**DATA SECURITY ENHANCEMENT OF THE ENHANCED ENCRYPTION STANDARD (AES) BY RIVEST – SHAMIN – ADLEMAN (RSA) METHOD OF ENCRYPTION & DECRPTION**

By

***Gurshaan Singh Bhasin – 21BLC1424***

***Rishita Agrawal – 21BLC1166***

***Mohammed Shoukat Ali – 21BLC1497***

***Harshita Sahu – 21BLC1558***

***Kumari Simran Devi – 21BLC1183***

***A project report submitted to***

**Dr. KIRUTHIKA**

in partial fulfilment of the requirements for the course of

**PJT PROJECT WORK**

in

**B.TECH ELECTRONICS AND COMMUNICATION / COMPUTER ENGINEERING**



**Vellore Institute of Technology, Chennai**

**Vandalur – Kelambakkam Road**

**Chennai – 600 127**

**TABLE OF CONTENTS**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **CHAPTER NO.** | **TITLE** | | | | **PAGE NO.** |
| **1** | **INTRODUCTION** | | | | 4-6 |
|  |  |
| **2** | **DATA SECURITY ENHANCEMENT OF THE ENHANCED ENCRYPTION STANDARD (AES) BY RIVEST – SHAMIN – ADLEMAN (RSA) METHOD OF ENCRYPTION & DECRPTION** | | | | 7-10 |
|  | 2.1 Incorporation of RSA in AES | | | | 7-8 |
|  | 2.2 Algorithm for AES Code Implementation in RSA | | | | 8-10 |
| **3** | **SIMULATION RESULTS AND ITS DISCUSSION** | | | | 11-12 |
|  | 3.1 Coding Language | | | | 11 |
|  | 3.2 Simulation parameters | | | | 12 |
|  | | |  |
| **4** | **CONCLUSION AND FUTURE WORK** | | | | 13-14 |
|  | 4.1 Conclusion  4.2 Future Works | | | | 13  13-14 |
| **5** | **REFERENCES** | | | | 15-16 |
|  |  | | | |  |

**6 APPENDIX** 16-21

Python Code

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURE NO.** | **TITLE** | **PAGE NO.** |
| 1  2  3 | RSA Flow  Algorithm Mind Map  Overall Flow | 6  8  10 |
|  |  |  |

**CHAPTER I**

**INTRODUCTION**

* **AES (Advanced Encryption Standard) :**

The Advanced Encryption Standard (AES) is a symmetric encryption algorithm widely recognized for its robust security and efficiency. It operates on fixed block sizes of 128 bits, supporting key lengths of 128, 192, or 256 bits. AES employs a series of transformation rounds, including SubBytes, ShiftRows, MixColumns, and AddRoundKey, to securely encrypt and decrypt data. One of its key strengths lies in its resistance to various cryptographic attacks, making it a popular choice for securing sensitive information in a wide range of applications, from data transmission over networks to securing stored data. Despite its strengths, AES relies on a single shared secret key for both encryption and decryption, which can pose challenges in key management and distribution, especially in large-scale systems.

* **RSA (Rivest-Shamir-Adleman) :**

RSA, named after its inventors Rivest, Shamir, and Adleman, is an asymmetric encryption algorithm commonly used for key exchange and digital signatures. Unlike symmetric encryption algorithms like AES, RSA employs two keys: a public key for encryption and a private key for decryption. This allows for secure communication between parties without the need to share a secret key beforehand. RSA relies on the computational difficulty of factoring large prime numbers, ensuring the security of encrypted data. Its applications extend beyond encryption, as RSA’s digital signatures can verify the integrity and authenticity of messages, making it a fundamental component of secure online transactions and communication protocols. However, RSA encryption is computationally more intensive compared to symmetric encryption, particularly for encrypting large amounts of data, which can impact performance in certain applications.as a complementary technique. VLS allows for dynamic adjustment of spreading code lengths for individual users, based on specific conditions or requirements.

AES (Advanced Encryption Standard) is a widely used symmetric encryption algorithm known for its strong security and efficiency. It operates on fixed block sizes of 128 bits and supports key sizes of 128, 192, or 256 bits. Despite its strengths, AES has some drawbacks, notably its reliance on a single shared secret key for both encryption and decryption. This can present challenges for securely sharing the key between parties and managing key distribution in large-scale systems, potentially leading to vulnerabilities if the key is compromised or mishandled.

o address these challenges and enhance the efficiency of AES, RSA (Rivest-Shamir-Adleman) encryption is often employed in conjunction with AES. RSA, as an asymmetric encryption algorithm, excels in key exchange and digital signatures. In the context of AES, RSA can facilitate secure key exchange by using public-key cryptography to securely share session keys. Instead of relying solely on symmetric encryption for key exchange, RSA allows for the secure transmission of session keys, which are then used in AES for the bulk encryption and decryption of data. This combined approach leverages the strengths of both AES and RSA, enhancing the overall security and efficiency of cryptographic systems, particularly in online transactions where secure key exchange is crucial for maintaining data confidentiality and integrity.

A diagram of a process

Description automatically generated

**FIGURE 1.1**

**A diagram of a blockchain

Description automatically generated**

**FIGURE 1.2**

**CHAPTER II**

**DATA SECURITY ENHANCEMENT OF THE ENHANCED ENCRYPTION STANDARD (AES) BY RIVEST – SHAMIN – ADLEMAN (RSA) METHOD OF ENCRYPTION & DECRPTION**

**2.1 INCORPORATION OF RSA IN AES**

* **Objectives** :

1. Importing Required Modules:

The Crypto library provides various cryptographic functions and algorithms. In this code, we import modules such as RSA, PKCS1\_OAEP, and hashlib to enable RSA encryption, PKCS#1 OAEP padding scheme for RSA encryption, and SHA256 hashing, respectively.

1. Financial Dataset:

This dataset represents hypothetical financial transactions, including sender and receiver information. Encrypting sensitive data like the receiver’s ID and phone number ensures that even if intercepted, this information remains confidential.

1. RSA Key Pair Generation:

RSA encryption relies on a pair of keys: a public key used for encryption and a private key used for decryption. The RSA.generate() method generates a key pair of the specified length (1024 bits in this case).

1. Exporting Keys:

After generating the RSA key pair, the code exports the public and private keys to PEM (Privacy Enhanced Mail) files. This step isn’t necessary for encryption and decryption but can be useful for storage and distribution of keys.

1. Encrypting Data:

RSA encryption is asymmetric, meaning different keys are used for encryption and decryption. Here, the PKCS1\_OAEP scheme is employed, which adds padding to the data before encryption, enhancing security. The encrypt() method encrypts the receiver\_id and phone\_no fields using the RSA public key.

1. Decrypting Data:

Decryption is performed using the RSA private key. The PKCS1\_OAEP scheme ensures that the decryption process is secure and accurate. After decryption, the original values of receiver\_id and phone\_no are obtained.

1. Printing Modified Dataset:

Finally, the code prints the modified financial dataset, showcasing the encrypted and decrypted values. This demonstrates the successful encryption and decryption process, ensuring the confidentiality and integrity of sensitive financial information.

This implementation illustrates the practical application of RSA encryption in securing sensitive data within financial transactions. While AES encryption is not directly involved in this code, RSA encryption serves as an essential cryptographic tool for ensuring secure communication and data protection.

A white rectangular object with colorful lines

Description automatically generated

**2.2 ALGORITHM FOR RSA IMPLEMENTATION IN AES**

# Step 1: Import Necessary Modules

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

# Step 2: Define Financial Dataset

financial\_dataset = [...]

# Step 3: Generate RSA Key Pair

key\_rsa = RSA.generate(1024)

# Step 4: Export Keys

public\_key\_rsa = key\_rsa.publickey().export\_key()

private\_key\_rsa = key\_rsa.export\_key()

# Step 5: Encrypt Data

for transaction in financial\_dataset:

# Encrypt receiver\_id

cipher\_rsa = PKCS1\_OAEP.new(RSA.import\_key(public\_key\_rsa))

encrypted\_receiver\_id = cipher\_rsa.encrypt(str(transaction["receiver\_id"]).encode())

transaction["receiver\_id"] = encrypted\_receiver\_id

# Encrypt phone\_no

encrypted\_phone\_no = cipher\_rsa.encrypt(str(transaction["phone\_no"]).encode())

transaction["phone\_no"] = encrypted\_phone\_no

# Step 6: Decrypt Data

for transaction in financial\_dataset:

# Decrypt receiver\_id

cipher\_rsa = PKCS1\_OAEP.new(key\_rsa)

decrypted\_receiver\_id = cipher\_rsa.decrypt(transaction["receiver\_id"])

print("Decrypted receiver\_id:", decrypted\_receiver\_id)

# Decrypt phone\_no

decrypted\_phone\_no = cipher\_rsa.decrypt(transaction["phone\_no"])

print("Decrypted phone\_no:", decrypted\_phone\_no)

# Step 7: Print Modified Dataset

print("Modified financial dataset:", financial\_dataset)

**A diagram of a diagram

Description automatically generated with medium confidence**

**CHAPTER III**

**SIMULATION RESULTS AND ITS DISCUSSION**

**3.1 CODING LANGUAGE**

Python is a suitable choice for this project due to several reasons:

1. Rich Cryptography Libraries: Python offers robust and extensive cryptography libraries like Crypto and PyCryptodome, providing high-level cryptographic functions and algorithms needed for secure data encryption and decryption.
2. Ease of Use and Readability: Python's simple syntax and readability make it easy to understand and maintain cryptographic code, ensuring that cryptographic operations are implemented correctly and securely.
3. Versatility and Flexibility: Python is a versatile language suitable for a wide range of applications, including cryptography. It allows developers to integrate cryptographic functionalities seamlessly into various projects, such as financial transactions, without significant overhead.
4. Large Community and Support: Python has a vast and active community of developers and contributors, offering abundant resources, tutorials, and support for cryptography-related tasks. This ensures that developers can quickly find solutions to any issues encountered during the project development.
5. Platform Independence: Python is a cross-platform language, meaning that code written in Python can run on various operating systems without modification. This ensures compatibility and portability, allowing the cryptographic solution to be deployed on different platforms seamlessly.

Overall, Python's combination of rich libraries, ease of use, versatility, community support, and platform independence makes it an excellent choice for implementing cryptographic solutions, including secure online transactions using RSA encryption as demonstrated in this project.

**3.2 SIMULATION PARAMETERS**

1. RSA Key Length: The RSA key length chosen for key generation is 1024 bits (key\_rsa = RSA.generate(1024)). This parameter determines the size of the RSA key pair, affecting the security level of the encryption.
2. Financial Dataset: This dataset contains sample financial transaction information, including transaction ID, sender's name, receiver's name, receiver's ID, and phone number. While not a simulation parameter per se, the dataset serves as input data for demonstrating RSA encryption and decryption.
3. Encryption Scheme: The RSA encryption scheme used in the code is PKCS1 OAEP (Optimal Asymmetric Encryption Padding), which provides enhanced security by adding padding to the data before encryption (PKCS1\_OAEP.new()).
4. Public and Private Keys: The public and private RSA keys are generated and used for encryption and decryption, respectively. These keys are essential parameters for RSA encryption and decryption operations.
5. Encryption and Decryption Process: The process involves encrypting sensitive data fields (receiver\_id and phone\_no) within the financial dataset using the RSA public key and decrypting them using the RSA private key. The encrypted data fields are replaced with their encrypted versions during encryption and decrypted back to their original values during decryption.

These parameters collectively facilitate the implementation of RSA encryption for securing sensitive data within the financial dataset, demonstrating the encryption and decryption process and ensuring data confidentiality and integrity.

**CHAPTER IV**

**CONCLUSION AND FUTURE WORKS**

**4.1 Conclusion**

Combining RSA implementation with AES encryption presents a robust method for secure data encryption in various applications. RSA excels in asymmetric encryption tasks such as key exchange and digital signatures, while AES offers efficient symmetric encryption for bulk data. By leveraging RSA for tasks like securely exchanging session keys and verifying the integrity of data, and utilizing AES for the actual encryption and decryption of data, the combination ensures both strong security and efficient performance. This approach provides a balance between the computational overhead of RSA and the speed of AES, offering a comprehensive solution for secure data transmission and storage. Overall, the RSA implementation on AES enhances data encryption by harnessing the strengths of both algorithms, thereby ensuring confidentiality, integrity, and efficiency in cryptographic operations.

**4.2 Future works**

1. Optimization for Performance: Investigate methods to optimize the performance of RSA encryption and decryption, such as using hardware acceleration or parallel processing techniques. This can help mitigate the computational overhead associated with RSA encryption, making it more suitable for high-volume data encryption.
2. Key Management: Develop more efficient and secure key management techniques, especially for RSA keys used in key exchange. This includes exploring key generation, distribution, rotation, and revocation strategies to enhance overall system security.
3. Hybrid Cryptosystems: Explore advanced hybrid cryptosystems that combine RSA with other symmetric encryption algorithms beyond AES. Research into novel cryptographic techniques can lead to enhanced security and performance benefits in specific use cases.
4. Post-Quantum Cryptography: Investigate the impact of quantum computing on RSA and AES encryption and explore post-quantum cryptographic algorithms as potential replacements or supplements. This is essential for ensuring long-term security against emerging threats.
5. Security Analysis and Vulnerability Assessment: Conduct thorough security analysis and vulnerability assessments of the implemented cryptographic system to identify potential weaknesses and mitigate them effectively. This includes evaluating resistance against known cryptographic attacks and continuously updating security measures accordingly.
6. Integration with Blockchain and Distributed Ledger Technologies: Explore the integration of RSA-AES encryption with blockchain and distributed ledger technologies to enhance data privacy and security in decentralized systems. This can involve research into cryptographic primitives tailored for blockchain environments and applications.
7. Standardization and Interoperability: Promote standardization and interoperability efforts to ensure compatibility and consistency across cryptographic implementations and platforms. This facilitates seamless integration and interoperability of cryptographic solutions in various systems and environments.
8. User Education and Awareness: Enhance user education and awareness regarding cryptographic best practices, including key management, secure communication protocols, and data protection measures. This empowers users to make informed decisions and effectively utilize cryptographic tools for secure data encryption.

By addressing these future works, we can further advance the effectiveness, efficiency, and security of RSA implementation on AES, ensuring robust cryptographic solutions for secure data encryption in various applications and domains.

**CHAPTER V**

**REFERENCES**

[1 ] Rahul Chavan - Financial Data Security in Cloud Computing

[2] NAVEEN KUNNATHUVALAPPIL HARIHARAN Sr. Hyperion SME & Department of Information Technology, United States January 2021- FINANCIAL DATA SECURITY IN CLOUD COMPUTING

[3] T. R. Mahesh , V. Vinoth Kumar , H. K. Shashikala , and S. Roopashree - ML Algorithms for Providing Financial Security in Banking Sectors with the Prediction of Loan Risks

[4] Shailendra Mishra - Exploring the Impact of AI-Based Cyber Security Financial Sector Management

[5] Habib Ullah Khan, Muhammad Sohail , Shah Nazir , Tariq Hussain, Babar Shah and Farman Ali - Role of authentication factors in Fin-tech mobile transaction security

[6] Kusworo Adi, Catur Edi Widodo, Aris Puji Widodo, Utami Sri Margiati - Detection of Foreign Object Debris (FOD) using Convolutional Neural Network (CNN), Journal of Theoretical and Applied Information Technology,**2022,** Vol.100. No 1,184-191.

[7] Jing Y, Zheng H, Lin C, Zheng W, Dong K, Li X. Foreign Object Debris Detection for Optical Imaging Sensors Based on Random Forest. Sensors (Basel). **2022** Mar 23;22(7):2463. doi: 10.3390/s22072463. PMID: 35408077; PMCID: PMC9002671.

[8] E. Papadopoulos and F. Gonzalez, "UAV and AI Application for Runway Foreign Object Debris (FOD) Detection," **2021** IEEE Aerospace Conference (50100), 2021, pp. 1-8, doi: 10.1109/AERO50100.2021.9438489.

[9] Qin, F.; Bu, X.; Liu, Y.; Liang, X.; Xin, J. Foreign Object Debris Automatic Target Detection for Millimeter-Wave Surveillance Radar. Sensors **2021**, 21, 3853.

[10] Ni P, Miao C, Tang H, Jiang M, Wu W. Small Foreign Object Debris Detection for Millimeter-Wave Radar Based on Power Spectrum Features. Sensors. **2020;** 20(8):2316. https://doi.org/10.3390/s20082316

**CHAPTER VI**

**APPENDIX**

* **Python Code**

!pip install Crypto

from Crypto.PublicKey import RSA

from Crypto.Cipher import PKCS1\_OAEP

from Crypto.Signature import pkcs1\_15

from Crypto.Hash import SHA256

import hashlib

# Sample financial dataset

financial\_dataset = [

    {

        "transaction\_id": 1,

        "name": "Gurshaan",

        "receiver\_name": "Shoukat",

        "receiver\_id": 123456,

        "phone\_no": "123-456-7890"

    },

    {

        "transaction\_id": 2,

        "name": "Ryan",

        "receiver\_name": "Kavin",

        "receiver\_id": 987654,

        "phone\_no": "987-654-3210"

    },

    {

        "transaction\_id": 3,

        "name": "Ajay",

        "receiver\_name": "Kavin",

        "receiver\_id": 555555,

        "phone\_no": "555-555-5555"

    }

]

# Generate RSA key pair

key\_rsa = RSA.generate(1024)  # Key length

# Export public and private keys

public\_key\_rsa = key\_rsa.publickey().export\_key()

private\_key\_rsa = key\_rsa.export\_key()

# Write RSA keys to files (optional)

with open('public\_rsa.pem', 'wb') as f:

    f.write(public\_key\_rsa)

with open('private\_rsa.pem', 'wb') as f:

    f.write(private\_key\_rsa)

# Encrypt the receiver\_id field of each transaction data

for transaction in financial\_dataset:

    receiver\_id = str(transaction["receiver\_id"]).encode()

    phone\_no = str(transaction["phone\_no"]).encode()

    # Encryption using RSA public key

    cipher\_rsa = PKCS1\_OAEP.new(RSA.import\_key(public\_key\_rsa))

    encrypted\_receiver\_id = cipher\_rsa.encrypt(receiver\_id)

    encrypted\_phone\_no= cipher\_rsa.encrypt(phone\_no)

    # Replace the original receiver\_id with the encrypted one

    transaction["receiver\_id"] = encrypted\_receiver\_id

    transaction["phone\_no"]= encrypted\_phone\_no

    # Decryption using RSA private key

    cipher\_rsa = PKCS1\_OAEP.new(key\_rsa)

    decrypted\_receiver\_id = cipher\_rsa.decrypt(encrypted\_receiver\_id)

    decrypted\_phone\_no= cipher\_rsa.decrypt(encrypted\_phone\_no)

    # Print decrypted receiver\_id

    print("Original receiver\_id:", receiver\_id)

    print("Encrypted receiver\_id:", encrypted\_receiver\_id)

    print("Decrypted receiver\_id:", decrypted\_receiver\_id)

    print("Original phone\_no:", phone\_no)

    print("Encrypted receiver\_id:", encrypted\_phone\_no)

    print("Decrypted receiver\_id:", decrypted\_phone\_no)

# Print the modified financial dataset

print("Modified financial dataset:", financial\_dataset)

* **Output :**

**Original receiver\_id: b'123456'**

**Encrypted receiver\_id:** b'L5\xf5\xe7\xe3<\x92/\x94\x02\xdcz\x99\xf71&\x14\x0f\xdb\x97nU\xf7l\xa8a\t\rMm\xce\x16\xb8\xdf\xcd\x07\xbdE&/6w;\x19}\xc3D\xc1\xa7!B\x1c\xc2\xd2\xf19\xb6\xa6t\xe0\xfc\xa9\xc6\x0c\xab\xda\xb6\x08\x02\xbb\x1a\xfc\x17w)\xfb\x07\x92\xd3\xca\xbd7\x80\x94\xb0$\xe4g\xaey+\xf4\xfd\x01u\xe2\xe3+\xd3\x84\xe8,3j\x97\x0eWM\x00\xaf\xa6\x16"\xe3\x8cA\x06\xaa-\xa1\x02\x99c\xf3\xbe\x08\xb2\x8a'

**Decrypted receiver\_id: b'123456'**

**Original phone\_no: b'123-456-7890'**

**Encrypted receiver\_id:** b'\*\xbb\x0f\xc0b>\xf5\xdb\x11\xe0\xd2\xfd\xc7\xa4ql\*t\xb8\x08\*\xb4\x12O[\x00B\x03\xd7=\xe6\x9f\x9b\xf1\xbe\xcb\x8e\x1d\xebx\xf0d\xf8rpO\xa5(V\x86\xffR\xe2$\x9e(\xed\x19Y\x92!;x\xb4\x15\xe3<`\x8b\x16&\xe1\x14w\xbc\xb1U:\x8e\xd9I\x8aY\xdc@1\x00<a\xcd\xbfI\xca,\x0c38q\xde5\xf1S\xbb\x82\x8aJ\x7f\x1e\rEQ\xe9X,n\x8a,y\xb5\x8b\xab\xf9o\xd7\xa78\xad\*'

**Decrypted receiver\_id: b'123-456-7890'**

**Original receiver\_id: b'987654'**

**Encrypted** **receiver**\_**id**: b'z.dCl5\x80\xd7\xa8\x98\xb4\x1e\xe0\xcf\x07O\x13[\x8cP\xfaL\t\xcb\xbe1\xfa\x0b\xec\x15\x96\xc0\xcdUg\x0f:\x95\xb2\xb44o\xfa!\r\x14\x1f\xf1f\\\x05\xab\x7f\x9b\xe74X\x11 NR\xa4,\x94=\xc6\xc7\xbcN/\\NS\'aa\'[\x91\xe4~\x05j\xd2\xcf\xc7\x87%\x83\xdd\x83\xc6P]C\xe1"\xf9)\xb1\x11\xf1\xa3z\x17\xef=\xb4\x80\xbd\xca\x07A\xfa\xaf\x8a\x02"\xfa\xe5\xa6;`\xee\x13%y\t'

**Decrypted receiver\_id: b'987654'**

**Original phone\_no: b'987-654-3210'**

Encrypted **receiver\_id**: b'\x9b-\xec\xe7T\x8a\x95\xd6z,\xb3\xa3\xb4o\xb4\xd7\x976s\xd8?\x94\_\x9f\x07s0\xec\x0eJZ\xc2\x0c\xe4\xcf%o\x08O\xaf{\x002\xce\x9e\x7f\x04M\xeb\x9a\x97G\x18\xe3S}\xcc\x15\x04e\xa8\xda\xd0k\xb5khX\xdb\xbf6!89pT\xa4kf\xf1;\xf9\xb72\xc68\x91\x99\xdb\xd0:\xa0\xf3:\xfe]{\xc3\xb8\xf6\xfe\x04\xa0Gs\xe0\xfch\x95\x95\xad\x98\x90\xe2\x97\x1a\x89U\n\x98\xee\xb68\xeat(v\xdb'

**Decrypted receiver\_id: b'987-654-3210'**

**Original receiver\_id: b'555555'**

**Encrypted receiver\_id:** b'\x95\xd1\xc3{\*&a\x82\xbc\xe8\xa9+\x0b[\xf6\x90\xe5h\x92\xce.H\xc1s\xba\x90\xc8y\xbd\xc8\xf9Q\x1cp\xacNl\xa1\xca}CV\x12\x9a\xe8\xaa\xb3\xe8\x9d\x8c,\x16;\xbc\x12PI\xeb\xe4G\x0b\x83\xaa\xac\xa2R\xb6\x90\x8c\x9c\xe4:\xfc\xa5\x18\xab\xbas\xfa\x1a\xc5\xf6\x9a\xac\x96#\x8d,\xcf\xb2\x8e\xdem\x97\x97sb\xfd\xa1\x80\xb8xc\xb5\x89\xdb\x9eLp\x85x\x82b\xfd\xe1\xa5\xbb!A)[\xddG|\xf7\xe0@@'

**Decrypted receiver\_id: b'555555'**

**Original phone\_no: b'555-555-5555'**

**Encrypted receiver\_id:** b'=\x9a\xac\x1dRTO\x81;\xf7\x95?c\xb9\xd7\xf0\xd4\x97\xa2\xe4\xbc\xc0\x12\xa4\xcf\xda|\x1d\xd4\xaa\xfe\xaaC\xa7\x99\xb0\xdc5\xa7}\x12I\xe4\x94\x14}\xff1\x17\xaf5\xea0}\xca2\x80\x05\x98\xaeZ\xf6\xddX`\xfb\x92d\x8fX\x8br\xa2\xeeB\x84\xb9\x98;\xbf\x17?\xfa\x9a\xc3\xbaU\_\xfe;\x17\t\xea\x8b;D \xc6{\xdd\x08\xea\xc4\xae\x8fk\xd8\xb2\x9a\xdc2\x88\xda\x93\x96\x17\xba\xaeU@\xa3\xe4\x83\xea\xad\x1a\x18\xb6'

Decrypted receiver\_id: b'555-555-5555'

Modified financial dataset: [{'transaction\_id': 1, 'name': 'Gurshaan', 'receiver\_name': 'Shoukat', 'receiver\_id': b'L5\xf5\xe7\xe3<\x92/\x94\x02\xdcz\x99\xf71&\x14\x0f\xdb\x97nU\xf7l\xa8a\t\rMm\xce\x16\xb8\xdf\xcd\x07\xbdE&/6w;\x19}\xc3D\xc1\xa7!B\x1c\xc2\xd2\xf19\xb6\xa6t\xe0\xfc\xa9\xc6\x0c\xab\xda\xb6\x08\x02\xbb\x1a\xfc\x17w)\xfb\x07\x92\xd3\xca\xbd7\x80\x94\xb0$\xe4g\xaey+\xf4\xfd\x01u\xe2\xe3+\xd3\x84\xe8,3j\x97\x0eWM\x00\xaf\xa6\x16"\xe3\x8cA\x06\xaa-\xa1\x02\x99c\xf3\xbe\x08\xb2\x8a', **'phone\_no':** b'\*\xbb\x0f\xc0b>\xf5\xdb\x11\xe0\xd2\xfd\xc7\xa4ql\*t\xb8\x08\*\xb4\x12O[\x00B\x03\xd7=\xe6\x9f\x9b\xf1\xbe\xcb\x8e\x1d\xebx\xf0d\xf8rpO\xa5(V\x86\xffR\xe2$\x9e(\xed\x19Y\x92!;x\xb4\x15\xe3<`\x8b\x16&\xe1\x14w\xbc\xb1U:\x8e\xd9I\x8aY\xdc@1\x00<a\xcd\xbfI\xca,\x0c38q\xde5\xf1S\xbb\x82\x8aJ\x7f\x1e\rEQ\xe9X,n\x8a,y\xb5\x8b\xab\xf9o\xd7\xa78\xad\*'}, {'transaction\_id': 2, 'name': 'Ryan', 'receiver\_name': 'Kavin', **'receiver**\_id': b'z.dCl5\x80\xd7\xa8\x98\xb4\x1e\xe0\xcf\x07O\x13[\x8cP\xfaL\t\xcb\xbe1\xfa\x0b\xec\x15\x96\xc0\xcdUg\x0f:\x95\xb2\xb44o\xfa!\r\x14\x1f\xf1f\\\x05\xab\x7f\x9b\xe74X\x11 NR\xa4,\x94=\xc6\xc7\xbcN/\\NS\'aa\'[\x91\xe4~\x05j\xd2\xcf\xc7\x87%\x83\xdd\x83\xc6P]C\xe1"\xf9)\xb1\x11\xf1\xa3z\x17\xef=\xb4\x80\xbd\xca\x07A\xfa\xaf\x8a\x02"\xfa\xe5\xa6;`\xee\x13%y\t', **'phone\_no'**: b'\x9b-\xec\xe7T\x8a\x95\xd6z,\xb3\xa3\xb4o\xb4\xd7\x976s\xd8?\x94\_\x9f\x07s0\xec\x0eJZ\xc2\x0c\xe4\xcf%o\x08O\xaf{\x002\xce\x9e\x7f\x04M\xeb\x9a\x97G\x18\xe3S}\xcc\x15\x04e\xa8\xda\xd0k\xb5khX\xdb\xbf6!89pT\xa4kf\xf1;\xf9\xb72\xc68\x91\x99\xdb\xd0:\xa0\xf3:\xfe]{\xc3\xb8\xf6\xfe\x04\xa0Gs\xe0\xfch\x95\x95\xad\x98\x90\xe2\x97\x1a\x89U\n\x98\xee\xb68\xeat(v\xdb'}, {'transaction\_id': 3, 'name': 'Ajay', 'receiver\_name': 'Kavin', 'receiver\_id': b'\x95\xd1\xc3{\*&a\x82\xbc\xe8\xa9+\x0b[\xf6\x90\xe5h\x92\xce.H\xc1s\xba\x90\xc8y\xbd\xc8\xf9Q\x1cp\xacNl\xa1\xca}CV\x12\x9a\xe8\xaa\xb3\xe8\x9d\x8c,\x16;\xbc\x12PI\xeb\xe4G\x0b\x83\xaa\xac\xa2R\xb6\x90\x8c\x9c\xe4:\xfc\xa5\x18\xab\xbas\xfa\x1a\xc5\xf6\x9a\xac\x96#\x8d,\xcf\xb2\x8e\xdem\x97\x97sb\xfd\xa1\x80\xb8xc\xb5\x89\xdb\x9eLp\x85x\x82b\xfd\xe1\xa5\xbb!A)[\xddG|\xf7\xe0@@', **'phone\_no':** b'=\x9a\xac\x1dRTO\x81;\xf7\x95?c\xb9\xd7\xf0\xd4\x97\xa2\xe4\xbc\xc0\x12\xa4\xcf\xda|\x1d\xd4\xaa\xfe\xaaC\xa7\x99\xb0\xdc5\xa7}\x12I\xe4\x94\x14}\xff1\x17\xaf5\xea0}\xca2\x80\x05\x98\xaeZ\xf6\xddX`\xfb\x92d\x8fX\x8br\xa2\xeeB\x84\xb9\x98;\xbf\x17?\xfa\x9a\xc3\xbaU\_\xfe;\x17\t\xea\x8b;D \xc6{\xdd\x08\xea\xc4\xae\x8fk\xd8\xb2\x9a\xdc2\x88\xda\x93\x96\x17\xba\xaeU@\xa3\xe4\x83\xea\xad\x1a\x18\xb6'}]