*Secure four in a row*

*Foundation of cybersecurity project A.A. 2019/20*

*Lorenzoni Dario, Nocerino Raffaele, Xefraj Riccardo*

Sommario

[**Introduction:** 3](#_Toc46501486)

[Project specifications: 3](#_Toc46501487)

[Implementation notes: 3](#_Toc46501488)

[Cryptographic Algorithms used 3](#_Toc46501489)

[**Protocol and Exchanged Messages:** 4](#_Toc46501490)

[Client-Server: 4](#_Toc46501491)

[*Authentication Phase – Protocol:* 4](#_Toc46501492)

[*Authentication Phase – Format of the messages used:* 4](#_Toc46501493)

[*Authentication Phase – General behaviour:* 5](#_Toc46501494)

[*Shared Secret Establishment – Protocol:* 6](#_Toc46501495)

[*Authentication Phase – Format of the messages used:* 6](#_Toc46501496)

[*Shared Secret Establishment – General behaviour:* 7](#_Toc46501497)

[*Communication between Client and Server - General behaviour:* 8](#_Toc46501498)

[*Communication between Client and Server - Protocol:* 8](#_Toc46501499)

[Peer-to-peer game: 12](#_Toc46501500)

[*P2P authentication:* 12](#_Toc46501501)

[*P2P Shared secret creation:* 12](#_Toc46501502)

[*P2P Game:* 12](#_Toc46501503)

[BAN logic proof of client-server authentication and key exchange 14](#_Toc46501504)

[Real protocol 14](#_Toc46501505)

[Assumptions 14](#_Toc46501506)

[Objectives 14](#_Toc46501507)

[Idealized protocol 1](#_Toc46501508)

[Proof 1](#_Toc46501509)

[M2: 1](#_Toc46501510)

[M3: 1](#_Toc46501511)

[M5: 2](#_Toc46501512)

[M6: 2](#_Toc46501513)

[M7: 2](#_Toc46501514)

[M8: 3](#_Toc46501515)

[M9: 3](#_Toc46501516)

[BAN logic proof of P2P authentication 4](#_Toc46501517)

[Real protocol 4](#_Toc46501518)

[Assumptions 4](#_Toc46501519)

[Objectives 5](#_Toc46501520)

[Idealized protocol 5](#_Toc46501521)

[Proof 5](#_Toc46501522)

[M0-A: 5](#_Toc46501523)

[M0-B: 6](#_Toc46501524)

[M2: 6](#_Toc46501525)

[M4: 6](#_Toc46501526)

[M5: 7](#_Toc46501527)

[M6: 7](#_Toc46501528)

[M7: 8](#_Toc46501529)

[M8: 8](#_Toc46501530)

# **Introduction:**

## Project specifications:

The application should implement a secure online version of the game “Four-in-a-row”. The protocol used for client-server and client-client communication must not be vulnerable to replay attacks, man-in-the-middle attacks, and perturbations (perturbations must be detected).

It is assumed that the server has the public key of every user subscribed to the service (a priori).

The server is used to:

* Give information about online users ready to receive a challenge
* Receive challenge and waiting request from the players
* Receive challenge response from the users
* Give the need information to make two users communicate with each other in a P2P fashion

When two clients are given the need information’s to contact each other, they can start playing the game.

At the end of the game, they disconnect from each other and can decide to quit the service or play another match with other opponents.

## Implementation notes:

The protocol used for the communication is UDP since we assume that the network reliability in term of packet loss is not relevant and UDP gave us a higher flexibility useful for the client management part in the server.

Before the client-server and client-client communication the peers are authenticated, and a share key is created.

Every message is encrypted by using this shared key.

## Cryptographic Algorithms used

**Authentication**

To authenticate the users (to the server and to each other) there are sent some random\_data to authenticate. The random\_data is sent in clear, on the random\_data is applied a Digital Signature using the hashing function sha256 and the private key of the sending user.

**Session key creation**

To create the session key has been used the standard version of Diffie-Hellman and on every message sent is applied the Digital Signature (since Diffie Hellman does not authenticate the parties). At the end when all the parties have the session key, the session key is hashed using sha256.

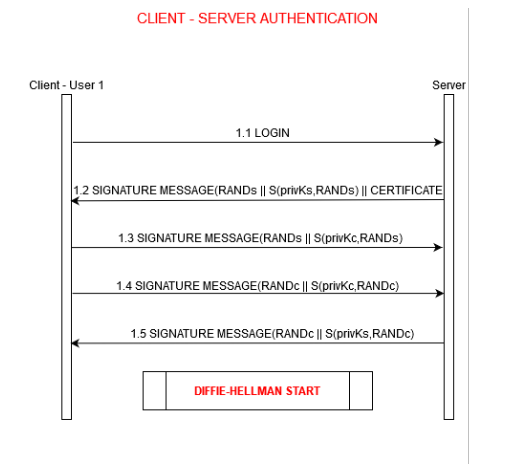
**Message Exchange**

Now both ends have the session key now the AES GCM is used to send the secrets. Since our messages are all encrypted despite the IV, the AAD used is the same as the IV of the messages.

# **Protocol and Exchanged Messages:**

## Client-Server:

### *Authentication Phase – Protocol:*



### *Authentication Phase – Format of the messages used:*

As wee can see from the picture below, there are two different messages for signature/authentication and certificate. Actually from the protocol point of view the signature message and the certificate are sent both in the same msg, but in order to implement this mechanism well on C++ we had to separate them in two different messages (to let the receiver know a priori the size of the received certificate allocating accordingly buffers).

Immagine che contiene testo

Descrizione generata automaticamente

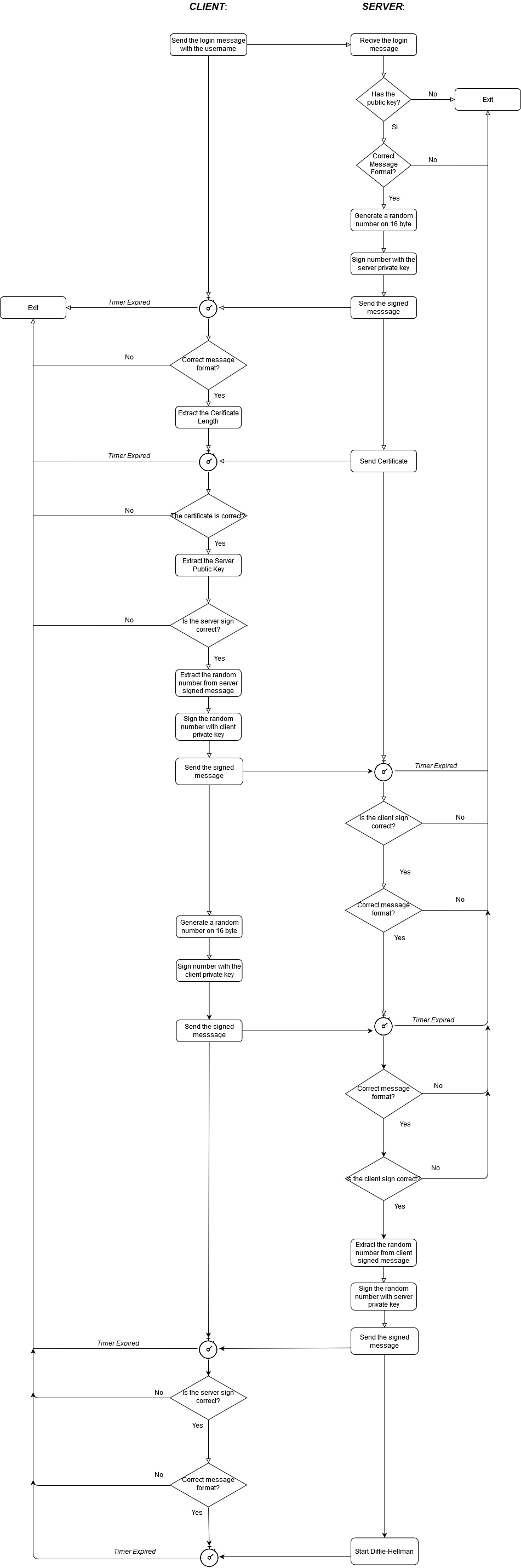
Immagine che contiene testo

Descrizione generata automaticamente

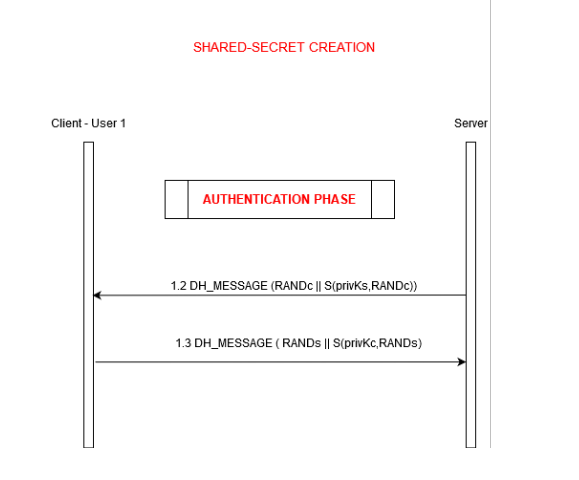
Immagine che contiene testo

Descrizione generata automaticamente

### *Authentication Phase – General behaviour:*

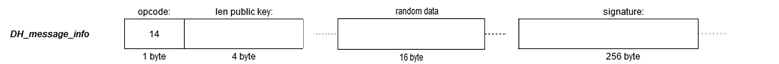


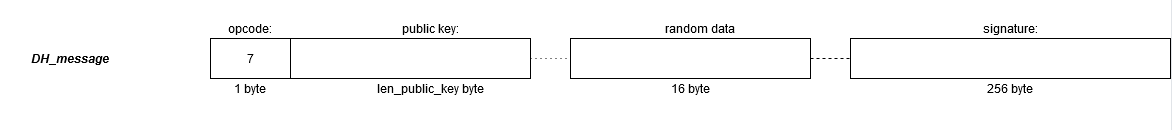
### *Shared Secret Establishment – Protocol:*



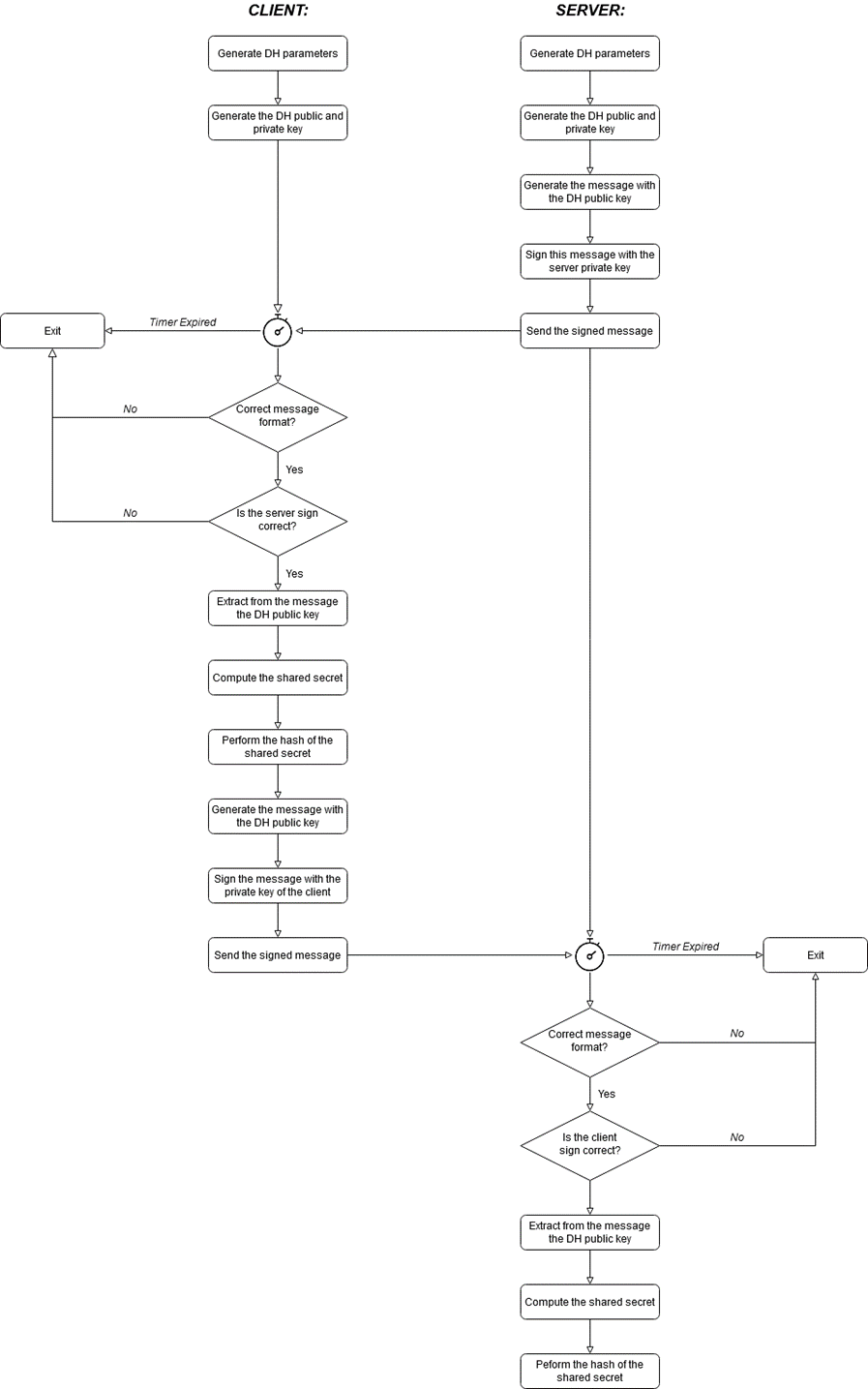
### *Authentication Phase – Format of the messages used:*

Also here there’s a little difference from the protocol and the message structure. Like before from the protocol point of view the DH message and the DH Message info (which contains the length of the DH public key) are sent both in the same msg, but to implement this mechanism well on C++ we had to separate them in two different messages (to let the receiver know a priori the size of the received DH public key allocating accordingly buffers).





### *Shared Secret Establishment – General behaviour:*



### *Communication between Client and Server - General behaviour:*

Immagine che contiene testo, mappa

Descrizione generata automaticamente

The image above describes the various action that the user can perform in particular after the authentication and secret key establishment phase the system will display to the user a message that ask for what the user wants to do. The user can choose if he wants to wait for a challenge or send a challenge to a waiting user (based also on the list of users that are currently waiting for a challenge that he receives). If he choose to send a new challenge -with the command **!challenge**- the system will ask for the username of the user that he wants to challenge and then will wait for the response from the other client. If instead the user chooses to wait for a challenge request -with the command **!wait**- the system will display an incoming request if any arrives but if instead no one arrives within 3 minutes the system will automatically come back to the first point and give the possibility to the user to choose between !challenge and !wait. Not shown in the picture in any moment the user can also choose to digit the command !exit in order to quit the session with the server in this case every structure that contains the information about the user will be deleted from the system.

### *Communication between Client and Server - Protocol:*

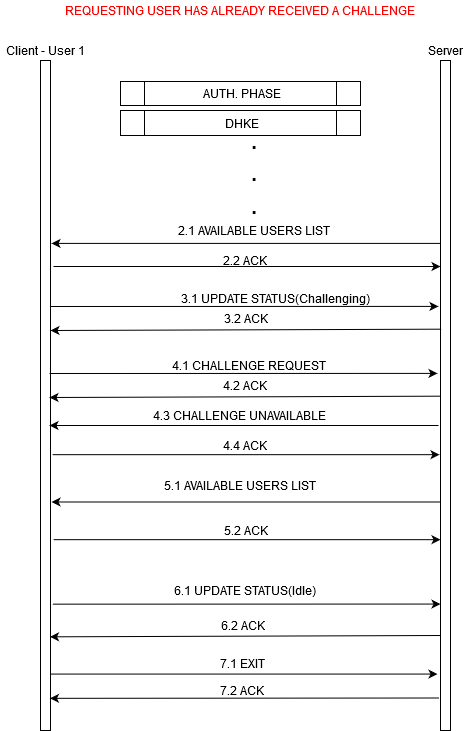
Each message during this phase is encrypted and decrypted using the shared secret derived in the previous phase. Moreover during this phase each time an entity -the user or the server- receives a message it immediately check -after decrypting the message- the format of the message received and if it corresponds to a message that it expect and if it is correct it replay with an ACK with the relative sequence number if instead the format of this message is altered the entity will replay with a malformed message and then close the socket. If the message received instead is a malformed message the connection with the user will be immediately close. We can have different behaviour depending on the choice of the involved users:

1. User 1 wants to play with User 2 that is waiting a challenge and as well wants to play with User 1:

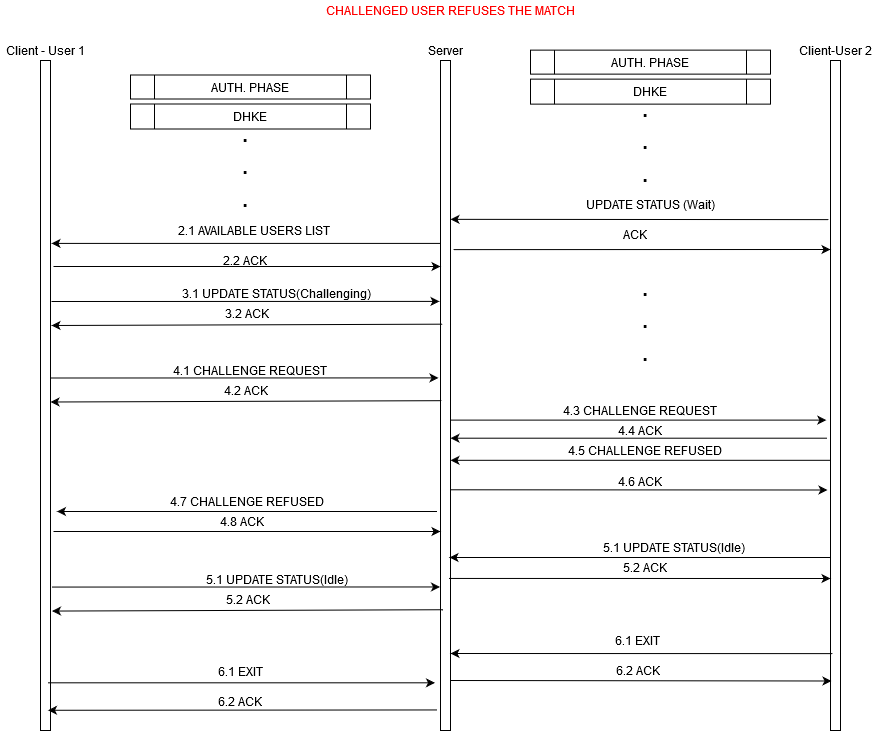
Immagine che contiene screenshot, disegnando

Descrizione generata automaticamente

1. User 1 wants to play with User 2 but meanwhile User 2 has already started a match with another user:



1. User 1 wants to play with User 2 that is waiting for a challenge, but User 2 refuse the challenge request:



1. User 1 wants to play with User 2 that is no more available:

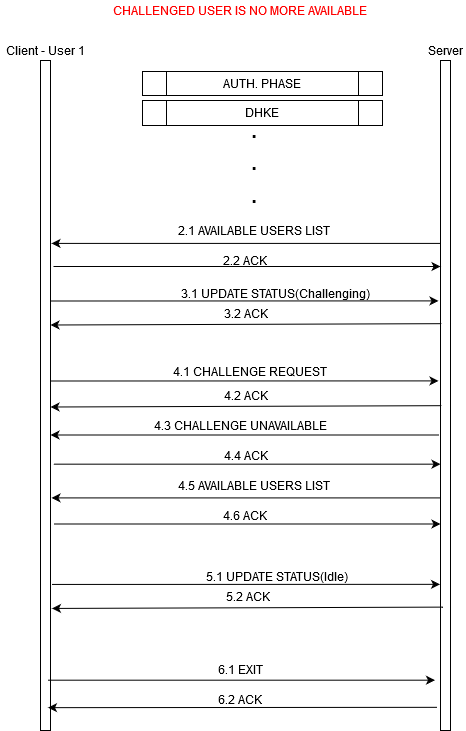


Immagine che contiene screenshot, sedendo

Descrizione generata automaticamente

Immagine che contiene screenshot, sedendo

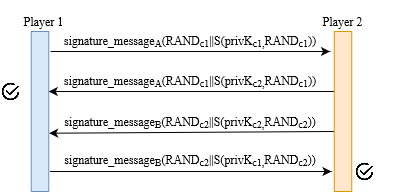
Descrizione generata automaticamente

## Peer-to-peer game:

The two players receive from the server the IP address and Public key of the adversary.

### *P2P authentication:*

The two players should authenticate each other before starting to play.



The format of the message used in this phase is the same used during the authentication phase of the client-server part. For convenience we report it below:

Immagine che contiene screenshot, sedendo

Descrizione generata automaticamente

In particular the field certificate\_len is not used and set to 0.

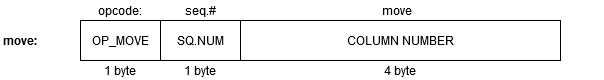
### *P2P Shared secret creation:*

For the creation of the shared secret is used the signed version of Diffie-Hellman realized for the client-server communication and described above.

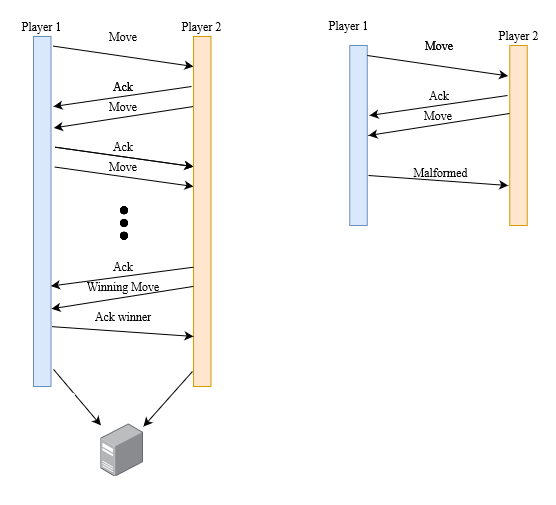
### *P2P Game:*

Now that the players have been successfully authenticated and the shared secret has been created, the two players are ready to play.

This is the move message they need to exchange:



If the move received is not allowed or there is some altered bit in the message a malformed message is sent.



# BAN logic proof of client-server authentication and key exchange

## 

## Real protocol

## Assumptions

## Objectives

## Idealized protocol

, #(RANDa)

## Proof

### M2:

Applying the 1st postulate:

If the certificate is valid and not revoked, we can assume it’s fresh and we can apply the 3nd postulate:

Now the client trusts the server’s public key.

### M3:

Applying 1st postulate:

Applying 2nd postulate:

Now the client is authenticated towards the server. In the following step we will authenticate the server towards the client.

### M5:

Applying the 1st postulate:

Applying the 2nd postulate:

Now the authentication phase is completed.

### M6:

Since the client trusts the server’s public key. Applying the 1st postulate:

Applying the 2nd postulate:

Applying the 3rd postulate:

And so A can compute the shared key Kas, proving objective 1.

We can also prove objective 5 by stating that:

### M7:

Applying 1st postulate:

Applying 2nd postulate:

Applying 3rd postulate:

And so S can compute the shared key Kas, proving objective 2.

We can also prove objective 6 by stating that:

### M8:

Applying 1st postulate:

Applying 2nd postulate:

Applying 3rd postulate we prove objective 3:

### M9:

Applying 1st postulate:

Applying 2nd postulate:

Applying 3rd postulate we prove objective 3:

# BAN logic proof of P2P authentication

## Real protocol

## Assumptions

## Objectives

## Idealized protocol

, #(RANDb)

## Proof

### M0-A:

Applying the 1st postulate:

Applying the 2nd postulate:

Applying the 3rd postulate:

### M0-B:

Applying the 1st postulate:

Applying the 2nd postulate:

Applying the 3rd postulate:

### M2:

Applying 1st postulate:

Applying 2nd postulate:

Now the client B is authenticated towards A.

### M4:

Applying the 1st postulate:

Applying the 2nd postulate:

Now both clients are authenticated.

### M5:

Since the clients trust each other public key. Applying the 1st postulate:

Applying the 2nd postulate:

Applying the 3rd postulate:

And so B can compute the shared key Kab, proving objective 1.

We can also prove objective 5 by stating that:

### M6:

Applying 1st postulate:

Applying 2nd postulate:

Applying 3rd postulate:

And so A can compute the shared key Kab, proving objective 2.

We can also prove objective 6 by stating that:

### M7:

Applying 1bt postulate:

Applying 2nd postulate:

Applying 3rd postulate we prove objective 3:

### M8:

Applying 1bt postulate:

Applying 2nd postulate:

Applying 3rd postulate we prove objective 3: