UNIVERSITÀ DI PISA

Computer Engineering

A Secure Bank Application

Giovanni Marrucci

Foundation of Cybersecurity - Project A.A. 2022-23

Contents

1. Introduction

My project is developed in C++ for Linux systems and consists of 2 different programs: client and server. Both programs need the server port number as an argument at execution time. I decided to use the localhost IP “127.0.0.1” as default, this has been written in the “defines.h” file and it is compiled with the server and client. To ease the development and concentrate on the security features I decided to use a single thread server. All packets traversing the network were properly serialized. I used TCP instead of UDP as the protocol for communication over the network because it ensures the reliability of packet delivery. It also provides the flow and the error control features. Moreover, it guarantees that a packet arrives at its destination without any duplication and with the same data order.

1.1 Server

When the program is launched listens on the main socket and waits for new connection requests from clients. Whenever a new client tries starts the authentication and afterwards, if the procedure was successful, starts accepting commands from it. The server can close the connection if there are some error situations but mainly waits for the client to disconnect.

1.2 Client

When the program is launched it asks for the user’s name and password to open the private key and start the communications with the server. If the authentication is successful, it will receive all the possible issuable commands:

1. Balance()
2. Transfer(username, amount)
3. History()
4. Logout()

Whenever ”Logout()” is issued the client disconnects from the server and the program terminates.

1.3 Communication between client and server

In general, the receive and send functions used in the programs always do 2 things:

1. Receive/send the length of the message.
2. Receive/send the message itself.

I decided to implement this functionality to be always sure to know beforehand how big is the packet that we are receiving.

2. Protocols and algorithms

As it was written in the requirements of the project, I generated for all users and the server a pair of keys. These keys were created using the openssl library issuing the following commands from the Linux terminal:

1. openssl genrsa -aes128 -f4 -out user_privK.pem
2. openssl rsa -in user_privK.pem -outform PEM -pubout -out user_pubK.pem

I used the station-to-station protocol to establish the symmetric keys necessary for the connection between the client and the server. Since we had pre-shared public keys, I didn't send the certificates with the signed messages. To verify the signatures both the client and server take the keys that they already have in storage. Using this protocol, we have PFS (Perfect Forward Secrecy) and DA (Direct authentication).

With the protocol above I decided to establish two session keys: the first to encrypt the messages that passed through the network and the second to use a HMAC policy. This was done to ensure message authentication, confidentiality, integrity, and non-malleability.

To avoid replay attacks I used a counter to ensure freshness.

Regarding the files stored in the server I used the digital envelope to crypt and decrypt them. Moreover, every user has 2 files, the first in which we have its balance and user Id, the second one in which we have all the transfers performed by him or towards him.

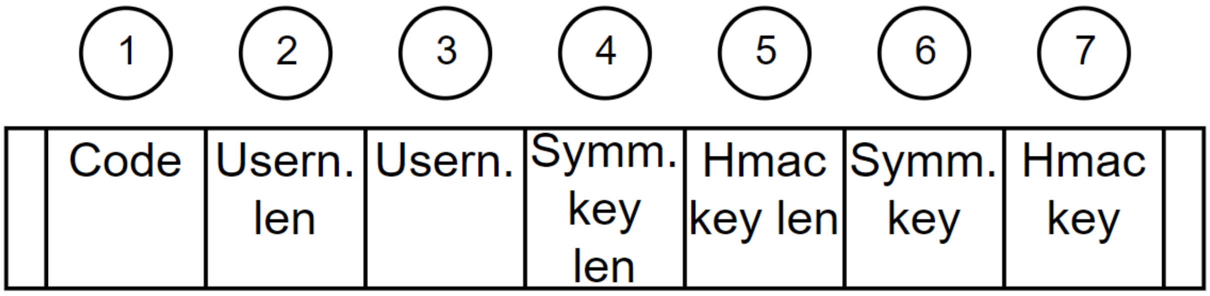
NOTE: The keys sent to establish the shared secret are serialized and deserialized using the BIO structures provided by the openssl library.

2.1 Format of the exchanged messages

I created 3 different structures to perform communication between the server and the client.

2.1.1 Wave packet

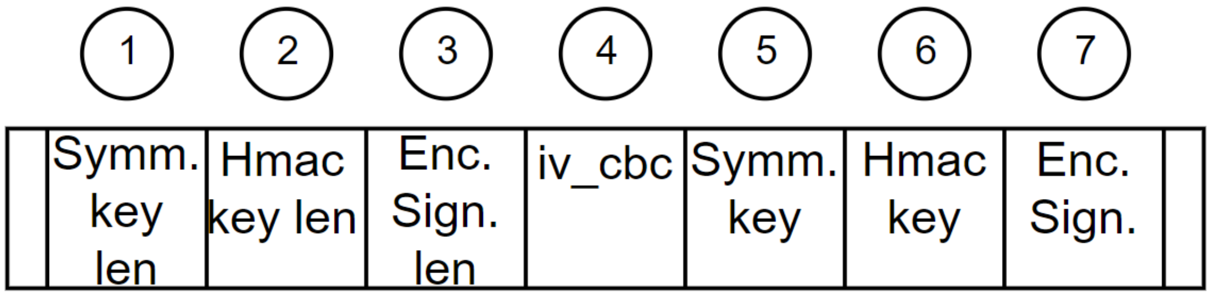
This packet is sent in clear by the client to the server to start the station-to-station protocol. The packet is as it follows:



1. Code: to identify the packet.
2. Username length: to know the length of the username.
3. Username: to know who is sending the communication request.
4. Symmetric key length: to know the length of the symmetric key sent.
5. Hmac key length: to know the length of the hash key sent.
6. Symmetric key: to establish the shared secret to encrypt the messages -> g^a.
7. Hmac key: to establish the hmac key to compute the Hmac of the messages ->g^b.

2.1.2 Login authentication packet

This packet is sent both by the server and the client. It is initially sent by the server to the client because it needs to generate the g^c for the symmetric key and g^d for the hash key. In fact, after generating the parameters, it creates the shared secrets using the g^a and g^b received with the wave packet. The resulting packet sent through the network is as it follows:

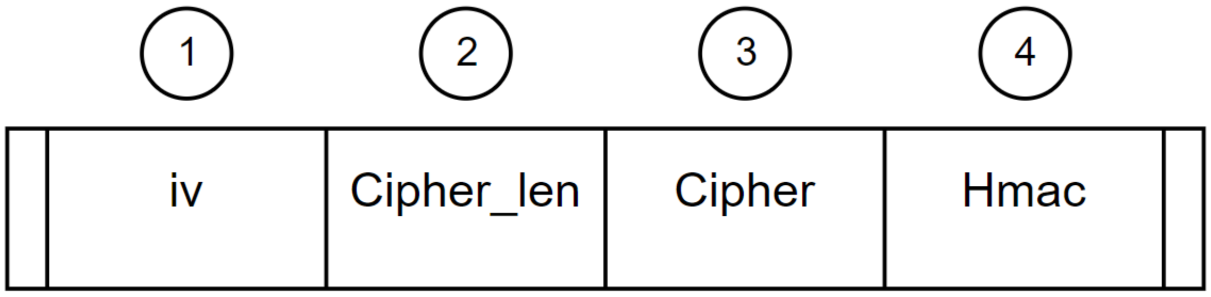


1. Symmetric key length: to know the length of the symmetric key sent.
2. Hmac key length: to know the length of the hash key sent.
3. Encrypted signature length: to know the length of the signature sent.
4. Iv_cbc: the initialization vector used to encrypt the signature.
5. Symmetric key: to establish the shared secret to encrypt the messages -> g^a or g^c .
6. Hmac key: to establish the hmac key to compute the Hmac of the messages ->g^b or g^d.
7. Encrypted signature: to verify the origin of the message.

NOTE: the signature is a concatenation of the 4 lengths of the keys plus the keys themselves, encrypted by the shared secret K = g^{ab} mod p.

2.1.3 Generic message

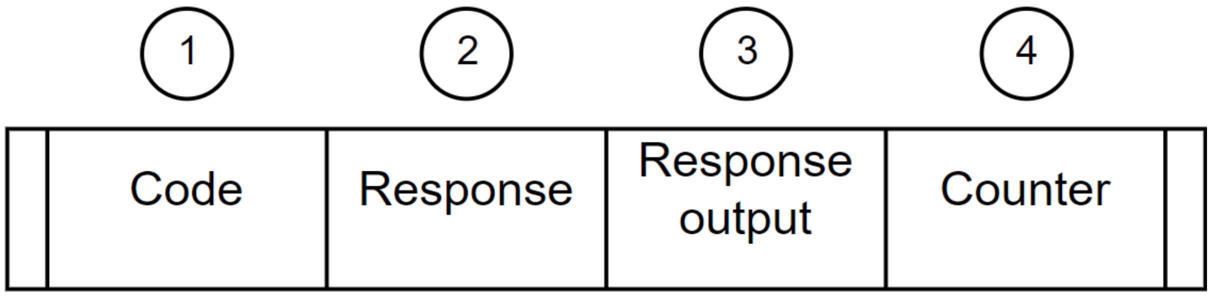
This packet is sent during normal communication. It is an encrypted version of the server packet and client packet (that I will explain later). The packet is as it follows:



1. Iv_cbc: the initialization vector used to encrypt the message sent.
2. Ciphertext length: to know the length of the ciphertext sent.
3. Ciphertext: the encrypted message with the symmetric key.
4. Hmac: the hmac of the ciphertext calculated with the hmac key.

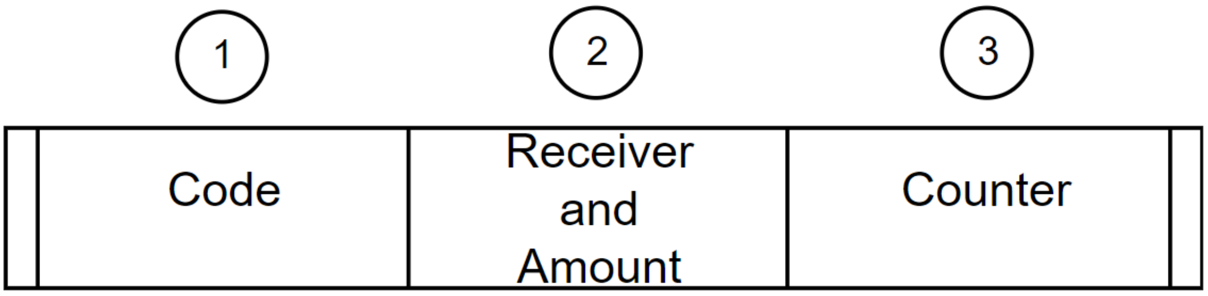
As I have previously mentioned these messages are the encrypted version of the plaintexts generated by the server and client. The packets are as it follows:

- Server:



1. Code: to identify the request.
2. Response: the response can be either a 1 or 0 depending on the result of the request.
3. Response output: the string that is returned to the client to give information.
4. Counter: used for the freshness of the message.

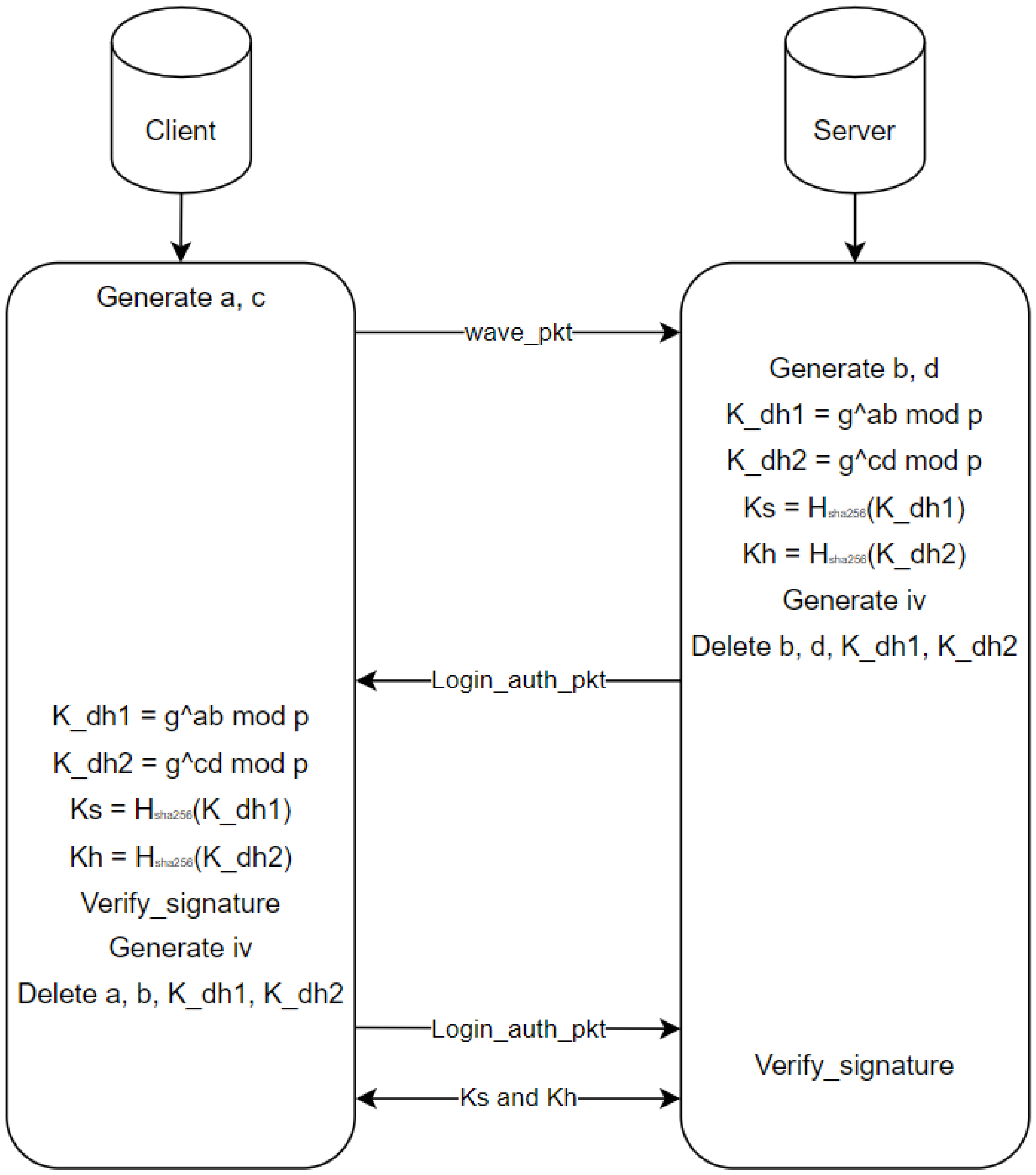
- Client:



1. Code: to identify the request.
2. Receiver and amount: used for the Transfer() command.
3. Counter: used for the freshness of the message.

2.2 Authentication and Key establishment

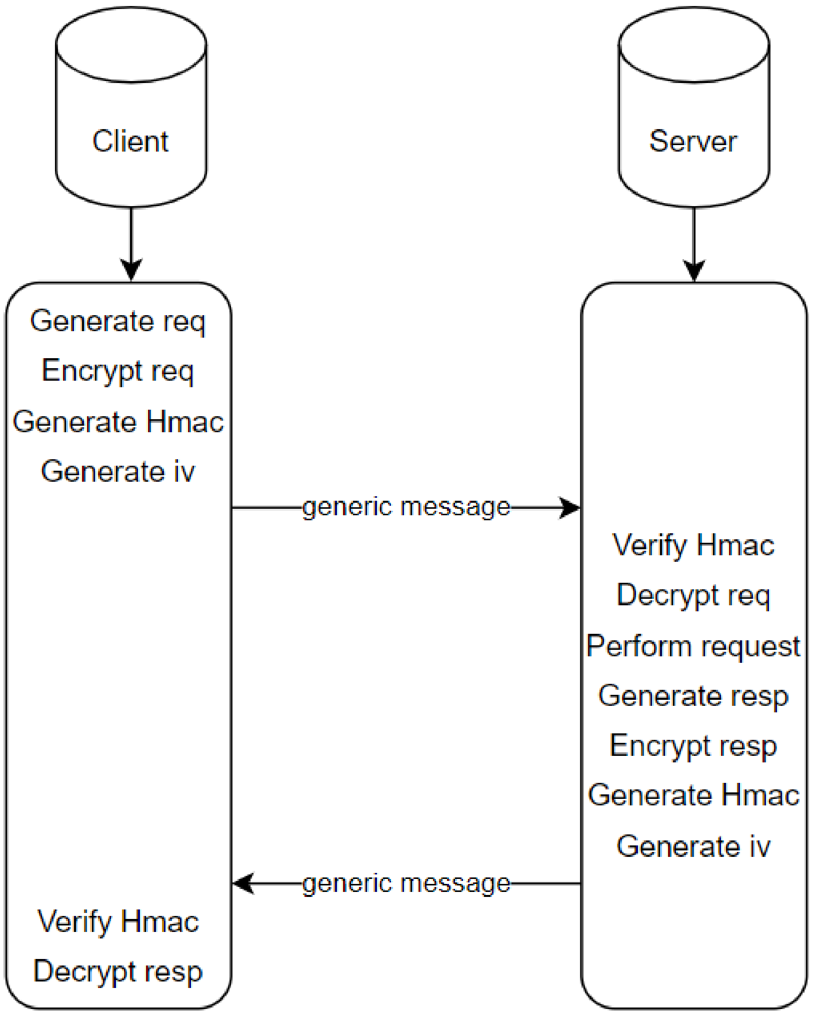
Using the previously mentioned packets this is what happens to establish the session keys.



The server before generating b and d checks for the username's validity. If the user is not registered with the bank, then it terminates the connection with the client. If either the client or the server can't verify the signature, then the connection terminates.

2.3 Message exchange

This is what happens after the key establishment. Encryption is always done using the symmetric key , while the Hmac is generated using the Hmac key. The previously mentioned keys were the ones established during the station-to-station protocol.



Both the server and client discard the generic message if they cannot verify the Hmac. Moreover, the messages can be discarded after the decryption whenever the counter, or the code is not correct.