CS432/532 Computer and Network Security
Spring 2022 - Term Project
Secure Distributed File System

Project Step 1 Due: May 12, 2022, Thursday, 21:00 (to be submitted to SUCourse+) – All group members will submit the same
Demos: Time and Schedule will be announced later

Project Step 2 Due: June 2, 2022, Thursday, 21:00 (to be submitted to SUCourse+) – All group members will submit the same
Demos: Time and Schedule will be announced later

Submission and Rules

- **CS532 students** should complete the project alone. CS532 students have an option to propose their own projects instead of doing this one; however, in such a case, the project proposal must be sent to the instructor before the deadline of the first step of the project. If no proposals are received, we assume that you will do this project.
- **CS432 students** can work in groups of 3-5 people in both steps of the project, not less not more. You have to group yourselves until the May 6, 2022. You will be informed by your TA Simge Demir about how to send the group information.
- 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.
- 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**. Please **do not use** other compression utilities like RAR, 7z, ICE, bz2, etc.
- 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
Barış Pakyürek – bpakyurek@sabanciuniv.edu
Sacit Emre Akca – sacitakca@sabanciuniv.edu

Introduction

In this project, you will implement a client-server application for establishing mechanisms for a *secure* and *authenticated* distributed file system. In this system, there will be three instances of *Server* module and an arbitrary amount of instances of a *Client* module.

The *Server* module:

- collects files from anonymous clients in encrypted way and acknowledge them by returning a digital signature (first step of the project);
- distributes the files received from clients to the other servers in a secure and authenticated way in order to replicate the file system in a distributed manner (second step of the project);
- sends the requested files to clients in an authenticated way via digital signatures (second step of the project).
- One of the servers is designated as the *Master Server*. The task of this Master Server is to securely distribute session key to the other servers at the beginning and whenever a disconnection occurs (first step of the project).

The *Client* module:

- connects to a Server in order to upload file(s) in encrypted way (first step of the project);
- requests files from any Server and downloads in authenticated way via digital signatures (second step of the project).

Roughly, session key distribution among the servers, starting up the servers, security configurations (loading the keys), secure file upload by clients and server acknowledgment of receipt 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 server and client 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 (all keying material) in hexadecimal format
- Digital signatures in hexadecimal format
- Client/server disconnections
- Verification results (digital signatures, HMAC verifications)
- File transfer operations and file details (such as names)
- HMAC values, session keys, etc. in hexadecimal format

Project Step 1 (Due: May 12, 2022, Thursday, 21:00)

In the first step of the project, the client performs *connection*, *secure file upload* and *disconnection* operations. Moreover, session key distribution among the servers is also performed in this step. The details are explained in the following subsections.

Connection Phase

In this project, there are both client-server and server-server communication. In the client-server part, 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. 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). Note that there are three servers in our system, but it is OK if one client instance connects to one of them only at a given time. However, it should be possible for a client to disconnect and later connect to another server.

In this project, there also are server-to-server connections. Eventually, all three servers will need to connect each other (however, in the first step of the project only master server to non-master server connections are sufficient). It is up to your design which will connect to which of them; you have to design your Server GUIs such that the necessary IP addresses and the listening ports are entered appropriately.

Session Key Distribution among the Servers

There are three servers. One of them is a designated *Master Server*. Secure key distribution among the servers is the responsibility of Master Server. It randomly generates a byte array with 48 bytes (16 bytes for AES-128 key, 16 bytes for AES-128 IV and 16 bytes for HMAC with SHA-256) to be used later in secure file replication phase (second step of the project). You may split the byte array indices [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. This random byte array should be generated using cryptographically secure random number generators, which are available in most crypto libraries.

The protocol for key distribution is as follows. A non-master server (Server1 or Server2) requests session key from the master server by sending a request message together with its server ID. When such a request is received by the master server, randomly generated byte array that includes the session key info (abovementioned random byte array with 48 bytes) is encrypted using the RSA-3072 public key of the requested server and signed with RSA-3072 private key of the master server. Note that RSA-3072 public-private key pairs of the servers are given in the project pack. The requesting server should perform the reverse operations (decryption and verification). If everything is fine, then session key transfer is considered complete; otherwise, another session key transfer protocol is initiated.

This protocol should run immediately after connection between the server and the master server is established.

At the end of this phase, all servers have the cryptographic material to securely replicate the files received from the clients; this secure replication will be performed in the second step of the project.

Secure File Upload and Verification

Each client can upload any type of file (text, executables, pdf, word, video, audio, etc.) of any length (the files can really be very big) to any of the servers including the master server. In order to upload and store a file to a server in a confidential way, firstly the client chooses the file to be uploaded by browsing his/her file system. In order to secure the file transfer, the client generates two random values: (i) a random 128-bit value to be used as AES key and, and (ii) another 128-bit random value to be used as IV for AES encryption in CFB mode.

The file to be uploaded will be encrypted in AES-128 CFB mode by the client using the 128-bit AES key and the IV mentioned in the previous paragraph. Since the server does not know the encryption key and IV, client should send them together with the encrypted file. The abovementioned randomly generated AES key and IV should be sent to the server by encrypting with the RSA-3072 public key of the server. Also, the name of the file should be sent to the server.

After all, the encrypted file, encrypted file name, encrypted AES key and IV are all concatenated and sent to the server. When the server receives them, first it decrypts the AES key and IV and then decrypts the file and its name using them. If the decryption(s) fail(s), the client must be informed by the server with a signed message and the file is not to be stored in the server. If everything is OK, the file will be stored at the server side and returns a positive acknowledgment in a signed message. This positive acknowledgment must include the server's signature on the received file. After this signature is received by the client, it is verified and an appropriate message is displayed at the client GUI.

The server should keep all of the received files in one folder using the names received from the clients. If the same file name is used more than once, previously stored ones are overwritten.

Disconnection

The client GUI will have a disconnect button. When a client disconnects from the server by pressing a disconnection button or by closing the GUI form window, he/she must clear server’s RSA public key values from the memory (not from the file system). After disconnection, the same client may want to connect again to the same or another server.

If disconnection occurs due to server side (when Server is somehow closed), the client should sense this case and act appropriately (e.g. going to initial state of no connection and trying brand new connection).

Server to server disconnection is also an issue to consider. Although there will not be a disconnect button for Server, any of the Servers may get closed (by closing the form window). If Server1 or Server2 disappears, Master Server does not need to take an action. However,

once the connection is restored, secure key distribution protocol should run again. If the Master Server gets disconnected, then the same procedure is applied for both Server1 and Server2. It is up to you how to restore the connections in such cases.

Provided RSA Keys

Master Server has:

- *MasterServer_pub_prv.txt*: This file includes Master Server's public/private key pair in *XML* format.
- *Server1_pub.txt*: This file includes Server1's public key in *XML* format.
- *Server2_pub.txt*: This file includes Server2's public key in *XML* format.

*Server*1has:

- *Server1_pub_prv.txt*: This file includes Server1's public/private key pair in *XML* format.
- *MasterServer_pub.txt*: This file includes Master Server's public key in *XML* format.
- *Server2_pub.txt*: This file includes Server2's public key in *XML* format.

*Server*2has:

- *Server2_pub_prv.txt*: This file includes Server2's public/private key pair in *XML* format.
- *MasterServer_pub.txt*: This file includes Master Server's public key in *XML* format.
- *Server1_pub.txt*: This file includes Server1's public key in *XML* format.

Client has:

- *MasterServer_pub.txt*: This file includes Master Server's public key in *XML* format.
- *Server1_pub.txt*: This file includes Server1's public key in *XML* format.
- *Server2_pub.txt*: This file includes Server2's public key in *XML* format.

Project Step 2 (Due: June 2, 2022, Thursday, 21:00)

In the second step of the project, mechanisms for file replication among the servers and secure file download by clients will be implemented on top of the first step (the mechanisms implemented in the first step should be functional in the second step as well).

One of the functionalities that you will implement in the second step of the project is file *replication* across all servers. The requirement here is that whenever one of the servers (Master or not) receives a file from a client, that server should send the same file to other two servers in encrypted and authenticated way (using the session keys distributed in Step 1).

In the second step, clients will be able to *download* any type of file (text, executables, pdf, word, video, audio, etc.) of any length (the files can really be very big) from any of the servers anytime. The files to be downloaded from the server are digitally signed but not encrypted. The client requests the files with the given file names. When requested, the file and the server signature of it are sent to the requesting client.

A particular client can request a file uploaded by himself/herself or another client. Moreover, since the uploaded files are distributed (replicated) to the other servers, a client can request a file from any of the servers including the Master Server.

Replication and *Download* operations are explained in the following subsections. Connection and disconnection issues are also explained below.

File Replication

As soon as a file is uploaded to any of the servers by a client, the file is shared with other two servers in a secure and authenticated way. To do so, the server first encrypts the file and file name in CFB mode of AES-128 algorithm. Also, server generates HMAC of the file and file name using SHA-256 as the hash algorithm. Then, the encrypted file, encrypted filename and the generated HMAC are concatenated and sent to the other two servers. Here the AES key, AES IV, HMAC key are the ones that are previously shared as session key in the first step of the project.

After getting the encrypted data and HMAC, the receiving server first decrypts the relevant parts of the message using the session key. In order to verify the authenticity, receiving server generates the HMAC with SHA-256 of the decrypted file and file name, and compares with the incoming HMAC.

- If the HMAC is verified successfully, receiving server returns a signed success message and stores the file in a folder with the given file name (same file name issues are addressed as in step 1). The sending server verifies the signed success message and displays the result on the GUI. Here the signatures are issued and verified using the RSA-3072 keys of the corresponding parties.
- If the HMAC verification fails, receiving server returns a signed failure message and does not store the file. The sending server verifies the signed failure message and

displays the result on the GUI. Here the signatures are issued and verified using RSA-3072 keys of the corresponding parties.

Download a File in Authenticated Way

Each client can request to download a file stored at the servers' side. Please note that since the files are replicated among the servers, a client may request the same file from any of them, not necessarily from the server to which it uploaded. Moreover, this system also allows to download the files uploaded by other clients.

The download procedure is explained below.

Requesting Client: X, Requested Server: S File: F

1) X → S: sends a download request for F using its name.
2) S checks whether F is in its folder or not.
3) If exists; S → X: sends F in clear but signed using RSA-3072 private key of S.
4) If not exists; S → X: sends an error message, signed using RSA-3072 private key of S.
5) In either case, X verifies the signature and displays the result of verification on GUI (file received successfully, file received but not genuine due to signature problem, file not found, file not found message received with invalid signature, etc.). The received file is stored in X's folder only if it is received with a valid signature; otherwise, no file will be stored.

Inıtial Server Connections and Dealing with Server-to-Server Disconnections

In the first step, Server1 and Server2 are connected to Master Server, but in the second step of the project, Server1 and Server2 must connect to each other as well. File replications must not start before all three servers are connected to each other. Once all three servers are connected to each other, first session key distribution protocol will run and then the file replications will start.

The servers will not have disconnect button. However, the servers (master or not) may get disconnected any time by closing the GUI form window. Other side of disconnection should be aware of this case. If there is such a server-to-server disconnection in the system, all file replications must stop temporarily until the connections are restored (even between the servers not disconnected). Once the server(s) come(s) back and connections are restored, first the session key distribution protocol runs from scratch (for both Server1 and Server2 even they are not disconnected) before continuing with file replication. Meanwhile, the clients may send files to the online servers and these servers should queue these files up to be replicated when the server-to-server connections are restored.