return int(password)
        else:
            concatenated_ascii = ''.join(str(ord(char)) for char in password)
            return concatenated_ascii
    password1=str(convert_to_int(password))
    key = PBKDF2(password1, b"salt", dkLen=32, count=1000000)
    public_key, private_key = generate_keypair(p,q)
    print("The public key in RSA is:",public_key)
    print("The private key in RSA is:",private_key)
    n=public_key[1]
    message1=int(password1)%n
    print(".............The Encryption/Decryption process.............")
    while True:
        print("\nSelect an operation:")
        print("1. Encrypt")
        print("2. Decrypt")
        print("3. Authenticate User")
        print("4.Exit")
        choice = int(input("Enter your choice: "))
        if choice == 1:
            encrypted_message = encrypt(key, message)
            print(".............Encrypt the E-mail content using AES Algorithm.............")
            print("The encrypted E-mail content:", encrypted_message)
            encrypted_password=encrypt1(public_key,message1)
            print(".............Encrypt the AES key using RSA Algorithm.............")
            print("The AES key is converted to integer for encryption:",message1)
            print("The encrypted AES key:",encrypted_password)
        elif choice == 2:
            decrypted_message = decrypt(key, encrypted_message)
            print(".............Decrypt the E-mail content using AES Algorithm.............")
            print("The decrypted E-mail content:", decrypted_message)
            print(".............Decrypt the AES key using RSA Algorithm.............")
            decrypted_password=decrypt1(private_key,encrypted_password)
            print("The decrypted AES key:",decrypted_password)
```

Experiment Screenshots

```
            print("Hybrid Decryption time: ",decrypt_time)
        elif choice == 3:
            print(".....Authenticating the user using Digital Signature.....")
            signature=sign(message, private_key)
            is_valid = verify(message, signature, public_key)
            print("Is the signature valid?", is_valid)
        elif choice == 4:
            break
        else:
            print("Invalid choice. Please try again.")
    message_lengths = [128,256,512,1024,2048,5096]  # Message lengths in bytes
    aes_times = [0.015,0.06,0.14,0.18,0.31,0.37]
    rsa_times = [0.04,0.11,0.21,0.31,0.49,0.59]  # Placeholder RSA encryption times
    hybrid_times = [0.025,0.07,0.15,0.19,0.32,0.38]

    # Plot line chart for encryption times
    plot_line_chart(message_lengths, aes_times, rsa_times, hybrid_times, 'Encryption Times')

    # Placeholder decryption times (modify these with actual times)
    aes_dec_times = [0.03,0.07,0.17,0.21,0.32,0.41]
    rsa_dec_times = [0.60,0.86,1.31,1.76,2.68,3.2]
    hybrid_dec_times = [0.51,0.55,0.65,0.69,0.81,0.88]

    # Plot line chart for decryption times
    plot_line_chart(message_lengths,aes_dec_times,rsa_dec_times,hybrid_dec_times,'Decryption Times')

if __name__ == "__main__":
    main()
```

Experiment Screenshots

Output:

```
Email Security System
.............AES ALGORITHM..............
Enter the AES key: abcd
Enter the E-mail content: Greetings from Infosys Springboard!Is Time Management a Hoax? Or is it a Paradox? At best it can just be
a misnomer. 'Management' of something usually means that you should be able to control it, Stop or Start it, Expand or Shrink it, C
reate More, or Create less of it. But when it comes to 'Time' these basic management principles do not work, yet 'Time Management'
is considered one of the most essential skills for success.
...............RSA ALGORITHM...................
Enter a prime number required to generate public and private keys: 23
Enter a prime number required to generate public and private keys: 53
The public key in RSA is: (3, 1219)
The private key in RSA is: (763, 1219)
```

```
............The Encryption/Decryption process.............

Select an operation:
1. Encrypt
2. Decrypt
3. Authenticate User
4.Exit
Enter your choice: 1
...........Encrypt the E-mail content using AES Algorithm...........
The encrypted E-mail content: km7SvfjZeR5idN7J7Nf7N18grMVrj2hYErBgQ9sePnsQS20PNp5c38dZ+vzZ9hZhOOQ9tQT1q1CeiYBFI/o8oT3TuCcyMUF+iZKqx
wlH6u0gdLlCY9+Ka8KVGewB1clpN84CtGK9CkWOQYQ3Mx0ojjGUbXL1jUngggM6marMJUTQ0IxZ0+KDdHGFIu22SQaHKEuoSZ9UxKxTpsca35hEPLIaUAOrw+K5hPd8DXPP
qNocSpMAo+DNUEdWHZ5CuGrpp88wu4Z/PH57Z/Qr6vTjQoMn0vtQLyDNPSyIN+xDA7SMXydOqloYiHQ5/pTDy3dKOWmhJpgYU/4L41RWgaS9BvoRZIWROkpK1+GHfCja1sb
MCi9oVBQBUovPadCYsVXaDdoj63jldcKl9ozjh2EL7Xw3uBFlg06ydyFiUEDb5r5frR6AWBWNxNxpHtjEEr2Y9HrxQ5x9Ou9TPlgV2+gAxT+UARq6+OwYl3QX0VhyDFDuRf
7MP6GeJRE9N5nfmX1BXLWr5cvgyebUwxflsxCCqHILJU1n312ySOOXl/Dfgsnd8H6TUUPsY9EWWT+M7lRA82lL1lNIikQk2YS1lReJ2pw==
...........Encrypt the AES key using RSA Algorithm.............
The AES key is converted to integer for encryption: 1074
The encrypted AES key: 94
```

Experiment Screenshots

ANURAG UNIVERSITY

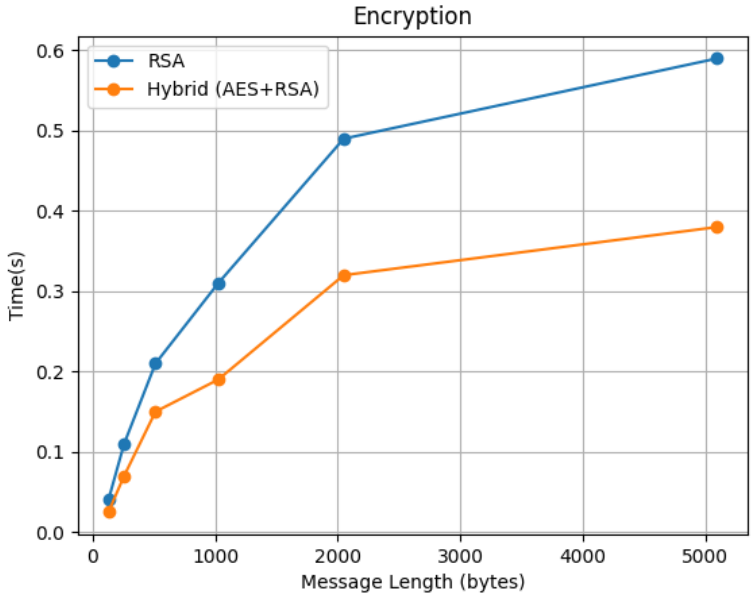
```
Select an operation:
1. Encrypt
2. Decrypt
3. Authenticate User
4.Exit
Enter your choice: 2
............Decrypt the E-mail content using AES Algorithm............
The decrypted E-mail content: Greetings from Infosys Springboard!Is Time Management a Hoax? Or is it a Paradox? At best it can just
be a misnomer. 'Management' of something usually means that you should be able to control it, Stop or Start it, Expand or Shrink it
, Create More, or Create less of it. But when it comes to 'Time' these basic management principles do not work, yet 'Time Managemen
t' is considered one of the most essential skills for success.
............Decrypt the AES key using RSA Algorithm............
The decrypted AES key: 1074
```

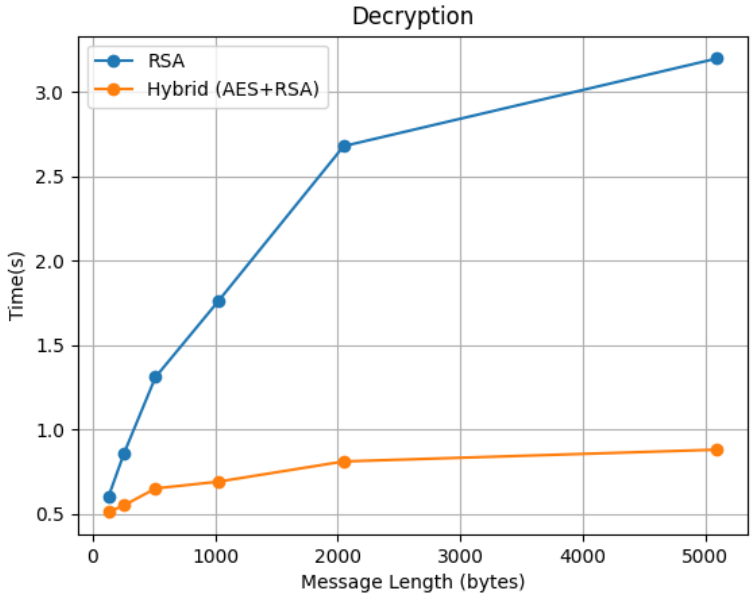
```
Select an operation:
1. Encrypt
2. Decrypt
3. Authenticate User
4.Exit
Enter your choice: 3
............Authenticating the user using Digital Signature..........
The generated hash value from the message for digital signature: b'\x0e\x9f\x0e\xd0\n\x87\x84\x19\xa2\x82m\xf40\x0f\xb3\xb6\xbc\x7f
/r!\x13\x87\xae\\c\xefRzs05'
The hash value which should be encrypted is: 1037
The encrypted signature is: 1189
The decrypted signature is: 1037
Is the signature valid? True

Select an operation:
1. Encrypt
2. Decrypt
3. Authenticate User
4.Exit
Enter your choice: 4
```

Experiment Results

Message length	RSA Encryption time	RSA Decryption time	Hybrid Encryption time	Hybrid Decryption time
128 bits	0.04 sec	0.60 sec	0.025 sec	0.51 sec
256 bits	0.11 sec	0.86 sec	0.07 sec	0.55 sec
512 bits	0.21 sec	1.31 sec	0.15 sec	0.65 sec
1024 bits	0.31 sec	1.76 sec	0.19 sec	0.69 sec
2048 bits	0.49 sec	2.68 sec	0.32 sec	0.81 sec
5096 bits	0.59 sec	3.2 sec	0.38 sec	0.88 sec

Experiment Results

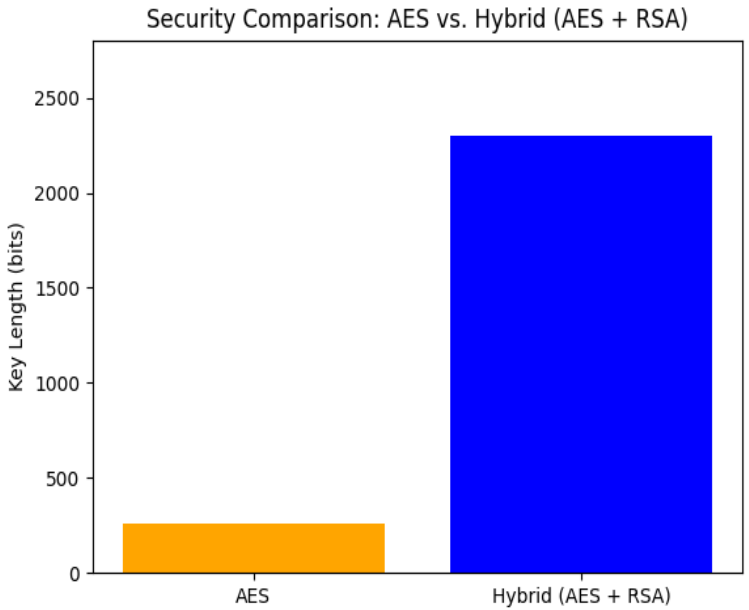




Experiment Results

From above experiment we can say that RSA computational time is more when compared to Hybrid algorithm. But the problem with using AES algorithm is prone to some attacks which are solved by using Hybrid algorithm.

Type of Attack	AES algorithm	Hybrid algorithm
Brute-force attack	As we are using 256 bits key length, there 2^{256} possible ways to identify which a hacker can find out.	We are using RSA to encrypt the AES key. If we use larger keys it is very difficult to decode the key, which ensures double security.
Man-in-the-middle attack	If an intruder eavesdrop on the sender and identify the key, it is very easy for him to decrypt the message	Even if the intruder eavesdrop there is no use because he cannot decrypt until he knows the receiver's private key



Findings

- Confidentiality: AES encryption ensures that only the intended recipient can decrypt and read the email, maintaining confidentiality.
- Authentication: RSA-based digital signatures can authenticate the sender of the email, confirming that the message indeed comes from the claimed source.
- Secure Key Exchange: The RSA algorithm can be used for secure key exchange, ensuring that both the sender and recipient share a secret key without interception.
- Mitigation of Man-in-the-Middle Attacks: By combining RSA with digital signatures, you can mitigate man-in-the-middle attacks, where an attacker intercepts and modifies email content.

Overall, the project ensures that email communication is secure, private, and trustworthy.

Justification

- **The parameters that are improved in our project**
  Security, Key Exchange mechanism, Speed and Performance
- **Mathematical Formula**
  Encryption time of Hybrid= Encryption time of AES+ Encryption time of RSA
  Decryption time of Hybrid= Decryption time of AES+ Decryption time of RSA
  Key length of Hybrid= Key length of AES+ Key length of RSA

```
import time
start_time=time.time()
encrypted_message = encrypt(key, message)
end_time=time.time()
encryption_time_aes=end_time-start_time
```

In what way parameters are improved

- **Security:** It provides confidentiality to the message using AES and authentication using RSA
- **Key Management:** Keys are managed and the keys are transferred securely using RSA.
- **Speed and Performance:** RSA takes longer time to encrypt large amount of data that's why we are using RSA only for key management, which ensures the less encryption/decryption time.