Connect Four Secure:
Project report

Leonardo Cecchelli
Zaccaria Essaid
Filippo Guggino

September 2020

Abstract

The objective of this project is to create a secure online implementation of the popular table game Connect Four. In particular, the main requirement was to guarantee that the communication must be confidential, authenticated, and protected against replay.

1 Introduction

The application we developed is an online secure version of the famous table game called Connect Four. We will now give a brief description of what a new player can do once he has launched the application. The player needs to go through different phases to play with another user. The first one is the authentication phase, during this one the player needs to be already registered on the server otherwise access will not be granted. In particular, the user needs to authenticate himself by using public keys authentication. Once the user has successfully logged, he is redirected to a menu where he can see all the other online users that are available for a match. Players that are already playing with other users will not be shown until they end the match session. From this menu the user can then access to the second phase, in which he can send a challenge request to another online player. During this phase, the server act as an intermediate between the two. If the opponent player accepts the challenge, the server sends the needed information to the challenger, this way the two opponents can establish a connection. Once the connection is established the players authenticate each other following a similar protocol used for the server authentication and after that they will be able to play. After the end of the match session a user will be reconnected to the server. After that he can either send a new challenge or disconnect from the server and close the application. In the next paragraph we will describe in detail each protocol used in the different phases of the game.

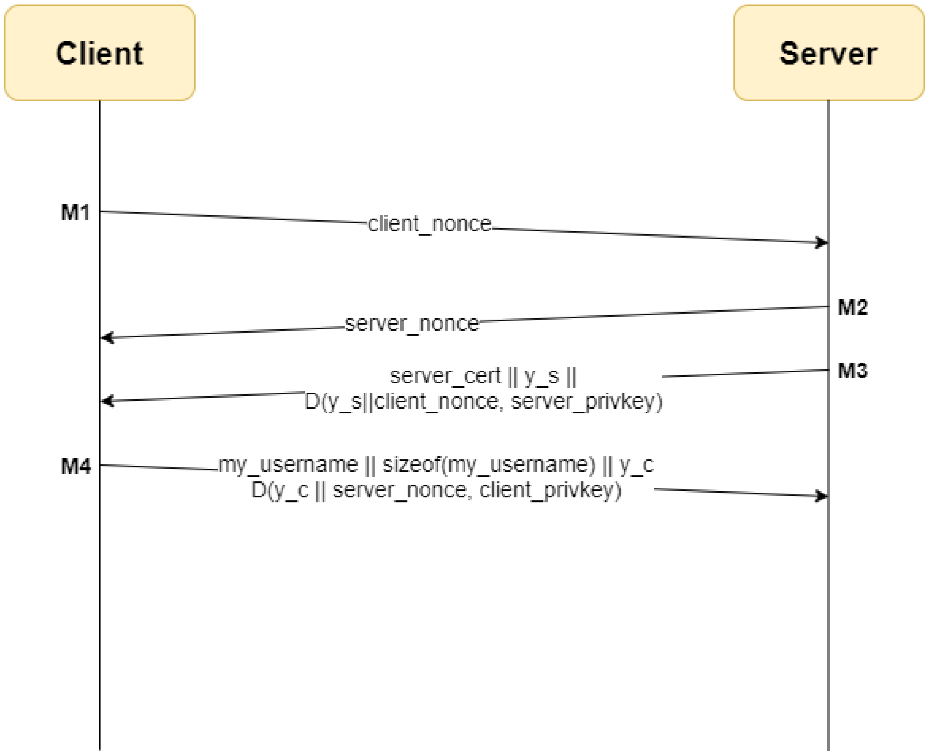
2 Protocols implementation

We will analyze the different protocols used in each phase starting from the authentication phase and ending with the disconnection phase.

2.1 Server authentication protocol

In this phase the authentication is done through the public key authentication. In particular, the server needs to have all the public keys of the users that are registered stored. The public keys are saved inside of *.pem* files and each one is protected with a password.

Let us now have a look to the scheme of the protocol:

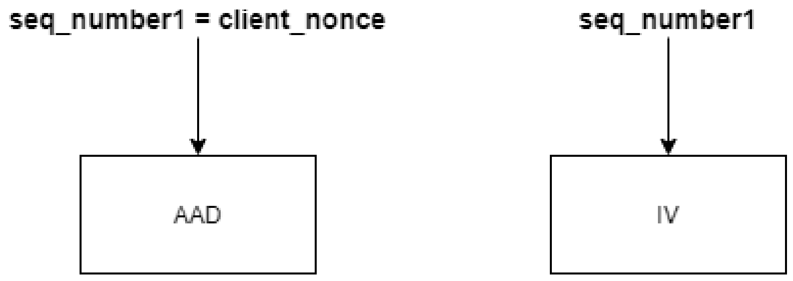

Client
Server
M1
client_nonce
M2
server_nonce
M3
server_cert || y_s ||
D(y_s||client_nonce, server_privkey)
M4
my_username || sizeof(my_username) || y_c
D(y_c || server_nonce, client_privkey)

With the first message M1 the client initiates the authentication, actually `client_nonce` is a randomly generated number. The server saves this information and replies with the message M2 that contains his own randomly generated nonce called `server_nonce`. Both M1 and M2 are send in clear, so without any type of encryption. Then the server send a new message M3 containing his own certificate (`server_cert`), the Diffie-Hellman public key (this will be later used to create a session key with the client but we will discuss about it in detail in the following paragraph) that we called `y_s` and finally the digital signature of `y_s` and `client_nonce`, in order to prove the authenticity of the data. When the client receives the message M3, he verifies the certificate and the digital signature, then he sends the message M4 containing his username (so the server knows which is the user that is logging in), the size of the username, his Diffie-Hellman public key `y_s` that he has generated and his digital signature of `y_s` and `server_nonce`.

2.2 Server session key generation protocol

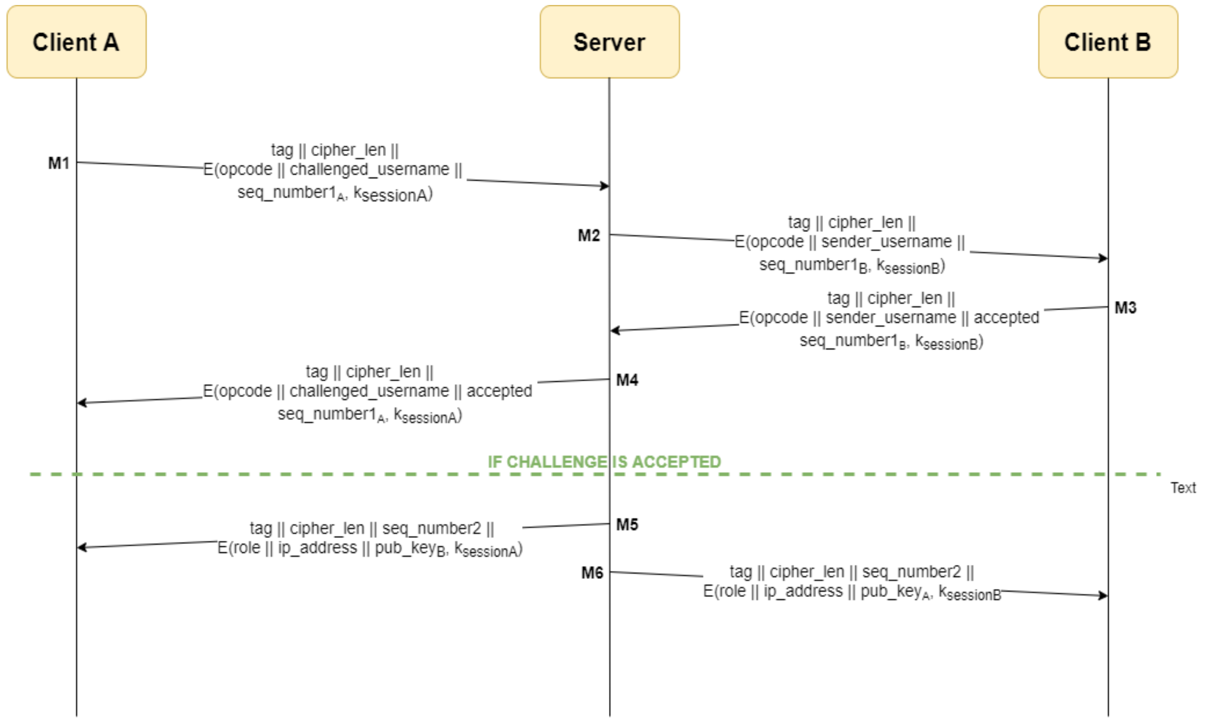
After the authentication phase server and client need to communicate using a symmetric encryption algorithm, in the specific we used the AES with Galois/Counter mode. In order to make client and server able to establish a session key we implement the Diffie-Hellman protocol. We have already seen that during the authentication phase both server and client generate and exchange their Diffie-Hellman public keys. The reason why this part is done during the authentication phase is to avoid the man in the middle attack: since plain Diffie-Hellman suffer from MIM attack we used the digital signature in an authenticated channel to avoid that and the nonce protection to avoid replay attack. Thanks to both elements an attacker that is able to intercept the messages containing the public keys cannot complete the MIM attack since he is not able to replicate the client and the server own digital signatures, nor he can replicate messages exchanged during previous sessions.

Note that from now on messages between server and client are encrypted using the server session key and client nonce is used as IV and seq_number1 as AAD for the GCM encryption. In the specific what both server and clients is assign `client_nonce` value to `seq_number1` and use them as following:



2.3 Challenge request protocol

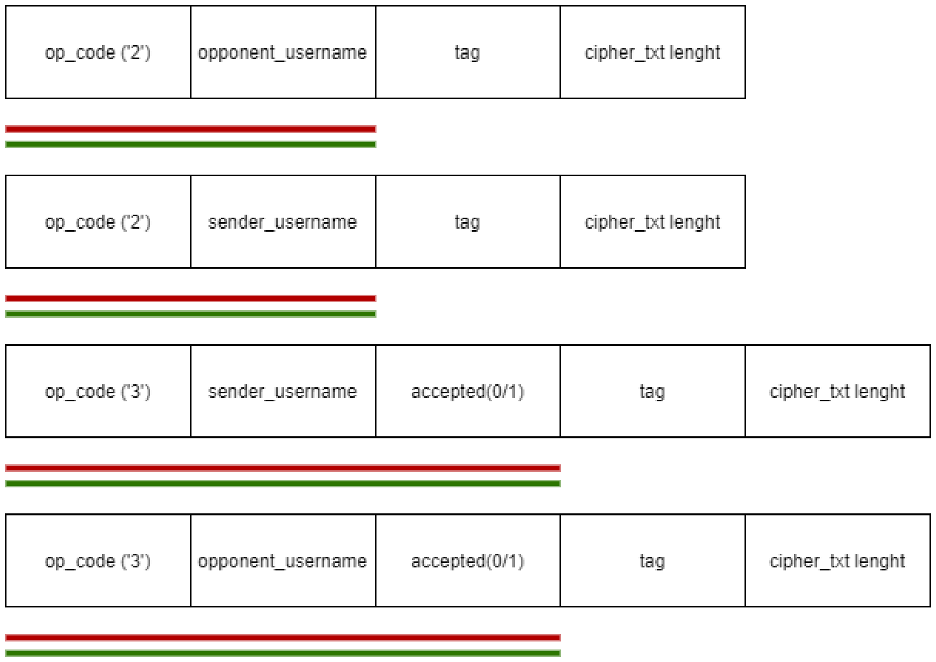
When the user has successfully logged in, he is able to send a challenge request to another user in order to play a match. Since two online users are not able to communicate directly, the server acts as an intermediate.



Client A
Server
Client B
M1
tag || cipher_len || E(opcode || challenged_username || seq_number1A, ksessionA)
M2
tag || cipher_len || E(opcode || sender_username || seq_number1B, ksessionB)
M3
tag || cipher_len || E(opcode || sender_username || accepted seq_number1B, ksessionB)
M4
tag || cipher_len || E(opcode || challenged_username || accepted seq_number1A, ksessionA)
IF CHALLENGE IS ACCEPTED
Text
M5
tag || cipher_len || seq_number2 || E(role || ip_address || pub_keyB, ksessionA)
M6
tag || cipher_len || seq_number2 || E(role || ip_address || pub_keyA, ksessionB

In the scheme we can see a user A that sends to user B a challenge request. As we have said previously both users need to be online and not inside of a match. Client A first sends a tag to authenticate the subsequent message and the ciphertext length; this last information is needed to handle messages with sockets. Client A sends a message M1 encrypted with the session key generated during the previous phase, in particular inside of the cipher text we can find the opcode (used to identify the nature of the message), the username of the player that A wants to challenge (B in this case) and the server nonce (with A) to protect against replay attacks. The message M1 is received by the server which substantially is in charge of decrypting and forwarding it to the challenged user after having done some changes. Basically the server substitute the username of the challenged user (`challenged_username`) with the name of the user that is sending the request (A in this case) and change the nonce with the client nonce of B, then the server encrypts the message with the session key established with B. At this point B receives the message M2 and he is able to decide whether to accept the challenge or to refuse it. Once he has made the choice, B crafts a new message M3 containing a new opcode (3 in this case), the `sender_username`, a boolean value 0/1 that indicates wheter he has accepted or not the challenge and the server nonce; then encrypts it with his session key. The server receives M3 and acts as for M2, it changes `sender_username` with `challenged_username`, `server_nonce`$_B$ with `client_nonce`$_A$ and encrypts it with the session key established with A. At this point if B has accepted the challenge the server addional informations to both clients, otherwise A is prompted with a message saying that B is not available to play at the moment.

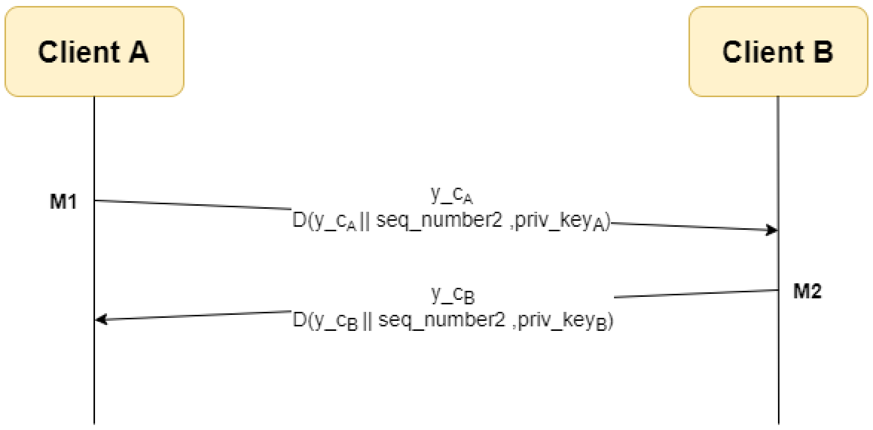
Below we can see the format of the messages we have described so far:



With the red line we highlight the part of the message that has been encrypted and with the green line the one that has been authenticated with `tag`.

2.4 Client – Client authentication protocol

Once each client has received the information from the server (messages M5 and M6 of the previous protocol), they start the authentication phase.

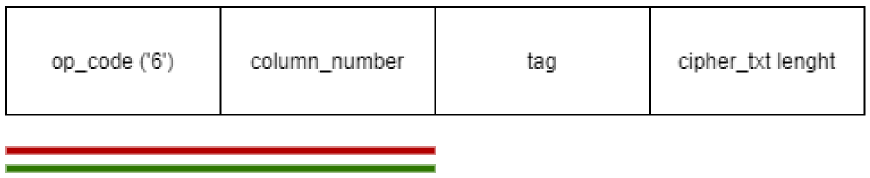

Client A
Client B
M1
y_c_A
D(y_c_A || seq_number2 ,priv_key_A)
y_c_B
D(y_c_B || seq_number2 ,priv_key_B)
M2

As we can see from the scheme Client A (which is the sender of the request) is the one in charge to start the authentication phase. He sends a message M1 that includes his generated Diffie-Hellman public key and the digital signature of that key concatenated with the `seq_number2` (used as a nonce) that the server has sent to A and B in the previous phase. After Client B has received the message, he verifies the digital signature of Client A he then sends the same message to B but with his Diffie-Hellman public key. Finally, Client A verifies the digital signature and both Clients can generate the session key for the match session.

2.5 Match protocol

During a match the two clients exchange messages in order to communicate each other the move that they want to do. This message are encrypted with the session key established in the client-client authentication phase and authenticated with a tag (basically it is the same mechanism we used for each session message sent between clients and between client and server).

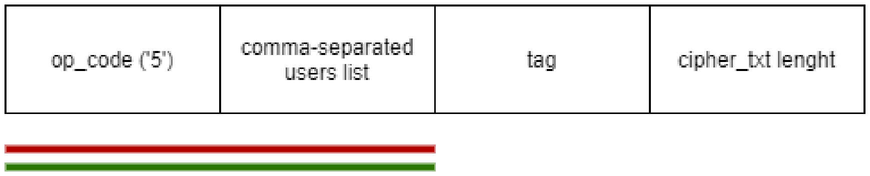
The format of the message is the following:



2.6 User-list update mechanism

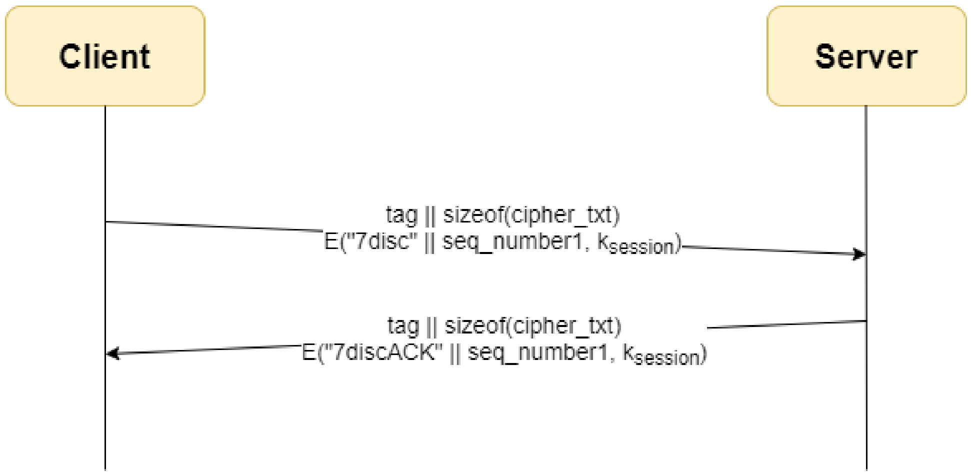
As we have said when a user is not playing a match, he can see a list of all online users that are not currently playing. So, the server has to send a list of all available players to all online users. This is done by sending a special message with opcode 5 that contains a comma-separated user list. The message is of course encrypted for each client with the corresponding session key and it is authenticated with a tag (that also prevents replay attacks by using the `seq_number1`).

Below we can see the format of this user-list update message:

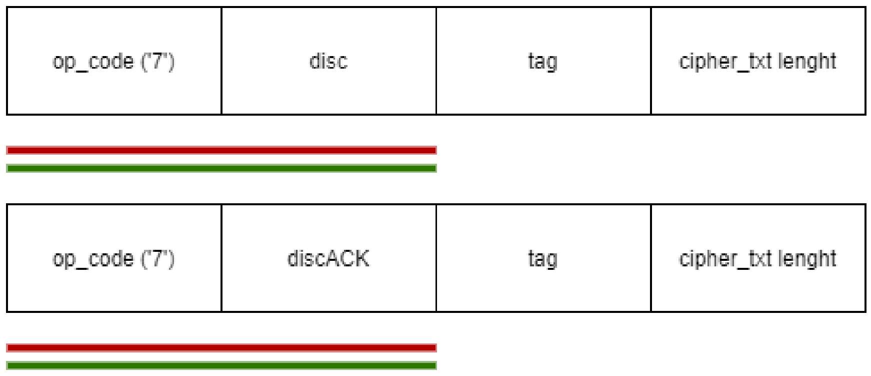


2.7 Disconnection protocol

Since the server keeps the list of all connected users inside of a special data structure, it is important that a client, who wants to disconnect, notifies the server. So, a simple disconnection protocol is important in order to keep the online user list in a consistent state.


Client
Server
tag || sizeof(cipher_txt)
E("7disc" || seq_number1, k_session)
tag || sizeof(cipher_txt)
E("7discACK" || seq_number1, k_session)

As we can see from the scheme the protocol is made by two messages, the first one being a disconnection request (sent by the client) and the second a disconnection ACK (sent by the server). The format of the first one is simply the opcode 7 together with the keyword “`disc`”. The message is encrypted and authenticated so there is no possibility of a replay attack. The server receives the message, checks the opcode and the keyword and updates the online user list. After that the server sends the ACK message (composed by opcode 7 and the keyword `discACK`) and closes the connection. At the end the client receives the ACK, checks the message and closes the connection with the server.



3 BAN logic proof

3.1 SERVER-CLIENT authentication

Real protocol:

$M.1 \quad C \to S: \quad N_C$

$M.2 \quad S \to C: \quad N_S$

$M.3 \quad S \to C: \quad sizeof(cert) \mid\mid cert \mid\mid Y_S \mid\mid D(Y_S \mid\mid N_C, K_S^{-1})$

$M.4 \quad C \to S: sizeof(username) \mid\mid username \mid\mid Y_C \mid\mid D(Y_C \mid\mid N_S, K_C^{-1})$

Assumptions

A1.$C \mid\equiv \#\{N_C\}$

A2.$C \mid\equiv \overset{K_{CA}}{\to} CA$

A3.$C \mid\equiv CA \Rightarrow \overset{K_S}{\to} S$

A4.$S \mid\equiv \#\{N_S\}$

A5.$S \mid\equiv \overset{K_C}{\to} C$

Idealized Protocol

$M.3 \quad S \to C: \quad \{\overset{K_S}{\to} S\}_{K_{CA}^{-1}}, \ \{N_C, Y_S, \#\{Y_S\}\}_{K_S^{-1}}$

After receiving Nc, S said that Ks is its public key and Ys is its DH public key and that it's fresh. #Ys freshness is implicit due to the presence of Nc in the message.

$M.4 \quad C \to S: \quad \{N_S, Y_C, \#\{Y_C\}\}_{K_S^{-1}}$

After receiving Ns, C said that Yc is its DH public key and that it's fresh. #Yc freshness is implicit due to the presence of Ns in the message.

Annotated protocol

$P.1 \quad C \triangleleft \{\overset{K_S}{\rightarrow} S\}_{K_{CA}^{-1}}$

$P.2 \quad C \triangleleft \{N_C, Y_S, \#\{Y_S\}\}_{K_S^{-1}}$

$P.3 \quad S \triangleleft \{N_S, Y_C, \#\{Y_C\}\}_{K_C^{-1}}$

Derivations

D1	$\dfrac{C \mid\equiv \overset{K_{CA}}{\rightarrow} CA, C \triangleleft \{\overset{K_S}{\rightarrow} S\}_{K_{CA}^{-1}}}{C \mid\equiv CA \mid\sim \overset{K_S}{\rightarrow} S}$	by message meaning applying (A2) and (P1)
D1.1	$\dfrac{C \mid\equiv CA \mid\sim \overset{K_S}{\rightarrow} S}{C \mid\equiv CA \mid\equiv \overset{K_S}{\rightarrow} S}$	by nonce verification applying (D1) (We imply that Ks is a valid key)
D2	$\dfrac{C \mid\equiv CA \mid\equiv \overset{K_S}{\rightarrow} S, C \mid\equiv CA \Rightarrow \overset{K_S}{\rightarrow} S}{C \mid\equiv \overset{K_S}{\rightarrow} S}$	by jurisdiction rule applying (A3) and (D1.1)
D3	$\dfrac{C \mid\equiv \overset{K_S}{\rightarrow} S, C \triangleleft \{N_C, Y_S, \#\{Y_S\}\}_{K_S^{-1}}}{C \mid\equiv S \mid\sim \{N_C, Y_S, \#\{Y_S\}\}}$	by message meaning rule applying (D2) and (P2)
D4	$\dfrac{C \mid\equiv \#\{N_C, Y_S, \#\{Y_S\}\}, C \mid\equiv S \mid\sim \{N_C, Y_S, \#\{Y_S\}\}}{C \mid\equiv S \mid\equiv \{N_C, Y_S, \#\{Y_S\}\}}$	by nonce verification rule applying (A1) and (D3)

Same goes with respect to the Server since the key establishment protocol is perfectly symmetrical.

D5. $S \mid\equiv C \mid\sim \{N_S, Y_C, \#\{Y_C\}\}$ by message meaning rule applying (A5) and (P3)

D6. $S \mid\equiv C \mid\equiv \{N_S, Y_C, \#\{Y_C\}\}$ by nonce verification rule applying (A4) and (D5)

Goal

$C \mid\equiv S \mid\equiv Y_S$

$C \mid\equiv S \mid\equiv \#\{Y_S\}$

$S \mid\equiv C \mid\equiv Y_C$

$S \mid\equiv C \mid\equiv \#\{Y_C\}$

$C \mid\equiv \overset{K_S}{\rightarrow} S$

3.2 CLIENT-CLIENT authentication

Real protocol:

$M.1 \quad S \rightarrow C_0 : \quad E(K_{C1} || N,\ K_{S-C_0}), MAC(E(.), N_0)$

$M.2 \quad S \rightarrow C_1 : \quad E(K_{C0} || N,\ K_{S-C_1}), MAC(E(.), N_1)$

$M.3 \quad C_0 \rightarrow C_1 : \quad Y_0 \mid\mid D(Y_0 \mid\mid N,\ K_{C_0}^{-1})$

$M.4 \quad C_1 \rightarrow C_0 : \quad Y_1 \mid\mid D(Y_1 \mid\mid N,\ K_{C_1}^{-1})$

Assumptions:

A1. $C_0 \mid\equiv C_0 \overset{K_{S-C_0}}{\leftrightarrow} S$

A2. $C_1 \mid\equiv C_1 \overset{K_{S-C_1}}{\leftrightarrow} S$

A3. $C_0 \mid\equiv \#\{N_0\}$

A4. $C_1 \mid\equiv \#\{N_1\}$

A5. $C_0 \mid\equiv S \Rightarrow \overset{K_{C_1}}{\rightarrow} C_1$

A6. $C_1 \mid\equiv S \Rightarrow \overset{K_{C_0}}{\rightarrow} C_0$

Idealized Protocol:

$$M.1 \qquad S \to C_0 : \qquad \{\overset{K_{C_1}}{\to} C_1, N, \#\{N\}\}_{K_{S-C_0}}$$

S said that Kc1 is C1’s public key.

$$M.2 \qquad S \to C_1 : \qquad \{\overset{K_{C_0}}{\to} C_0, N, \#\{N\}\}_{K_{S-C_1}}$$

S said that Kc0 is C0’s public key.

$$M.3 \qquad C_0 \to C_1 : \qquad \{Y_0, \#\{Y_0\}, N\}_{K_{C_0}^{-1}}$$

C0 said that Y0 is its DH public key and that it’s fresh. Y0’s freshness is implicit due to the presence of N in the message.

$$M.4 \qquad C_1 \to C_0 : \qquad \{Y_1, \#\{Y_1\}, N\}_{K_{C_1}^{-1}}$$

(#Y1 is implicit)

C1 said that Y1 is its DH public key and that it’s fresh. Y1’s freshness is implicit due to the presence of N in the message.

Derivations

D1	$\dfrac{C_0 \mid\equiv S \overset{K_{S-C_0}}{\leftrightarrow} C_0, C_0 \triangleleft \{\overset{K_{C_1}}{\to} C_1, N, \#\{N\}\}_{K_{S-C_0}}}{C_0 \mid\equiv S \mid\sim \{\overset{K_{C_1}}{\to} C_1, N, \#\{N\}\}}$	by message meaning applying (A1) and (M1)
D2	$\dfrac{C_0 \mid\equiv \#\{\overset{K_{C_1}}{\to} C_1, N, \#\{N\}\}, C_0 \mid\equiv S \mid\sim \{\overset{K_{C_1}}{\to} C_1, N, \#\{N\}\}}{C_0 \mid\equiv S \mid\equiv \{\overset{K_{C_1}}{\to} C_1, N, \#\{N\}\}}$	by nonce verification rule applying (A3) and (D1)

D3	$\dfrac{C_0 \mid\equiv S \mid\equiv \{\overset{K_{C_1}}{\rightarrow} C_1, N, \#\{N\}\}}{C_0 \mid\equiv \{\overset{K_{C_1}}{\rightarrow}, N, \#\{N\}}$	since S is an authenticated authority over X then C believes X
D4	$\dfrac{C_0 \mid\equiv \overset{K_{C_1}}{\rightarrow} C_1, C_0 \triangleleft \{Y_1, \#\{Y_1\}, N\}_{K_{C_1}^{-1}}}{C_0 \mid\equiv C_1 \mid\sim \{Y_1, \#\{Y_1\}, N\}}$	by message meaning applying (M4) and (D3)
D5	$\dfrac{C_0 \mid\equiv \#\{Y_1, \#\{Y_1\}, N\}, C_0 \mid\equiv C_1 \mid\sim \{Y_1, \#\{Y_1\}, N\}}{C_0 \mid\equiv C_1 \mid\equiv \{Y_1, \#\{Y_1\}, N\}}$	by nonce verification applying (D4)

Same goes with respect to the Server since the key enstablishment protocol is perfectly symmetrical.

D6. $C_1 \mid\equiv S \mid\sim \{\overset{K_{C_0}}{\rightarrow} C_0\}$ by message meaning applying (A2) and (M2)

D7. $C_1 \mid\equiv S \mid\equiv \{\overset{K_{C_0}}{\rightarrow} C_0\}$ by nonce verification rule applying (a.4) and (D6)

D8. $C_1 \mid\equiv \{\overset{K_{C_0}}{\rightarrow} C_0\}$ by jurisdiction rule applying (A6) and (D7)

D9. $C_1 \mid\equiv C_0 \mid\sim \{Y_0, \#\{Y_0\}, N\}$ by message meaning applying (M3) and (D8)

D10. $C_1 \mid\equiv C_0 \mid\equiv \{Y_0, \#\{Y_0\}, N\}$ by nonce verification applying (D9)

Goal

$C_0 \mid\equiv C_1 \mid\equiv Y_1$

$C_0 \mid\equiv C_1 \mid\equiv \#\{Y_1\}$

$C_1 \mid\equiv C_0 \mid\equiv Y_0$

$C_1 \mid\equiv C_0 \mid\equiv \#\{Y_0\}$

$$C_0 \models\overset{\kappa_{C_1}}{\rightarrow} C_1$$

$$C_1 \models\overset{\kappa_{C_0}}{\rightarrow} C_0$$