Name: Mohammed Aziz Thabit Saeed

ID: 2202144828

Secure Communication System Report

Introduction

The Secure Communication System project focuses on implementing encrypted communication between parties using cryptographic techniques. The primary objectives are to ensure data confidentiality, secure key exchange, and robust message integrity. The system can be utilized in various scenarios, such as secure messaging applications and confidential data transmission.

Cryptographic Techniques Used

1. RSA Encryption:
   - RSA is employed for secure key exchange, ensuring that private keys remain confidential while facilitating public key distribution through a server.
   - It involves modular exponentiation and adheres to standard practices for cryptographic strength.
   - Ensures that sensitive AES keys are exchanged securely without exposure to eavesdroppers.
2. AES Encryption:
   - Advanced Encryption Standard (AES) is used for encrypting communication after the key exchange.
   - Operates in CBC (Cipher Block Chaining) mode with a 256-bit encryption key, ensuring high security against brute-force attacks.
   - Employs an Initialization Vector (IV) to randomize encryption, enhancing resistance to pattern recognition.
3. SHA-256 Hashing:
   - Provides message integrity through cryptographic hashes.

    - Produces a 256-bit hash from any input, which is used to verify data integrity.
    - Ensures that even small changes in the input message result in a completely different hash.

4. Digital Signatures:
    - RSA-based signatures verify the authenticity of messages.
    - A digital signature is generated by hashing the message and encrypting the hash with the sender's private key. Recipients verify the signature by decrypting it with the sender's public key and comparing it to the hash of the received message.
    - Ensures both sender authentication and message integrity.

System Design

The system architecture includes a server for managing key exchanges and clients for communication. The design integrates multiple layers of security to address potential threats.

1- Server Design:

- Acts as an intermediary to facilitate RSA public key exchanges.
- Maintains a database of connected clients, their public keys, and socket information.
- Handles initial connections and ensures secure communication channels.

2- Client Design:

- Each client generates a 1024-bit RSA key pair at startup.
- Implements AES for encrypting messages post-key exchange.
- Uses SHA-256 for integrity verification and RSA for signing messages to enhance trust.

3- Key Exchange Mechanism:

- During connection setup, clients exchange RSA public keys through the server.
- Clients then generate a random AES session key, encrypt it using the recipient's RSA public key, and transmit it securely.

Implementation Details

1. Server Implementation:
    - Developed using Python's socket library for managing TCP connections.
    - Multithreaded to handle multiple clients concurrently without bottlenecks.
    - Stores client data in a secure dictionary for efficient communication.

2. Client Implementation:
    - RSA key pairs are generated using “sympy” for modular arithmetic.
    - Encrypts messages with AES and digitally signs them using RSA.

3. Digital Signature Workflow:
    - A message is hashed using SHA-256 to produce a digest.
    - The digest is encrypted with the sender's RSA private key to create the signature.
    - Recipients validate the signature by decrypting it with the sender's public key and comparing it to their computed hash.

4. Message Encryption and Decryption:
    - Messages are encrypted using the AES session key.
    - Decryption is performed on the receiver’s side using the same AES key, ensuring message confidentiality.

5. Key Exchange Process:
    - Ensures secure transmission of the AES key via RSA encryption, eliminating the risk of interception.

Testing and Validation

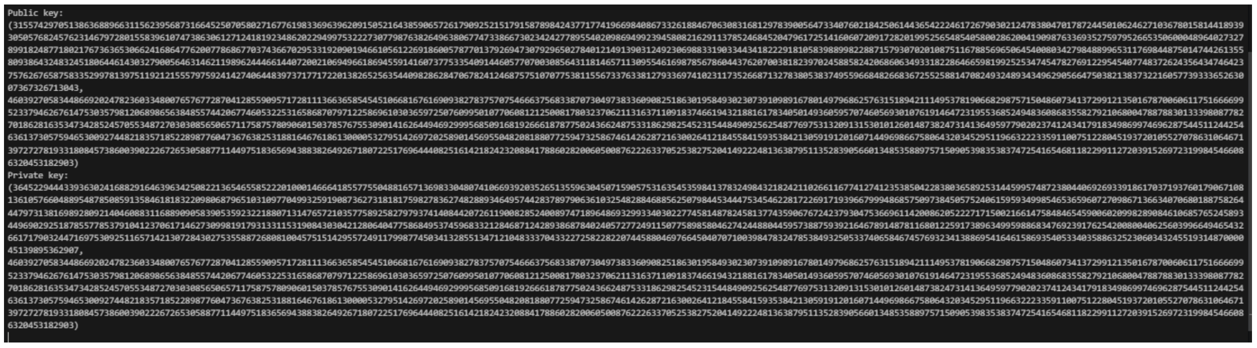
a- Testing each component:

For this part, we tested each component to make sure each component works without problems, then we build the project by using these components together.

Here are some of the component's tests:

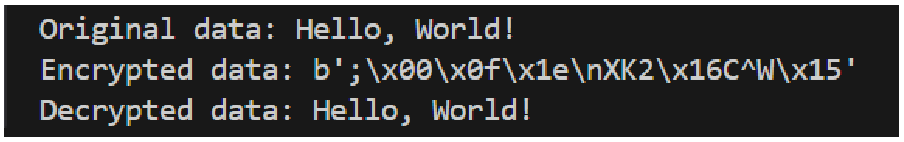
1. RSA Encryption:

   Example of how the public and the private keys will look links in the RSA:



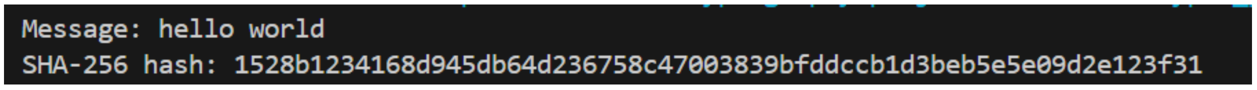
```
Public key:
(315574297051386368896631156239568731664525070580271677619833696396209150521643859065726179092521517915878984243771774196698408673326188467063083168129783900564733407602184250614436542224617267903021247838047017872445010624627103678015814418939
305057682457623146797280155839610747386306127124181923486202294997532227307798763826496380677473386673023424277895540209869499239458082162911378524684520479617251416060720917282019952565485405800286200419098763369352759795266535060000489640273 27
899182487718021767363653066241686477620077868677037436670295331920901946610561226918600578770137926947307929650278401214913903124923069883319033443418222918105839889982288715793070201087511678856965064540008034279848899653117698448750147442613 55
809386432483245180644614303279005646314621198962444661440720021069496618694559141607377533540914460577070030856431181465711309554616987856786044376207003818239702458858242068606349331822864665981992525347454782769122954540774837262435643474642 3
757626765875833529978139751192121555797592414274064483973717717220138265256354409828628470678241246875751070775381155673376338127933697410231173526687132783805383749559668482668367255258814708249324893434962905664750382138373221605773933365263 0
07367326713043,
460392705834486692024782360334800765767728704128559095717281113663658545451066816761690938278375707546663756833870730497383360908251863019584930230739109891678014979686257631518942111495378190668298757150486073413729912135016787006061175166 6699
523379462676147530357981206898656384855744206774605322531658687079712258696103036597250760995010770608121250081780323706211316371109183746619432188161783405014936059570746056930107619146472319553685249483608683558279210680047887883013339808778 2
701862816353473428524570553487270303085650657117587578090601503785767553090141626449469299995685091681926661878775024366248753318629825452315448490925625487769753132091315301012601487382473141364959779020237412434179183498699746962875445112442 54
636137305759465300927448218357185228987760473676382531881646761861300005327951426972025890145695504820818807725947325867461426287216300264121845584159353842130591912016071449698667580643203452951196632223359110075122804519372010552707863106467 1
397272781933180845738600390222672653058877114497518365694388382649267180722517696444082516142182423208841788602820060500876222633705253827520414922248136387951135283905660134853588975715090539835383747254165468118229911272039152697231998454660 8
6320453182903)
Private key:
(364522944433936302416882916463963425082213654655852220100014666418557755048816571369833048074106693920352651355963045071590575316354535984137832498432182421102661167741274123538504228380365892531445995748723804406926933918617037193760179067108
136105766048895487850859135846181832209806879651031097704993259190873627318181759827836274828893464957442837897906361032548288468856250798445344475345462281722691719396679994868575097384505752406159593499854653659607270986713663407068018875826 4
447973138169892809214046088311688909058390535923221880713147657210357758925827979374140844207261190082852400897471896486932993340302277458148782458137743590676724237930475366961142008620522271715002166147584846545900602099828908461068576524589 3
449690292518785577853791041237061714627309981917931331153190843030421280640477586849537459683321284687124289386878402405727724911507758985804627424488044595738875939216467891487811680122591738963499598868347692391762542008004062560399664946543 2
661717903244716975309251165714213072843027535588726808100457515142955724911799877450341328551347121048333704332272582282207445880469766450407071003984783247853849325053374065846745769323413886954164615869354053340358863252306034324551931487000 0
45139895362907,
460392705834486692024782360334800765767728704128559095717281113663658545451066816761690938278375707546663756833870730497383360908251863019584930230739109891678014979686257631518942111495378190668298757150486073413729912135016787006061175166 6699
523379462676147530357981206898656384855744206774605322531658687079712258696103036597250760995010770608121250081780323706211316371109183746619432188161783405014936059570746056930107619146472319553685249483608683558279210680047887883013339808778 2
701862816353473428524570553487270303085650657117587578090601503785767553090141626449469299995685091681926661878775024366248753318629825452315448490925625487769753132091315301012601487382473141364959779020237412434179183498699746962875445112442 54
636137305759465300927448218357185228987760473676382531881646761861300005327951426972025890145695504820818807725947325867461426287216300264121845584159353842130591912016071449698667580643203452951196632223359110075122804519372010552707863106467 1
397272781933180845738600390222672653058877114497518365694388382649267180722517696444082516142182423208841788602820060500876222633705253827520414922248136387951135283905660134853588975715090539835383747254165468118229911272039152697231998454660 8
6320453182903)
```

2. AES Encryption:



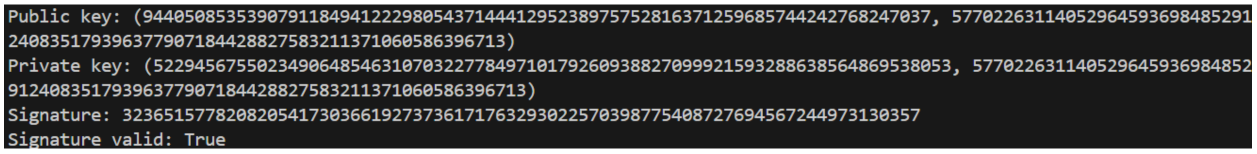
```
Original data: Hello, World!
Encrypted data: b';\x00\x0f\x1e\nXK2\x16C^W\x15'
Decrypted data: Hello, World!
```

3. SHA-256 Hash:



```
Message: hello world
SHA-256 hash: 1528b1234168d945db64d236758c47003839bfddccb1d3beb5e5e09d2e123f31
```

4. Digital Signature

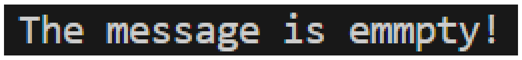


```
Public key: (94405085353907911849412229805437144412952389757528163712596857442427682470 37, 57702263114052964593698485291
240835179396377907184428827583211371060586396713)
Private key: (52294567550234906485463107032277849710179260938827099921593288638564869538053, 577022631140529645936984852
91240835179396377907184428827583211371060586396713)
Signature: 32365157782082054173036619273736171763293022570398775408727694567244973130357
Signature valid: True
```

b- Test with edge cases:

1- Empty messages:

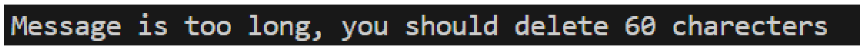
It will ignore the message since it is empty.



```
The message is emmpty!
```

2- Extremely long messages:

If the user exceeds the maximum number of characters, there will be an alert message, that tells user to decrease the proper number of characters such that the message is within the range of allowed number of characters.



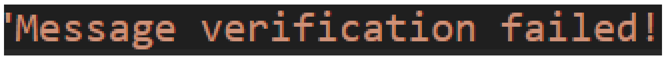
```
Message is too long, you should delete 60 charecters
```

3- Invalid or mismatched keys:

If we manually modify the public key of one client before sending a message, the other client will verify that the message verification fails.

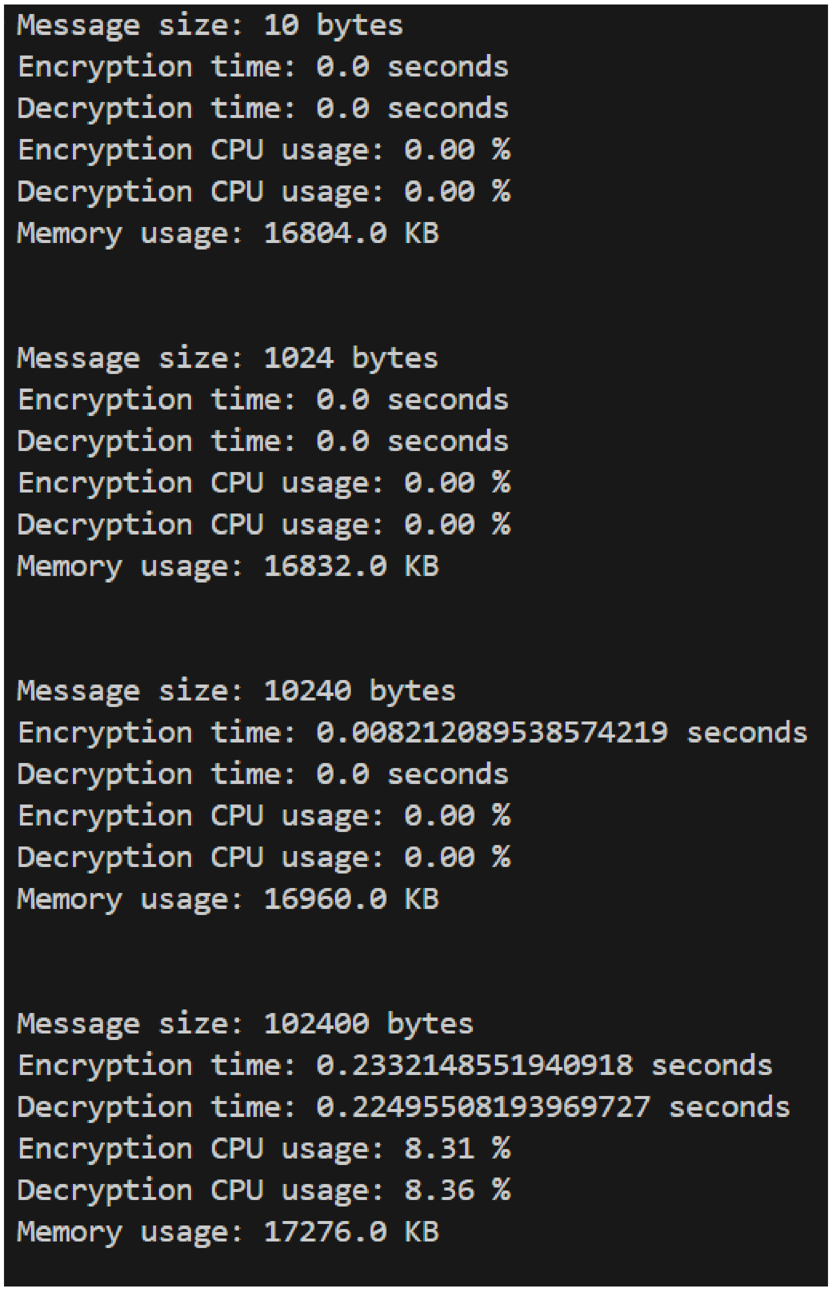
4- Test with Corrupted Messages:

If we manually corrupt a message during transmission like altering the ciphertext or signature, the receiving client will detect the corruption and fail verification.



```
'Message verification failed!
```

c- Efficiency:



```
Message size: 10 bytes
Encryption time: 0.0 seconds
Decryption time: 0.0 seconds
Encryption CPU usage: 0.00 %
Decryption CPU usage: 0.00 %
Memory usage: 16804.0 KB


Message size: 1024 bytes
Encryption time: 0.0 seconds
Decryption time: 0.0 seconds
Encryption CPU usage: 0.00 %
Decryption CPU usage: 0.00 %
Memory usage: 16832.0 KB


Message size: 10240 bytes
Encryption time: 0.008212089538574219 seconds
Decryption time: 0.0 seconds
Encryption CPU usage: 0.00 %
Decryption CPU usage: 0.00 %
Memory usage: 16960.0 KB


Message size: 102400 bytes
Encryption time: 0.2332148551940918 seconds
Decryption time: 0.22495508193969727 seconds
Encryption CPU usage: 8.31 %
Decryption CPU usage: 8.36 %
Memory usage: 17276.0 KB
```

The numbers may look too small, but that is because we are using a powerful computer.

Note: “We added a time counter, so if you send from one client, you will be able to see the time it takes for encryption and creating a signature. Same thing in the other client, it will show you the decryption and verification time”.

Challenges and Solutions:

a- Cryptography Implementation

1. Correctness and Security of Algorithms:
    - Implementing cryptographic algorithms (RSA, SHA-256, AES, Digital Signature) manually requires a deep understanding of their inner workings. Any mistakes can compromise the security of the system.
    - Ensuring that the implementation is both correct and secure against various attack vectors is challenging.
2. RSA Key Generation and Management:
    - Generating large prime numbers efficiently and securely (1024-bit per prime number).
    - Managing key pairs securely, especially handling the distribution of public keys and keeping private keys confidential.
3. Random Number Generation:
    - Securely generating random numbers for keys and initialization vectors (IVs). Ensuring randomness is crucial to prevent predictability in keys and encrypted data.

b- Network Programming

1. Concurrent Connections:
    - Handling multiple clients concurrently and ensuring that public key exchange and message routing between clients work seamlessly.
    - Managing threads safely and efficiently to avoid race conditions or deadlocks.
2. Data Integrity and Authentication:
    - Implementing digital signatures and verification to ensure data integrity and authenticity. Ensuring that signatures cannot be forged or tampered with.
3. Error Handling:
    - Properly handling errors in network communication, encryption/decryption processes, and during key exchange. Ensuring the system fails gracefully and provides meaningful error messages.

c- Communication Protocol

1. Public Key Exchange:
    - Securely exchanging public keys between clients without relying on a trusted third party.
    - Ensuring that the public key exchange is not susceptible to man-in-the-middle attacks.
2. Message Encryption and Decryption:
    - Efficiently encrypting and decrypting messages with AES after securely exchanging the AES key using RSA.
    - Ensuring that the messages are correctly formatted and encoded/decoded during transmission.

d- Security Considerations

1. Man-in-the-Middle Attacks:
    - Ensuring that the initial key exchange is secure to prevent attackers from intercepting or altering the public keys.
    - Verifying the authenticity of the keys received from the server.
2. Replay Attacks:
    - Implementing mechanisms to prevent attackers from replaying old messages. This might involve timestamping messages or using nonces.
3. Data Confidentiality:
    - Ensuring that the encryption provides strong confidentiality, and that encrypted data cannot be easily decrypted without the correct key.

e- Testing and Validation

1. Comprehensive Testing:
    - Thoroughly testing the system to ensure all cryptographic operations work as expected. This includes unit tests for individual components and integration tests for the entire system.
    - Testing under various network conditions to ensure robustness.
2. Performance Considerations:
    - Ensuring that the encryption and decryption operations are performant and do not introduce significant latency in communication.

        - Balancing security and performance, particularly with RSA operations which are computationally intensive.

f- Documentation and Maintainability

1. Code Documentation:
    - Providing clear documentation for the implemented cryptographic algorithms and network protocols.
    - Ensuring that the code is maintainable and understandable for future developers.
2. Security Audits:
    - Regularly reviewing the code for potential security vulnerabilities and keeping up with best practices in cryptography and secure communication.

Conclusion

The project successfully demonstrates secure communication using RSA, AES, and SHA-256 algorithms. It ensures confidentiality, secure key exchange, and message integrity. The implemented system highlights the strength of combining symmetric and asymmetric cryptographic techniques.