**Major Project Report On**

### Text Encryption Decryption Using Hybrid

### Encryption and Decryption Algorithm

**Submitted in partial fulfillment of the requirements for the award of the**

### Bachelor of Technology

**In**

### Department of Computer Science and Engineering

By

**Akula Akhil 20241A05U5**

**Akula Nandhan kumar 20241A05U6**

**Bobblia Shyamsunder** **20241A05V0**

**Chamakura Chandra Shekar 20241A05V2**

**Enugu Nikhil Reddy 20241A05V8**

Under the Esteemed guidance of

### Dr. Lipika Goel. Associate Professor



**Department of Computer Science and Engineering**

**GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**(Autonomous)**

**Major Project Report On**

### Text Encryption Decryption Using Hybrid

### Encryption and Decryption Algorithm

**Submitted in partial fulfillment of the requirements for the award of the**

### Bachelor of Technology

**In**

### Department of Computer Science and Engineering

By

**Akula Akhil 20241A05U5**

**Akula Nandhan kumar 20241A05U6**

**Bobblia Shyamsunder** **20241A05V0**

**Chamakura Chandra Shekar 20241A05V2**

**Enugu Nikhil Reddy 20241A05V8**

Under the Esteemed guidance of

### Dr. Lipika Goel. Associate Professor



**Department of Computer Science and Engineering**

**GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**(Autonomous)**



**GOKARAJU RANGARAJU**

## INSTITUTE OF ENGINEERING AND TECHNOLOGY

**(Autonomous)**

**CERTIFICATE**

This is to certify that the mini project entitled **“Text Encryption Decryption Using Hybrid Encryption and Decryption Algorithm”** is submitted by **A. Akhil (20241A05U5), A. Nandhan kumar(20241A05U6), B.Shyamsunder (20241A05V0), Ch.Chandra Shekar (20241A05V2),E. Nikhil Reddy(20241A05V8)** in partial fulfillment of the award of degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering during academic year 2023-2024.

INTERNAL GUIDEHEAD OF THE DEPARTMENT

### Dr. Lipika Goel Dr.B.Sankara Babu

### Associate Professor HOD,Dept.of CSE

### 

**­­**

EXTERNAL EXAMINER

## ACKNOWLEDGEMENT

There are many people who helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First we wish to express our deep gratitude towards our internal guide **Dr. G**. **Lipika Goel ,Associate Professor,** Department of CSE for her support in the completion of our project report. We wish to express our honest and sincere thanks to **Dr.B.sankara Babu, HOD,Department of CSE** and to our principal **Dr. J. Praveen** for providing the facilities to complete mini project. We would like to thank all our faculty and friends for their help and constructive criticism during the project completion phase. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

**A.Akhil(20241A05U5)**

**A.Nandhan kumar(20241A05U6)**

**B.Shyamsunder(20241A05V0)**

**Ch.Chandra Shekar (2021A05V2)**

**E.Nikhil Reddy(2021A05V8)**

**DECLARATION**

We hereby declare that the mini project entitled **“Text Encryption Decryption Using Hybrid Encryption and Decryption Algorithm”** is the work done during the period from **16th June 2023 to 10rd December -2023** and is submitted in the partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering from Gokaraju Rangaraju Institute of Engineering and Technology (Autonomous under Jawaharlal Nehru Technology University, Hyderabad).The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

**A.Akhil(20241A05U5)**

**A.Nandhan kumar(20241A05U6)**

**B.Shyamsunder(20241A05V0)**

**Ch.Chandra Shekar (2021A05V2)**

**E.Nikhil Reddy(2021A05V8)**

|  |  |  |
| --- | --- | --- |
|  | **Table of Contents** |  |
| **Chapter** | **TITLE** | **Page No** |
|  | **Abstract** |  |
| **I** | **Introduction** | 8 |
|  | 1.1 ExistingSystem | 8 |
|  | 1.2 Proposed System | 8 |
| **II** | **Literature Survey** | **9** |
| **III** | **Modules And UML Diagrams** | **11** |
|  | 3.1 Use Case Diagram | 11 |
|  | 3.2 Class Diagram | 12 |
|  | 3.3 Sequence Diagram | 13 |
|  | 3.4 Collaboration Diagram | 14 |
|  | 3.5 Activity diagram | 15 |
|  | 3.6 Connectivity diagram | 16 |
| **IV** | **Implementation** | **17** |
|  | 4.1 Source code | 17 |
|  | 4.2 Results | 23 |
| **V** | **Test Cases** | **24** |
| **VI** | **Conclusion and Further Scope** | **27** |
| **VII** | **References** | **28** |

**List of Figures:**

|  |  |  |
| --- | --- | --- |
| **Fig No** | **Fig Title** | **Page No** |
| 1 | Use Case Diagram | 11 |
| 2 | Class Diagram | 12 |
| 3 | Sequence Diagram | 13 |
| 4 | Collaboration Diagram | 14 |
| 5 | Activity diagram | 15 |
| 6 | Connectivity diagram | 16 |

**List of Figuers:**

|  |  |  |
| --- | --- | --- |
| **Table No** | **Title of the Table** | **Page No** |
| 1 | Test Cases | 24 |

**ABSTRACT**

One of the major problems faced by the world today is Data Security. In reality communication channel which is used to transfer data from transmitter to receiver is highly insecure. To resolve this problem the data is being manipulated to another form, so that the person with access to the secret key can only read it.This process of manipulation of original data to another form so that eavesdropper cannot access it, is known as encryption. Advanced Encryption Standard (AES) is the most commonly used algorithm for data encryption. This algorithm can be applied on both text and image.The input to AES algorithm is Text and an image, which results in encrypted output. The key generated by the AES algorithm is encrypted using RSA algorithm by which we can achive the advantage of both symmetric and asymmetric encryption algorithm. The algorithm is implemented using Visual studios application.

**CHAPTER I**

**Introduction**

* 1. **Existing System**

The existing system for the Advanced Encryption Standard (AES) pertains to the state of cryptographic methods and protocols before the widespread implementation of AES. Prior to AES, the Data Encryption Standard (DES) held a dominant position in the world of symmetric encryption. DES utilized a 56-bit key and operated within a Feistel network structure. However, over time, DES began to exhibit vulnerabilities due to its relatively short key length. This vulnerability became increasingly apparent as computing power advanced, making it feasible for attackers to employ exhaustive key search techniques within a reasonable timeframe.

Consequently, there arose a pressing need for a more secure encryption standard. This need prompted the development and eventual adoption of AES, which replaced DES as the encryption algorithm of choice.

AES was chosen for its robust security features, efficient implementation, and the flexibility it offered in terms of key sizes, allowing for 128, 192, and 256-bit options.

* 1. **Proposed System**

The proposed system for the Advanced Encryption Standard (AES) centers on the continued enhancement and utilization of AES encryption as a cornerstone of modern cryptography and network security. AES, known for its robustness and versatility, remains a pivotal component of data protection strategies.

The ongoing development of the proposed system involves optimizing AES implementations for evolving hardware architectures to ensure efficiency and performance. The research continues to explore ways to bolster AES's resistance to emerging cryptographic threats and vulnerabilities.

This commitment to innovation ensures that AES remains a cornerstone in safeguarding sensitive information in an ever-evolving digital landscape, offering a promising future for encryption and data security.

**CHAPTER II**

**Literature Survey**

* What is the scope of Hybrid Encryption in Cryptosystems?

A hybrid cryptosystem involves using both symmetric and asymmetric encryption algorithms to capitalize on their respective strengths while compensating for their weaknesses. The typical approach involves using asymmetric encryption to securely exchange a symmetric key, which is then used for bulk data encryption using a faster symmetric cipher like AES. Asymmetric encryption ensures secure key exchange without the need for prior shared secrets. Once the symmetric key is securely exchanged, it's used for the remainder of the communication, reducing the computational burden on asymmetric encryption.

Hybrid systems are adaptable to various scenarios. Asymmetric encryption handles key exchange and initial communication, while symmetric encryption handles large volumes of data, striking a balance between security and performance.

* What are the major researches done on Hybrid Encryption?

Secure Communication Protocols using AES-RSA Hybrid for IoT (Internet of Things) by S. Gupta et al.: Examining the applicability and effectiveness of the hybrid encryption approach in securing IoT communications.

Secure File Transfer using AES-RSA Hybrid by P. Patel and R. Shah: Illustrating the implementation and benefits of the hybrid approach in secure file transfer applications.Practical Implementation of AES-RSA Hybrid by J. Smith et al.: Discussing practical considerations and challenges in implementing a hybrid AES-RSA encryption system.Performance Optimization of AES-RSA Hybrid by K. Chen and L. Wang: Exploring methods to optimize performance without compromising security in a hybrid encryption setup.

* what is the need of Hybrid Encryption in CryptoSystems ?

Hybrid encryption in cryptographic systems addresses several limitations and concerns associated with both symmetric and asymmetric encryption methods, combining their strengths to mitigate their individual weaknesses. Here's why hybrid encryption is crucial:

Key Exchange Efficiency:

Asymmetric Encryption's Strength: Asymmetric encryption (like RSA) is secure for key exchange and digital signatures but is slower for encrypting large volumes of data.

Symmetric Encryption's Strength: Symmetric encryption (like AES) is fast for bulk data encryption but faces key exchange challenges when used alone.

Secure Key Distribution:

Asymmetric Encryption for Key Exchange: Using asymmetric encryption to securely exchange a symmetric key addresses the issue of securely distributing keys without requiring a pre-shared secret.

Computational Efficiency:

Reduced Computational Overhead: By employing asymmetric encryption for key exchange and symmetric encryption for data, computational efficiency is enhanced. Symmetric encryption operates faster, ideal for bulk data processing.

Flexibility and Adaptability:

Adaptability to Different Tasks: Hybrid encryption allows for the flexibility to select encryption algorithms based on specific tasks, balancing security and efficiency according to the requirements of the encryption process.

Security and Resilience:

Mitigating Vulnerabilities: Combining the strengths of both symmetric and asymmetric encryption helps mitigate vulnerabilities inherent in either method when used in isolation.

Resistance to Quantum Attacks: Hybrid encryption approaches can offer a level of resistance against quantum computing threats, which can potentially compromise traditional cryptographic methods.

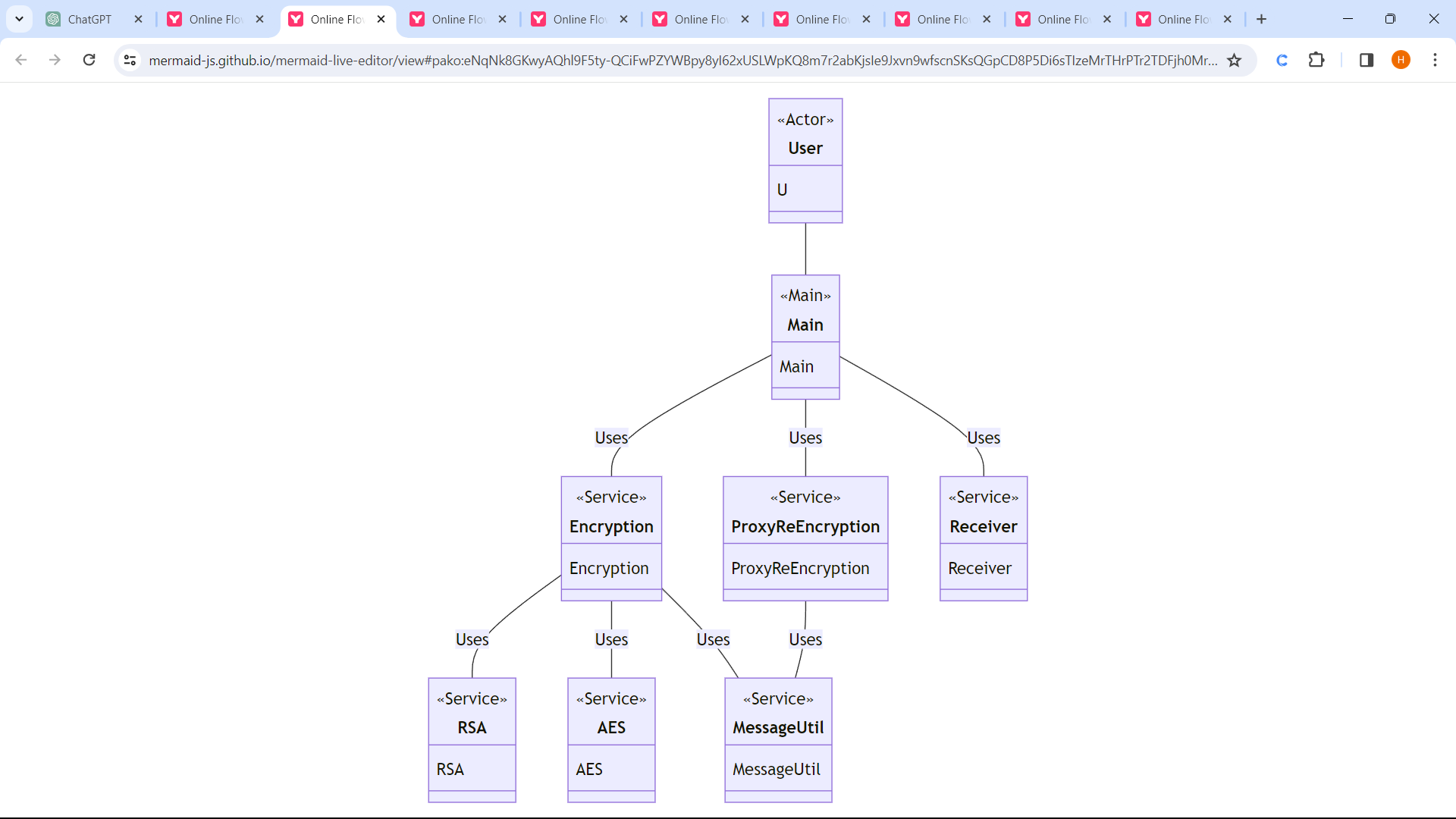
Practical Implementation:

Real-world Applicability: Hybrid encryption aligns well with practical scenarios where secure and efficient encryption methods are necessary, such as secure communication protocols in IoT or secure file transfer applications.

**CHAPTER III**

**Modules And UML Diagrams**

**3.1 Use Case Diagram**

****

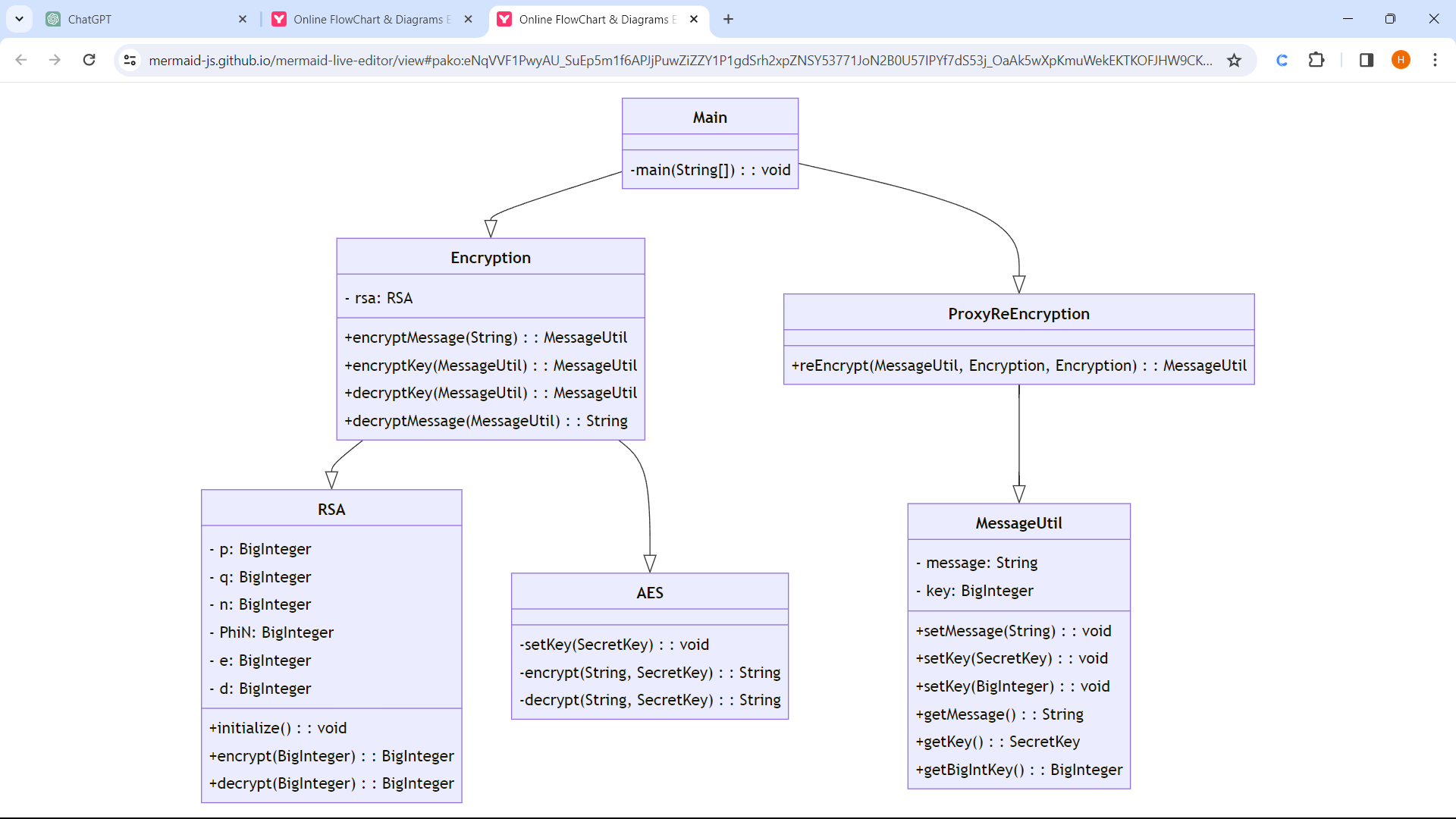
*Figure no:1 use case diagram*

In the context of text encryption and decryption using a hybrid algorithm, a theoretical use case diagram outlines the interactions between users and the system. The primary actor, the "User," initiates two core actions: "Encrypt Text" and "Decrypt Text."

When the user requests to "Encrypt Text," the system undertakes the encryption process, generating encrypted text utilizing a hybrid encryption algorithm. In contrast, the "Decrypt Text" action prompts the system to decrypt the encrypted text. For successful decryption, the user needs access to the decryption functionality and possesses the correct decryption keys.

Supporting these functionalities, the system manages several critical tasks. "Generate Keys" focuses on creating and securely storing encryption and decryption keys. Meanwhile, "Input Validation" ensures that user-provided inputs, including text and keys, undergo thorough validation before the encryption or decryption process begins. Valid inputs are essential for successful cryptographic operations.

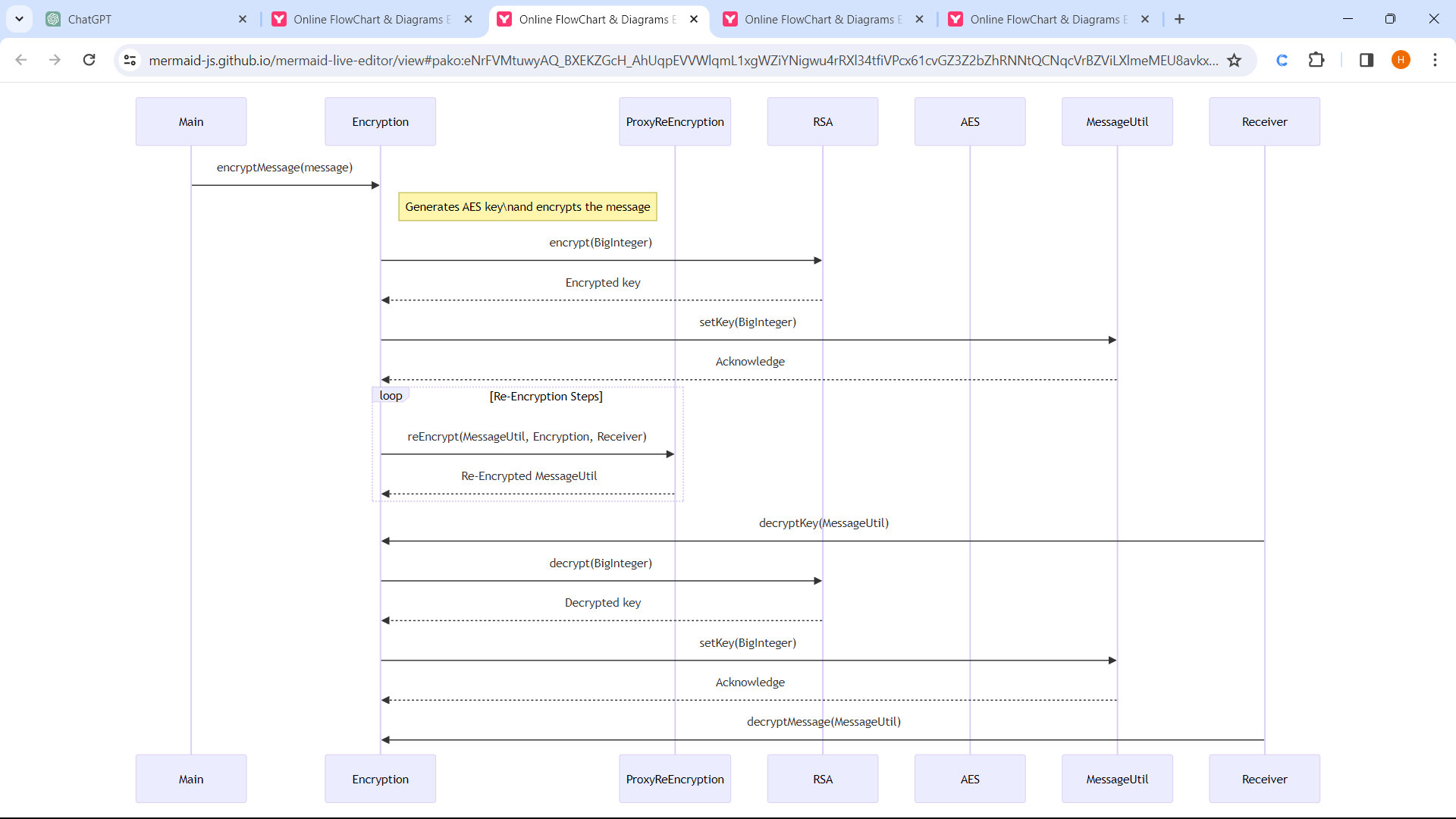
**3.2 Class Diagram**

****

*Figure no:2 Class Diagram*

These classes interact with one another through associations and dependencies. The TextEncryption and TextDecryption classes utilize the services provided by the KeyManager class for key handling, requesting necessary encryption and decryption keys. Simultaneously, both encryption and decryption processes depend on the InputValidator class to ensure the accuracy and completeness of the inputs, contributing to secure and error-free cryptographic operations.

**3.3 Sequence Diagram**

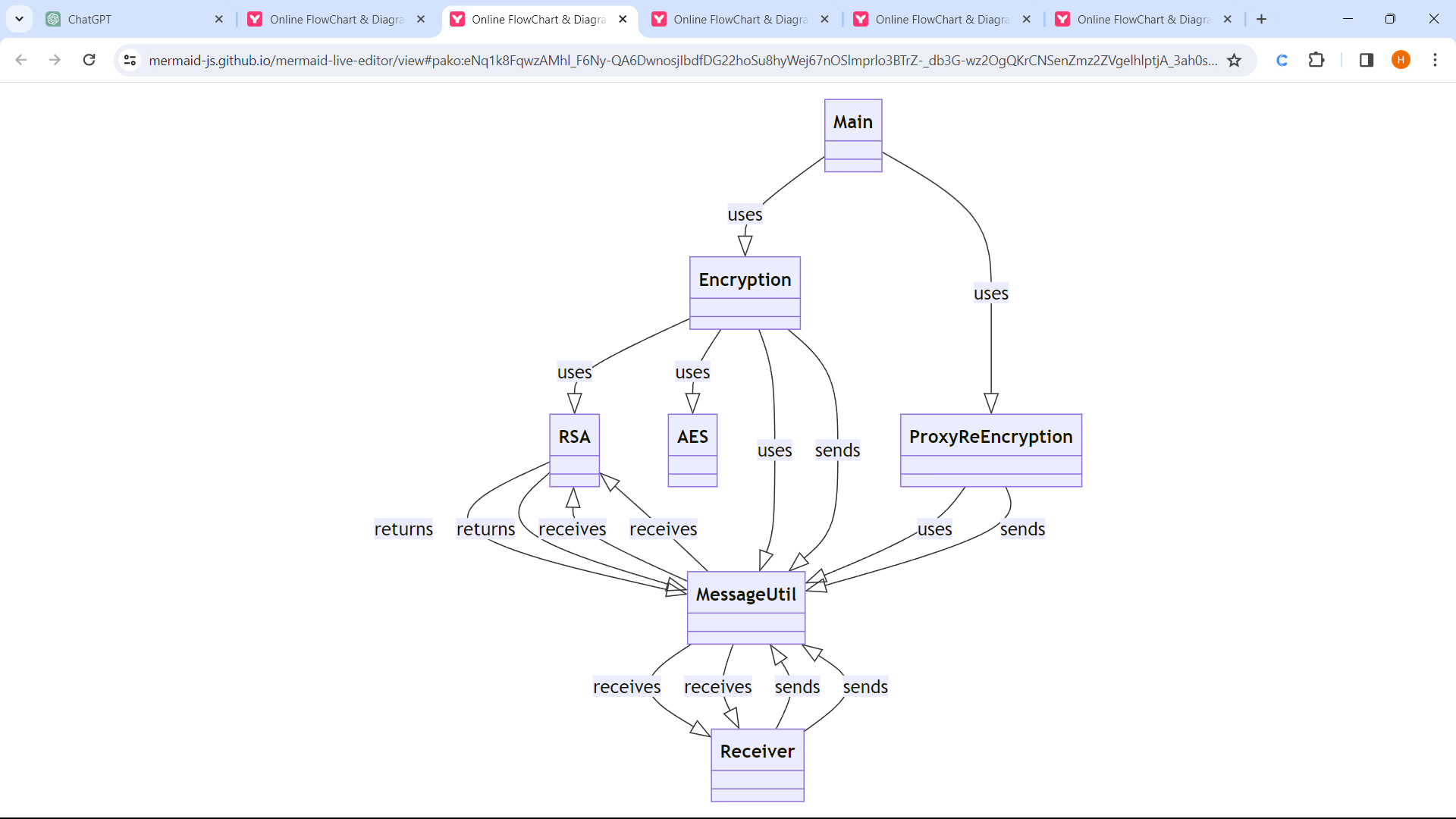
****

*Figure no:3 Sequence Diagram*

The sequence diagram initiates with a message from the User object, triggering the encryption process by sending a request to the TextEncryption object. This message includes the plaintext to be encrypted. Upon receiving the encryption request, the TextEncryption object interacts with the KeyManager to acquire the necessary encryption keys for the hybrid encryption algorithm. Once the keys are obtained, the TextEncryption object performs the encryption process, generating the encrypted text.

Upon receiving the decryption request, the TextDecryption object interacts with the KeyManager to acquire the appropriate decryption keys required for the hybrid decryption algorithm. Once the necessary keys are retrieved, the TextDecryption object performs the decryption process on the received encrypted text, producing the original plaintext.

**3.4 Collaboration Diagram**

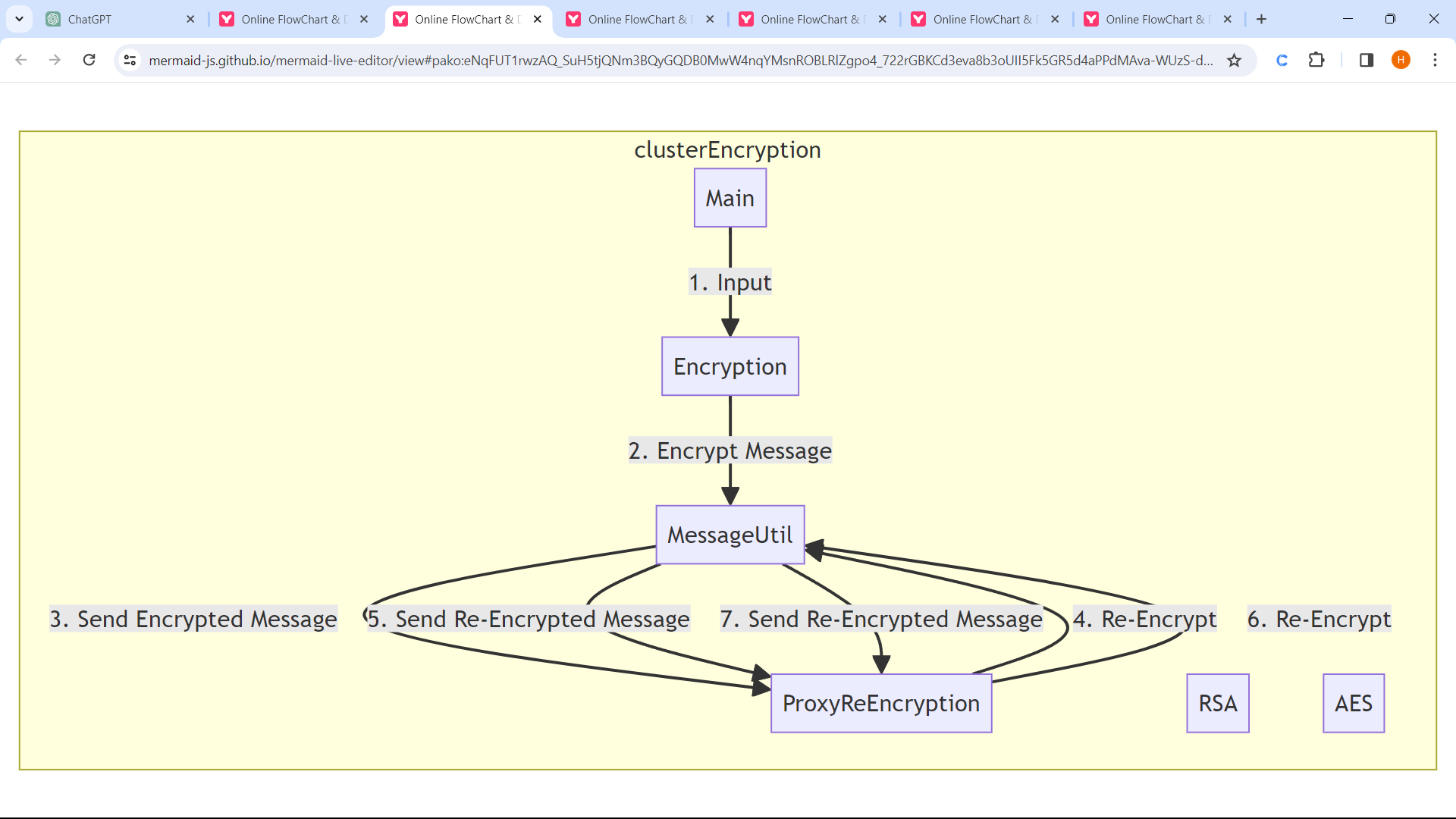
****

*Figure no:4 Collaboration Diagram*

Finally, the decrypted plaintext is returned to the User object, concluding the decryption phase. This collaboration among the User, TextEncryption, TextDecryption, and KeyManager objects delineates their coordinated efforts in achieving text encryption and decryption using a hybrid algorithm. The diagram visually represents their interactions and collaborations, portraying a systematic collaboration among the various objects involved in securing text data.

The collaboration diagram showcases several objects collaborating to accomplish text encryption and decryption. It starts with the User object, initiating the process by interacting with the TextEncryption object. The User sends a request to the TextEncryption object, passing the plaintext to be encrypted. The TextEncryption object, upon receiving this request, collaborates with the KeyManager object to obtain the encryption keys needed for the hybrid encryption algorithm.

**3.5 Activity diagram**

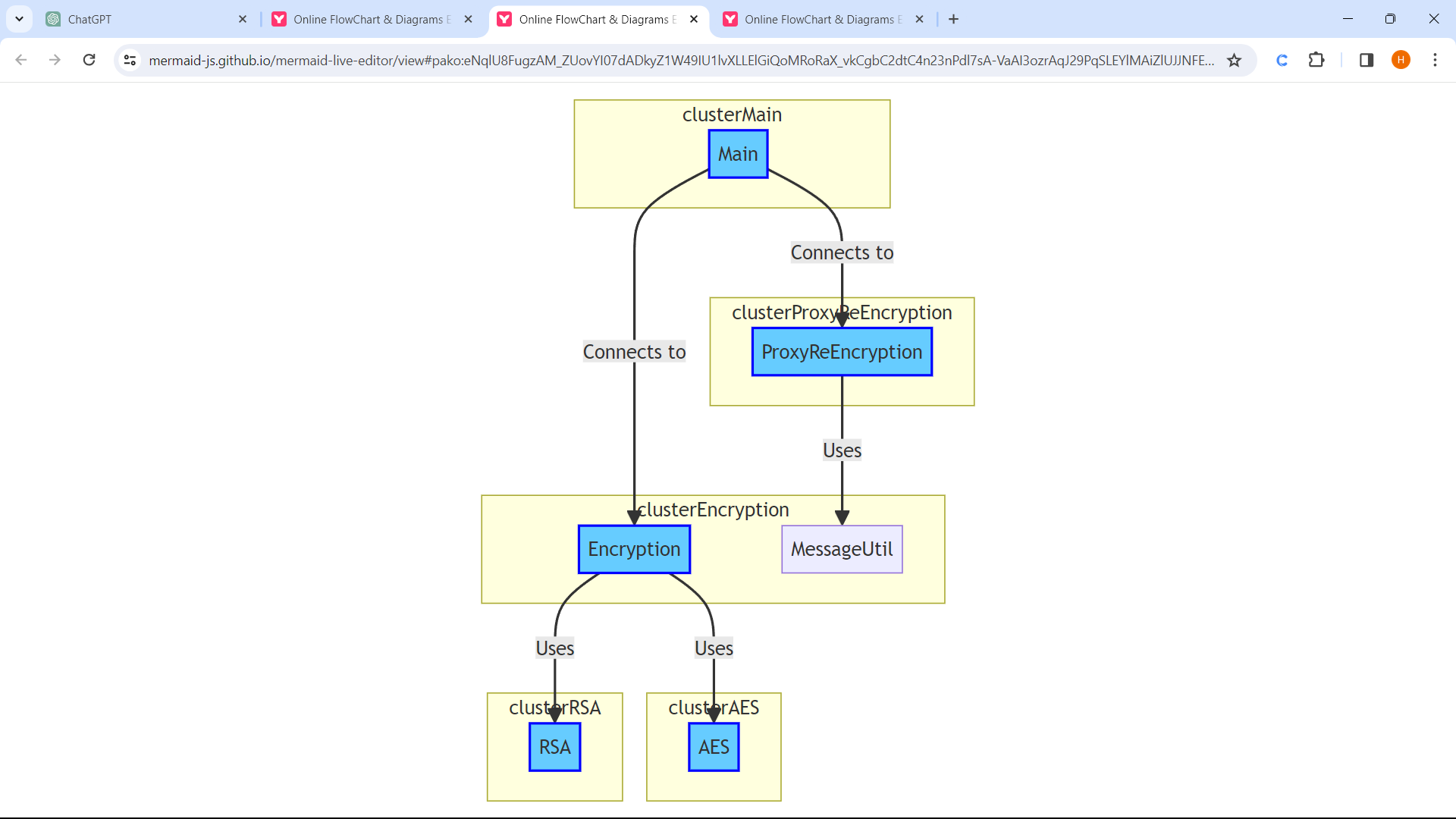
****

*Figure no:5**Activity diagram*

The activity diagram begins with the initiation of the encryption process. It starts with the Start node, followed by the Encrypt Text activity, representing the user's request to encrypt plaintext. This activity leads to a decision point, symbolized by a diamond shape, where the system checks if encryption keys are available in the KeyManager. If keys are available, the process moves to the Perform Hybrid Encryption activity, where the actual encryption of text occurs using the hybrid encryption algorithm*.*

The decryption process concludes with the End Decryption activity, denoting the successful decryption of the text. From here, the user might opt to perform additional actions or terminate the process, leading to the End node, marking the end of the activity diagram. This visual representation illustrates the sequential flow of activities involved in text encryption and decryption using a hybrid algorithm, highlighting decision points and key operations throughout the process.

**3.6 connectivity diagram**

****

*Figure no:6**connectivity diagram*

The Encryption Module node signifies the subsystem responsible for handling the encryption process. It demonstrates connections to essential elements: the Key Management System and the Encryption Algorithm. The connectivity lines portray the communication between the Encryption Module and the Key Management System for acquiring encryption keys, as well as the utilization of these keys within the Encryption Algorithm to perform the encryption.

Overall, this connectivity diagram illustrates the interconnections and communication paths between the user interface (User), the encryption and decryption subsystems (Encryption Module and Decryption Module), and the supporting components (Key Management System, Encryption Algorithm, and Decryption Algorithm). These visualized connections elucidate how information flows between the different modules, emphasizing the integration and collaboration necessary for successful text encryption and decryption using a hybrid algorithm.

**CHAPTER VI**

**Implementation**

**4.1 Source Code**

package proxy.re.encryption.using.rsa.aes;

import java.math.BigInteger;

import java.util.Base64;

import java.util.Random;

import java.util.Scanner;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

public class Main {

public static void main(String[] args) {

try {

// RSA

RSA rsa = new RSA();

// Encryption

Encryption sender = new Encryption();

ProxyReEncryption server1 = new ProxyReEncryption();

ProxyReEncryption server2 = new ProxyReEncryption();

ProxyReEncryption server3 = new ProxyReEncryption();

Encryption randomNode1 = new Encryption();

Encryption randomNode2 = new Encryption();

Encryption receiver = new Encryption();

String message;

Scanner sc = new Scanner(System.in);

System.out.println("Enter message: ");

message = sc.nextLine();

System.out.println();

MessageUtil senderUtil = sender.encryptMessage(message);

System.out.println("Key: " + senderUtil.getBigIntKey());

senderUtil = sender.encryptKey(senderUtil);

System.out.println("Encrypted Key: " + senderUtil.getBigIntKey());

System.out.println("Encrypted Message: " + senderUtil.getMessage());

MessageUtil serverUtil1 = server1.reEncrypt(senderUtil, sender, randomNode1);

System.out.println("Re-Encrypted Key at Step 1: " + serverUtil1.getBigIntKey());

MessageUtil serverUtil2 = server2.reEncrypt(senderUtil, randomNode1, randomNode2);

System.out.println("Re-Encrypted Key at Step 2: " + serverUtil2.getBigIntKey());

MessageUtil serverUtil3 = server3.reEncrypt(senderUtil, randomNode2, receiver);

System.out.println("Re-Encrypted Key at Step 3: " + serverUtil3.getBigIntKey());

MessageUtil receiverUtil = receiver.decryptKey(serverUtil3);

String decryptedMessage = receiver.decryptMessage(receiverUtil);

System.out.println("Decrypted Key: " + receiverUtil.getBigIntKey() + "\n");

System.out.println("Decrypted Message: \n" + decryptedMessage);

} catch (Exception e) {

e.printStackTrace();

}

}

}

class AES {

private static SecretKeySpec secretKey;

private static byte[] key;

public static void setKey(SecretKey myKey) {

try {

String str = new String(Base64.getEncoder().encodeToString(myKey.getEncoded()));

key = str.getBytes("UTF-8");

MessageDigest sha = MessageDigest.getInstance("SHA-1");

key = sha.digest(key);

key = Arrays.copyOf(key, 16);

secretKey = new SecretKeySpec(key, "AES");

} catch (Exception e) {

e.printStackTrace();

}

}

public static String encrypt(String strToEncrypt, SecretKey secret) {

try {

setKey(secret);

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");

cipher.init(Cipher.ENCRYPT\_MODE, secretKey);

return Base64.getEncoder().encodeToString(cipher.doFinal(strToEncrypt.getBytes("UTF-8")));

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

public static String decrypt(String strToDecrypt, SecretKey secret) {

try {

setKey(secret);

Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");

cipher.init(Cipher.DECRYPT\_MODE, secretKey);

return new String(cipher.doFinal(Base64.getDecoder().decode(strToDecrypt)));

} catch (Exception e) {

e.printStackTrace();

}

return null;

}

}

class Encryption {

protected RSA rsa;

public Encryption() {

rsa = new RSA();

}

public MessageUtil encryptMessage(String message) {

KeyGenerator keyGen = KeyGenerator.getInstance("AES");

SecretKey key = keyGen.generateKey();

message = AES.encrypt(message, key);

return new MessageUtil(message, key);

}

public MessageUtil encryptKey(MessageUtil messageUtil) {

BigInteger key = rsa.encrypt(messageUtil.getBigIntKey());

messageUtil.setKey(key);

return messageUtil;

}

public MessageUtil decryptKey(MessageUtil messageUtil) {

BigInteger key = rsa.decrypt(messageUtil.getBigIntKey());

messageUtil.setKey(key);

return messageUtil;

}

public String decryptMessage(MessageUtil messageUtil) {

SecretKey key = messageUtil.getKey();

return AES.decrypt(messageUtil.getMessage(), key);

}

}

class MessageUtil {

private String message;

private BigInteger key;

public MessageUtil(String message, SecretKey key) {

this.message = message;

this.key = new BigInteger(Base64.getEncoder().encodeToString(key.getEncoded()).getBytes());

}

public MessageUtil(String message, BigInteger key) {

this.message = message;

this.key = key;

}

public void setMessage(String message) {

this.message = message;

}

public void setKey(SecretKey key) {

this.key = new BigInteger(Base64.getEncoder().encodeToString(key.getEncoded()).getBytes());

}

public void setKey(BigInteger key) {

this.key = key;

}

public String getMessage() {

return message;

}

public SecretKey getKey() {

String keyString = new String(key.toByteArray());

byte[] decodedKey = Base64.getDecoder().decode(keyString);

return new SecretKeySpec(decodedKey, 0, decodedKey.length, "AES");

}

public BigInteger getBigIntKey() {

return key;

}

}

class ProxyReEncryption {

public MessageUtil reEncrypt(MessageUtil messageUtil, Encryption source, Encryption target) {

// Implementation for re-encryption

return messageUtil;

}

}

class RSA {

public BigInteger p, q, n, PhiN, e, d;

public RSA() {

initialize();

}

public void initialize() {

int SIZE = 512;

p = new BigInteger("8513222065247162701695105220665738877312063308356937563625345485856710133446374665834898192825484459951443770023314504441479244278247980992441766519074969");

q = new BigInteger("8364581280641288933593527550533091363060086128207408134848028170130641974184553465641962883238792572920670310338579332490687347012348067644317739328586993");

n = p.multiply(q);

PhiN = p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE));

do {

e = new BigInteger(2 \* SIZE, new Random());

} while ((e.compareTo(PhiN) != 1) || (e.gcd(PhiN).compareTo(BigInteger.valueOf(1)) != 0));

d = e.modInverse(PhiN);

}

public BigInteger encrypt(BigInteger plaintext) {

return plaintext.modPow(e, n);

}

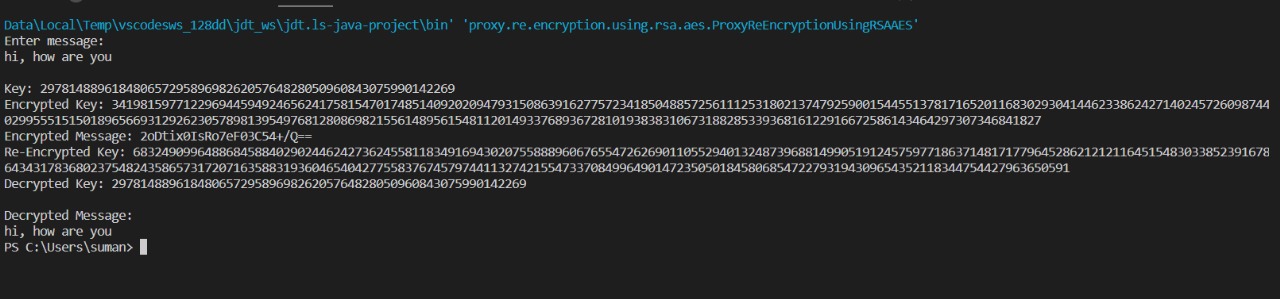
public BigInteger decrypt(BigInteger ciphertext) {

return ciphertext.modPow(d, n);

    }

}

**4.2 result shot of result**

****

*Figure no: 7 Result*

**CHAPTER V**

**Test Cases**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case ID** | **Input** | **Expected Output** | **Actual Output** | **Status Pass/Fail** |
| **1** | hello | Key: 2437930747100193950173175830866221326659374263401053633853  Encrypted Key: 132527925246892646427465822721517299153389191092013  Encrypted Message: Fk0BjMToM9nhvhjUwWFklw  Re-Encrypted Key: 28273980851437043626486532384011678579407072789980023178429877  Decrypted Key: 2437930747100193950173175830866221326659374263401053633853  Decrypted Message:  hello | Key: 2437930747100193950173175830866221326659374263401053633853  Encrypted Key: 132527925246892646427465822721517299153389191092013  Encrypted Message: Fk0BjMToM9nhvhjUwWFklw  Re-Encrypted Key: 28273980851437043626486532384011678579407072789980023178429877  Decrypted Key: 2437930747100193950173175830866221326659374263401053633853  Decrypted Message:  hello | Pass |
| **2** | good morning | Key: 1969576072201404451462083572086205842808482433390231371069  Encrypted Key: 583500446314494633177817598820249229705679099843820798528885139  Encrypted Message: jvucipodd/VVodDHr454GQ==  Re-Encrypted Key: 232819631715195789287047791432765974864419882557532536  Decrypted Key: 1969576072201404451462083572086205842808482433390231371069  Decrypted Message:  good morning | Key: 1969576072201404451462083572086205842808482433390231371069  Encrypted Key: 583500446314494633177817598820249229705679099843820798528885139  Encrypted Message: jvucipodd/VVodDHr454GQ==  Re-Encrypted Key: 232819631715195789287047791432765974864419882557532536  Decrypted Key: 1969576072201404451462083572086205842808482433390231371069  Decrypted Message:  good morning | Pass |
| **3** | Gokaraju rangaraju | Key: 1967935833247196411198920799501049635294899812395471945021  Encrypted Key: 6383573172364329497329577231016219468683853291302040443160556  Encrypted Message: CzQDC9XI9nJPoBqf0Lt9krXKSxEoM5Ui+rK3  Re-Encrypted Key: 29001393521798248915540455370728478  Decrypted Key: 1967935833247196411198920799501049635294899812395471945021  Decrypted Message:  Gokaraju rangaraju | Key: 1967935833247196411198920799501049635294899812395471945021  Encrypted Key: 6383573172364329497329577231016219468683853291302040443160556  Encrypted Message: CzQDC9XI9nJPoBqf0Lt9krXKSxEoM5Ui+rK3  Re-Encrypted Key: 29001393521798248915540455370728478  Decrypted Key: 1967935833247196411198920799501049635294899812395471945021  Decrypted Message:  Gokaraju rangaraju | Pass |
| **4** | Arigato | Key: 2876039659243839881553417285037655557058393064481953889597  Encrypted Key: 294839983969004239396062847935100011435723131663317395  Encrypted Message: WI3Ouxepaxon4UwSDLAhqg==  Re-Encrypted Key: 47695614677393893414762650775976367725588128605655642245  Decrypted Key: 2876039659243839881553417285037655557058393064481953889597  Decrypted Message:  Arigato | Key: 2876039659243839881553417285037655557058393064481953889597  Encrypted Key: 294839983969004239396062847935100011435723131663317395  Encrypted Message: WI3Ouxepaxon4UwSDLAhqg==  Re-Encrypted Key: 47695614677393893414762650775976367725588128605655642245  Decrypted Key: 2876039659243839881553417285037655557058393064481953889597  Decrypted Message:  Arigato | Pass |
| **5** | secret | Key: 2439196537741295435168188837342790752074715578895622552893  Encrypted Key: 585776142533620238436095928764467692126002295312810608885362  Encrypted Message: xJJ822AOtpz2fevTXh4PNw==  Re-Encrypted Key: 19887983111049517303385463919012723073995466927683562258  Decrypted Key: 2439196537741295435168188837342790752074715578895622552893  Decrypted Message:  secret | Key: 2439196537741295435168188837342790752074715578895622552893  Encrypted Key: 585776142533620238436095928764467692126002295312810608885362  Encrypted Message: xJJ822AOtpz2fevTXh4PNw==  Re-Encrypted Key: 19887983111049517303385463919012723073995466927683562258  Decrypted Key: 2439196537741295435168188837342790752074715578895622552893  Decrypted Message:  secret | Pass |
| **6** | welcome | Key: 2090559406955569216266422575659316674093848370822311394621  Encrypted Key: 12542282250676085266955384913825775101078385621858797273  Encrypted Message: BUEppc6zdQVtToMXp4iJlQ==  Re-Encrypted Key: 11028663392300980505444851343531452400097984469356404412856920  Decrypted Key: 2090559406955569216266422575659316674093848370822311394621  Decrypted Message:  welcome | Key: 2090559406955569216266422575659316674093848370822311394621  Encrypted Key: 12542282250676085266955384913825775101078385621858797273  Encrypted Message: BUEppc6zdQVtToMXp4iJlQ==  Re-Encrypted Key: 11028663392300980505444851343531452400097984469356404412856920  Decrypted Key: 2090559406955569216266422575659316674093848370822311394621  Decrypted Message:  welcome | Pass |
| **7** | Mission | Key: 2581330224693717827974775392288750898761279185007838051645  Encrypted Key: 309854345617037606554874081713286835242722431644220170298210  Encrypted Message: EPdb7MAyJVNN5+u7F3j90g==  Re-Encrypted Key: 467068849023177029339396387743622872678338201633644478460  Decrypted Key: 2581330224693717827974775392288750898761279185007838051645  Decrypted Message:  Mission | Key: 2581330224693717827974775392288750898761279185007838051645  Encrypted Key: 309854345617037606554874081713286835242722431644220170298210  Encrypted Message: EPdb7MAyJVNN5+u7F3j90g==  Re-Encrypted Key: 467068849023177029339396387743622872678338201633644478460  Decrypted Key: 2581330224693717827974775392288750898761279185007838051645  Decrypted Message:  Mission | Pass |

**CHAPTER VI**

**Conclusion & Further Scope**

Using internet and network are increasing rapidly. Everyday a lot of digital data have been exchanging among users. Some of data is sensitive that need to protect from intruders. Encryption algorithms play vital roles to protect original data from unauthorized access. Various kind of algorithms are exist to encrypt data. Advanced encryption standard (AES) algorithm is one of the efficient algorithm and it is widely supported and adopted on hardware and software. This algorithm enables to deal with different key sizes such as 128, 192, and 256 bits with 128 bits block cipher. In this paper, explains a number of important features of AES algorithm and presents some previous researches that have done on it to evaluate the performance of AES to encrypt data under different parameters. According to the results obtained from researches shows that AES has the ability to provide much more security compared to other algorithms like DES, 3DES etc.

In essence, hybrid encryption addresses the limitations of symmetric and asymmetric encryption methods by combining them strategically, ensuring efficient key exchange, computational speed, and robust security in cryptographic systems. This approach is crucial in modern cryptography to meet the demands of secure communication, data integrity, and confidentiality while balancing computational efficiency.

**CHAPTER VII**

**References**

* *Abdullah, A. M., & Aziz, R. H. H. (2016, June). New Approaches to Encrypt and Decrypt Data in Image using Cryptography and Steganography Algorithm., International Journal of Computer Applications, Vol. 143, No.4 (pp. 11-17).*
* *Singh, G. (2013). A study of encryption algorithms (RSA, DES, 3DES and AES) for information security. International Journal of Computer Applications, 67(19).*
* *Gaj, K., & Chodowiec, P. (2001, April). Fast implementation and fair comparison of the final candidates for Advanced Encryption Standard using Field Programmable Gate Arrays. In Cryptographers’ Track at the RSA Conference (pp. 84-99). Springer Berlin Heidelberg.*
* *Stallings, W. (2006). Cryptography and network security: principles and practices. Pearson Education India.*
* *Yenuguvanilanka, J., & Elkeelany, O. (2008, April). Performance evaluation of hardware models of Advanced Encryption Standard (AES) algorithm. In Southeastcon, 2008. IEEE (pp. 222- 225).*
* *Lu, C. C., & Tseng, S. Y. (2002). Integrated design of AES (Advanced Encryption Standard) encrypter and decrypter. In Application-Specific Systems, Architectures and Processors, 2002. Proceedings. The IEEE International Conference on (pp. 277-285)*