2019-2020

Four in a Row

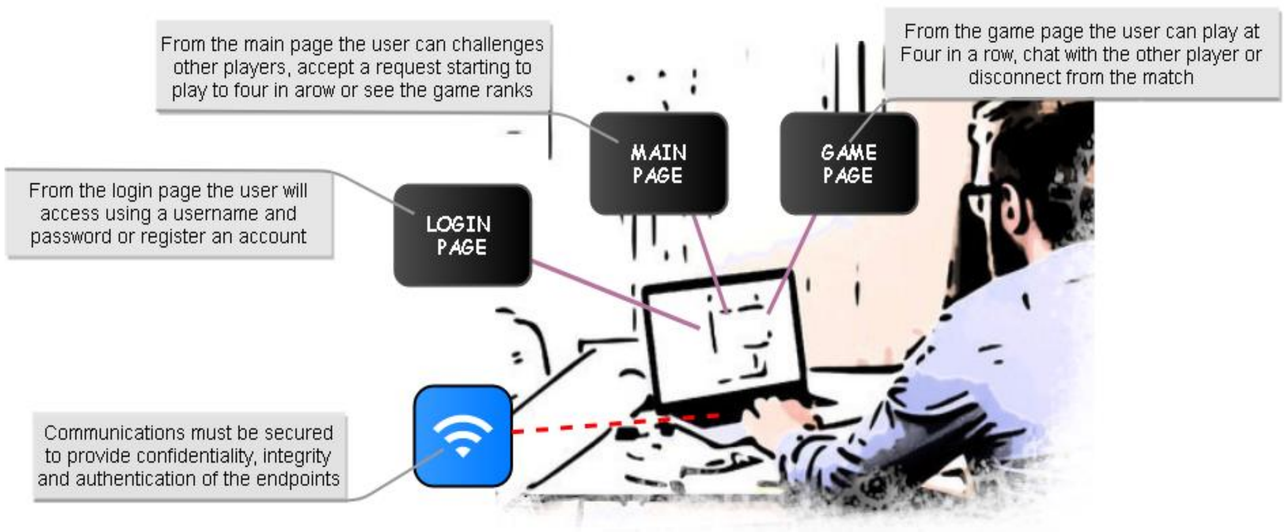
Cybersecurity Project

Nicola Barsanti Gianluca Tumminelli

Contents

User Requirement Analysis

This paper will document the development of the application **Four in a Row Online**. The application under development is a multiplayer online game accessed by a prompt interface. Each user can register a new account or directly access to the application from the *Login Page* giving a username and a password. Then, from the *Main Page,* he can see all the active players and all the users game statistics, he can choose a different player and challenge him or see all the received pending challenges and accept one. The implemented game is the classic *Four in a Row*, the game is based on a shared Gameboard in which the first player who puts four token in a row wins. The game is divided in rounds of 15 seconds and each time only one player can choose a column and put a token. During a game the players can also talk to each other using a chat or disconnect returning to the *Main Page*. The application must be confidential, authenticated and resilient to corruption and replay attack. To guarantee confidentiality it will use symmetric encryption and the exchange of the symmetric key will be done using the Diffie-Hellman Key Generation algorithm. Moreover to guarantee the authenticity of the exchanged messages and their resistance to corruption and replay attack a signature method and a nonce will be used.



User Specification Document

The following document is obtained from the formalization and analysis of the user requirements:

- The **user** *will* access the application through a prompt-like interface
- The **user** *will* access the application remotely
- The **user** *will* use a username and a password to access to the application
- The **user** *will* see a timer which indicates the remaining time to make a move
- The **user** *will* logout from the application
- The **user** *will* see all the active players
- The **user** *will* send a challenge to an active player
- The **user** *will* send only a challenge a time
- The **user** *will* withdraw a sent challenge
- The **user** *will* see all the user's statistics
- The **user** *will* have more than a request of challenge a time
- The **user** *will* see all the pending challenge requests
- The **user** *will* reject a pending challenge
- The **user** *will* reject all the pending challenges
- The **user** *will* play a four in a row game with another user
- The **user** *will* chat with the adversary during a game
- The **user** *will* see the active state of the game board
- The **user** *will* know when it is his turn to play
- The **user** *will* see the remaining time of a round
- The **user** *will* logout from a match
- The **user** *will* win a match by inserting 4 tokens in a row, a column or a diagonal
- A **match** *will* be composed by **rounds**
- A **round** *will* rough at most 15s
- Only one **user** *will* make a movement during a **round**
- A **user** *must* authenticate each other before they can play a match
- All the **users** *must* be authenticated with the server
- All the **messages** *must* authenticated by a signature
- All the **messages** of the service *must* be encrypted with a symmetric encryption
- All the **messages** *must* be able to identify corruption and prevent replay attack

System Requirement Analysis

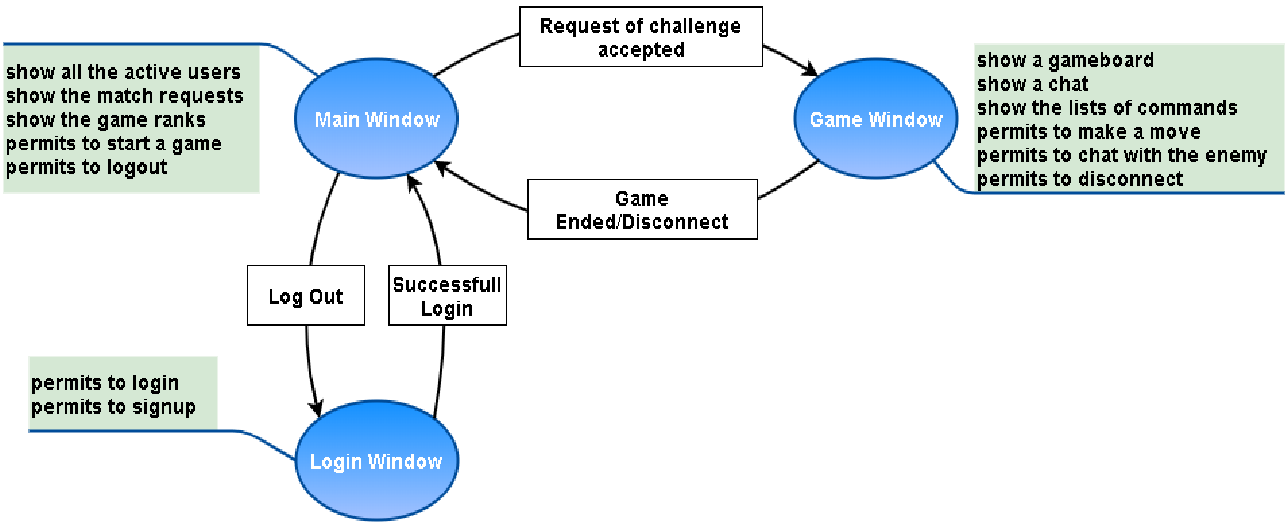
The following requirements are obtained starting from the User Specification Document.

- The **application** *will* be composed by a server and several clients which communicate remotely using an hybrid protocol
    - A **client-server** protocol *will* be adopted for the communication to the service
- A **peer-to-peer** protocol *will* be adopted for the communication between users
- A **client** *will* be able to generate a new pair of RSA keys
- A **client** *will* be able to generate new Diffie-Hellman parameters
- A **client** *will* use a symmetric encryption to communicate with the server
- Each **client** *will* use a different symmetric key to communicate with the server
- A **client** *will* use a symmetric encryption to communicate with another client during a Match
- Each **Match** *will* have different symmetric keys for the communication
- The exchange of the keys *will* be implemented using a Diffie-Hellman key generation algorithm
- Each **message** *will* have a signature to guarantee end-point authentication realized by an hash of the message signed by RSA server/user private key
- Each **message** not encrypted *will* have a nonce field to prevent replay attack
- Each **message** *will* be sanitized before being used by the application
- Each **user** *will* have a personal RSA key stored into the filesystem
- The **server** *will* have a personal RSA key stored into the filesystem and encrypted using the 'admin' password
- The **server** *will* have Diffie-Hellman key generation parameters stored into the filesystem
- Each **user** *will* have Diffie-Hellman key generation parameters stored into the filesystem
- The **server** *will* have all the user's public keys stored in a relational database
- The **server** *will* have all the client's connection information
- The **server** *will* have all the client's statistics stored into a database
- The **private RSA key** of the user *will* be stored encrypted by the user password
- The **application** *will* be composed by three windows
    - Login window:
        - **It** *will* be the first window showed
        - **It** *will* permit to login to an account
        - **Login** *will* be performed giving a correct username and password
        - The **password** *will* be used to decrypt and load the user RSA key
        - The **password** *will* be used to decrypt and load the user Diffie-Hellman parameters
        - The **username** *will* be used to identify the RSA key and parameters associated to the user
        - After a **correct login** it *will* be changed with the Main Window
        - After a **failed login** it *will* print an error message e permits a new login

    - Main window

- **It** *will* be the first window showed after a correct login
- **It** *will* be able to show all the available players
- **It** *will* be able to send a challenge to an available player
- **It** *will* be able to see the received challenges
- **It** *will* be able to accept or reject a received challenge
- **It** *will* be able to reject all the received challenges
- **It** *will* be able to show the gamer ranks
- **It** *will* be able to logout from the application

- Game window
  - **It** *will* be showed after accepting of a challenge
  - **It** *will* have a matrix 6x7 as Gameboard
  - **It** *will* generate automatically a move for the user at the timer expiration
  - **It** *will* be able to close the match and return to the Main Window
  - **It** *will* have a Chat
  - **It** *will* be able to receive and update the chat
  - **It** *will* be able to send a message from the users
  - **It** *will* be able to insert a token into a column of the Gameboard
  - **It** *will* have a timer to show the currently round duration
  - The **Gameboard** *will* be updated in real-time
  - The **Gameboard** *will* be divided in column
  - The **match** is composed by rounds
  - During a **round** only one user can play
  - The **user** which can play *will* be changed in each round
  - A **round** *will* rough at most 15s
  - The first **player** which insert four tokens consecutively in a row, column or diagonal wins
  - If the **gameboard** *will* be full with no winner the match will end with a tie
  - If a **player** disconnects from a match it *will* automatically lose



System Specification Document

The following document is obtained from the System Requirement and it will be used as a formal specification of the system behavior. It is divided into two parts to describe separately the two main components. For each component there will be its specification documents and a simple flow to describe in a high level approach its basic logic.

Client

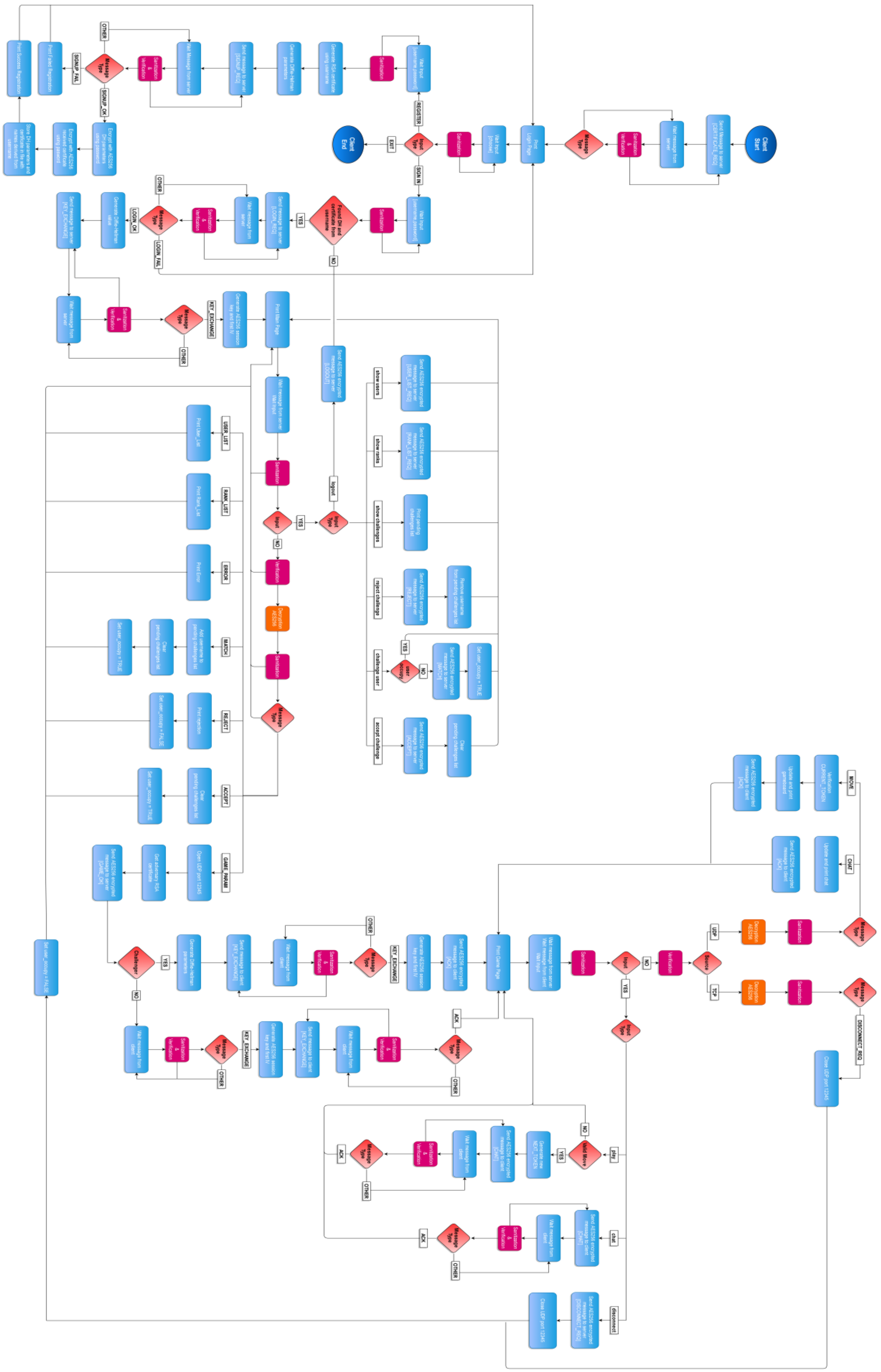
- The **client** *will* be implemented in C++ with Secure Coding
- The **client** *will* use OpenSSL library for crypto algorithms
- The **client** *will* use a TCP connection to the server on port 12345
- The **client** *will* use a UDP connection for the peer-to-peer communications on port 12345
- The **client** RSA private key *will* be encrypted with AES256 using the user login password
- The **client** *will* send ***CERTIFICATE_REQ*** messages to the server
- The **client** *will* send ***LOGIN_REQ*** messages to the server
- The **client** *will* send ***USER_LIST_REQ*** messages to the server
- The **client** *will* send ***RANK_REQ*** messages to the server
- The **client** *will* send ***MATCH*** messages to the server
- The **client** *will* send ***KEY_EXCHANGE*** messages to the server
- The **client** *will* send ***ACCEPT*** messages to the server
- The **client** *will* send ***WITHDRAW_REQ*** messages to the server
- The **client** *will* send ***REJECT*** messages to the server
- The **client** *will* send ***GAME_OK*** messages to the server
- The **client** *will* send ***DISCONNECT_REQ*** messages to the server
- The **client** *will* send ***LOGOUT_REQ*** messages to the server
- The **client** *will* send ***MOVE*** messages to clients
- The **client** *will* send ***CHAT*** messages to clients
- The **client** *will* send ***ACK*** messages to clients
- The **client** *will* receive ***GAME_PARAM*** messages from the server
- The **client** *will* receive ***LOGOUT_OK*** messages to the server
- The **client** *will* receive ***CERTIFICATE*** messages from the server
- The **client** *will* receive ***LOGIN_OK*** messages from the server
- The **client** *will* receive ***LOGIN_FAIL*** messages from the server
- The **client** *will* receive ***USER_LIST*** messages from the server
- The **client** *will* receive ***RANK_LIST*** message from the server
- The **client** *will* receive ***MATCH*** messages from the server
- The **client** *will* receive ***WITHDRAW_OK*** messages from the server
- The **client** *will* receive ***ACCEPT*** messages from the server
- The **client** *will* receive ***REJECT*** messages from the server
- The **client** *will* receive ***ERROR*** messages from the server
- The **client** *will* receive ***KEY_EXCHANGE*** messages from the server

- The **client** *will* receive ***KEY_EXCHANGE*** messages from a client
- The **client** *will* receive ***MOVE*** messages from a client
- The **client** *will* receive ***ACK*** messages from a client
- The **client** *will* be composed by three windows
  - **Login Window**
    - **It** *will* be the first window showed by the application
    - **It** *will* show a menu of all the possible actions the user can perform
    - **It** *will* require the insertion of a username and a password to perform a login
    - **It** *will* require the insertion of a username and a password to perform a registration
    - **It** *will* use the user password to crypt/decrypt the user RSA public and private key
    - **It** *will* use the username to search the user files which contains RSA and Diffie-Hellman parameters
    - **It** *will* use the password to crypt/decrypt the Diffie-Hellman user parameters
    - **It** *will* generate a pair of RSA keys during the registration of a new user
    - **It** *will* generate Diffie-Hellman parameters during the registration of a new user
    - **It** *will* store RSA keys and Diffie-Hellman parameters into a files named as the username
    - **It** *will* send a ***CERTIFICATE_REQ*** message in clear to the server
    - **It** *will* send a ***LOGIN_REQ*** message in clear to the server containing a username, a nonce and an HMAC signed by the user RSA private key
    - **It** *will* send/receive after a successful login a ***KEY_EXCHANGE*** message in clear using Diffie-Hellman to generate an AES256 symmetric key and an IV to secure the session. The message will contain the Diffie-Hellman key, a nonce and an HMAC encrypted by the user/server private RSA key
    - **It** *will* receive a ***CERTIFICATE*** message in clear from the server containing a certificate
    - **It** *will* receive ***LOGIN_OK*** message in clear from the server after a successful login. The message will contain a nonce and an HMAC encrypted by the server RSA private key
    - **It** *will* receive ***LOGIN_FAIL*** message in clear from the server after a failed login. The message will contain a nonce and an HMAC signed by the server RSA private key
  - **Main Window**
    - **It** *will* be the first window showed after a successful login
    - **It** *will* show a menu of all the possible actions a user can perform
    - **It** *will* show all the active users
      - An **user** will be represented by the tuple:
        (username, Total Played Games, Total Won Games)
    - **It** *will* show all the challenge pending requests
    - **It** *will* show the rank table
      - A **rank** *will* be represented by the tuple:
        (*username, Total Match, Won Match, Lost Match, Tied Match)*
    - **It** *will* be able to logout from the application and return to the login window
    - **It** *will* be able to accept a challenge
    - **It** *will* be able to discard a challenge

        - **It** *will* be able to discard all the received challenges
        - **It** *will* send a ***USER_LIST_REQ*** message encrypted with AES256 to the server containing a nonce and an HMAC
        - **It** *will* send a ***RANK_REQ*** message encrypted with AES256 to the server containing a nonce and an HMAC
        - **It** *will* send a ***MATCH*** message encrypted with AES256 to the server containing the username of the adversary and an HMAC
        - **It** *will* send an ***ACCEPT*** message encrypted with AES256 to the server containing the username of the adversary and an HMAC
        - **It** *will* send an ***WITHDRAW_REQ*** message encrypted with AES256 to the server containing a nonce and an HMAC
        - **It** *will* send a ***REJECT*** message encrypted with AES256 to the server containing the username of the adversary and an HMAC
        - **It** *will* send a ***GAME_OK*** message encrypted with AES256 to the server containing a nonce and an HMAC
        - **It** *will* send a ***LOGOUT_REQ*** message encrypted with AES256 to the server containing a nonce and an HMAC
        - **It** *will* receive a ***GAME_PARAM*** message encrypted with AES256 to the server containing a user's public key, an username an IP address and an HMAC
        - **It** *will* receive a ***USER_LIST*** message encrypted with AES256 from the server containing the user list and an HMAC
        - **It** *will* receive a ***RANK_LIST*** message encrypted with AES256 from the server containing the rank list and an HMAC
        - **It** *will* receive an ***ACCEPT*** message encrypted with AES256 from the server containing a username and an HMAC
        - **It** *will* receive a ***REJECT*** message encrypted with AES256 from the server containing a username and an HMAC
        - **It** *will* receive a ***WITHDRAW_OK*** message encrypted with AES256 from the server containing a nonce and an HMAC
        - **It** *will* receive an ***ERROR*** message encrypted with AES256 from the server containing the type of error generated, a nonce and an HMAC. The message will be received if some action performed with the server are invalid
        - **It** *will* receive a ***LOGOUT_OK*** message encrypted with AES256 from the server containing a nonce and an HMAC
    - **Game Window**
        - **It** *will* be showed after the accepting/acceptance of a challenge
        - **It** *will* show a menu of all the possible commands the user can perform
        - **It** *will* show the player/adversary name and his total played games and percentage of wins
        - **It** *will* be able to close the match and return to the Main Window
            - The **user** *will* automatically lose
        - **It** *will* show a 6X7 char matrix
            - A **token** *will* always be inserted into the lowest available column position
            - The **first player** which insert four tokens consecutively in a row, column or diagonal wins

    - **It** *will* be able to automatically close a match with a tie when the matrix is full and there aren’t winners
    - **It** *will* have only two possible values
        - ‘ ’ to represents an available position
        - ‘O’ to represents a token
        - The ‘O’ used by the adversary will be red colored
        - The ‘O’ used by the player will be blue colored
- **It** *will* show a timer set to 15s at the beginning of each round
    - At the timer expiration the application will choose automatically a column and insert a token
- **It** *will* show a chat
- **It** *will* be able to control if the user is in charge to make the next move using a token mechanism
    - Only a client with the **CURRENT_TOKEN** can make the move
    - The **CURRENT_TOKEN** is generated by a counter
    - The client which receive a move controls if the **CURRENT_TOKEN** has the correct value
- **It** *will* send a ***CHAT*** message encrypted with AES256 and containing the text, a nonce, an acknowledgement and an HMAC to send a message to the other player
- **It** *will* send/receive a ***KEY_EXCHANGE*** message in clear using Diffie-Hellman to generate an AES256 symmetric key and an IV to secure the game session. The message will contain the Diffie-Hellman key, an acknowledgement and an HMAC signed by the user private RSA key
- **It** *will* send a ***MOVE*** message encrypted with AES256 and containing a column field, a **CURRENT_TOKEN**, an acknowledgement and an HMAC
- **It** *will* send a ***DISCONNECT_REQ*** message encrypted with AES256 and containing a nonce and an HMAC from the server when it wants to leave the current game or inform that the current game is completed
- **It** *will* receive as first communication from an UDP session a ***KEY_EXCHANGE*** message in clear using Diffie-Hellman to generate an AES256 symmetric key to secure their connection. The messages will contain an acknowledgement field and an HMAC signed by the user private RSA key
- **It** will reply to each message exchanged into the UDP connection with an ***ACK*** message in clear containing an acknowledgement, an HMAC field
- **It** *will* receive an ***ACK*** message after each communication using the UDP connection,
- **It** *will* resend a message after 10s without receiving an ACK message
- **It** *will* resend an ***ACK*** message after 5s without receiving new messages
- **It** *will* receive a ***DISCONNECT_REQ*** message encrypted with AES256 and containing a nonce and an HMAC field from the server to inform it that the game is ended
- **It** *will* receive a ***CHAT*** message encrypted with AES256 and containing the message, a nonce, an acknowledgement and an HMAC field signed with the user RSA private key from another client

Flow Diagram of Client Application

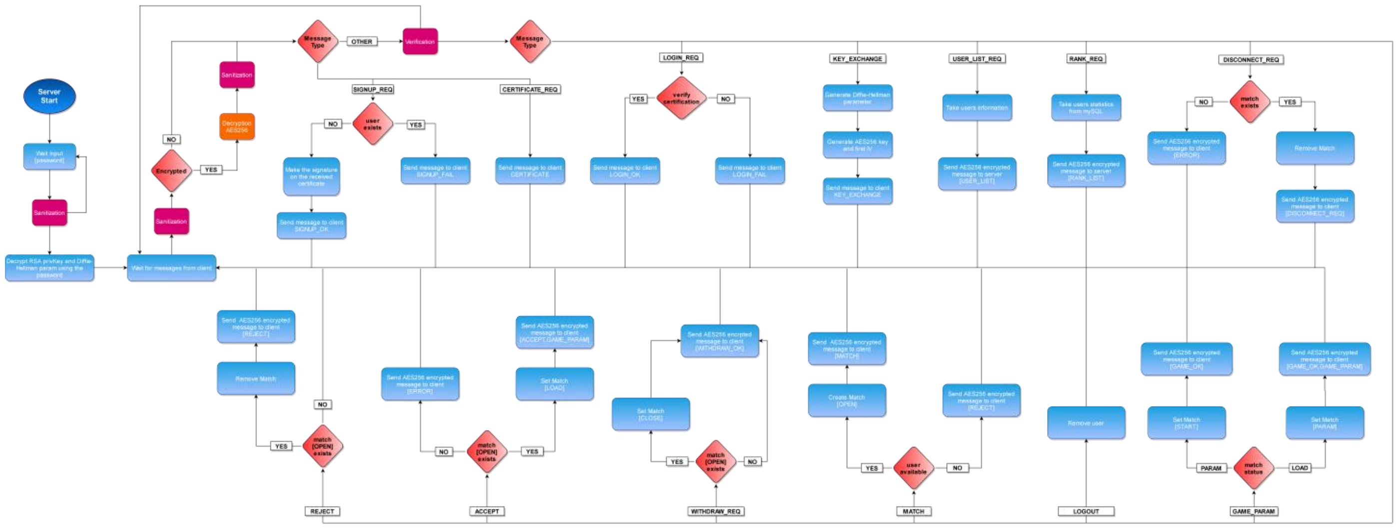


Server

- The **server** *will* be implemented in C++ with Secure Coding
- The **server** *will* use OpenSSL library for crypto algorithms
- The **server** *will* use a MySQL database to store information about registered accounts and their public RSA keys
- The **server** *will* use MySQL database to store the user's statistics
- The **server** *will* have an RSA certificate
- The **server** RSA certificate *will* be signed by a certification authority
- The **server** *will* wait for new connections on TCP port 12345
- The **server** *will* record all the connected clients into the Client Register
    - Each **client** *will* be defined as:
      (client ID, IP address, port, communication socket)
    - Each **client** *will* be recorded during the first connection
    - Each **client** *will* be removed after the closing of its connection
- The **server** *will* record all the logged users into the application in the User Register
    - Each **user** *will* be defined as:
      (client ID, username, public RSA key, AES256key)
    - Each **user** *will* be recorded after the login
    - Each **user** *will* be removed after the logout or the closing of its connection
- The **server** *will* record all the signed users ranks into the MySQL Database
    - Each **rank** *will* be defined as
      (username, public RSA, Total Played Games, Total Won, Total Lose, Total Tie)
    - Each **rank** *will* be updated after a match
- The **server** will record all the matches in progress into the MatchRegister
    - Each **match** *will* be defined as
      (username(challenger), username(challenged), status)
    - A **match** status can be OPEN, ACCEPT, LOAD, START, REJECT
    - Each **match** *will* be recorded after the accepting of a challenge
    - Each **match** *will* be removed after the receiving of a ***DISCONNECT_REQ*** message
    - Each **match** will be added after a valid MATCH request with status OPEN
    - Each **match** that belong to the request *will* be removed after the receiving of a ***REJECT***/***DISCONNECT_REQ***/***WITHDRAW*** message
    - Each **match** that belong to the request *will* be updated to status ACCEPT after the receipt of an ***ACCEPT*** message otherwise to status REJECT
    - Each **match** that belong to the request and which has an ACCEPT status *will* be update to status LOAD after the receiving of a ***GAME_OK*** message
    - Each **match** that belong to the request and which has a LOAD status *will* be update to status START after the receiving of a ***GAME_OK*** message
- The **server** *will* send a ***LOGIN_OK*** message in clear containing a nonce and an HMAC signed with the server private RSA key after a successful login

- The **server** *will* send a ***LOGIN_FAIL*** message in clear containing a nonce and an HMAC signed with the server private RSA key after a incorrect login
- The **server** *will* send a ***CERTIFICATE*** message in clear composed by a certificate, a nonce and an HMAC signed with the server RSA private key
- The **server** *will* send a ***USER_LIST*** message encrypted with AES256 and containing a list of all the active users and an HMAC signed with the server RSA private key
- The **server** *will* send a ***RANK_UPDATE*** message encrypted with AES256 and containing the user ranking list and an HMAC signed with the server RSA private key
- The **server** *will* send a ***DISCONNECT_REQ*** message encrypted with AES256 and containing a username, a type, a nonce and an HMAC signed with the server RSA private key
- The **server** *will* send a ***REJECT*** message encrypted with AES256 and containing a username and an HMAC signed with the server RSA private key
- The **server** *will* send an ***ACCEPT*** message encrypted with AES256 and containing a username and an HMAC signed with the server RSA private key
- The **server** *will* send a ***GAME_PARAM*** message encrypted with AES256 and containing a public key, a username and an HMAC signed with the server RSA private key
- The **server** *will* receive a ***CERTIFICATE_REQ*** message in clear
- The **server** *will* receive a ***LOGIN_REQ*** message in clear containing a username, a nonce and a HMAC signed with the user private RSA key
- The **server** *will* receive a ***USER_LIST_REQ*** message encrypted with AES256 and containing a nonce and an HMAC signed with the user RSA private key
- The **server** *will* receive a ***RANK_REQ*** message encrypted with AES256 and containing a nonce and an HMAC signed with the user RSA private key
- The **server** *will* receive a ***MATCH*** message encrypted with AES256 and containing a username, a nonce and an HMAC signed with the user RSA private key
- The **server** *will* receive an ***ACCEPT*** message encrypted with AES256 containing the username and an HMAC signed by a user RSA private key
- The **server** *will* receive a ***REJECT*** message encrypted with AES256 and containing a username and an HMAC signed by a user RSA private key
- The **server** *will* receive a ***GAME_OK*** message encrypted with AES256 and containing a nonce and an HMAC signed with a user RSA private key

Flow Diagram of Server Application



Mockup

Login Page

On this page the user can login to his account by using the command Login, a username and password.

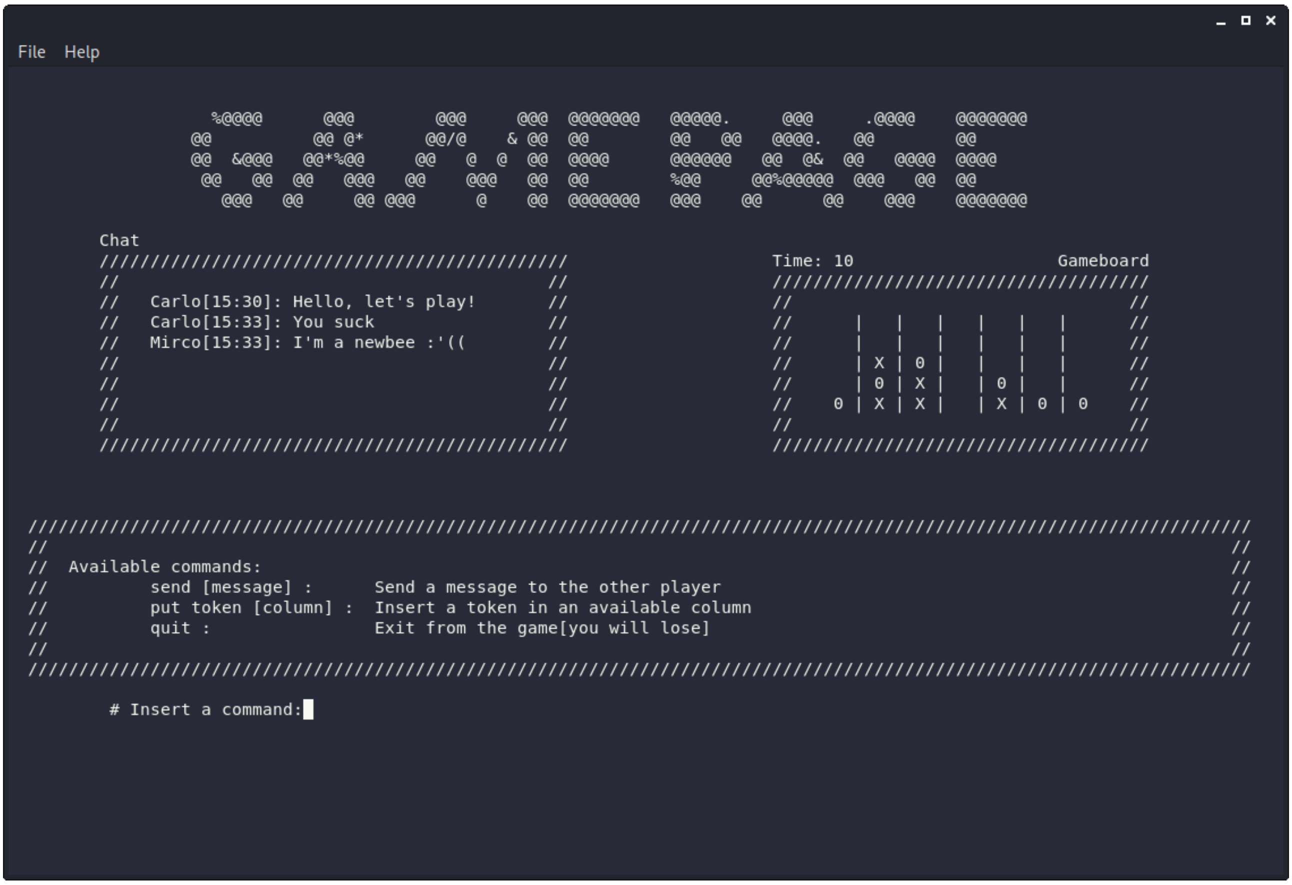
Main Page

In this page the user can execute the following commands:

- show
  - show[user]: show all the available users
  - show [rank]: show all the user's game statistics
- challenge [user]: invite a user to play a four in a row match
- logout: the user quit from the application and return to the login page

Game Page

In this page users after accepting a game, they can perform a move with the “**put token**” command or they can chat with an opponent with the command “**send**” and they can withdraw from a game with the “quit” command.



Protocol Analysis

In the following section there will be described in detail the exchange of the application messages and their structure. We have designed four extra postulates to manage particular situation not covered by the base postulates. During all the analysis with the symbol H we mean an HMAC of all or partial of the message fields which we consider equivalent to the encryption of all the included fields.

Certificate Postulate

We need a postulate to link the receive of a server authorized certificate to the obtaining of a valid user public key.

$$\frac{\stackrel{K_q}{\longmapsto} S, \{H_{\stackrel{K_q}{\longmapsto} Q}\}_{K_{ca}^{-1}}, \#(N), \{H_{all}\}_{K_q^{-1}}}{P \mid\equiv \stackrel{K_q}{\longmapsto} Q, P \mid\equiv Q \mid\equiv \stackrel{K_q}{\longmapsto} Q}$$

Signature Postulate

To simplify the BAN Analysis we have made a simple postulate to link the presence of a signature of all the fields of the message to their trustability.

$$\frac{P \mid\equiv \stackrel{k}{\mapsto} Q, P \triangleleft X, \{H_{all}\}_k}{P \mid\equiv Q \mid\sim X}$$

$$\frac{P \mid\equiv P \stackrel{k}{\leftrightarrow} Q, P \triangleleft X, \{H_{all}\}_K}{P \mid\equiv Q \mid\sim X}$$

Diffie-Hellman Postulates

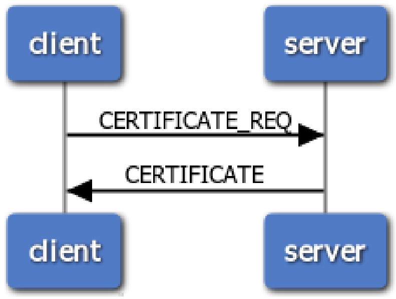
We need a postulate to link the possession of two Diffie-Hellman parameters to the generation of a shared session key. To simplify the analysis we consider the message and the key the two components to be shared by the two parts independently from what they really are(key,messages)

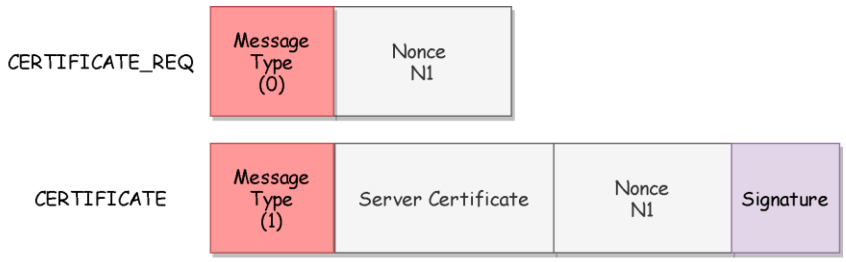
$$\frac{P \mid\equiv D_1, P \mid\equiv D_2}{P \mid\equiv A \stackrel{K_{AB}}{\longleftrightarrow} B}$$

We need a postulate to link the freshness of the Diffie-Hellman parameters to the freshness of the shared session key

$$\frac{P \mid\equiv \#(D_1), P \mid\equiv \#(D_2)}{P \mid\equiv \#(P \stackrel{K_{pq}}{\longleftrightarrow} Q)}$$

The protocol is used during the client initialization to obtain the server certificate. The certificate requires to be authenticated from the server so for the reply message we will use a signature which will be based on a fresh nonce sent by the client.





BAN Logic Analysis

Real Protocol

$$M_1 \quad C \rightarrow S : \; M, N_1$$

$$M_2 \quad S \rightarrow C : \; M, (C_s, \{H_{C_s}\}_{K_{ca}^{-1}}), N_1, \{H_{all}\}_{K_s^{-1}}$$

Ideal Protocol

$$M_2 \quad S \rightarrow C : \xmapsto{K_s} S, \{H_{\xmapsto{K_s} S}\}_{K_{ca}^{-1}}, \#(N_1), \{H_{all}\}_{K_s^{-1}}$$

Goals

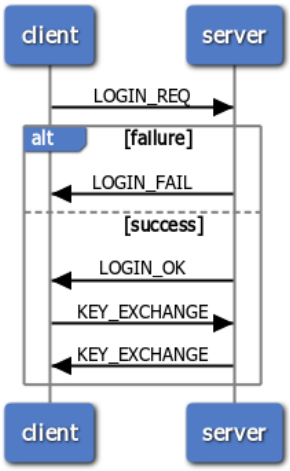
$$C \mid\equiv \xmapsto{K_s} S$$

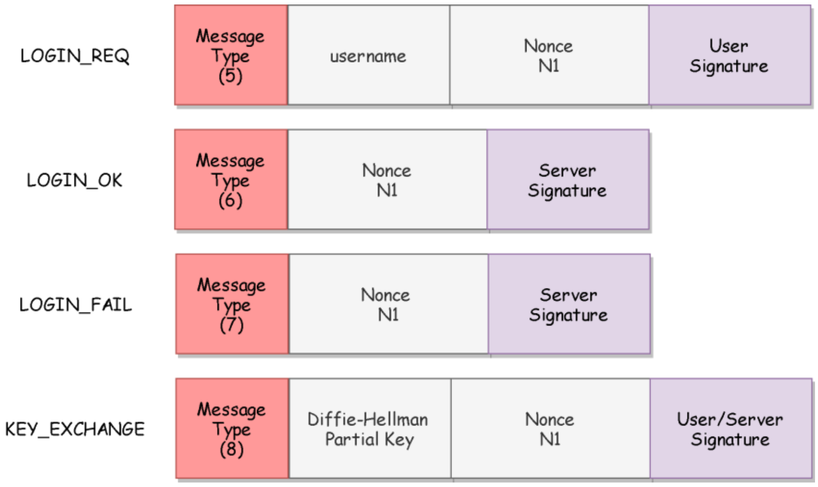
$$C \mid\equiv S \mid\equiv \xmapsto{K_s} S$$

Analysis

M2	$\dfrac{C \triangleleft (\xmapsto{K_S} S, \{H_{\xmapsto{K_s} S}\}_{K_s^{-1}}), \#N_1, \{H_{all}\}_{K_s^{-1}}}{\boxed{C \mid\equiv \xmapsto{K_s} S,} \boxed{C \mid\equiv S \mid\equiv \xmapsto{K_s} S}}$	The client has received the server certificate which is validate by the CA. Moreover the message is fresh due to the nonce field and it contains a signature made by the server RSA private key. We can apply the **certificate postulate** to derive that the certificate belongs to the server

The protocol will be used to access the application. The client has to give a proof of its authenticity by sending a signature made by its private RSA key. Then it will create a session key to communicate with the server by the Diffie-Hellman key-generation algorithm. All the messages don't need confidentiality, only the end-point authenticity and the impossibility of reply attack must be granted.





BAN Logic Analysis

Real Protocol

$M_1 \quad C \to S: \ M, U, N_1, \{H_{all}\}_{K_c^{-1}}$

$M_2 \quad S \to C: \ M, N_1, \{H_{all}\}_{K_s^{-1}}$

$M_3 \quad S \to C: \ M, D_1, N_1, \{H_{all}\}_{K_s^{-1}}$

$M_4 \quad C \to S: \ M, D_2, N_1, \{H_{all}\}_{K_c^{-1}}$

Ideal Protocol

$M_1 \quad C \to S: \ M, \#(N_1), \{H_{all}\}_{K_c^{-1}}$

$M_2 \quad S \to C: \ M, \#(N_1), \{H_{all}\}_{K_s^{-1}}$

$M_3 \quad S \to C: \ M, \#(D_1, N_1), \{H_{all}\}_{K_s^{-1}}$

$M_4 \quad C \to S: \ M, \#(D_2, N_1), \{H_{all}\}_{K_c^{-1}}$

Goals

$C \mid\equiv C \xleftrightarrow{K_{cs}} S \qquad C \mid\equiv S \mid\equiv C \xleftrightarrow{K_{cs}} S$

$S \mid\equiv C \xleftrightarrow{K_{cs}} B \qquad S \mid\equiv C \mid\equiv C \xleftrightarrow{K_{cs}} S$

$S \mid\equiv C \mid\equiv U$

$C \mid\equiv S \mid\equiv N_1$

Assumptions

$C \mid\equiv \xmapsto{K_s} S \qquad C \mid\equiv S \mid\equiv \xmapsto{K_s} S$

$S \mid\equiv \xmapsto{K_c} C \qquad S \mid\equiv C \mid\equiv \xmapsto{K_c} C$

Analysis

M1

$$\frac{S \mid\equiv \xmapsto{K_c} C, \ S \triangleleft \{H_{all}\}_{K_c^{-1}}}{S \mid\equiv C \mid\sim (U, \#(N_1))}$$

The server has received a message containing a signature made by all the fields with the client private RSA key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(N_1), S \mid\equiv C \mid\sim (U, N_1)}{\boxed{S \mid\equiv C \mid\equiv U}}$$

M2

$$\frac{C \mid\equiv \overset{K_s}{\longmapsto} S, C \triangleleft \{H_{all}\}_{K_s^{-1}}}{C \mid\equiv S \mid\sim (\#(N_1))}$$

The client has received a message containing a signature made by all the fields with the client private RSA key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C \mid\equiv \#(N_1), C \mid\equiv S \mid\sim N_1}{\boxed{C \mid\equiv S \mid\equiv N_1}}$$

M3

$$\frac{S \mid\equiv \overset{K_c}{\longmapsto} C, S \triangleleft \{H_{all}\}_{K_c^{-1}}}{S \mid\equiv C \mid\sim \#(D_1, N_1)}$$

The server has received a message containing a signature made by all the fields with the client private RSA key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(D_1, N_1), S \mid\equiv C \mid\sim (D_1, N_1)}{\boxed{S \mid\equiv C \mid\equiv D_1}}$$

M4

$$\frac{C \mid\equiv \overset{K_s}{\longmapsto} S, C \triangleleft \{H_{all}\}_{K_s^{-1}}}{C \mid\equiv S \mid\sim \#(D_2, N_1)}$$

The client has received an HMAC encrypted by the server certificate. We can apply the **second message meaning postulate** to derive that it believes the message is sent by the server

The received HMAC contains a fresh timestamp, so it is fresh and we can apply the **nonce verification postulate** to derivate that the client believes that only the server could have sent the message

$$\frac{C \mid\equiv \#(D_2, N_1), C \mid\equiv S \mid\sim (D_2, N_1)}{\boxed{C \mid\equiv S \mid\equiv D_2}}$$

$$\frac{C \mid\equiv D_1, C \mid\equiv S \mid\equiv D_2}{\boxed{C \mid\equiv C \xleftrightarrow{K_{cs}} S}}$$

We have the two Diffie-hellman components, we can use the first **Diffie-Hellman postulate** to derive that the client has generate the shared session key

We have almost one fresh Diffie-Hellman partial key, we can use the *second* **Diffie-Hellman postulate** to derive that the shared key is unique and believing to that session

$$\frac{C \mid\equiv \#(D_1), C \mid\equiv S \mid\equiv D_2}{C \mid\equiv S \mid\equiv (C \xleftrightarrow{K_{cs}} S)}$$

$$\frac{S \mid\equiv D_2, S \mid\equiv C \mid\equiv D_1}{\boxed{S \mid\equiv C \xleftrightarrow{K_{cs}} S}}$$

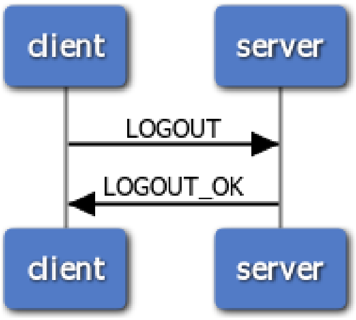
We have the two Diffie-hellman components, we can use the first **Diffie-Hellman postulate** to derive that the client has generate the shared session key

We have almost one fresh Diffie-Hellman partial key, we can use the *second* **Diffie-Hellman postulate** to derive that the shared key is unique and believing to that session

$$\frac{S \mid\equiv \#(D_2), S \mid\equiv C \mid\equiv D_1}{S \mid\equiv C \mid\equiv (C \xleftrightarrow{K_{cs}} S)}$$

Logout Protocol

The protocol will be used to quit from the application. The messages requires only authenticity and protection to reply attack so they have a signature made by AES256 GCM based on a fresh nonce.



LOGOUT	Message Type (10)	Nonce N1	AES Signature
LOGOUT_OK	Message Type (11)	Nonce N1	AES Signature

BAN Logic Analysis

Real Protocol

$$M_1 \quad C \rightarrow S: \; M, N_1, \{H_{all}\}_{Kcs}$$
$$M_2 \quad S \rightarrow C: \; M, N_1, \{H_{all}\}_{Kcs}$$

Ideal Protocol

$$M_1 \quad C \rightarrow S: \; \#(N_1), \{H_{all}\}_{Kcs}$$
$$M_2 \quad S \rightarrow C: \; \#(N_1), \{H_{all}\}_{Kcs}$$

Goals

$$S \mid\equiv C \mid\equiv N_1$$
$$C \mid\equiv S \mid\equiv N_1$$

Assumptions

$$C \mid\equiv C \xleftrightarrow{K_{cs}} S$$
$$S \mid\equiv C \xleftrightarrow{K_{cs}} S$$

Analysis

M1

$$\frac{S \mid\equiv C \xleftrightarrow{K_{cs}} S, S \triangleleft \{H_{all}\}_{K_{cs}}}{S \mid\equiv C \mid\sim N_1}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(N_1), S \mid\equiv C \mid\sim \{N_1\}}{\boxed{S \mid\equiv C \mid\equiv N_1}}$$

M2

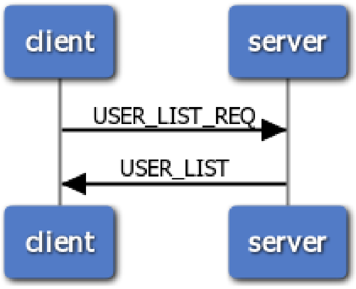
$$\frac{C \mid\equiv C \xleftrightarrow{K_{cs}} S, C \triangleleft \{H_{all}\}_{K_{cs}}}{C \mid\equiv S \mid\sim N_1}$$

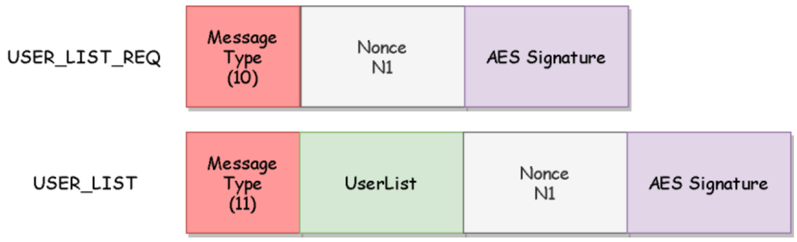
The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C \mid\equiv \#(N_1), C \mid\equiv S \mid\sim N_1}{\boxed{C \mid\equiv S \mid\equiv N_1}}$$

The protocol will be used from the clients to obtain a list of the users currently available to be challenged. The user list requires to be confidential and it will be encrypted. All the other fields requires only authentication and will be protected by a signature based on a fresh nonce and made by AES256 GCM.





BAN Logic Analysis

Real Protocol

$$M_1 \quad C \rightarrow S: \; M, N_1, \{H_{all}\}_{K_{cs}}$$

$$M_2 \quad S \rightarrow C: \; M, N_1, \{L, H_{all}\}_{K_{cs}}$$

Ideal Protocol

$$M_1 \quad C \rightarrow S: \; N_1, \{H_{all}\}_{K_{cs}}$$

$$M_2 \quad S \rightarrow C: \; N_1, \{L, H_{all}\}_{K_{cs}}$$

Goals

$$C \mid\equiv S \mid\equiv L$$

$$S \mid\equiv C \mid\equiv N_1$$

Assumptions

$$C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S$$

$$S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S$$

Analysis

M1

$$\frac{S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S, S \triangleleft \{N_1, H_{all}\}_{K_{cs}}}{S \mid\equiv C \mid\sim \{N_1, H_{all}\}}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S), S \mid\equiv C \mid\sim N_1}{\boxed{S \mid\equiv C \mid\equiv N_1}}$$

M2

$$\frac{C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S, C \triangleleft \{H_{all}\}_{K_{cs}}}{C \mid\equiv S \mid\sim (N_1, L)}$$

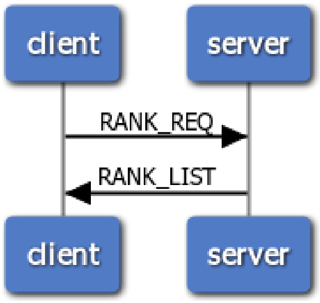
The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C \mid\equiv \#(N_1), C \mid\equiv S \mid\sim (N_1, L)}{\boxed{C \mid\equiv S \mid\equiv L}}$$

Rank List Protocol

The protocol will be used from the clients to obtain a list of the users game statistics. The rank list requires to be confidential and it will be encrypted. All the other fields requires only authentication and will be protected by a signature based on a fresh nonce and made by AES256 GCM.



RANK_LIST_REQ

Message Type (12)	Nonce N1	AES Signature

RANK_LIST

Message Type (13)	RankList	Nonce N1	AES Signature

BAN Logic Analysis

Real Protocol

$M_1 \quad C \to S: \; M, N_1, \{H_{all}\}_{K_{cs}}$

$M_2 \quad S \to C: \; M, N_1, \{L, H_{all}\}_{K_{cs}}$

Ideal Protocol

$M_1 \quad C \to S: \; N_1, \{H_{all}\}_{K_{cs}}$

$M_2 \quad S \to C: \; N_1, \{L, H_{all}\}_{K_{cs}}$

Goals

$C \mid\equiv S \mid\equiv L$

$S \mid\equiv C \mid\equiv N_1$

Assumptions

$C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S$

$S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S$

Analysis

M1

$$\frac{S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S, S \triangleleft \{N_1, H_{all}\}_{K_{cs}}}{S \mid\equiv C \mid\sim \{N_1, H_{all}\}}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S), S \mid\equiv C \mid\sim N_1}{\boxed{S \mid\equiv C \mid\equiv N_1}}$$

M2

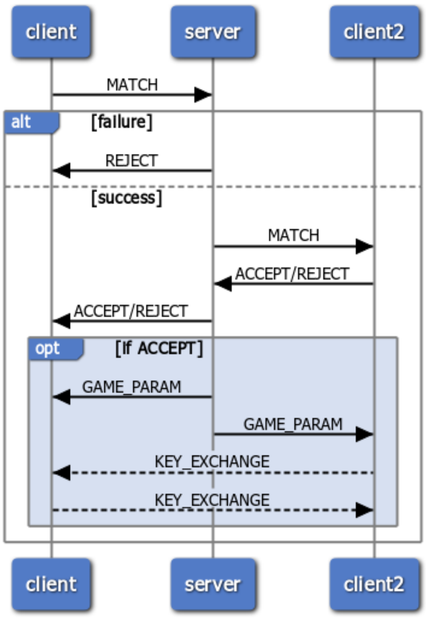
$$\frac{C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S, C \triangleleft \{H_{all}\}_{K_{cs}}}{C \mid\equiv S \mid\sim (N_1, L)}$$

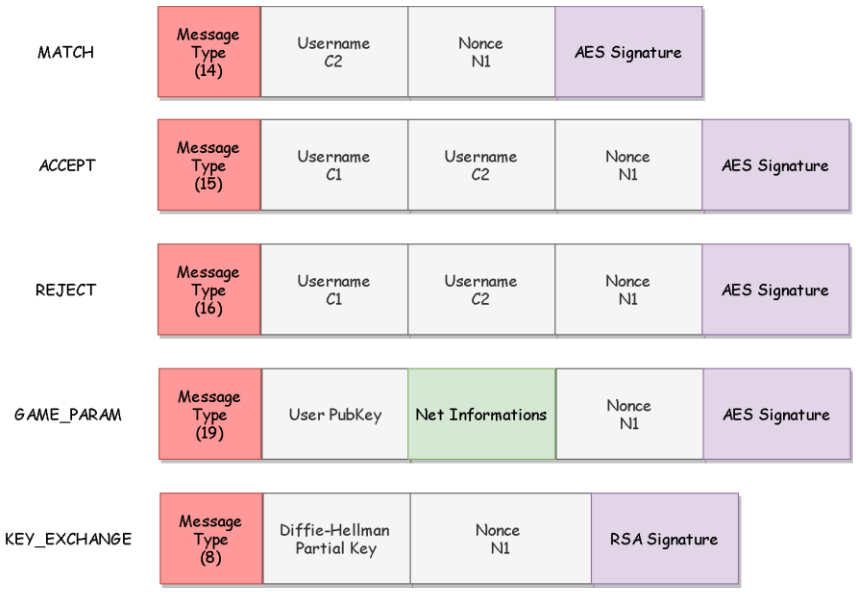
The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C \mid\equiv \#(N_1), C \mid\equiv S \mid\sim (N_1, L)}{\boxed{C \mid\equiv S \mid\equiv L}}$$

The protocol will be used from the clients to request to another player to join a game. The messages requires authenticity so a signature based on a fresh nonce and made by AES256 GCM is applied on each message. The only fields that require confidentiality are the net information of the users and so they will be encrypted.





BAN Logic Analysis

Real Protocol

$$
\begin{array}{ll}
M_1 & C_1 \rightarrow S: \ M, C_2, N_1, \{H_{all}\}_{K_{sc_1}} \\
M_2 & S \rightarrow C_2: \ M, C_1, N_1, \{H_{all}\}_{K_{sc_2}} \\
M_3 & C_2 \rightarrow S: \ M, C_1, C_2, N_1, \{H_{all}\}_{K_{sc_2}} \\
M_4 & S \rightarrow C_1: \ M, C_1, C, 2, N_1, \{H_{all}\}_{K_{sc_1}} \\
M_5 & S \rightarrow C_1: \ M, K_{C_2}, N_1, \{I_{C_2}, H_{all}\}_{K_{sc_1}} \\
M_6 & S \rightarrow C_2: \ M, K_{C_1}, N_1, \{I_{C_1}, H_{all}\}_{K_{sc_1}} \\
M_7 & C_2 \rightarrow C_1: \ M, D_1, N_1, \{H_{all}\}_{K_{c_2}^{-1}} \\
M_8 & C_1 \rightarrow C_2: \ M, D_2, N_1, \{H_{all}\}_{K_{c_1}^{-1}}
\end{array}
$$

Goals

$$
\begin{array}{lll}
S \mid\equiv C_1 \mid\equiv' C_2' & S \mid\equiv C_2 \mid\equiv' C_1' & \\
C_1 \mid\equiv S \mid\equiv' C_2' & C_1 \mid\equiv S \mid\equiv \overset{c_2}{\longmapsto} C_2, I_{C_2} & C_1 \mid\equiv C_2 \mid\equiv C_1 \overset{c_1 c_2}{\longleftrightarrow} C_2 \\
C_2 \mid\equiv S \mid\equiv' C_1' & C_2 \mid\equiv S \mid\equiv \overset{c_1}{\longmapsto} C_1, I_{C_1} & C_2 \mid\equiv C_1 \mid\equiv C_1 \overset{c_1 c_2}{\longleftrightarrow} C_2
\end{array}
$$

Ideal Protocol

$$
\begin{array}{ll}
M_1 & C_1 \rightarrow S: \ \#(N_1), \{H_{all}\}_{K_{sc_1}} \\
M_2 & S \rightarrow C_2: \ \#(N_1), \{H_{all}\}_{K_{sc_2}} \\
M_3 & C_2 \rightarrow S: \ \#(N_1), \{H_{all}\}_{K_{sc_2}} \\
M_4 & S \rightarrow C_1: \ \#(N_1), \{H_{all}\}_{K_{sc_1}} \\
M_5 & S \rightarrow C_1: \overset{C_2}{\longmapsto} C_2, \#(N_1), \{I_{C_2}, H_{all}\}_{K_{sc_1}} \\
M_6 & S \rightarrow C_2: \overset{C_1}{\longmapsto} C_1, \#(N_1), \{I_{C_1}, H_{all}\}_{K_{sc_1}} \\
M_7 & C_2 \rightarrow C_1: \ \#(D_1, N_1), \{H_{all}\}_{K_{c_2}^{-1}} \\
M_8 & C_1 \rightarrow C_2: \ \#(D_2, N_1), \{H_{all}\}_{K_{c_1}^{-1}}
\end{array}
$$

Assumptions

$$
\begin{array}{ll}
C_1 \mid\equiv C_1 \overset{K_{cs_1}}{\longleftrightarrow} S & C_2 \mid\equiv C_2 \overset{K_{sc_2}}{\longleftrightarrow} S \\
S \mid\equiv C_1 \overset{K_{sc_1}}{\longleftrightarrow} S & S \mid\equiv C_2 \overset{K_{sc_2}}{\longleftrightarrow} S
\end{array}
$$

Analysis

M1

$$\frac{S \mid\equiv C_1 \stackrel{K_{sc_1}}{\longleftrightarrow} S, S \triangleleft N_1\{H_{all}\}_{K_{sc_1}}}{S \mid\equiv C_1 \mid\sim N_1}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(N_1), S \mid\equiv C_1 \mid\sim (N_1, C_2)}{\boxed{S \mid\equiv C_1 \mid\equiv C_2}}$$

M2

$$\frac{C_2 \mid\equiv C_2 \stackrel{K_{sc_2}}{\longleftrightarrow} S, C_2 \triangleleft N_1, \{H_{all}\}_{K_{sc_2}}}{C_2 \mid\equiv S \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C_2 \mid\equiv \#(N_1), C_2 \mid\equiv S \mid\sim (N_1, C_1)}{\boxed{C_2 \mid\equiv S \mid\equiv C_1}}$$

M3

$$\frac{S \mid\equiv C_2 \stackrel{K_{sc_2}}{\longleftrightarrow} S, C_2 \triangleleft N_1, \{H_{all}\}_{K_{sc_2}}}{S \mid\equiv C_2 \mid\sim C_1, C_2, N_1}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(N_1), S \mid\equiv C_2 \mid\sim C_1, C_2, N_1}{\boxed{S \mid\equiv C_2 \mid\equiv C_1, C_2}}$$

M4

$$\frac{C_1 \mid\equiv C_1 \stackrel{K_{sc_1}}{\longleftrightarrow} S, C_1 \triangleleft N_1, \{H_{all}\}_{K_{sc_1}}}{C_1 \mid\equiv S \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C_1 \mid\equiv \#(N_1), C_1 \mid\equiv S \mid\sim C_1, C_2, N_1}{\boxed{C_1 \mid\equiv S \mid\equiv C_1, C_2}}$$

M5

$$\frac{C_1 \mid\equiv C_1 \stackrel{K_{sc_1}}{\longleftrightarrow} S, C_1 \triangleleft N_1, \{H_{all}\}_{K_{sc_1}}}{C_1 \mid\equiv S \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C_1 \mid\equiv \#(N_1), C_1 \mid\equiv S \mid\sim \stackrel{c_2}{\longmapsto} C_2, I_{c_2}, N_1}{\boxed{C_1 \mid\equiv S \mid\equiv \stackrel{c_2}{\longmapsto} C_2, I_{c_2}}}$$

M6

$$\frac{C_2 \mid\equiv C_1 \stackrel{K_{sc_2}}{\longleftrightarrow} S, C_2 \triangleleft N_1, \{H_{all}\}_{K_{sc_2}}}{C_2 \mid\equiv S \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C_2 \mid\equiv \#(N_1), C_2 \mid\equiv S \mid\sim \stackrel{c_1}{\longmapsto} C_1, I_{c_1}, N_1}{\boxed{C_2 \mid\equiv S \mid\equiv \stackrel{c_1}{\longmapsto} C_1, I_{c_1}}}$$

M7

$$\frac{C_1 \mid\equiv \stackrel{K_{c_2}}{\longmapsto} C_2,\ S \triangleleft \{H_{all}\}_{K_{c_2}^{-1}}}{S \mid\equiv C_2 \mid\sim \#(D_1, N_1)}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the other client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the other client could have sent that message

$$\frac{{}_1 \mid\equiv \#(D_1, N_1),\ C_1 \mid\equiv C_2 \mid\sim (D_1, N_1)}{\boxed{S \mid\equiv C \mid\equiv D_1}}$$

M8

$$\frac{C_2 \mid\equiv \stackrel{K_{c_1}}{\longmapsto} C_1,\ C_2 \triangleleft \{H_{all}\}_{K_{c_1}^{-1}}}{C_2 \mid\equiv C_1 \mid\sim \#(D_2, N_1)}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the other client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the other client could have sent that message

$$\frac{C_2 \mid\equiv \#(D_2, N_1),\ C_2 \mid\equiv C_1 \mid\sim (D_2, N_1)}{C_2 \mid\equiv C_1 \mid\equiv D_2}$$

$$\frac{C_1 \mid\equiv D_1,\ C_1 \mid\equiv C_2 \mid\equiv D_2}{\boxed{C_1 \mid\equiv C_1 \stackrel{K_{c_1c_2}}{\longleftrightarrow} C_2}}$$

We have the two Diffie-Hellman components, we can use the first **Diffie-Hellman postulate** to derive that the client has generate the shared session key

We have almost one fresh Diffie-Hellman partial key, we can use the *second* **Diffie-Hellman postulate** to derive that the shared key is unique and believing to that session

$$\frac{\#(D_1),\ C_1 \mid\equiv C_2 \mid\equiv D_2}{\boxed{C_2 \mid\equiv (C_1 \stackrel{K_{c_1c_2}}{\longleftrightarrow} C_2)}}$$

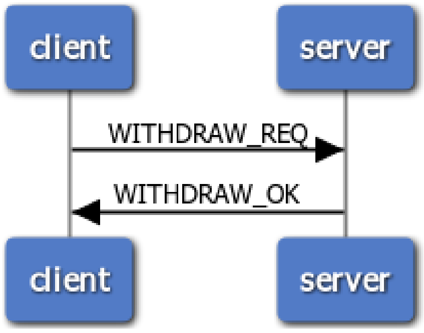
$$\frac{C_2 \mid\equiv D_2,\ C_2 \mid\equiv C_1 \mid\equiv D_1}{\boxed{C_2 \mid\equiv C_1 \stackrel{K_{c_1c_2}}{\longleftrightarrow} C_2}}$$

We have the two Diffie-Hellman components, we can use the first **Diffie-Hellman postulate** to derive that the client has generate the shared session key

We have almost one fresh Diffie-Hellman partial key, we can use the *second* **Diffie-Hellman postulate** to derive that the shared key is unique and believing to that session

$$\frac{C_2 \mid\equiv \#(D_2),\ C_2 \mid\equiv C_1 \mid\equiv D_1}{\boxed{C_1 \mid\equiv C_2 \mid\equiv (C_1 \stackrel{K_{c_1c_2}}{\longleftrightarrow} C_2)}}$$

The protocol will be used from the clients to undo a previously sent challenge. The messages requires authenticity so a signature based on a fresh nonce and made by AES256 GCM is applied on each message.



WITHDRAW_REQ

Message Type (17)	Username	Nonce N1	AES Signature

WITHDRAW_OK

Message Type (18)	Nonce N1	AES Signature

BAN Logic Analysis

Real Protocol

$M_1 \quad C \rightarrow S: \; M, U, N_1, \{H_{all}\}_{K_{cs}}$

$M_2 \quad S \rightarrow C: \; M, N_1, \{H_{all}\}_{K_{cs}}$

Ideal Protocol

$M_1 \quad C \rightarrow S: \; \#(N_1), \{H_{all}\}_{K_{cs}}$

$M_2 \quad S \rightarrow C: \; \#(N_1), \{H_{all}\}_{K_{cs}}$

Goals

$C \mid\equiv S \mid\equiv N_1$

$S \mid\equiv C \mid\equiv U$

Assumptions

$C \mid\equiv \xleftrightarrow{K_{cs}} S$

$S \mid\equiv \xleftrightarrow{K_{cs}} S$

Analysis

M1

$$\frac{S \mid\equiv C \xleftrightarrow{K_{sc}} S, S \triangleleft N_1, \{H_{all}\}_{K_{sc}}}{S \mid\equiv C \mid\sim U}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(N_1), S \mid\equiv C \mid\sim U, N_1}{\boxed{S \mid\equiv C \mid\equiv U}}$$

M2

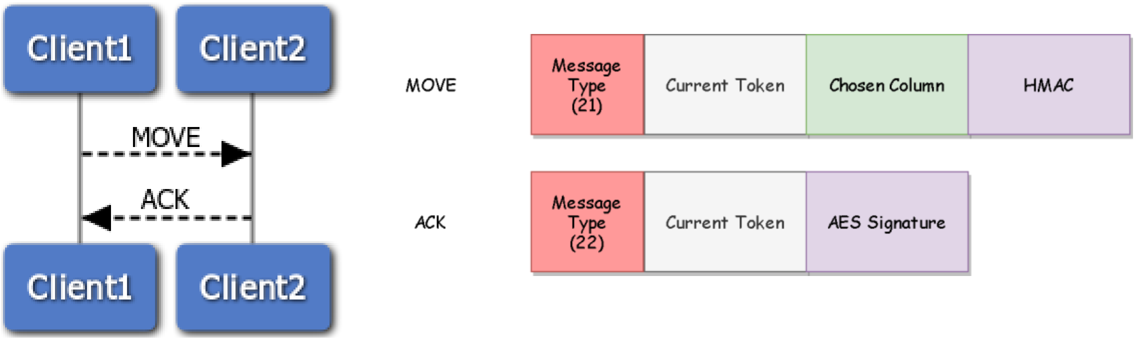
$$\frac{C \mid\equiv C \xleftrightarrow{K_{sc}} S, C \triangleleft N_1, \{H_{all}\}_{K_{sc}}}{C \mid\equiv S \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C \mid\equiv \#(N_1), C \mid\equiv S \mid\sim N_1}{\boxed{C \mid\equiv S \mid\equiv N_1}}$$

The protocol will be used from the clients to make a move during the match. The messages requires authenticity so a signature based on a fresh nonce and made by AES256 GCM is applied on each message. The only field that requires confidentiality is the chosen column of the user and so it will be encrypted.



BAN Logic Analysis

Real Protocol

M_1	$C_1 \to C_2 : M, CT, \{(C, H_{all}\}_{K_{c_1c_2}}$
M_2	$C_2 \to C_1 : M, CT, \{H_{all}\}_{K_{c_1c_2}}$

Ideal Protocol

M_1	$C_1 \to C_2 : \#(CT), \{(C, H\}_{K_{c_1c_2}}$
M_2	$C_2 \to C_1 : \#(CT), \{H_{all}\}_{K_{c_1c_2}}$

Goals

$C_2 \mid\equiv C_1 \mid\equiv C$

$C_1 \mid\equiv C_2 \mid\equiv CT$

Assumptions

$C_1 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2$

$C_2 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2$

Analysis

M1

$$\frac{C_2 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2, C_2 \triangleleft CT, \{(C, H_{all}\}_{K_{c_1c_2}}}{C_2 \mid\equiv C_1 \mid\sim CT, C}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the other client has sent the fields of the mes-

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the other client could have sent that message

$$\frac{C_2 \mid\equiv \#(CT), C_2 \mid\equiv C_1 \mid\sim CT, C}{\boxed{C_2 \mid\equiv C_1 \mid\equiv C}}$$

M2

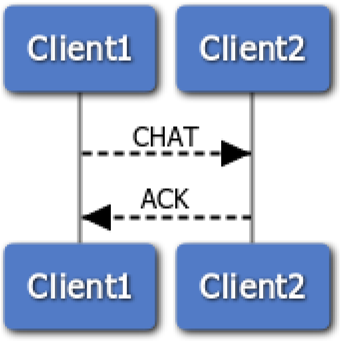
$$\frac{C_1 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2, C_1 \triangleleft CT, \{H_{all}\}_{K_{c_1c_2}}}{C_2 \mid\equiv C1 \mid\sim CT}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the other client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the other client could have sent that message

$$\frac{C_1 \mid\equiv \#(CT), C_1 \mid\equiv C_2 \mid\sim CT}{\boxed{C_2 \mid\equiv C_1 \mid\equiv CT}}$$

The protocol will be used from the clients to send a message the adversary during the match. The messages requires authenticity so a signature based on a fresh nonce and made by AES256 GCM is applied on each message. The only field that requires confidentiality is the sent message and so it will be encrypted



CHAT	Message Type (24)	Nonce N1	Message	AES Signature
ACK	Message Type (22)	Nonce N1	AES Signature	

BAN Logic Analysis

Real Protocol

$$M_1 \quad C_1 \rightarrow C_2 : M, N_1, \{(C,H\}_{Kc_1c_2}$$

$$M_2 \quad C_2 \rightarrow C_1 : M, N_1, \{H_{all}\}_{K_{c_1c_2}}$$

Goals

$$C_2 \mid\equiv C_1 \mid\equiv C$$

$$C_1 \mid\equiv C_2 \mid\equiv N_1$$

Ideal Protocol

$$M_1 \quad C_1 \rightarrow C_2 : \#(N_1), \{(C,H\}_{Kc_1c_2}$$

$$M_2 \quad C_2 \rightarrow C_1 : \#(N_1), \{H_{all}\}_{K_{c_1c_2}}$$

Assumptions

$$C_1 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2)$$

$$C_2 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2)$$

Analysis

M1

$$\frac{C_2 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2, C_2 \triangleleft N_1, \{(C,H\}_{K_{c_1c_2}}}{C_2 \mid\equiv C_1 \mid\sim N_1, C}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the other client has sent the fields of the mes-

The message contains a fresh field(nonce), se we can apply th **nonce verification postulate** to derive that the client will believes that only the other client could have sent that message

$$\frac{C_2 \mid\equiv \#(N_1), C_2 \mid\equiv C_1 \mid\sim N_1, C}{\boxed{C_2 \mid\equiv C_1 \mid\equiv C}}$$

M2

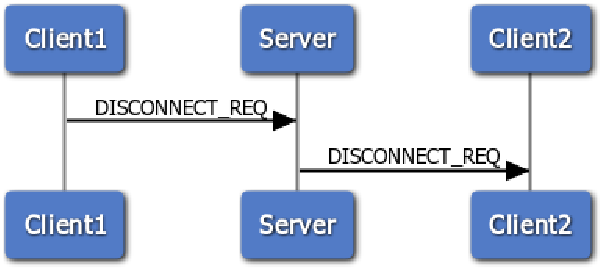
$$\frac{C_1 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2, C_1 \triangleleft N_1, \{H_{all}\}_{K_{c_1c_2}}}{C_1 \mid\equiv C_2 \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the other client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the other client could have sent that message

$$\frac{C_1 \mid\equiv \#(N_1), C_1 \mid\equiv C_2 \mid\sim N_1}{\boxed{C_1 \mid\equiv C_2 \mid\equiv N_1}}$$

The protocol will be used from the clients to exit from a match. The messages requires authenticity so a signature based on a fresh nonce and made by AES256 GCM is applied on each message.



BAN Logic Analysis

Real Protocol

$M_1 \quad C_1 \to S: \; M, N_1, \{(H_{all}\}_{KC_1S}$

$M_2 \quad S \to C_2: \; M, N_1, \{H_{all}\}_{K_{c_2s}}$

Ideal Protocol

$M_1 \quad C_1 \to C_2: \; \#(N_1), \{(H_{all}\}_{KC_1S}$

$M_2 \quad C_2 \to C_1: \; \#(N_1), \{(H_{all}\}_{KC_2S}$

Assumptions

$C_1 \mid\equiv C_1 \xleftrightarrow{K_{c_1s}} S$

$C_2 \mid\equiv C_2 \xleftrightarrow{K_{c_2s}} S$

$S \mid\equiv C_1 \xleftrightarrow{K_{c_1s}} S$

$S \mid\equiv C_2 \xleftrightarrow{K_{c_2s}} S$

Goals

$S \mid\equiv C_1 \mid\equiv N_1$

$C_2 \mid\equiv S \mid\equiv N_1$

Analysis

M1

$$\frac{S \mid\equiv C_1 \xleftrightarrow{K_{c_1s}} S, S \triangleleft N_1, \{(H\}_{K_{c_1s}}}{S \mid\equiv C_1 \mid\sim N_1}$$

The server has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the server will believes that the client has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the server will believes that only the client could have sent that message

$$\frac{S \mid\equiv \#(N_1), S \mid\equiv C_1 \mid\sim N_1}{\boxed{S \mid\equiv C_1 \mid\equiv N_1}}$$

M2

$$\frac{C_2 \mid\equiv C_2 \xleftrightarrow{K_{c_2s}} S, C_2 \triangleleft N_1, \{H_{all}\}_{K_{c_2s}}}{C_2 \mid\equiv S \mid\sim N_1}$$

The client has received a message containing a signature made by all the fields with the AES session key. We can apply the **signature postulate** to derive that the client will believes that the server has sent the fields of the message

The message contains a fresh field(nonce), se we can apply the **nonce verification postulate** to derive that the client will believes that only the server could have sent that message

$$\frac{C_2 \mid\equiv \#(N_1), C_2 \mid\equiv S \mid\sim N_1}{\boxed{C_2 \mid\equiv S \mid\equiv N_1}}$$

UML Diagram