**INTERNSHIP REPORT**

**Intel-Unnati-Industrial-Training**

**Cryptography Simulation with OpenSSL/mbedTLS**

**By**

**VIPPAGUNTA HIMACHARITHA - BU21CSEN0300232**

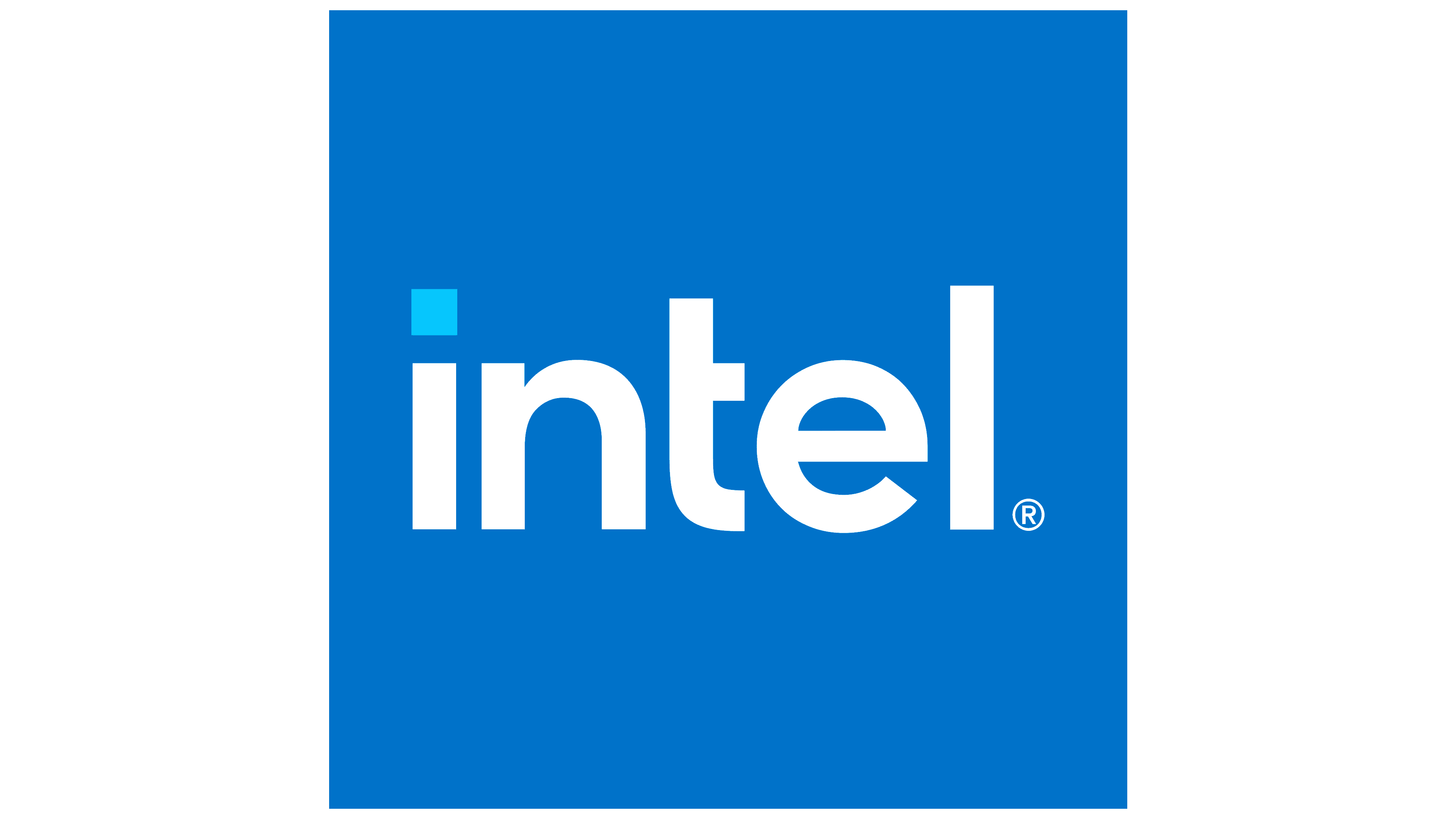
**PAVANA MADDIREDDY - BU21CSEN0600142**

**PUTTA VENKATA PUSHPA - BU21CSEN0600048**

**ARIGELA YASHASWINI - BU21CSEN0600056**

**MINI GADHIRAJU - BU21CSEN0100837**

**(Duration: 14-05-2024 to 15-07-2024)**

**** ****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**Gandhi Institute of Technology and Management**

**(DEEMED TO BE A UNIVERSITY)**

**BENGALURU, KARNATAKA, INDIA**

**SESSION: 2021-2025**

**CONTENTS**

|  |  |
| --- | --- |
| S.No | Index |
| 1. | **Introduction** |
| 2. | **Problem Statement** |
| 3. | **Scope** |
| 4. | **Infrastructure Requirements** |
| 5. | **Methodology** |
| 6. | **Application workflow** |
| 7. | **Algorithm Implementation** |
| 8. | **Justification for Crypto Algorithms Used** |
| 9. | RSA Key Generation and Digital Certificates |
| 10. | Source Code Implementation |
| 11. | **Test Methodologies** |
| 12. | **Process flow** |
| 13. | **Learning Outcomes** |
| 14. | **Results** |
| 15. | **Conclusion** |
| 16. | **References** |

**Abstract:**

This project aims to implement and simulate cryptographic algorithms and protocols using the mbedTLS or OpenSSL libraries. It focuses on demonstrating encryption, decryption, and secure communication techniques essential for ensuring data privacy and integrity in various applications. Key cryptographic algorithms such as RSA, AES, and ECC will be implemented to showcase asymmetric and symmetric encryption methods. The project will also cover the use of Digital Certificates, TLS/SSL protocols for secure communication, and Hash Functions like SHA-256 for data integrity verification. This project illustrates a strong implementation of cryptographic algorithms that result in an interactive tool that serves as an educational resource for understanding and experimenting with cryptography.

**Introduction:**

Our project goal is to implement and simulate key cryptographic algorithms and protocols using mbedTLS and OpenSSL libraries, focusing on RSA, AES, ECC, TLS, SSL, Digital Certificates, and SHA-256. The successful completion will result in an interactive tool for understanding and experimenting with cryptography. This tool will benefit students and professionals in cybersecurity, offering hands-on experience with cryptographic concepts. It will enhance understanding of how cryptography secures digital communication and data.

**Problem Statement:**

Cryptography Simulation with mbedTLS/OpenSSL Library Usage andUser Interaction

**Scope:**

The scope of this project includes the connection of the communication between server and client and building conversation about psychological problems with AI chatbot called Eliza and securing that communication using SIGMA. The project needs knowledge of Key generation, cryptography algorithms, encryption and C/C++ programming.

**Infrastructure Requirements**

**Hardware:**

**1. Development Machine:**

* Modern multi-core processor (Intel Core i5 or equivalent recommended)
* Minimum 8GB of RAM (16GB recommended for better performance)
* Sufficient storage space for development tools, libraries, and datasets

**2. Testing Environment:**

* Similar specifications as development machine to ensure consistent performance during testing phases

**Software:**

* Microsoft VS code
* C/C++
* OpenSSL

**Networking Requirements:**

* Internet connectivity for library updates, documentation references, and security patches

**Security Considerations:**

* Ensure secure handling of cryptographic keys, sensitive data, and digital certificates within the development and testing environments
* Regular updates and patches for OS and cryptographic libraries to mitigate security vulnerabilities

### **Methodology**

* **Choice of OpenSSL Libraries:** Selected for their robust support and widely trusted implementations of cryptographic algorithms and protocols.
* **Implementation Details for Each Algorithm and Protocol:** Utilize OpenSSL for implementing AES-256 encryption, RSA for key exchange, HMAC with SHA-256 for message integrity, and secure hashing with SHA-256, ensuring secure and efficient cryptographic operations.
* **Security Best Practices:** Follow cryptographic security best practices, including proper key management, secure random number generation, and regular updates to cryptographic libraries.

**Application Workflow**

1. **Design and Architecture**

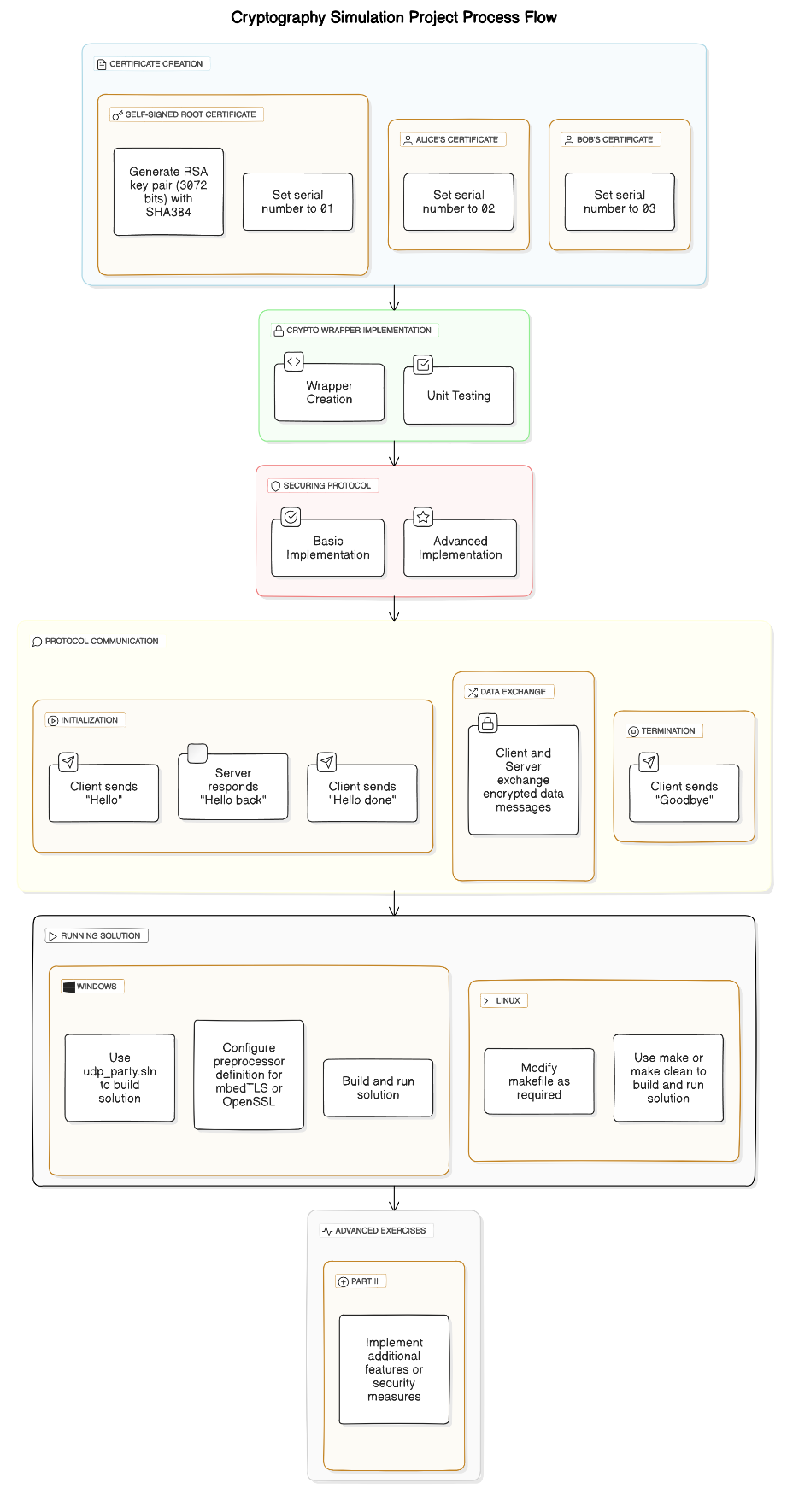
* **System Architecture Design:**
  + Create an architectural diagram outlining components and their interactions (e.g., key generation, encryption, decryption, certificate management).
* **File Encryption Application Design:**
  + Design the structure and flow for the command-line interface (CLI) application, specifying user interactions and outputs.

1. **Implementation**

* **Setup Development Environment:**
  + Install necessary software tools (mbedTLS, OpenSSL, Python).
  + Configure development IDE or text editor (e.g., Visual Studio Code, Vim).
* **Algorithm and Protocol Implementation:**
  + Implement RSA for key generation, encryption, and decryption.
  + Implement AES for symmetric encryption and decryption.
  + Implement ECC for key exchange or digital signatures.
  + Implement SHA-256 for hashing messages.
  + Integrate TLS/SSL protocols for secure communication channels.
* **Digital Certificate Management:**
  + Develop functions to create, manage, and verify digital certificates.

1. **Testing and Validation**

* **Unit Testing:**
  + Develop and execute unit tests for each cryptographic algorithm and protocol.
* **Integration Testing:**
  + Test the integration of algorithms and protocols within the CLI application.
* **Security Testing:**
  + Perform security analysis and vulnerability assessment.
  + Validate secure handling of cryptographic keys and sensitive data.



**Algorithm Implementation**

**Crypto Wrapper file:**

### Key Generation: Generate AES-GCM and RSA key pairs using a secure random number generator for cryptographic operations.

### AES-GCM Encryption and Decryption: Encrypt plaintext with AES-GCM using a randomly generated Initialization Vector (IV) and compute an authentication tag for integrity verification. Decrypt ciphertext by verifying and decrypting with the correct key, IV, and authentication tag.

### HMAC (Hash-based Message Authentication Code): Generate HMAC using SHA-256 to compute a hash of messages with a shared secret key for integrity verification. Verify integrity by recalculating HMAC and comparing with received HMAC.

### SHA-256 (Secure Hash Algorithm): Compute SHA-256 hash of data to produce a fixed-size hash value, ensuring data integrity and secure hashing capabilities.

### RSA Encryption and Decryption: Generate RSA key pairs for secure communication. Encrypt sensitive data using RSA public key encryption and decrypt using RSA private key decryption for secure key exchange and data protection.

### 

### **Client Session:**

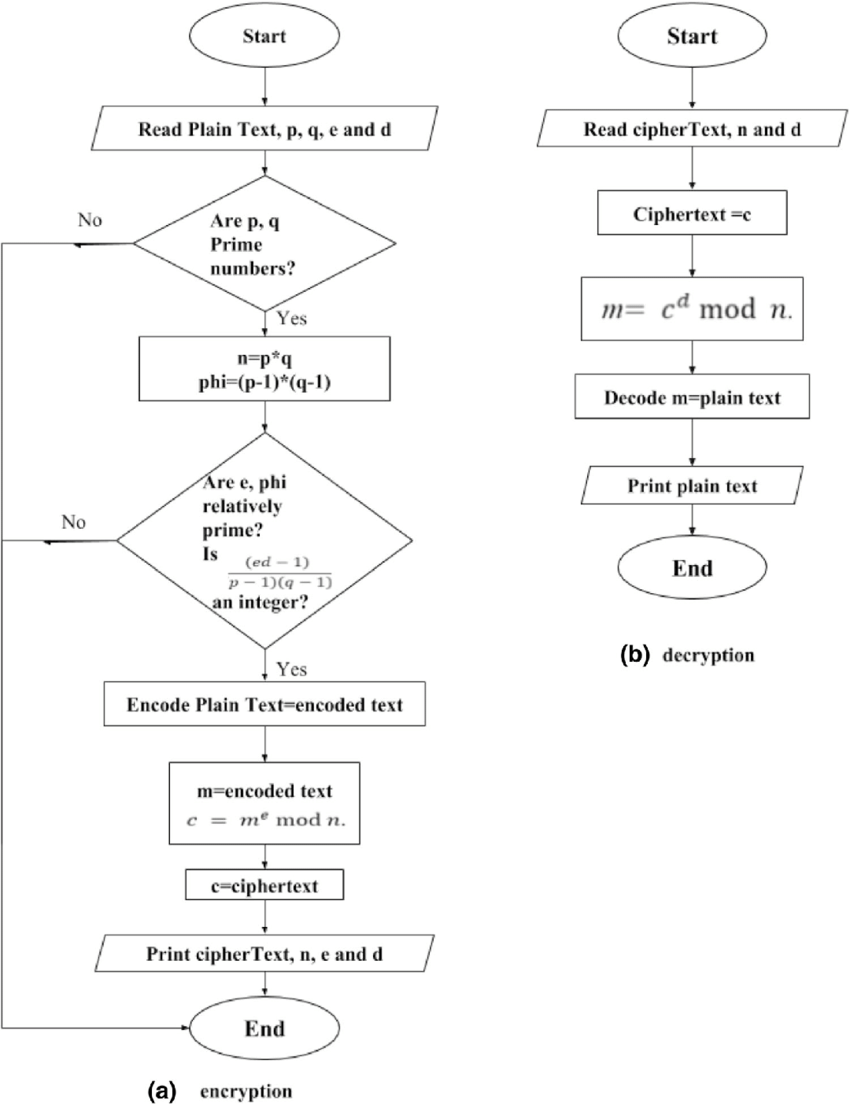
* Establish secure communication using AES-GCM for encryption and HMAC with SHA-256 for message integrity verification.

### **Server Session:**

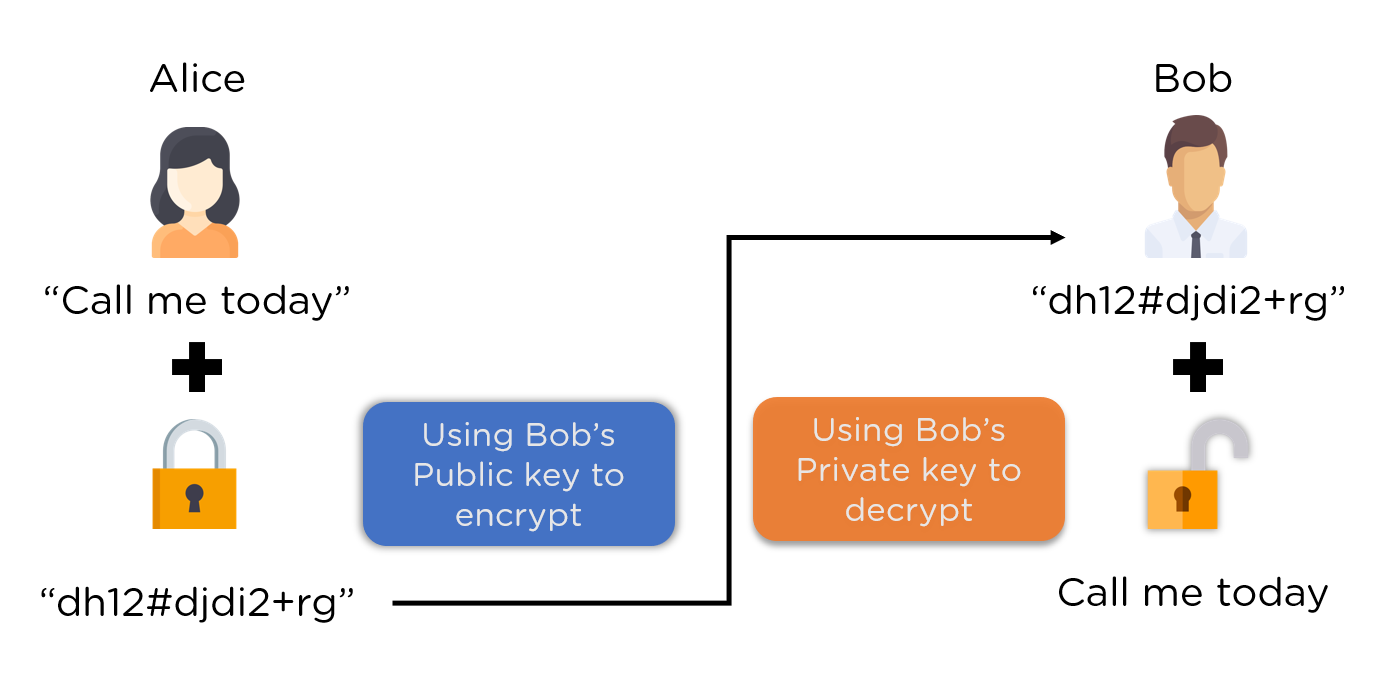
* Decrypt client messages with RSA and AES-GCM using key pairs for secure data exchange, ensuring data integrity with HMAC and SHA-256.

**Justification for Crypto Algorithms Used**

* Generate **AES-GCM** and **RSA key pairs** using a secure random number generator for cryptographic operations. Encrypt plaintext with AES-GCM using a randomly generated Initialization Vector (IV) and compute an authentication tag for integrity verification. Decrypt ciphertext by verifying and decrypting with the correct key, IV, and authentication tag. Generate **HMAC** using SHA-256 to compute a hash of messages with a shared secret key for integrity verification, ensuring data integrity by recalculating HMAC and comparing with received HMAC. Compute **SHA-256** hash of data to produce a fixed-size hash value, facilitating secure hashing capabilities. Use RSA key pairs for secure communication, employing RSA public key encryption to encrypt sensitive data and RSA private key decryption for secure key exchange and data protection in both client and server sessions.
* **Creation of RSA Keys**



Cipher Text (256 bits)



## RSA Key Generation and Digital Certificates

Creating a self-signed root certificate:

* Generate a private key.
* Create a certificate signing request (CSR).
* Sign the CSR with the private key to generate the self-signed certificate.

To generate an RSA key pair for "Alice" and sign it with the root CA:

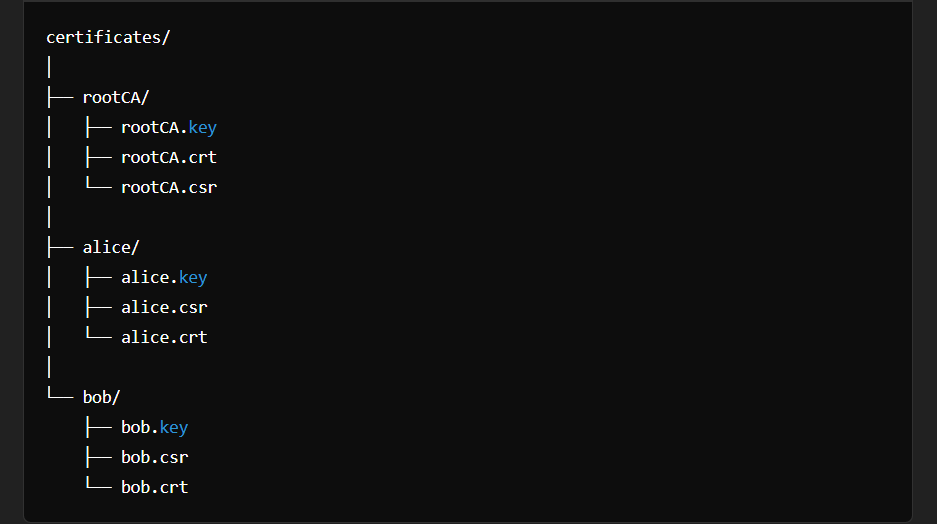
* Generate Alice's private key.
* Create a Certificate Signing Request (CSR) for Alice.
* Sign Alice's CSR with the root CA's private key to generate Alice's certificate.

To generate an RSA key pair for "Bob" and sign it with the root CA:

* Generate Bob's private key.
* Create a Certificate Signing Request (CSR) for Bob.
* Sign Bob's CSR with the root CA's private key to generate Bob's certificate.







**Source Code implementation**

****

## Test Methodologies

**1. Unit Testing:**

* **Objective:** Ensure each cryptographic algorithm and protocol functions correctly in isolation.
* **Implementation:**
  + Develop comprehensive unit tests for individual components, such as AES-256 encryption/decryption, RSA key generation, HMAC computation, and SHA-256 hashing.
  + Use mock data to validate the correctness of each function.
  + Verify that encryption and decryption processes are reversible and produce consistent results.
  + Test boundary cases and potential error conditions, such as invalid inputs or corrupted data.
* **Tools:** Utilize unit testing frameworks such as unit test or pytest for Python, ensuring automated and repeatable test execution.

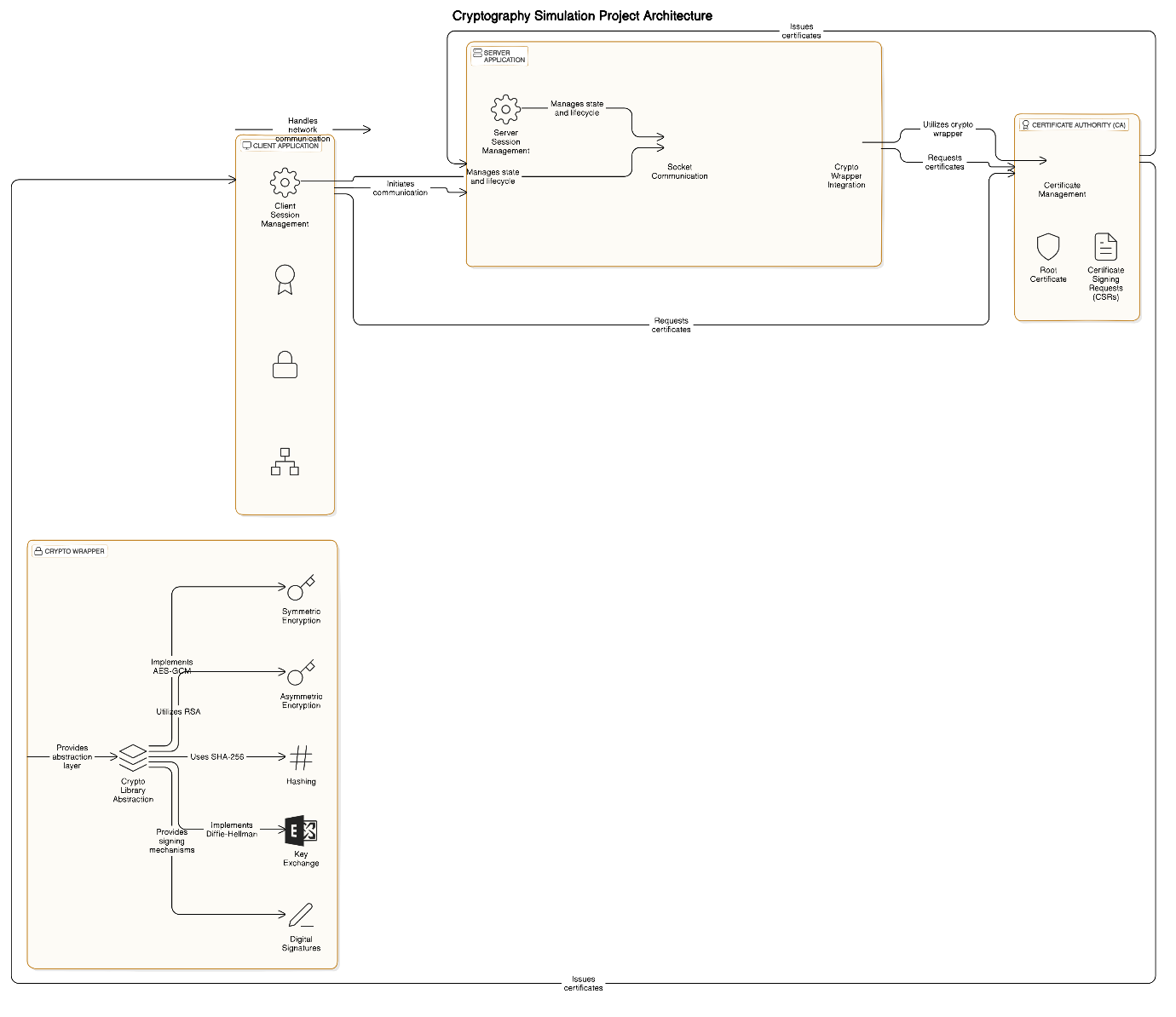
**2. Integration Testing:**

* **Objective:** Validate the seamless integration of various cryptographic algorithms and protocols within the CLI application.
* **Implementation:**
  + Combine individual components and test their interactions within the application workflow.
  + Ensure that key generation, encryption, decryption, and HMAC verification work together as intended.
  + Simulate end-to-end scenarios, such as file encryption followed by decryption using the CLI.
  + Test the handling of user passphrases, including key derivation and retrieval of encrypted FEK.
* **Tools:** Use integration testing tools and frameworks to automate the testing process, verifying that the integrated system functions correctly as a whole.

**Security Testing:**

* **Objective:** Ensure the application’s security, focusing on the confidentiality, integrity, and authenticity of the data.
* **Implementation:**
  + Perform a thorough security analysis to identify potential vulnerabilities and weaknesses in the cryptographic implementation.
  + Conduct vulnerability assessments to test for common cryptographic flaws, such as weak key generation, improper key storage, and insecure handling of sensitive data.
  + Validate the secure handling of cryptographic keys, ensuring they are not exposed or stored in plain text.
  + Test the application’s resistance to attacks, such as brute-force attacks on user passphrases, cryptanalysis attempts, and man-in-the-middle attacks during data transmission.
* **Tools:** Use security testing tools such as OpenVAS, Nmap, and custom scripts to perform in-depth security assessments. Employ static and dynamic analysis tools to detect and mitigate potential security issues.

## Process flow



**Learning Outcomes**

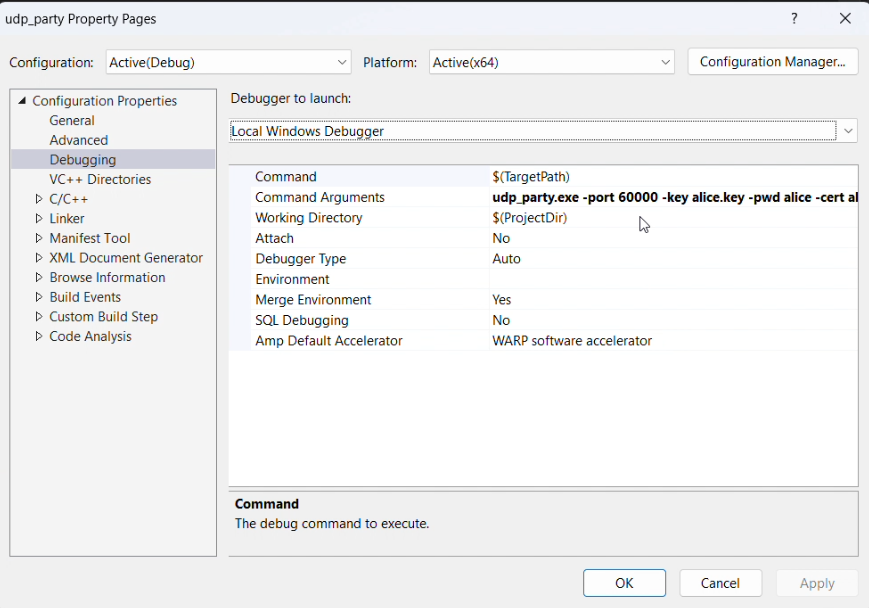
1. **Understanding Cryptographic Algorithms:**

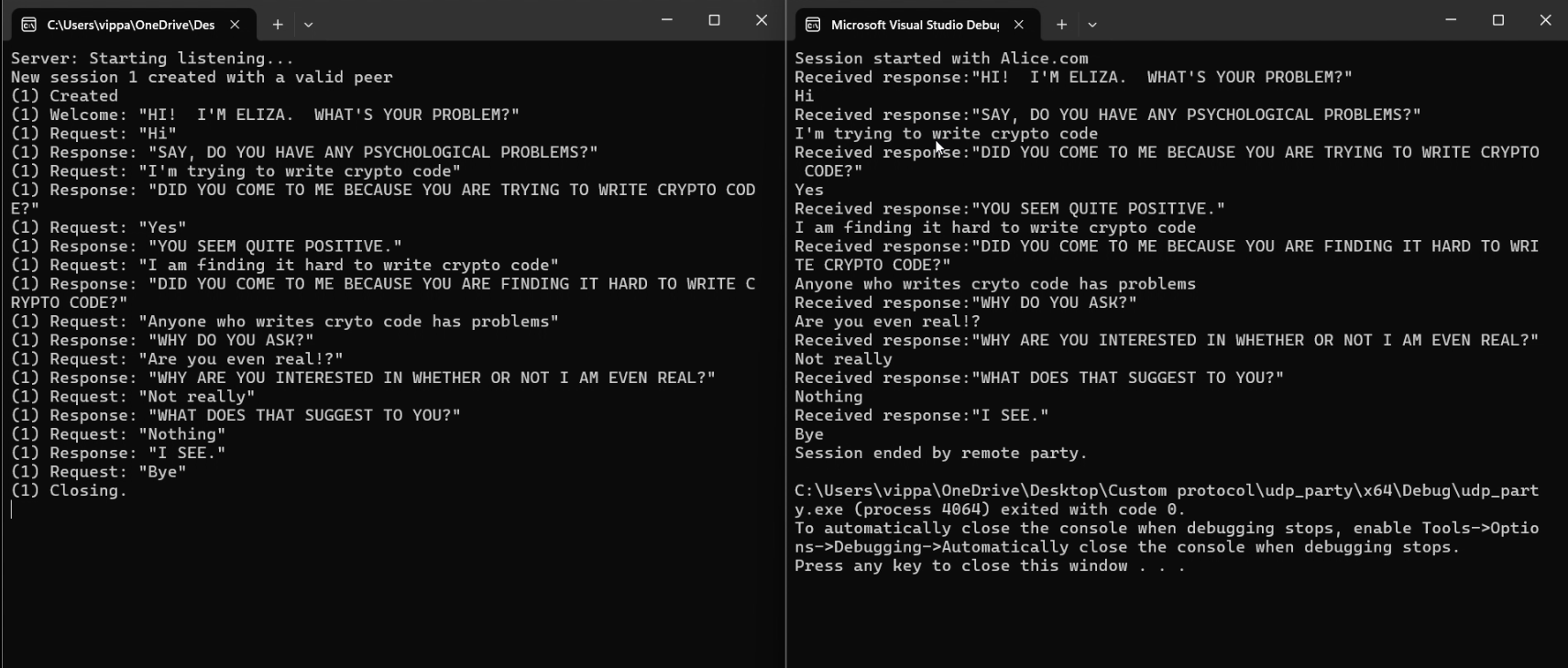
* Gain a comprehensive understanding of AES-256, RSA, HMAC, and SHA-256, including their implementation details and practical applications.

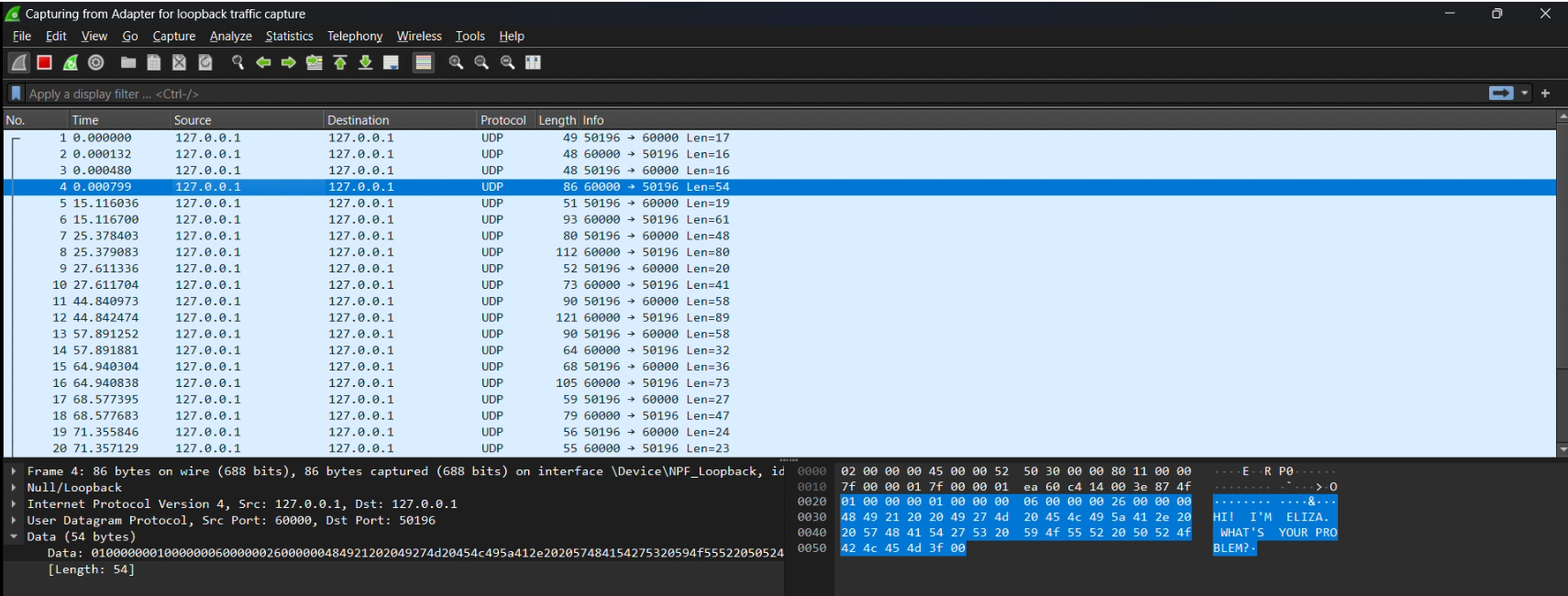
2. **Secure Coding Practices:**

* Learn and apply secure coding practices to protect sensitive information, prevent vulnerabilities, and ensure robust cryptographic implementations using OpenSSL or mbedTLS libraries.

**Result**

****

****

****

**Demo**



**Conclusion**

This project showcases the creation of an interactive educational tool focused on cryptographic techniques using C code and mbedTLS/OpenSSL libraries. It emphasizes hands-on exploration of RSA, AES, DES, and SHA algorithms for encryption, decryption, key generation, and management. By offering a user-friendly interface and comprehensive documentation, the project aims to enhance understanding of cybersecurity concepts among students and professionals, promoting practical skills in cryptographic implementation and education.

**References**

<https://www.intel.com/content/www/us/en/resources-documentation/developer.html>

<https://wiki.openssl.org/index.php/OpenSSL_3.0>

<https://www.rfc-editor.org/rfc/rfc3526>