2019-2020

Four in a Row

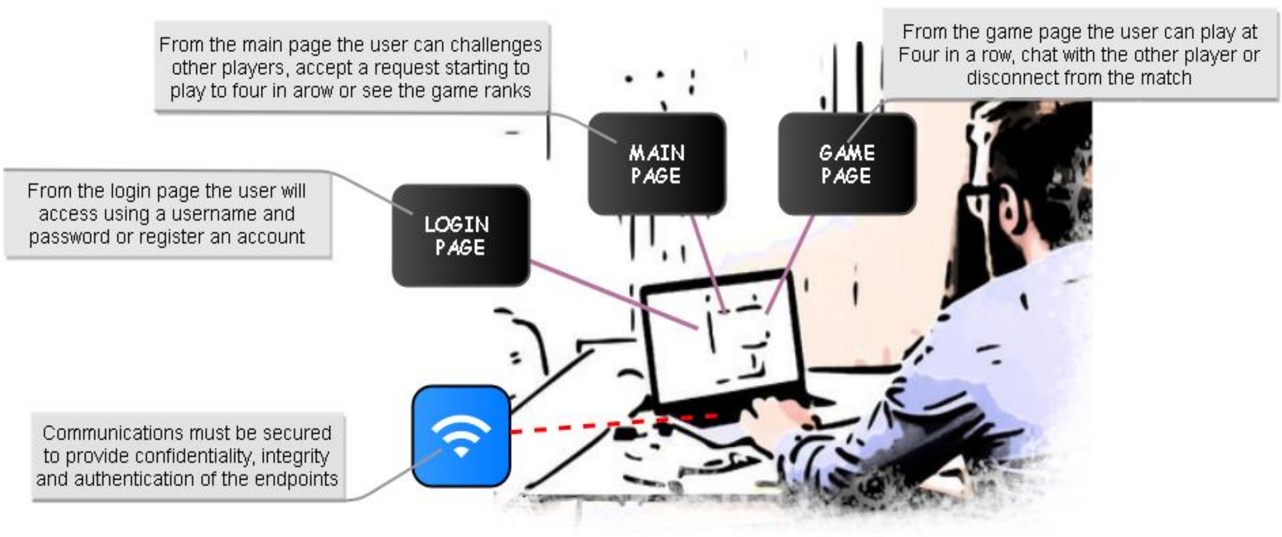
Cybersecurity Project

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 timestamp 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* register a new account giving a username and a password
- 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 users 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 **users** *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

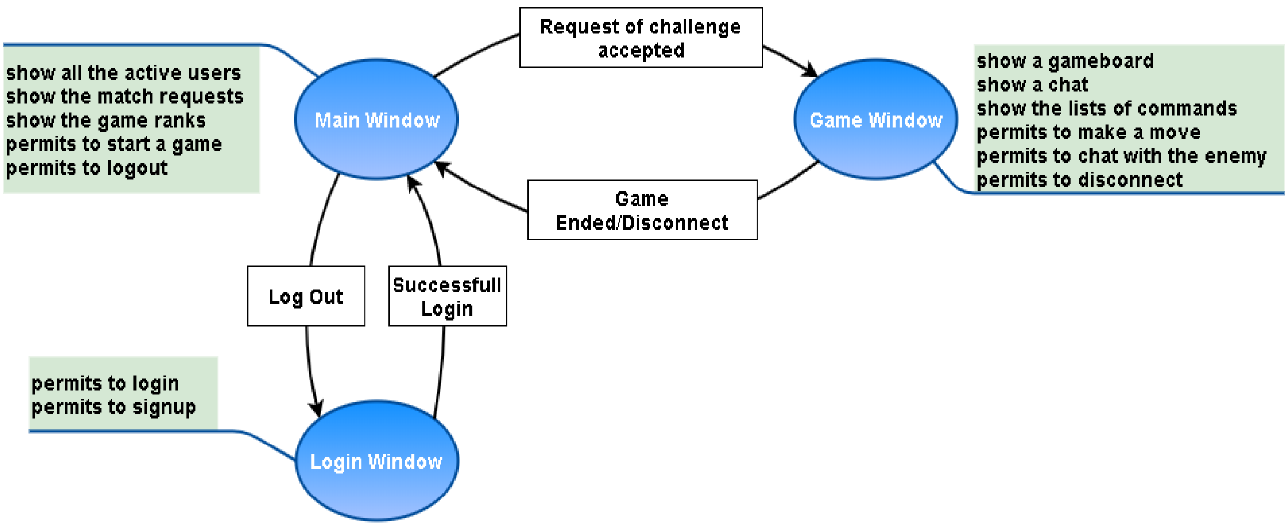
The following requirements are obtained starting from the User Specification Document.

- 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 encrypted by RSA server/user private key
- Each **message** not encrypted *will* have a timestamp 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 users public keys stored in a relational database
- The **server** *will* have all the clients connection information
- The **server** *will* have all the clients 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 window
  - Login window:
    - **It** *will* be the first window showed
    - **It** *will* permits to login and to sign up a new 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
    - **Signup** *will* be performed giving a username and a password
    - During the **signup** RSA keys and Diffie-Hellman parameters *will* be generated and stored
    - During the **signup** RSA keys *will* be certified by the server

    - 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 messages 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** disconnect 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* have a personal RSA certificate
- 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 ***SIGNUP_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 ***SIGNUP_OK*** messages from the server
- The **client** *will* receive ***SIGNUP_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

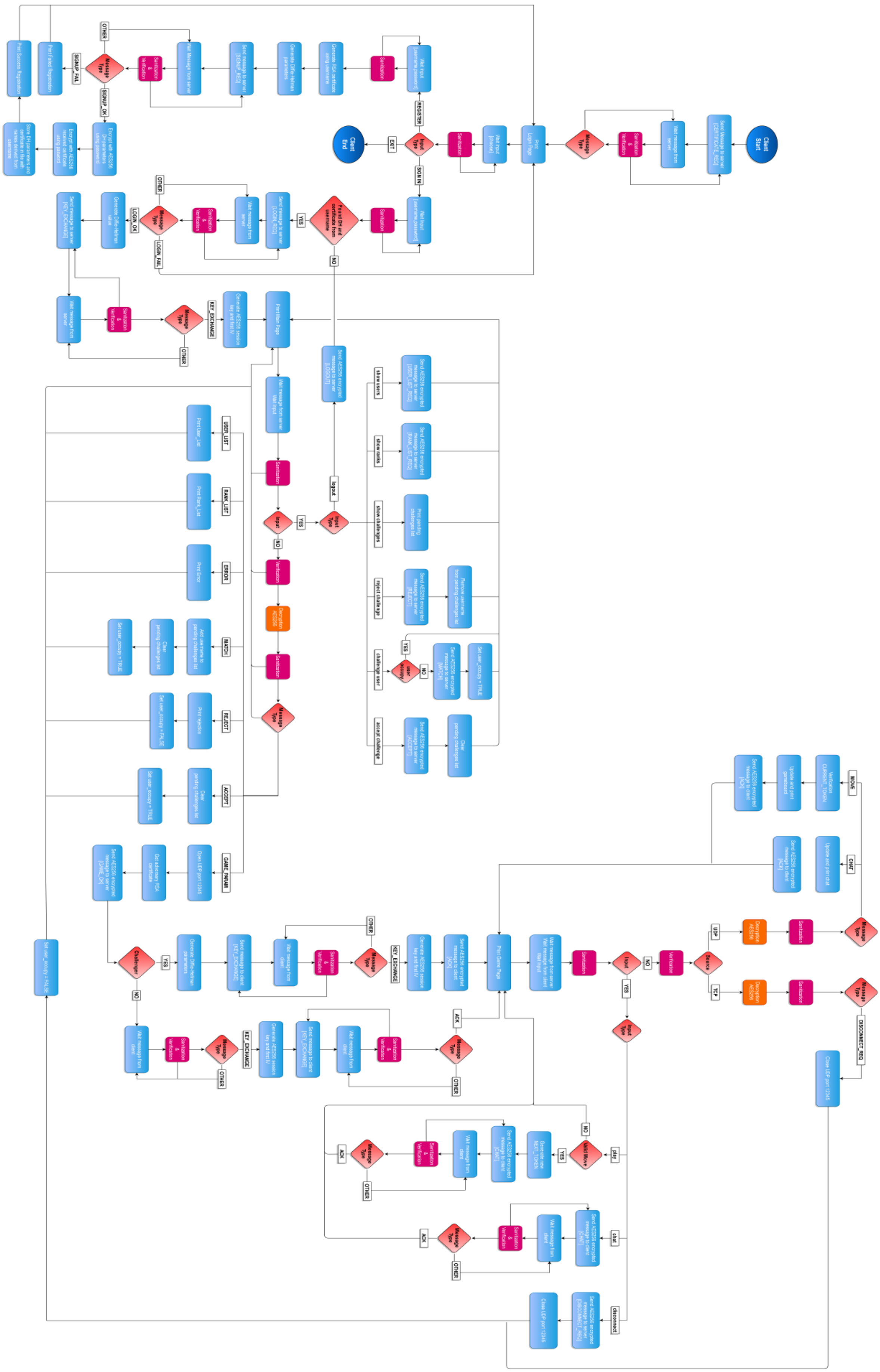
- The **client** *will* receive ***ACCEPT*** messages from the server
- The **client** *will* receive ***REJECT*** messages from the server
- The **client** *will* receive ***ERROR*** messages from the server
- The **client** *will* receive ***KEY_EXCHANGE*** messages from the server
- The **client** *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 window
    - **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 certificate, a timestamp and an HMAC encrypted by the user RSA private key
        - **It** *will* send a ***SIGNUP_REQ*** message in clear to the server containing a certificate and an HMAC encrypted 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 timestamp 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 timestamp 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 timestamp and an HMAC encrypted by the server RSA private key
        - **It** *will* receive a ***SIGNUP_OK*** message in clear from the server after a successful account registration. The message will contain the user certificate verified by the server, a timestamp and an HMAC encrypted by the server RSA private key
        - **It** *will* receive a ***SIGNUP_FAIL*** message in clear from the server after a failed account registration. The message will contain a timestamp and an HMAC encrypted 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 timestamp and an HMAC
  - **It** *will* send a ***RANK_REQ*** message encrypted with AES256 to the server containing a timestamp 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 timestamp 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 timestamp and an HMAC
  - **It** *will* send a ***LOGOUT_REQ*** message encrypted with AES256 to the server containing a timestamp and an HMAC
  - **It** *will* receive a ***GAME_PARAM*** message encrypted with AES256 to the server containing a certificate, 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 a ***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 timestamp and an HMAC
  - **It** *will* receive an ***ERROR*** message encrypted with AES256 from the server containing the type of error generated, a timestamp 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 timestamp 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 the other client during the previously move and given as **NEXT_TOKEN**
        - The client which receive a move controls if the **CURRENT_TOKEN** match with the previously sent **NEXT_TOKEN**
    - **It** *will* send a ***CHAT*** message encrypted with AES256 and containing the text, a timestamp, 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 encrypted by the user private RSA key
    - **It** *will* send a ***MOVE*** message encrypted with AES256 and containing a column field, a **CURRENT_TOKEN**, a **NEXT_TOKEN** , an acknowledgement and an HMAC
    - **It** *will* send a ***DISCONNECT_REQ*** message encrypted with AES256 and containing a timestamp 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 encrypted 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 timestamp 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 timestamp, an acknowledgement and an HMAC field encrypted with the user RSA private key from another client

Flow Diagram of Client Application

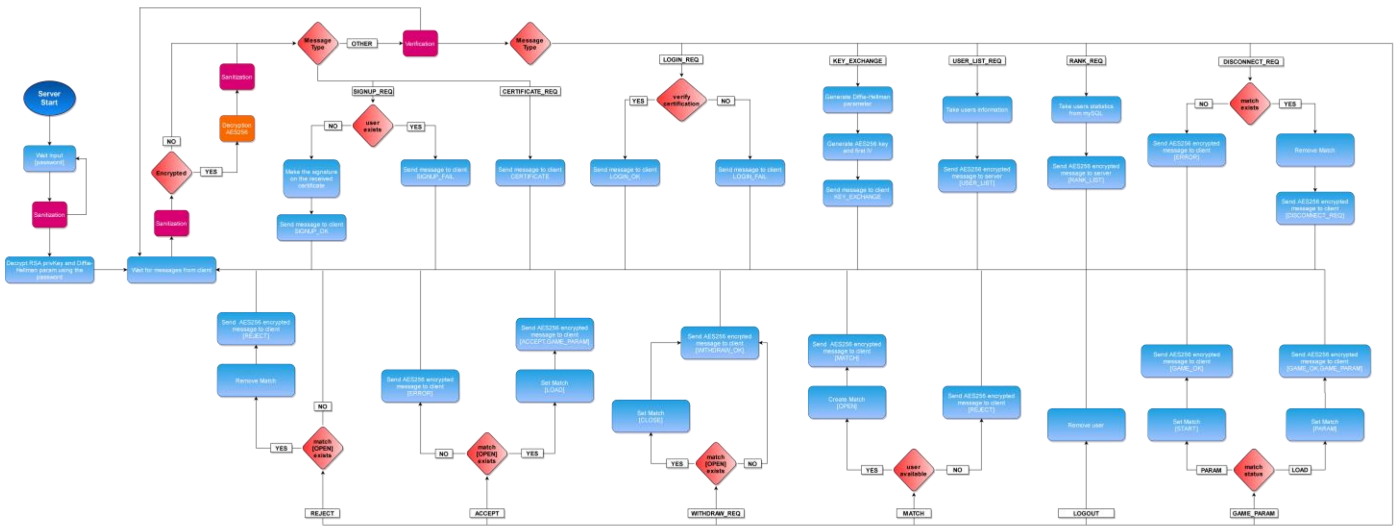


Server

- 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 users statistics
- The **server** *will* have a RSA certificate
- The **server** RSA certificate *will* be encrypted with AES256 using ‘admin” password
- 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 recorded after the signup
    - 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 an 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 timestamp and an HMAC encrypted with the server private RSA key after a successful login
- The **server** *will* send a ***LOGIN_FAIL*** message in clear containing a timestamp and an HMAC encrypted with the server private RSA key after a incorrect login
- The **server** *will* send a ***SIGNUP_OK*** message in clear containing a timestamp and an HMAC encrypted with the server RSA private key after a correct registration
- The **server** *will* send a ***SIGNUP_FAIL*** message in clear containing a timestamp and an HMAC encrypted with the server RSA private key after an invalid registration
- The **server** *will* send a ***CERTIFICATE*** message in clear composed by a certificate, a timestamp and an HMAC encrypted 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 encrypted 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 encrypted with the server RSA private key
- The **server** *will* send a ***DISCONNECT_REQ*** message encrypted with AES256 and containing a username, a type, a timestamp and an HMAC encrypted with the server RSA private key
- The **server** *will* send a ***REJECT*** message encrypted with AES256 and containing a username and an HMAC encrypted with the server RSA private key
- The **server** *will* send a ***ACCEPT*** message encrypted with AES256 and containing a username and an HMAC encrypted with the server RSA private key
- The **server** *will* send a ***GAME_PARAM*** message encrypted with AES256 and containing a certificate and an HMAC encrypted with the server RSA private key
- The **server** *will* receive a ***SIGNUP_REQ*** message in clear containing a certificate and an HMAC encrypted with a user 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 certificate, a timestamp and a HMAC encrypted with the user private RSA key
- The **server** *will* receive a ***USER_LIST_REQ*** message encrypted with AES256 and containing a timestamp and an HMAC encrypted with the user RSA private key
- The **server** *will* receive a ***RANK_REQ*** message encrypted with AES256 and containing a timestamp and an HMAC encrypted with the user RSA private key
- The **server** *will* receive a ***MATCH*** message encrypted with AES256 and containing a username, a timestamp and an HMAC encrypted with the user RSA private key
- The **server** *will* receive a ***ACCEPT*** message encrypted with AES256 containing the username and an HMAC encrypted by a user RSA private key
- The **server** *will* receive a ***REJECT*** message encrypted with AES256 and containing a username and an HMAC encrypted by a user RSA private key
- The **server** *will* receive a ***GAME_OK*** message encrypted with AES256 and containing a timestamp and an HMAC encrypted with a user RSA private key

Flow Diagram of Server Application



Protocol Analysis

In the following section there will be described in detail the exchange of the application messages and their adopted 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.

$$\frac{P \triangleleft (\overset{K_q}{\longmapsto} Q, \{H_{\overset{K_q}{\longmapsto} Q}\}_{K_s^{-1}}), \{\#(X)\}_{\overset{K_q}{\longmapsto} Q}}{P \mid\equiv \overset{K_q}{\longmapsto} Q, P \mid\equiv Q \mid\equiv \overset{K_q}{\longmapsto} Q}$$

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)

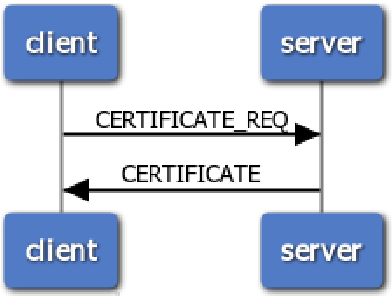
$$\frac{P \mid\equiv D_1, P \mid\equiv D_2}{P \mid\equiv A \overset{K_{AB}}{\longleftrightarrow} B}$$

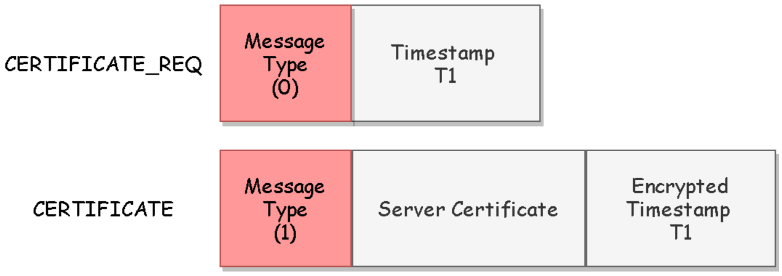
We need a postulate to link the freshness of the Diffie-Hellman parameters to the freshness of the shared session key

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

Certificate Protocol

The protocol is used during the client initialization to obtain the server certificate. All the messages are in clear but since everyone can request the server certificate no protection is required for the request messages.





BAN Logic Analysis

Real Protocol

$M_1 \quad C \rightarrow S: \ M, T_1$

$M_2 \quad S \rightarrow C: \ (C_s, \{H_{C_s}\}_{K_s^{-1}}), \{T_1\}_{K_s^{-1}}$

Ideal Protocol

$M_2 \quad S \rightarrow C: \xmapsto{K_s} S, \{H_{\xmapsto{K_s} S}\}_{K_s^{-1}}, \{\#(T_1)\}_{K_s^{-1}}$

Goals

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

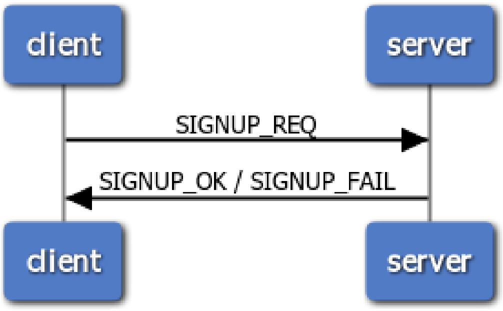
$C \mid\equiv S \mid\equiv \xmapsto{K_s} S$

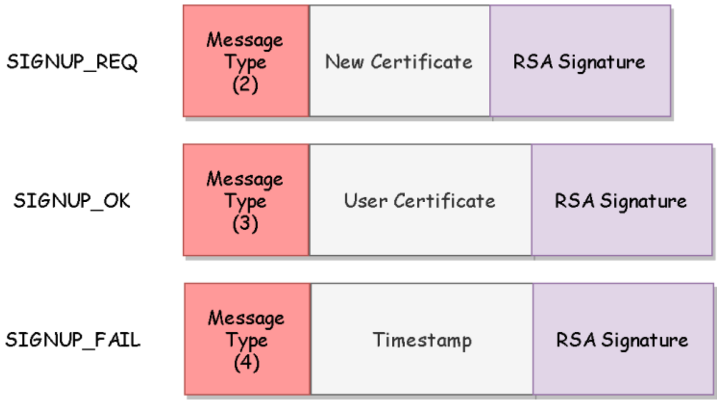
Analysis

M2		
	$\dfrac{C \triangleleft (\xmapsto{K_s} S, \{H_{\xmapsto{K_s} S}\}_{K_s^{-1}}), \{\#(T_1)\}_{K_s^{-1}}}{\boxed{C \mid\equiv \xmapsto{K_s} S} \ \boxed{C \mid\equiv S \mid\equiv \xmapsto{K_s} S}}$	The client has received a certificate containing the signature of validity of the certification authority(the server) and a fresh encrypted timestamp. We can apply the **certificate postulate** to derive that the certificate belongs to the server

Registration Protocol

The protocol is used the register a new account on the server. The client will generate a certificate containing a username that must be unique into the server. In the registration is accepted the server will sign the user certificate to the dominion. The communication are all in clear so to protect messages from MIM, corruptions and replay attack the messages will contain a signed HMAC field and eventually a timestamp.





BAN Logic Analysis

Real Protocol

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

$M_{2a} \quad S \to C:\ M, (C_c, \{H_{C_c}\}_{K_s^{-1}}), \{H_{all}\}_{K_s^{-1}}$

$M_{2b} \quad S \to C:\ M, T, \{H_{all}\}_{K_s^{-1}}$

Ideal Protocol

$M_1 \quad C \to S: \stackrel{K_c}{\longmapsto} C, \{H_{all}\}_{K_c^{-1}}$

$M_{2a} \quad S \to C:\ (\stackrel{K_c}{\longmapsto} C, \{H_{\stackrel{K_c}{\longmapsto} C}\}_{K_s^{-1}}), \{H_{all}\}_{K_s^{-1}}$

$M_{2b} \quad S \to C:\ \#(T), \{H_{all}\}_{K_s^{-1}}$

Goals

$S \mid\equiv C \mid\equiv \stackrel{K_c}{\longmapsto} C$

$C \mid\equiv S \mid\equiv \stackrel{K_c}{\longmapsto} C$

$C \mid\equiv S \mid\equiv T$

Assumptions

$C \mid\equiv \stackrel{K_s}{\longmapsto} S \quad C \mid\equiv \stackrel{K_c}{\longmapsto} C \quad C \mid\equiv S \mid\equiv \stackrel{K_s}{\longmapsto} S \quad C \mid\equiv \#(\stackrel{K_c}{\longmapsto} C)$

$S \mid\equiv \stackrel{K_c}{\longmapsto} C \quad S \mid\equiv \#(\stackrel{K_c}{\longmapsto} C)$

Analysis

M1

$$\frac{S \mid\equiv \stackrel{K_c}{\longmapsto} C,\ S \triangleleft \{H_{all}\}_{K_c^{-1}}}{S \mid\equiv C \mid\sim \stackrel{K_c}{\longmapsto} C}$$

The server has received a public key from a client that for assumption it believes is true. We can apply the **second message meaning postulate** to derive that it believes the key is sent by the client

For assumption the key is fresh(otherwise the registration will fail because the key is already registered) so using the **nonce verification postulate** we can assume that only the client could have sent the certificate

$$\frac{S \mid\equiv \#(\stackrel{K_c}{\longmapsto} C), \{H_{all}\}_{K_c^{-1}})}{\boxed{S \mid\equiv C \mid\equiv \stackrel{K_c}{\longmapsto} C}}$$

M2.a

$$\frac{C \mid\equiv \stackrel{K_s}{\longmapsto} S,\ C \triangleleft \{H_{all}\}_{K_s^{-1}}}{C \mid\equiv S \mid\sim \stackrel{K_c}{\longmapsto} C}$$

The client has received a certificate containing the signature of validity of the certification authority(the server). We can apply the **second message meaning postulate** to derive that the certificate is authorized from the server

For assumption the key is fresh(otherwise the registration will fail because the key is already registered) so using the **nonce verification postulate** we can assume that only the client could have sent the certificate

$$\frac{C \mid\equiv \#(\stackrel{K_c}{\longmapsto} C),\ C \triangleleft \{H_{all}\}_{K_s^{-1}}}{\boxed{C \mid\equiv S \mid\equiv \stackrel{K_c}{\longmapsto} C}}$$

M2.b

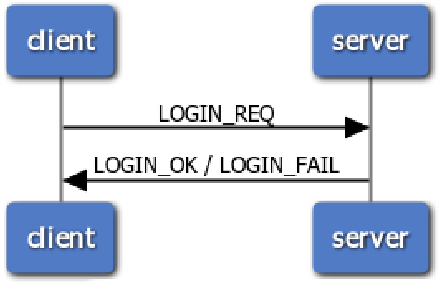
The client has received a message containing an HMAC encrypted by the server public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

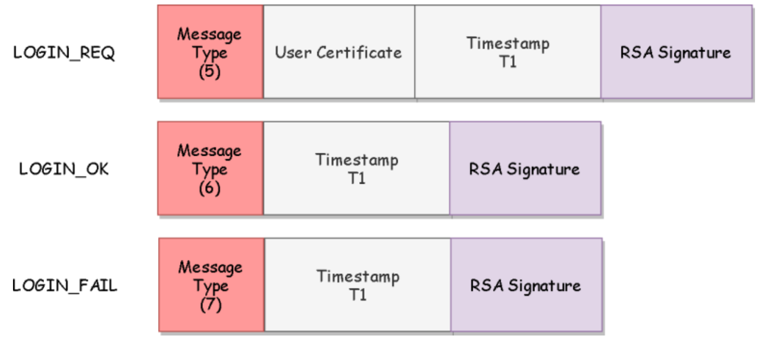
$$\frac{C \mid\equiv \stackrel{K_s}{\longmapsto} S,\ C \triangleleft \{H_{all}\}_{K_s^{-1}}}{C \mid\equiv S \mid\sim T}$$

$$\frac{C \mid\equiv \#(T),\ C \triangleleft \{H_{all}\}_{K_s^{-1}}}{\boxed{C \mid\equiv S \mid\equiv T}}$$

The client believes that the timestamp is fresh so using the **nonce verification postulates** we can assume that it believes that only the server could have sent the message

The protocol is used to access the main application. The client will send its certificate and the server the certificate validity. The messages are all in clear so to protect them from MIM, corruptions and replay attack the messages will contain a signed HMAC field and a timestamp.





BAN Logic Analysis

Real Protocol

$M_1 \quad C \rightarrow S: \; M, (C_c, \{H_{C_c}\}_{K_s^{-1}}), \{T, H_{all}\}_{K_c^{-1}}$

$M_2 \quad S \rightarrow C: \; M, T, \{H_{all}\}_{K_s^{-1}}$

Ideal Protocol

$M_1 \quad C \rightarrow S: \; (\xmapsto{K_c} C, \{H_{\xmapsto{K_c} C}\}_{K_s^{-1}}), \#(T), \{H_{all}\}_{K_c^{-1}}$

$M_2 \quad S \rightarrow C: \; \#(T), \{H_{all}\}_{K_s^{-1}}$

Goals

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

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

$C \mid\equiv S \mid\equiv T$

Assumptions

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

Analysis

M1

$$\frac{S \triangleleft (\xmapsto{K_c} C, \{H_{\xmapsto{K_c} C}\}_{K_s^{-1}}), \{\#(T)\}_{K_c^{-1}}}{\boxed{S \mid\equiv \xmapsto{K_c} C}, \boxed{S \mid\equiv C \mid\equiv \xmapsto{K_c} C}}$$

The client has received a message containing a certificate signed by the certification authority(the server) and a fresh encrypted timestamp. We can apply the **certificate postulate** to derive that the certificate belongs to the client

M2

$$\frac{C \mid\equiv \xmapsto{K_s} S, C \triangleleft \{H_{all}\}_{K_s^{-1}}}{C \mid\equiv S \mid\sim T}$$

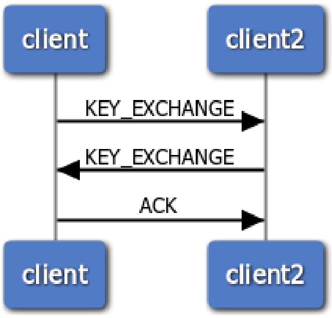
The client has received an HMAC encrypted by the server certificate. We can apply the **second message meaning postulate** to derive that it believes the message is sent by the client

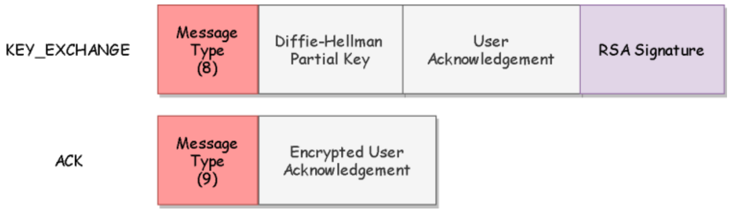
The received HMAC contains a fresh timestamp, so it is fresh and we can apply the **nonce verification postulate** to derivate that the client believes that only the server could have sent the message

$$\frac{C \mid\equiv \#(T), C \triangleleft \{H_{all}\}_{K_s^{-1}}}{\boxed{C \mid\equiv S \mid\equiv T}}$$

Key Exchange Protocol

The protocol will be used for the generation and exchange of an AES256 session key using Diffie-Hellman. The protocol will be is used for the client-server and client-client sessions with small differences. The messages are all in clear so to prevent MIMs attack, corruptions and replays we have inserted a Timestamp and an HMAC signed field. The ACK messages are already encrypted so the doesn't need to give user authentication





BAN Logic Analysis

Real Protocol

$$M_1 \quad A \to B: \; M, D_1, T, \{H_{all}\}_{K_a^{-1}}$$
$$M_2 \quad B \to A: \; M, D_2, T, \{H_{all}\}_{K_b^{-1}}$$
$$M_3 \quad A \to B: \; M, \{T\}K_{ab}$$

Ideal Protocol

$$M_1 \quad A \to B: \; \#(D_1, T), \{H_{all}\}_{K_a^{-1}}$$
$$M_2 \quad B \to A: \; \#(D_2, T), \{H_{all}\}_{K_b^{-1}}$$
$$M_3 \quad A \to B: \; \{\#(T)\}_{K_{ab}}$$

Goals

$$A \mid\equiv B \mid\equiv D_2 \quad A \mid\equiv A \xleftrightarrow{K_{ab}} B \quad A \mid\equiv B \mid\equiv A \xleftrightarrow{K_{ab}} B \quad A \mid\equiv \#(A \xleftrightarrow{K_{ab}} B)$$
$$B \mid\equiv A \mid\equiv D_1 \quad B \mid\equiv A \xleftrightarrow{K_{ab}} B \quad B \mid\equiv A \mid\equiv A \xleftrightarrow{K_{ab}} B \quad B \mid\equiv \#(A \xleftrightarrow{K_{ab}} B)$$

Assumptions

$$A \mid\equiv \overset{K_b}{\longmapsto} B \quad A \mid\equiv B \mid\equiv \overset{K_a}{\longmapsto} A \quad A \mid\equiv D_1 \quad A \mid\equiv \#(T)$$
$$B \mid\equiv \overset{K_a}{\longmapsto} A \quad B \mid\equiv A \mid\equiv \overset{K_b}{\longmapsto} B \quad B \mid\equiv D_2 \quad B \mid\equiv \#(T)$$

Analysis

M1

$$\frac{B \mid\equiv \overset{K_a}{\longmapsto} A, B \triangleleft \{D_1, T\}_{K_a^{-1}}}{B \mid\equiv A \mid\sim (D_1, T)}$$

The client has received a message containing an HMAC encrypted by the user/server public key. We can apply the **second message meaning postulate** to derive that the client believes that the user/server has sent the message

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

$$\frac{B \mid\equiv \#(D_1, T), B \mid\equiv A \mid\sim (D_1, T)}{B \mid\equiv A \mid\equiv (D_1, T)}$$

M2

$$\frac{A \mid\equiv \xrightarrow{K_b} B, A \triangleleft \{D_2, T\}_{K_b^{-1}}}{A \mid\equiv B \mid\sim (D_2, T)}$$

The client has received a message containing an HMAC encrypted by the user/server public key. We can apply the **second message meaning postulate** to derive that the client believes that the user/server has sent the

The received HMAC contains a fresh element, 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

$$\frac{A \mid\equiv \#(D_2, T), A \mid\equiv B \mid\sim (D_2, T)}{\boxed{A \mid\equiv B \mid\equiv (D_2}, T)}$$

$$\frac{A \mid\equiv D_1, A \mid\equiv B \mid\equiv D_2}{\boxed{A \mid\equiv A \xleftrightarrow{K_{ab}} B}}$$

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

$$\frac{A \mid\equiv \#(D_1), A \mid\equiv B \mid\equiv D_2}{\boxed{A \mid\equiv \#(A \xleftrightarrow{K_{ab}} B)}}$$

$$\frac{B \mid\equiv D_2, B \mid\equiv A \mid\equiv D_1}{\boxed{B \mid\equiv A \xleftrightarrow{K_{ab}} B}}$$

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

$$\frac{B \mid\equiv \#(D_2), B \mid\equiv A \mid\equiv D_1}{\boxed{B \mid\equiv \#(A \xleftrightarrow{K_{ab}} B)}}$$

$$\frac{A \mid\equiv (A \xleftrightarrow{K_{ab}} B), A \triangleleft \{T\}_{K_{ab}}}{A \mid\equiv B \mid\sim T}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the user/server has sent the message

The received HMAC contains a fresh element, so it is fresh and we can apply the **nonce verification postulate** to derive that the client believes that only the user/server could have sent the key

$$\frac{A \mid\equiv \#(T), A \mid\equiv B \mid\sim T}{\boxed{A \mid\equiv B \mid\equiv (A \xleftrightarrow{K_{ab}} B)}}$$

M3

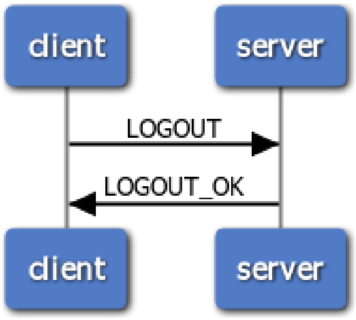
$$\frac{B \mid\equiv (A \xleftrightarrow{K_{ab}} B), B \triangleleft \{T\}_{K_{ab}}}{B \mid\equiv A \mid\sim T}$$

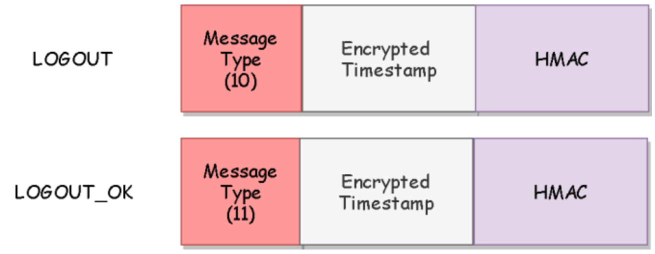
The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

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

$$\frac{B \mid\equiv \#(T), B \mid\equiv A \mid\sim T}{\boxed{B \mid\equiv A \mid\equiv (A \xleftrightarrow{K_{ab}} B)}}$$

The protocol will be used to quit from the application. The messages are all protected using AES256 CBC encryption which encrypts all the field excepts for the message type. We have only increased the dimension of the encrypted message inserting a timestamp and an HMAC field to prevent corruptions.





BAN Logic Analysis

Real Protocol

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

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

Ideal Protocol

$M_1 \quad C \rightarrow S: \ \{T, H_{all}\}_{Kcs}$

$M_2 \quad S \rightarrow C: \ \{T, H_{all}\}_{Kcs}$

Goals

$S \mid\equiv C \mid\equiv \{T, H\}$

$C \mid\equiv S \mid\equiv \{T, H\}$

Assumptions

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

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

Analysis

M1

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

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C \xleftrightarrow{K_{cs}} S), S \mid\equiv C \mid\sim \{T, H_{all}\}}{\boxed{S \mid\equiv C \mid\equiv \{T, H_{all}\}}}$$

M2

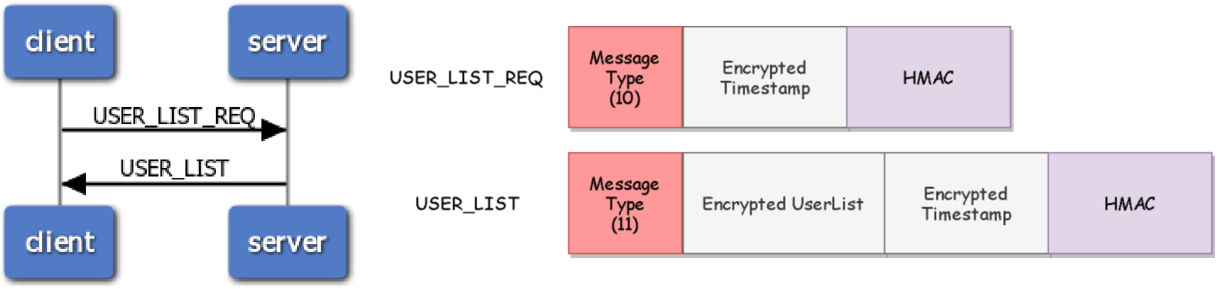
$$\frac{C \mid\equiv C \xleftrightarrow{K_{cs}} S, C \triangleleft \{T, H_{all}\}_{K_{cs}}}{C \mid\equiv S \mid\sim \{T, H_{all}\}}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C \mid\equiv \#(C \xleftrightarrow{K_{cs}} S), C \mid\equiv S \mid\sim \{T, H_{all}\}}{\boxed{C \mid\equiv S \mid\equiv \{T, H_{all}\}}}$$

The protocol will be used from the clients to obtain a list of the users currently available to be challenged. The messages are all protected using AES256 CBC encryption which encrypts all the field excepts for the message type. We have only increased the dimension of the message inserting a timestamp and an HMAC field to prevent corruptions.



BAN Logic Analysis

Real Protocol

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

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

Ideal Protocol

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

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

Goals

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

$S \mid\equiv C \mid\equiv T$

Assumptions

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

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

Analysis

M1

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

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C \xleftrightarrow{K_{cs}} S), \; S \mid\equiv C \mid\sim \{T, H_{all}\}}{S \mid\equiv C \mid\equiv \{T, H_{all}\}}$$

M2

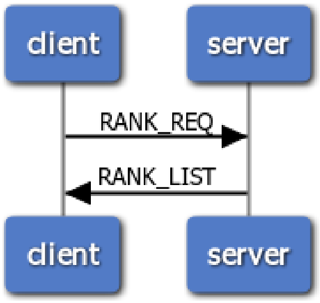
$$\frac{C \mid\equiv C \xleftrightarrow{K_{cs}} S, \; C \triangleleft \{T, H_{all}\}_{K_{cs}}}{C \mid\equiv S \mid\sim \{T, H_{all}\}}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C \mid\equiv \#(C \xleftrightarrow{K_{cs}} S), \; C \mid\equiv S \mid\sim \{T, H_{all}\}}{C \mid\equiv S \mid\equiv \{T, L, H_{all}\}}$$

The protocol will be used from the clients to obtain a list of the users game statistics. The messages are all protected using AES256 CBC encryption which encrypts all the field excepts for the message type. The messages will have an HMAC field to identify corruptions.



RANK_REQ

Message Type (12)	Encrypted Timestamp	HMAC

RANK

Message Type (13)	Encrypted RankList	Encrypted Timestamp	HMAC

BAN Logic Analysis

Real Protocol

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

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

Ideal Protocol

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

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

Goals

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

$$S \mid\equiv C \mid\equiv T$$

Assumptions

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

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

Analysis

M1

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

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C \xleftrightarrow{K_{cs}} S), S \mid\equiv C \mid\sim \{T, H_{all}\}}{\boxed{S \mid\equiv C \mid\equiv \{T,} H_{all}\}}$$

M2

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

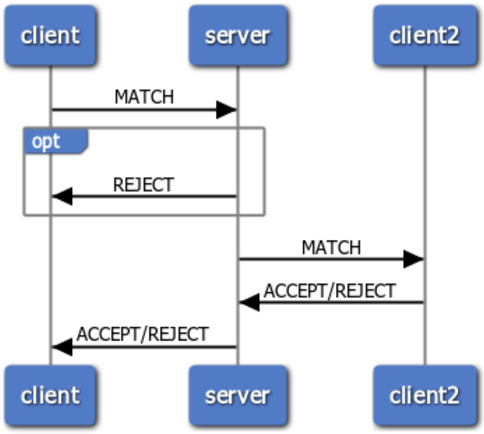
The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

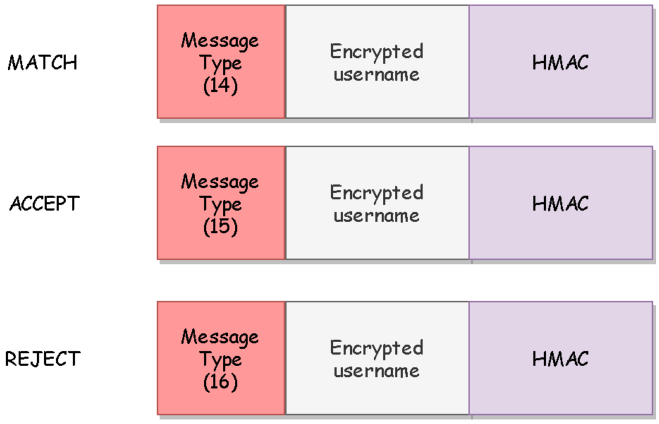
The received HMAC contains a fresh element, so it is fresh and we can apply the **nonce verification postulat** to derivate that the client believes that only the server could have sent the message

$$\frac{C \mid\equiv \#(C \xleftrightarrow{K_{cs}} S), C \mid\equiv S \mid\sim \{T, H_{all}\}}{\boxed{C \mid\equiv S \mid\equiv \{T, L,} H_{all}\}}$$

Challenge Protocol

The protocol will be used from the clients to request to another player to play a game. The messages are all protected using AES256 CBC encryption which encrypts all the field excepts for the message type. The messages will have an HMAC field to prevent corruptions.





BAN Logic Analysis

Real Protocol

$M_1 \quad C_1 \rightarrow S : M, \{U_2, H_{all}\}_{K_{sc_1}}$

$M_{2a} \quad S \rightarrow C_2 : M, \{U_2, H_{all}\}_{K_{sc_2}} \qquad M_{2b} \quad S \rightarrow C_1 : M, \{U_2, H_{all}\}_{K_{sc_1}}$

$M_3 \quad C_2 \rightarrow S : M, \{U_1, H_{all}\}_{K_{sc_2}}$

$M_4 \quad S \rightarrow C_1 : M, \{U_1, H_{all}\}_{K_{sc_1}}$

Ideal Protocol

$M_1 \quad C_1 \rightarrow S : \{U_2, H_{all}\}_{K_{sc_1}}$

$M_{2a} \quad S \rightarrow C_2 : \{U_2, H_{all}\}_{K_{sc_2}} \qquad M_{2b} \quad S \rightarrow C_1 : \{U_2, H_{all}\}_{K_{sc_1}}$

$M_3 \quad C_2 \rightarrow S : \{U_1, H_{all}\}_{K_{sc_2}}$

$M_4 \quad S \rightarrow C_1 : \{U_1, H_{all}\}_{K_{sc_1}}$

Goals

$C_1 \mid\equiv S \mid\equiv U_1 \qquad C_2 \mid\equiv S \mid\equiv U_2 \qquad S \mid\equiv C_1 \mid\equiv U_2 \qquad S \mid\equiv C_2 \mid\equiv U_1$

$C_1 \mid\equiv S \mid\equiv U_2 \qquad S \mid\equiv C_1 \mid\equiv U_2$

Assumptions

$C_1 \mid\equiv C_1 \xleftrightarrow{K_{cs_1}} S \qquad C_1 \mid\equiv S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S \qquad C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S)$

$C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad C_2 \mid\equiv S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad C_2 \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S)$

$S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S \qquad S \mid\equiv C_1 \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S \qquad S \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S)$

$S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad S \mid\equiv C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad S \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S)$

Analysis

M1

$$\frac{S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S, S \triangleleft \{U_2, H_{all}\}_{K_{sc_1}}}{S \mid\equiv C_1 \mid\sim U_2}$$

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S), S \mid\equiv C_1 \mid\sim U_2}{\boxed{S \mid\equiv C_1 \mid\equiv U_2}}$$

M2

$$\frac{C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S, C_2 \triangleleft \{U_2, H_{all}\}_{K_{sc_2}}}{C_2 \mid\equiv S \mid\sim U_2}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C_2 \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S), C_2 \mid\equiv S \mid\sim U_2}{\boxed{C_2 \mid\equiv S \mid\equiv U_2}}$$

M3

$$\frac{C_1 \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S, C_1 \triangleleft \{U_2, H_{all}\}_{K_{sc_1}}}{C_1 \mid\equiv S \mid\sim U_2}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S), C_1 \mid\equiv S \mid\sim U_2}{\boxed{C_1 \mid\equiv S \mid\equiv U_2}}$$

M4

$$\frac{S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S, C_2 \triangleleft \{U_1, H_{all}\}_{K_{sc_2}}}{S \mid\equiv C_2 \mid\sim U_1}$$

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S), C_2 \mid\equiv S \mid\sim U_1}{\boxed{S \mid\equiv C_2 \mid\equiv U_1}}$$

M5

$$\frac{C_1 \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S, C_1 \triangleleft \{U_1, H_{all}\}_{K_{sc_1}}}{C_1 \mid\equiv S \mid\sim U_1}$$

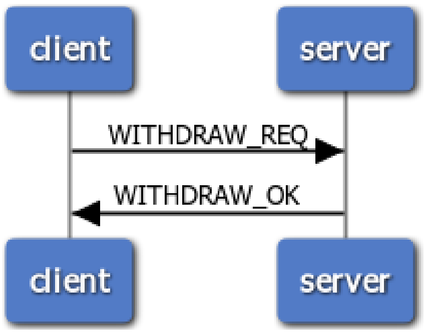
The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

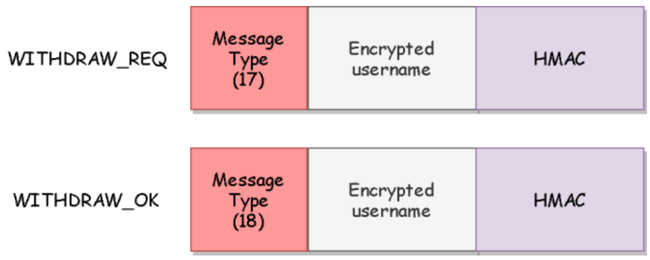
The received HMAC contains a fresh element, so it is fresh and we can apply the **nonce verification postulate** to derivate that the client believes that only the server could have sent the message

$$\frac{C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S), C_1 \mid\equiv S \mid\sim U_1}{\boxed{C_1 \mid\equiv S \mid\equiv U_1}}$$

Withdraw Protocol

The protocol will be used from the clients to undo a previously sent challenge. The messages are all protected using AES256 CBC encryption which encrypts all the field excepts for the message type. The messages will have an HMAC field to prevent corruptions.





BAN Logic Analysis

Real Protocol

$M_1 \quad C \to S: \; M, \{T, H_{all}\}_{Kcs}$

$M_2 \quad S \to C: \; M, \{T, H_{all}\}_{Kcs}$

Ideal Protocol

$M_1 \quad C \to S: \; \{T, H_{all}\}_{Kcs}$

$M_2 \quad S \to C: \; \{T, H_{all}\}_{Kcs}$

Goals

$S \mid\equiv C \mid\equiv \{T, H\}$

$C \mid\equiv S \mid\equiv \{T, H\}$

Assumptions

$C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad C \mid\equiv S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad C \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S)$

$S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad S \mid\equiv C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad S \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S)$

Analysis

M1

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

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S), \; S \mid\equiv C \mid\sim \{T, H_{all}\}}{\boxed{S \mid\equiv C \mid\equiv \{T, H_{all}\}}}$$

M2

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

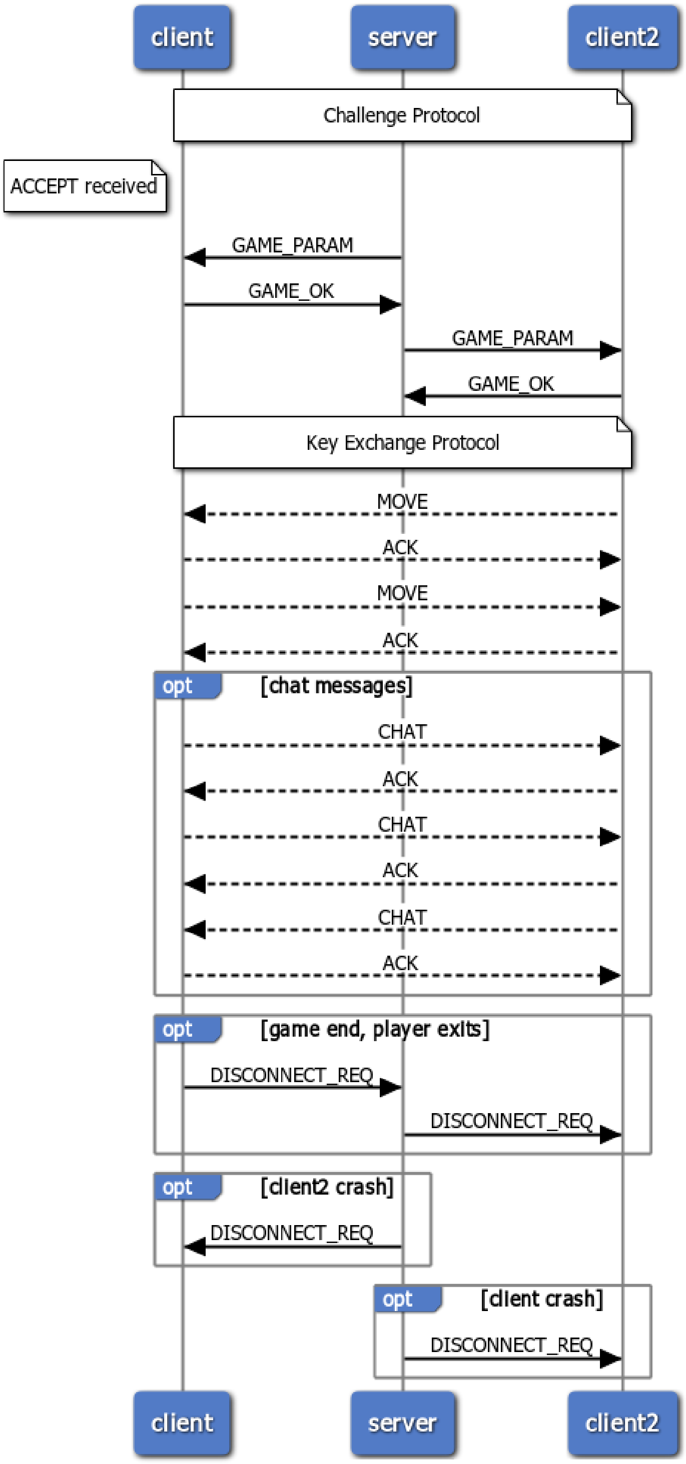
The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

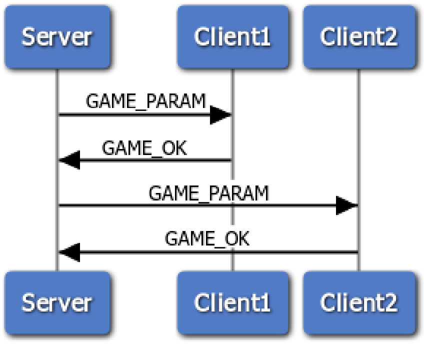
$$\frac{C \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S), \; C \mid\equiv S \mid\sim \{T, H_{all}\}}{\boxed{C \mid\equiv S \mid\equiv \{T, H_{all}\}}}$$

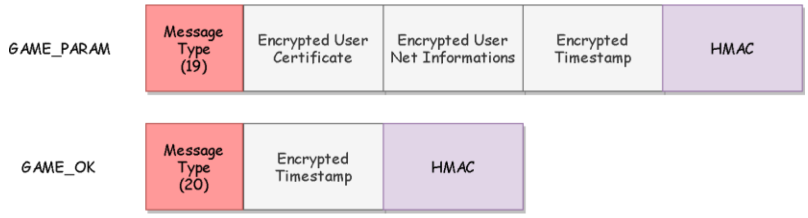
Game Protocol

The protocol will be used to play a math with another player. The messages are all encrypted using AES256 CBC. We have inserted where needed a timestamp to increase the message size and an HMAC field to prevent corruptions.



Game Param Protocol





BAN Logic Analysis

Real Protocol

$$M_1 \quad S \rightarrow C_1 : \; M, \{(C_{U_2}, \{H_{C_{u_2}}\}_{K_S^{-1}}), T_1, IP_{u_2}, H_{all}\}_{K_{sc_1}}$$

$$M_2 \quad C_1 \rightarrow S : \; M, \{T_1, H_{all}\}_{K_{sc_1}}$$

$$M_3 \quad S \rightarrow C_2 : \; M, \{(C_{U_1}, \{H_{C_{U_1}}\}_{K_S^{-1}}), T_2, IP_{u_1}, H_{all}\}_{K_{sc_2}}$$

$$M_4 \quad C_2 \rightarrow S : \; M, \{T_2, H_{all}\}_{K_{sc_2}}$$

Goals

$$S \mid\equiv C_1 \mid\equiv T_1$$

$$S \mid\equiv C_2 \mid\equiv T_2$$

$$C_1 \mid\equiv S \mid\equiv \xmapsto{K_{u_2}} U_2, IP_{u_2}$$

$$C_2 \mid\equiv S \mid\equiv \xmapsto{K_{u_1}} U_1, IP_{u_2}$$

Ideal Protocol

$$M_1 \quad S \rightarrow C_1 : \; \#(T_1), \{(\xmapsto{K_{u_2}} U_2, \{H_{\xmapsto{K_{u_2}} U_2}\}_{K_S^{-1}}), T_1, IP_{u_2}, H_{all}\}_{K_{sc_1}}$$

$$M_2 \quad C_1 \rightarrow S : \; \{T_1, H_{all}\}_{K_{sc_1}}$$

$$M_3 \quad S \rightarrow C_2 : \; \#(T_2), \{(\xmapsto{K_{u_1}} U_1, \{H_{\xmapsto{K_{u_1}} U_1}\}_{K_S^{-1}}), T_2, IP_{u_1}, H_{all}\}_{K_{sc_2}}$$

$$M_4 \quad C_2 \rightarrow S : \; \{T_2, H_{all}\}_{K_{sc_2}}$$

Assumptions

$$C_1 \mid\equiv C_1 \xleftrightarrow{K_{cs_1}} S \qquad C_1 \mid\equiv S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S \qquad C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S)$$

$$C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad C_2 \mid\equiv S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad C_2 \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S)$$

$$S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S \qquad S \mid\equiv C_1 \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S \qquad S \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S)$$

$$S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad S \mid\equiv C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S \qquad S \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S)$$

Analysis

M1

$$\frac{C_1 \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S, C_1 \triangleleft \{(\xmapsto{K_{u_2}} U_2, \{H_{\xmapsto{K_{u_2}} U_2}\}_{K_s^{-1}}), IP_{u_2}\}_{K_{sc_1}}}{C_1 \mid\equiv S \mid\sim \xmapsto{K_{u_2}} U_2, IP_{u_2}}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S), C_1 \mid\equiv S \mid\sim \xmapsto{K_{u_2}} U_2, IP_{u_2}}{\boxed{C_1 \mid\equiv S \mid\equiv \xmapsto{K_{u_2}} U_2, IP_{u_2}}}$$

M2

$$\frac{S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S, S \triangleleft \{T_1, H_{all}\}_{K_{sc_1}}}{S \mid\equiv C_1 \mid\sim T_1}$$

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S), S \mid\equiv C_1 \mid\sim T_1}{\boxed{S \mid\equiv C_1 \mid\equiv T_1}}$$

M3

$$\frac{C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S, C_2 \triangleleft \{(\xmapsto{K_{u_1}} U_1, \{H_{\xmapsto{K_{u_1}} U_1}\}_{K_s^{-1}}), IP_{u_1}\}_{K_{sc_2}}}{C_2 \mid\equiv S \mid\sim \xmapsto{K_{u_1}} U_1, IP_{u_1}}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C_2 \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S), C_2 \mid\equiv S \mid\sim \xmapsto{K_{u_1}} U_1, IP_{u_1}}{\boxed{C_2 \mid\equiv S \mid\equiv \xmapsto{K_{u_1}} U_1, IP_{u_1}}}$$

M4

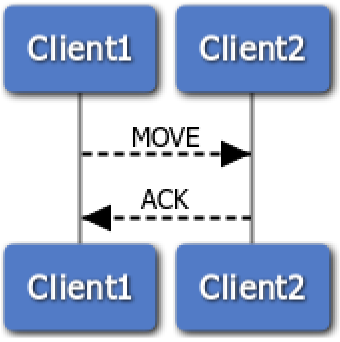
$$\frac{S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S, S \triangleleft \{T_2, H_{all}\}_{K_{sc_2}}}{S \mid\equiv C_2 \mid\sim T_2}$$

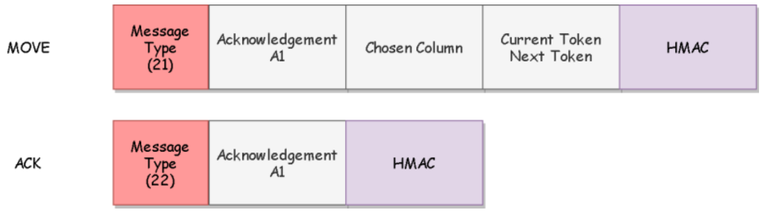
The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has

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

$$\frac{S \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S), S \mid\equiv C_2 \mid\sim T_2}{\boxed{S \mid\equiv C_2 \mid\equiv T_2}}$$

Move Protocol





BAN Logic Analysis

Real Protocol

$$M_1 \quad C_1 \rightarrow C_2 : \; M, A, \{(C, CT, NT, H_{all}\}_{K_{c_1c_2}}$$

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

Ideal Protocol

$$M_1 \quad C_1 \rightarrow C_2 : \; \#(NT), \{(C, CT, NT, H\}_{K_{c_1c_2}}$$

$$M_2 \quad C_2 \rightarrow C_1 : \; \{H_{all}\}_{K_{c_1c_2}}$$

Goals

$$C_2 \mid\equiv C_1 \mid\equiv \{(C, CT, NT\}$$

$$C_1 \mid\equiv C_2 \mid\equiv \{H_{all}\}$$

Assumptions

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

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

Analysis

M1

$$\frac{C_2 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2, C_2 \triangleleft \{(C, CT, NT, H\}_{K_{c_1c_2}}}{C_2 \mid\equiv C1 \mid\sim \{(C, CT, NT\}}$$

The clientB has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the clientB believes that the clientA has sent the message

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

$$\frac{C_2 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} C_2), C_2 \mid\equiv C_1 \mid\sim \{(C, CT, NT\}}{\boxed{C_2 \mid\equiv C_1 \mid\equiv \{(C, CT, NT, H\}}}$$

M2

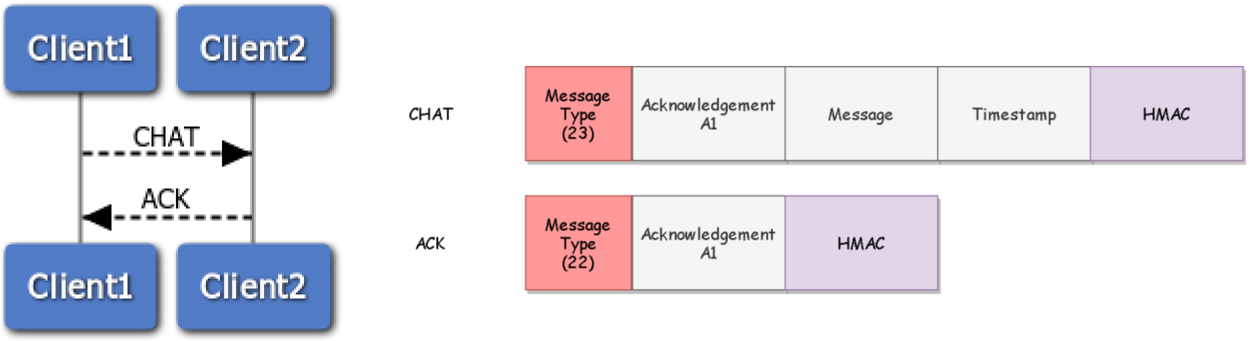
$$\frac{C_1 \mid\equiv C_1 \xleftrightarrow{K_{c_1c_2}} C_2, C_1 \triangleleft \{H_{all}\}_{K_{c_1c_2}}}{C_2 \mid\equiv C1 \mid\sim \{H_{all}\}}$$

The clientA has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the clientA believes that the clientB has sent the message

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

$$\frac{C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} C_2), C_1 \mid\equiv C_2 \mid\sim \{H_{all}\}}{\boxed{C_2 \mid\equiv C_1 \mid\equiv \{H_{all}\}}}$$

Chat Protocol



BAN Logic Analysis

Real Protocol

$$M_1 \quad C_1 \rightarrow C_2 : \; M, A, \{(C, T, H\}_{Kc_1c_2}$$
$$M_2 \quad C_2 \rightarrow C_1 : \; M, A, \{H_{all}\}_{K_{c_1c_2}}$$

Goals

$$C_2 \mid\equiv C_1 \mid\equiv \{(C, T\}$$
$$C_1 \mid\equiv C_2 \mid\equiv \{H_{all}\}$$

Ideal Protocol

$$M_1 \quad C_1 \rightarrow C_2 : \; \#(T), \{(C, T, H\}_{Kc_1c_2}$$
$$M_2 \quad C_2 \rightarrow C_1 : \; \{H_{all}\}_{K_{c_1c_2}}$$

Assumptions

$$C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad C \mid\equiv S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad C \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S)$$
$$S \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad S \mid\equiv C \mid\equiv C \overset{K_{cs}}{\longleftrightarrow} S \qquad S \mid\equiv \#(C \overset{K_{cs}}{\longleftrightarrow} S)$$

Analysis

M1

$$\frac{C_2 \mid\equiv C_1 \overset{K_{c_1c_2}}{\longleftrightarrow} C_2, C_2 \triangleleft \{(C, T, H\}_{K_{c_1c_2}}}{C_2 \mid\equiv C1 \mid\sim \{(C, T\}}$$

The clientB has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the clientB believes that the clientA has sent the message

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

$$\frac{C_2 \mid\equiv \#(C_1 \overset{K_{sc_1}}{\longleftrightarrow} C_2), C_2 \mid\equiv C_1 \mid\sim \{(C, T\}}{\boxed{C_2 \mid\equiv C_1 \mid\equiv \{(C, T\}}}$$

M2

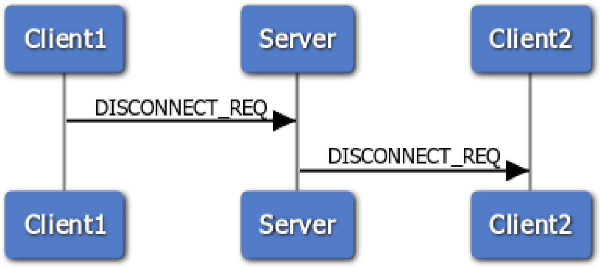
$$\frac{C_1 \mid\equiv C_1 \overset{K_{c_1c_2}}{\longleftrightarrow} C_2, C_1 \triangleleft \{H_{all}\}_{K_{c_1c_2}}}{C_1 \mid\equiv C_2 \mid\sim \{H_{all}\}}$$

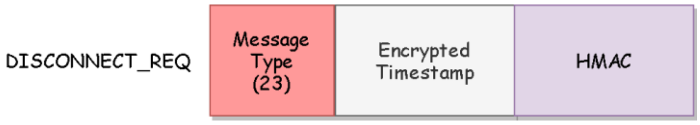
The clientB has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the clientA believes that the clientB has sent the message

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

$$\frac{C_1 \mid\equiv \#(C_1 \overset{K_{sc_1}}{\longleftrightarrow} C_2), C_1 \mid\equiv C_2 \mid\sim \{H_{all}\}}{\boxed{C_1 \mid\equiv C_2 \mid\equiv \{H_{all}\}}}$$

Disconnect Protocol





BAN Logic Analysis

Real Protocol

M_1	$C_1 \rightarrow S: \; M, \{(T_1, H_{all}\}_{K}sc_1$
M_2	$S \rightarrow C_2: \; M, \{(T_2, H_{all}\}_{K}sc_2$

Goals

$S \mid\equiv C_1 \mid\equiv \{(T_1, H_{all}\}$

$C_2 \mid\equiv S \mid\equiv \{(T_2, H_{all}\}$

Ideal Protocol

M_1	$C_1 \rightarrow S: \; \#(NT), \{(T_1, H_{all}\}_{K}sc_1$
M_2	$S \rightarrow C_2: \; \#(NT), \{(T_2, H_{all}\}_{K}sc_2$

Assumptions

$C_1 \mid\equiv C_1 \xleftrightarrow{K_{cs_1}} S$	$C_1 \mid\equiv S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S$	$C_1 \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S)$
$C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S$	$C_2 \mid\equiv S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S$	$C_2 \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S)$
$S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S$	$S \mid\equiv C_1 \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S$	$S \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S)$
$S \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S$	$S \mid\equiv C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S$	$S \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S)$

Analysis

M1

$$\frac{S \mid\equiv C_1 \xleftrightarrow{K_{sc_1}} S, \; S \triangleleft \{(T_1, H_{all}\}_{K_{sc_1}}}{S \mid\equiv C_1 \mid\sim \{(T_1, H_{all}\}}$$

The server has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the server believes that the client has sent the message

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

$$\frac{S \mid\equiv \#(C_1 \xleftrightarrow{K_{sc_1}} S), \; S \mid\equiv C_1 \mid\sim \{(T_1, H_{all}\}}{\boxed{S \mid\equiv C_1 \mid\equiv \{(T_1, H_{all}\}}}$$

M2

$$\frac{C_2 \mid\equiv C_2 \xleftrightarrow{K_{sc_2}} S, \; C_2 \triangleleft \{(T_2, H_{all}\}_{K_{sc_2}}}{C_2 \mid\equiv S \mid\sim \{(T_2, H_{all}\}}$$

The client has received a message containing an HMAC encrypted by the shared public key. We can apply the **second message meaning postulate** to derive that the client believes that the server has sent the message

The received HMAC contains a fresh element, 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

$$\frac{C_2 \mid\equiv \#(C_2 \xleftrightarrow{K_{sc_2}} S), \; C_2 \mid\equiv S \mid\sim \{(T_2, H_{all}\}}{\boxed{C_2 \mid\equiv S \mid\equiv \{(T_2, H_{all}\}}}$$

UML Diagram