CS432 Computer and Network Security
Spring 2023 - Term Project

Secure Channel Application

Project Step 1 Due: May 4, 2023, Thursday, 22:00 (to be submitted to SUCourse)
Demos: Time and Schedule will be determined later

Project Step 2 Due: May 26, 2023, Friday, 22:00 (to be submitted to SUCourse)
Demos: Time and Schedule will be determined later

Submission and Rules

- You can work in groups of at most 3 people in both steps of the project. We will collect group information before step 1, but we will not group people by ourselves; if you cannot find a group, you can do the project alone. The project requirements will **not** change by the group size.
- Equal distribution of the work among the group members is essential.
- All group members should appear at the demos.
- All group members should submit the complete project in each step to SUCourse+. No email submissions please. Make sure that all group members submit the same version. If a group member does not submit, we will assume that this person has been kicked out of the group and he/she will receive zero. On the other hand, if you allow a group member submit the same code as you, then you acknowledge his/her contribution. Consequently, the group members who submit step 1 together should continue to work together for step 2 as well.
- The submitted code will be used during the demo. No modification on the submitted code will be allowed.
- Submission steps:
  - Delete the content of debug/release folders in your project directory before submission.
  - Create a folder named ***Server*** and put your file server related codes here.
  - Create a folder named ***Client*** and put your client related codes here.
  - Create a folder named ***XXXXX_Surname_Name***, where *XXXXX* is your SUNet ID (e.g. *simgedemir_Demir_Simge)* Put your *Server* and *Client* folders.
  - Compress your *XXXXX_Lastname_Name* folder **using ZIP**.
- You will be invited for a demonstration of your work. Date and other details about the demo will be announced later.
- Late submission is allowed only for one day (24 hours) with the cost of 10 points (out of 100).
- In case there is an unclear part in the project, please use the WhatsApp group to ask questions so that everyone gets benefited.

Programming Rules

- Preferred languages are C#, Java and Python, but C# is recommended.
- Your application should have a graphical user interface (GUI). **It is not a console application!**
- Your code should be clearly commented. This may affect up to 5% of your grade.
- Your program should be portable. It should not require any dependencies specific to your computer. We will download, compile and run it. If it does not run, it means that your program is not running. So do test your program before submission.
- In your application when a window is closed, **all threads related to this window should be terminated.**

We encourage the use of course's WhatsApp group for the general questions related to the project and for finding group members. For personal questions and support, you can send email to course TAs.

Good luck!

Albert Levi
Simge Demir – simgedemir@sabanciuniv.edu
Emre Ekmekçioğlu - eekmekcioglu@sabanciuniv.edu
Burcu Tanış - burcutanis@sabanciuniv.edu

Introduction

In this project, you will implement a client-server application for establishing a *secure* and *authenticated* message broadcast mechanism among subscribers of different channels (in total there will be three channels). In the application that you will implement, there will be *Client* and *Server* modules.

The *Server* module:

- receives the messages from clients and broadcasts them to the subscribed clients on a specific channel.

The *Client* module:

- registers to the Server and subscribes to a channel,
- gets authenticated to the Server,
- connects to the Server to send and receive messages in a secure and authenticated way through the channel subscribed.

Roughly, the enrollment of the clients, client authentication to the server and starting up the server in a secure way constitute the first step of the project. The rest is the second step.

You should design and implement a user-friendly and functional graphical user interface (GUI) for client and server programs. In both of the steps, all activities and data generated by the server and the clients should be reported in text fields at their GUIs. These include (but not limited to):

- RSA public and private keys in hexadecimal format,
- AES keys and IVs in hexadecimal format,
- Random challenges and responses calculated for the challenge-response protocol runs in hexadecimal format,
- Digital signatures in hexadecimal format,
- List of online clients (on the server side),
- Verification results (digital signatures, HMAC verifications, etc.),
- Message transfer operations and message texts,
- HMAC values, session keys, etc. in hexadecimal format,
- And all other details that are needed to follow the flow of execution. We cannot explain all the details of the things that need to be showed on the GUI within the project document; thus, you may take this as the golden rule: "i**f we cannot follow what is going on, we cannot grade**". And please do not ask us if you should display this or that; please use your own reasoning thinking of the golden rule above. Another related rule is that we will not follow the execution of your code in debug console; we will grade using whatever you put on the GUIs.

Project Step 1 (Due: May 4, 2023, Thursday, 22:00)

In the first step of the project, the client performs *enrollment*, *user authentication* and *disconnection* operations. You will implement enrollment and authentication protocols between client and the server. The initial contact of each client to the server is the *enrollment* process. Only the enrolled users (clients) can be authenticated by the server; this authentication is done via a challenge-response protocol. Enrollment, authentication and disconnection processes are explained in the following subsections.

We will be providing you with two RSA-3072 public-private key pairs in the project pack. These key pairs will be used by the server; one for encryption/decryption and one for signing/verification purposes. You have to read the RSA keys on server start-up.

Enrollment Phase

Before doing anything, a client has to enroll to server in a secure way if he/she has not done it yet. Before making the connection to the server, the client program should load the server's public keys (one for encryption and the other for signature verification) from the file system (these keys are also provided in the project pack).

Then, the user enters the IP address and the port number of the server, as well as his/her selected username, selected password and selected channel to get subscribed from the client GUI. Note that there will be only three channels that can be subscribed (via selecting from client GUI): IF100, MATH101 and SPS101 ☺. Channel subscription will be completed in enrollment phase and cannot be changed later.

The client should connect to the server using the IP address and the port number entered. If the connection is not successful, it should show an error message and ask the user to enter the port number and IP address information again. If the connection is successful, then user takes the hash of the entered password using SHA-512 and concatenates this hash along with the username and the selected channel name. Then, it encrypts this concatenated message by using the RSA public key (the one for encryption/decryption) of the server and sends the encrypted message to the server.

Server receives this encrypted message and decrypts it using the corresponding RSA private key. Then, it parses the decrypted message and gets the username, the hash of the password that the user entered and the selected channel name. After that, Server should check if there is another enrolled client with the same username. If so, it should send back an error message to the client stating that the username has already been taken. Otherwise (if the username is unique amongst enrolled clients), it should save the username, the hash of its password and subscribed channel name to its database (it is up to you how to implement this), and send a message to the client stating that the enrollment is successful. The success/error messages sent by the server must be signed by server's signature RSA key.

Once the client receives signed response from the server, it first verifies the server's signature using the corresponding public key. If the response is verified, client checks whether the response is "error" or "success". If "error" is received, then the client can try again with another username. If "success" is received, that means the client has been successfully enrolled to the system.

After successful enrollment, user should be able to login to the system with the username and the password it selected. Please note that, the clients do not save or store their passwords in any step of the project and the user should always enter it from scratch when the password is asked.

Enrollments are permanent, meaning that after any type of disconnections (Server or clients), the enrolled users must be remembered.

Authentication (Secure Login) Phase

This phase is the implementation of a secure login protocol for the client to get authenticated to the server. In this phase, a challenge-response protocol is run between the client and the server.

If not already entered, first the client should enter the IP address of the server and the port number. Then, the client connects to the server and sends an authentication request to it, together with the username (not the password!). This request is in clear text and initiates the challenge-response protocol. After that, the server generates and sends a 128-bit random number (i.e. the challenge) to this client. Then, the client takes the HMAC of this random number using the lower quarter of the hash of his/her password as the key (128-bit). Remember that we always use SHA-512 as the hash algorithm on the password. Then, the client sends this HMAC to the server. The hash algorithm used in HMAC is also SHA-512. If the server cannot verify the HMAC, it sends a negative acknowledgment message such as “Authentication Unsuccessful” to the client. If the server can verify the HMAC, the client is authenticated. In this case, the server sends a positive acknowledgment message such as “Authentication Successful”. Both positive and negative acknowledgments must be encrypted using AES-128 in CBC mode and then ciphertext must be signed by server's signature RSA key. As AES encryption key and IV, you will use the most significant (i.e. upper) half (256-bit) of the hash of the password that was received during enrollment phase. You have to split this 256-bit hash equally for key and IV (e.g. first 16 bytes for key and next 16 bytes for IV). In other words, both AES key and IV will be 128-bit.

When the client verifies the signature on the acknowledgment, he/she makes sure that the sender of this acknowledgment message is the valid server. If the response is verified, then the client decrypts the message using the key and IV extracted from the hash of the password that it enters (use the same splitting logic as in the server side). After decryption, client checks whether the response is "error" or "success". If it is a failed authentication, the client may try again.

During the decryption of the server message, if the client enters a wrong password, decryption fails of course. You should catch this exception so that the client can enter another password after an error message.

Disconnection

A client may disconnect from the server by pressing a disconnect button or by closing the window. After disconnection, the same user may want to login again by running a brand-new authentication phase. But he/she will not enroll again.

If the Server disconnects by pressing a disconnect button or by closing the window, the connected clients must understand this and get disconnected as well. Later, when the Server opens again, the clients can make fresh connections.

The Client and Server applications must not crash while handling disconnections.

Some Caveats

In the project, some random numbers will be generated. These random numbers must be cryptographically secure ones.

We will test various failed authentication and wrong password cases (such as wrong keys, modified messages, etc.) during the demos; so, please test your own codes accordingly.

So far, it should be clear that the client needs to load the server's RSA public keys from files provided. Moreover, the server needs to load its own RSA public/private key pairs from files. For these purposes, both client and server need to <u>browse</u> the file system to choose the key file(s).

Please remark that this is a client-server application, meaning that the server listens on a predefined port and accepts incoming client connections. The listening port number of the server must be entered via the server GUI. Clients connect to the server on the corresponding port and identify themselves with usernames. Server needs to keep the usernames of currently connected clients in order to avoid the same username to be connected more than once at a given time. There might be one or more different clients connected to the server at the same time. Each client knows the IP address and the listening port of the server (to be entered through the GUI).

This step may be tested with one server and multiple clients in the Demos.

Provided RSA Keys

Client has:

- *server_enc_dec_pub.txt*: This file includes Server’s RSA-3072 <u>public key for encryption operations</u> in XML format.
- *server_sign_verify_pub.txt*: This file includes Server’s RSA-3072 <u>public key for signature verification purposes</u> in XML format.

Server has:

- *server_enc_dec_pub_prv.txt*: This file includes Server’s RSA-3072 <u>public/private key pair</u> for <u>encryption and decryption</u> operations. The keys are in XML format.
- *server_signing_verification_pub_prv.txt*: This file includes Server’s RSA-3072 <u>public/private key pair</u> for <u>signing and verification</u> operations. The keys are in XML format.

Project Step 2 (Due: May 26, 2023, Friday, 22:00)

In the second step of the project, channel key distribution and secure message broadcast mechanisms will be implemented on top of the first step.

Secure broadcasting is basically sending encrypted and authenticated messages among the connected clients via the server. After secure login (authentication), the authenticated clients will receive *channel keys* from the server for the channels they subscribed to. Then, clients will be able to send secure and authenticated messages to the Server. After receiving such a message, the Server will send it to all connected clients that are subscribed to the same channel. While doing so, the server will not decrypt/verify the encrypted messages, it will only relay them to the subscribed clients, who will make the decryptions and message verifications.

Creating a Channel Key and Distributing to Subscribed Clients

In the second step of this project, there will be some changes in **Authentication (Secure Login) Phase** that you implemented in the first step of the project. In the first step of this project, at the end of authentication phase, the server sends a signed "Authentication Successful" or "Authentication Unsuccessful" message depending on the HMAC verification of the challenge. In the second step, however, this process changes a little bit. If the HMAC is not verified, you do not have to change anything. However, if the HMAC is verified successfully, the server must send AES-128 symmetric encryption key, 128-bit IV and 128-bit HMAC key that are specific to the channel that the client subscribed. To accomplish this, Server GUI should include a master secret part for each channel and the entered master secret will be used to generate these keys and IV for the corresponding channel. When generate key button is clicked for the specific channel, server takes the hash of the master secret using SHA-512 and splits it to generate the required keys and IV (16 bytes for AES key, 16 bytes for IV, 16 bytes for HMAC key, 16 bytes not used). You may split the byte array indices of the generated hash [0…15] being the AES key, the byte array indices [16…31] being the IV, and the byte array indices [32....47] being the HMAC key.

Then, the server encrypts the channels keys and IV using AES-128 algorithm in CBC mode. The key and IV used for this encryption are the same as Step 1 authentication phase (derived from client password). The server should append the encrypted keys to "Authentication Successful" message and sign the concatenated message using the signing RSA key of the server. Server, finally, sends (Authentication Successful || encrypted keys and IV) message and its signature to the client.

When the client receives this message, first of all, it verifies that the received signature is valid. If the signature is not valid, it disregards the message. Otherwise, it decrypts the content and load the keys and IV for secure channel communication The cryptographic details of this decryption mechanism are the same as Step 1 (i.e. password entry, etc.).

As you can deduce from the flow above, channel keys and IVs must had been generated before the client authentication. You should handle this case as well. One way of doing this is that during login, instead of sending channel keys and IV, a message saying "Channel

Unavailable" can be sent to the client instead of successful/unsuccessful authentication message. In this way, the clients may try over and over again.

Another thing that you might already deduced is that the channel keys and IV must not be permanently stored at the Server side; they are generated when the Server opens up through the GUI (master key entry and generate key button). When the Server GUI is closed, then these keys and IVs are forgotten. Moreover, once they are created, they should not change during the lifetime of Server GUI; you should handle this.

Secure Broadcasting among Clients via Server

Secure broadcasting a message by a client is basically sending an encrypted and authenticated message to the server. Then the server relays this encrypted and authenticated message to all currently logged in subscribed clients of the channel. Since each client is subscribed to only one channel, he/she does not need to select a channel before sending.

On the client GUI, this feature should be enabled if and only if the user is logged in to the server successfully. In other words, only authenticated clients can broadcast.

When a client wants to broadcast a message, it should encrypt its message using 128-bit AES algorithm in CBC mode using the channel encryption key and IV. Then it should take the HMAC of this encrypted message using the channel HMAC key. The hash algorithm used in HMAC is also SHA-512. User concatenates encrypted message and its HMAC together and sends to the server.

After the Server receives this package, it broadcasts it to all of the connected clients that are subscribed to the same channel as the sending client (including the sending client). Here please remark that the Server does not decrypt and verify HMAC; it broadcasts unchanged.

When a client receives a broadcasted message from the server, it decrypts the message using channel key and checks the validity of the HMAC. If everything is all right, the decrypted message is displayed in the GUI; otherwise, the message is discarded, and error prompt is displayed in the GUI.

Since there are three different channels that clients can subscribe, on the Server GUI we should see three Rich Text boxes for easily capturing the sent/received messages.