departamento de electrónica, telecomunicações e informática
universidade de aveiro
theoria poiesis praxis

Secure Messaging Repository System

Security 2017/2018

Verónica Rocha (68809)

Miguel Ferreira (72583)

Class 1, Group 1

MIECT

INDEX

1. Introduction

This project was developed in the context of the Security course of the fourth year of Mestrado Integrado em Engenharia de Computadores e Telemática da Universidade de Aveiro. The goal of this project is to develop a system that enables users to exchange messages asynchronously through a non-trustworthy repository that saves the user's messages.

The Secure Message Repository System is designed to allow user to register, list all users that are already registered, send and receive messages, as well as list all the messages exchanged between users and if they were read by the other user or not. This must be implemented in a way that assures message confidentiality, integrity and authentication, message delivery confirmation and identity preservation.

The system has two main components: the clients and the server. Multiple clients interact with each other through the server. Through the connection between the components, several processes are to be supported, such as creating a user's message box, list other user's message boxes, list all the new messages received by the user, list all messages received (even the ones that were already read), send a message to a specific user, receive a message from a user message box, send receipt for a received message and list all messages that were sent, along with the respective receipts.

The entire system was implemented in Python 2.7.11.

2. Libraries and Algorithms

2.1. EXTERNAL LIBRARIES

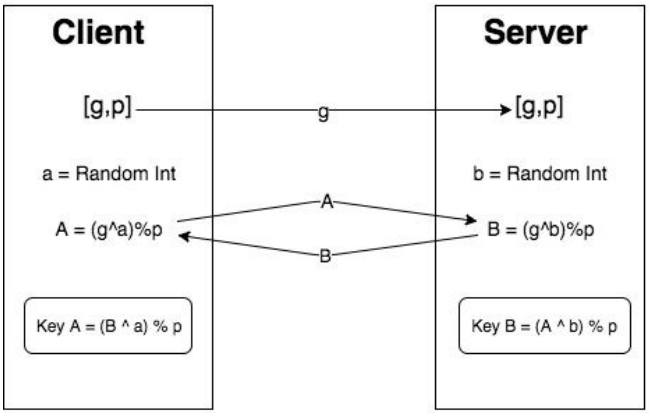
- Crypto
- OpenSSL
- PyKCS11
- RSA

2.2. ALGORITHMS

All the symmetrical keys were created using the AES cipher mode CBC.

The asymmetric key pair was created using the RSA algorithm.

Diffie-Hellman was used to exchange keys between server and client.



3. System Functionality

When a user opens the program, he/she chooses a username. If the user already exists, it returns the user's id from the previous session, otherwise, a new one is created.

```
Please choose a username: vr
Your uuid is : c873da24-8748-52ba-9687-ba33225401c1

1 See new messages.
2 See all messages.
3 List all users.
4 Send message.
5 Status.
6 Exit.
Please Select:
```

If the user selects the first option, See new message, the following menu will appear with the list of new messages.

```
New messages:
        e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1
        e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_2
Choose message you want to read:
```

After choosing which message to read, in this case the e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1, the message that the user sent will be displayed together with the verification. If the verification is ok, it means that the verification of the citizen's card signature of the user who sent the message was verified.

```
e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1:

        test message

        Verification: ok
```

When the user vr reads the message, a pop up will appear to ask for the authentication card code of authentication. This happens so that a receipt message is signed and sent, this way, the other user will know if the status of the message.

If the user selects the option to list users, the system will print the id of all the users connected to message box.

```
1 See new messages.
2 See all messages.
3 List all users.
4 Send message.
5 Status.
6 Exit.
Please Select: 3

Users connected to the Message Box:
        e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322
        e2cf2e1d-3d57-57e5-890b-ca375b61a1d6
        1b17aeb2-7fe8-5832-85df-a6967e76dbbe
        c873da24-8748-52ba-9687-ba33225401c1 -> Me
```

If the user wants to send a message to another user, the option 4 has to be selected and the list of users registered in the system will appear. It is necessary to insert the id of the destiny user and, afterwards, the message to send. Also, the citizen card authentication pop up will appear again so that the user puts the code, this will allow the signature of the message.

```
1 See new messages.
2 See all messages.
3 List all users.
4 Send message.
5 Status.
6 Exit.
Please Select: 4

Please choose an user to communicate with:
e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322
e2cf2e1d-3d57-57e5-890b-ca375b61a1d6
1b17aeb2-7fe8-5832-85df-a6967e76dbbe
Please insert the uuid of the user: e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322
Please insert your message:
```

If the user wants to see all messages, the ones that recieved and the ones that he/she sent, the client selects 2 and a list of all the sent messages and all received messages will be displayed.

```
1 See new messages.
2 See all messages.
3 List all users.
4 Send message.
5 Status.
6 Exit.
Please Select: 2

Recieved message list:
        e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_2
        _e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1
Sent message list:
        e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1
```

By choosing the status option, a list of the messages that were sent to other users will appear. The user chooses the message to read and it will be presented. In the example presented below, the content sent is presented, as well as the receipt of the message that includes the time it was received, the id of the destination user, the receipt itself and the verification of the signature.

```
1 See new messages.
2 See all messages.
3 List all users.
4 Send message.
5 Status.
6 Exit.
Please Select: 5

Sent message list:
        e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1
Choose message you want to read: e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322_1

Message sent:
        hello
Receipts:
        Time received: 1514744035302
        Destination id: e9ba9c1e-6a5e-5b92-bd82-ab9be25e3322
        Receipt: VnfgjzIAw6e6uOdWJarBZvgcFwzS3+en1ySB/VuEZK/6n31Za5Yy1t1pSXpa8ZGdxvXsfFPi
RFMR/4dgesd7G596TfzuaodId3aK2PgrJxSOWiH2QOF7N5JMIEAWFF1PMuYsOPCdLtvAvFzJ/3sSxTbzpNzdI0306
ltJLWURyaQ=
        Verification: ok
```

4. Client

The client connects to the server through socket communication.

When a client connects, it is necessary to retrieve client information according to a given username or, in case this client doesn't exist yet, create a new one. If the user already exists in the system, the client keys folder will be checked to inquire if the uuid already exists in the respective file. If the key exists, it is retrieved, otherwise, a new pair of RSA keys will be generated and retrieved. Finally, the citizen card public key is retrieved and a dict with the user base info, such as the uid, user uuid, username and the citizen card public key, is returned.

After this, a connection with the server needs to be established in order for it to return the symmetric key associated with the connection. In order to accomplish this, a generator word, that is equal to the one that will be used by the server, is used in the AES algorithm to generate an AES cipher. This cipher will be used to cipher the value g and a, which are parameters of the Diffie-Hellman algorithm. These values are sent to the server, which, in turn, will answer with the value of b. Afterwards, the user's public key and respective citizen card key are encrypted with the session key and sent to the server, which will return the user uuid.

From this point forward, all the messages exchanged between server and client are ciphered using the aes cipher with the session key.

After the client is registered, it is presented with 5 different menu options: See new message, see all messages, list users, send message and message status.

If the user chooses to see the new messages, a request message with the type 'new', encrypted with the AES cipher, is sent to the server with the respective user id. If the server doesn't send new messages, a information will be printed notifying the user that there are no unread messages, otherwise, it prints the id of the sender's identification and the number of the message and asks the user which message to read. Once the user selects a message, a request of the type 'recv' is sent to the server in order to get the message received that is stored in the mbox of the client. A verification of the respective signature is made, returning an ok or fail depending on the result.

After the visualization of the content, a receipt message is elaborated with the signature and certificate of the user who read it before being sent to the server with the type 'receipt'.

To see all the messages in the message box of the user, the process is similar to the process described above except for the fact that the server returns the list of messages in the user's message box to a 'all' request type, however, the user cannot read the same message twice.

In order to get the list of clients registered in the system, the client sends a 'list' message to the Server so that it responds with the list of users.

When the user wants to send a message, the client will send a message requesting the list of clients to the server (i.e. the same process as to list the clients) and when it retrieves it, presents the list so that the user can choose the id of the user to whom he/she wants to send the message. Having the uuid of the destiny user and the message to send to said user, the signature and certificate of the client's citizen card are retrieved and concatenated with the message in a python dict. For the communication to work, it is necessary to encode the message in base64. In addition, a copy of the message to send is encrypted using the RSA asymmetric key algorithm with the client's public key. The client sends a request for message 'send' to the server.

The status will request all the messages sent and received to the server and will present the ones that were sent. After choosing the id of the message, the content of the message itself, as well as information about the message, the receipt itself, the verification of the signature and validation of the certificates.

5. Server

The server, in terms of code, is divided in four different files that were given by the teachers. The server.py file was not modified during the execution of this project.

5.1. Server Client

In the server_client, alterations were made when the results of a certain request is being sent to the client. It was necessary to verify the type of the message and encrypt if the type is any other than ‘create’. A ‘to’ field was added to the object with the address of the client to which the message will be delivered to identify the client, because the client who receives the message may not be the one that sent it.

5.2. Server Actions

The handleRequest function was altered in order to decrypt the message using the AES cipher algorithm with the session key for that client.

Also, the processCreate function, called when the server receives a message from from the client requesting the creation of the communication, was altered to decrypt the data that was encrypted by the client, in the case that the ciphers is ‘START_AES’, the g and a parameters of the Diffie-Hellman algorithm, with an AES cipher using a generator key that is the same for every client and is not shared. Using the values retrieved, the session key and the b parameter of DH are determined, the b is encrypted using the same AES key and cipher and sent to the client.

Apart from this, many verifications and validations of other fields were added, for example, validations to the signature and certificate fields when the message type is ‘send’.

5.3. Server Registry

In this file, many changes were made, but the main difference was to change from sequential ids to unique identifiers of 32 bytes generated by a specific function, which required changes in most of the remaining code.

On the sendMessage, each time a message is received, it is encrypted and the public key from the citizen card is added to the message. It was necessary to perform an

encoding operation on the message to prevent any problems with any different characters in the message sent by the client, whatever the encoding of the message is.

In the stored receipt, the citizen's card public key was also added to the message and a new verification of the state of the message in the message box (whether it was read or not) before being stored in the receipt box.

Lastly, in the function used to get the receipts, the citizen's card public key was also added to the receipt itself for verification purposes.

6. User

There was no class User created specifically and separately, however, a client_info folder was created to store the client's public and private keys (encrypted), username, server uid, uuid and the public key from CC.

The server stores the ID and description of the user's public keys, both the client's unique one and the citizen's card one.

7. Citizen Card

The citizen card class is used both in the client and server to sign messages, verify the signature, get the certificate, validate said certificate and get the citizen's card public key.

The sign function, given the data that is to be signed, loads the PKCS11 library, gets the slots available and uses one of the slots to open a session. Using the session keys and information, it signs the giver data and returns the result.

The verifySignature function uses the original data, the signature and the public key to verify the signature.

For the function that gets the certificates, it takes a label as a parameter that will be "CITIZEN AUTHENTICATION CERTIFICATE". As it occurs in the sign function, it is necessary to load the library, get the slot list, open session and use that session to retrieve data. In this case, retrieve the certificate, that will be loaded using OpenSSL and then returned.

The getPublicKey function returns the public key associated with the citizen card that is currently in the slot.

The validateCert function was supposed to validate the certificate using a chain validation. However, the validation didn't work as expected, so the code was commented so that the result of this function was always true. This way, it didn't affect the functionality of the program.

8. Run the project

To execute the project, it is necessary to run the server first. After this, it is possible to run multiple clients. The execution is made in the command line.

9. Conclusions

In the development of this project, there were several problems. One of them was the encoding and decoding of the messages, keys and signatures, that a lot of times were not compatible and were a lot of work to fix.

Also, in the end, the certificate validation did work as expected.

Even with the problems, this assignment was a challenge and allowed to better understand the practical and theory classes and enabled us to learn more while solving the problems and challenges presented in the course of this project development.

10. References

- André Zúquete, Segurança em Redes Informáticas , 4a Edição Aumentada, FCA - Editora de Informática, Lda.
- Portuguese Citizen Card Official Software, https://www.autenticacao.gov.pt/o-cartao-de-cidadao
- PyKCS11 , https://bitbucket.org/PyKCS11/pykcs11/src