UNIVERSITÀ DEGLI STUDI DI PISA

School of Engineering
Master of Science in Computer Engineering
Foundation of Cybersecurity

PROJECT DOCUMENTATION

SECURE FOUR IN A ROW

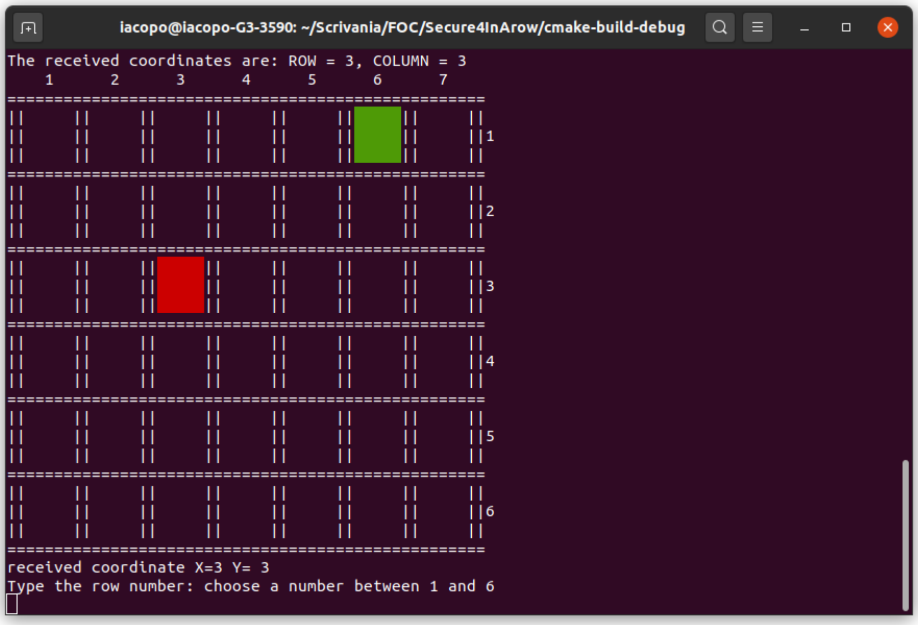
WORKGROUP:
Laura Lemmi
Iacopo Pacini

ACADEMIC YEAR 2019/2020

Index

1. DESCRIPTION

In this document is presented a simple implementation of the game “Four-in-a-Row”, in this implementation the focus is not the user experience but the security of the overall communications. Especially when the client application starts, Server and Client
First authenticate with their public keys: in this scenario anyone who wants to play must ask first to the server administrator to register his public key, and only after that he(or she) can use it to authenticate himself. Server public key is certified by SimpleAuthority CA, for this reason every player has the burden to check the validity of said public key.
Once done all the verifications, those keys are used together with a 2048-bits DH public key to compute a shared secret hence a symmetric key to encrypt and authenticate all the exchanged messages.
After the authentication and negotiation phases a newly joined user can see other available users logged to the server and he (or she) can challenge one of them, the latter can either accept or refuse. If the challenge is accepted, a secure channel is being established by means of a protocol pretty similar to C/S one. The users proceed to play using a peer-to-peer communication and for each move a 6X7 grid is printed to terminal with all the moves done until that moment.



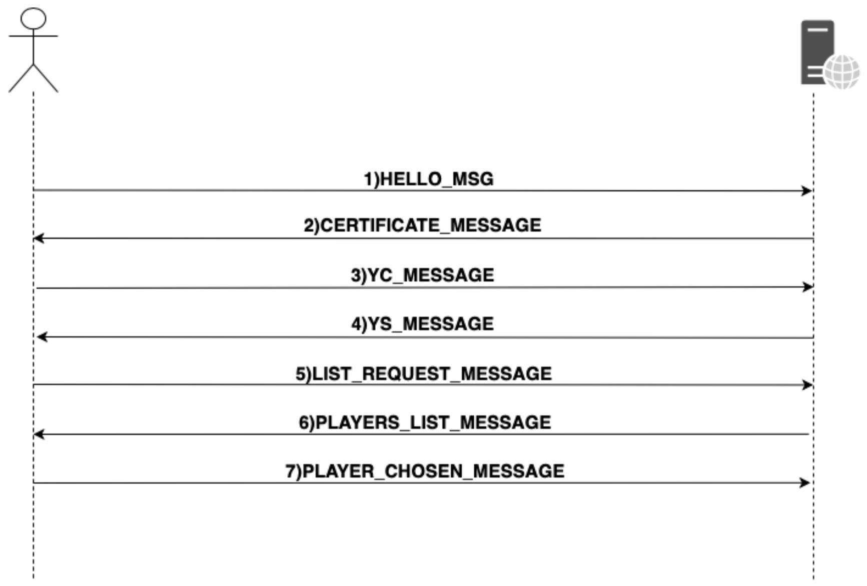
```
iacopo@iacopo-G3-3590: ~/Scrivania/FOC/Secure4InArow/cmake-build-debug
The received coordinates are: ROW = 3, COLUMN = 3
    1      2      3      4      5      6      7
===================================================
||     ||     ||     ||     ||     ||🟩🟩 ||     ||
||     ||     ||     ||     ||     ||🟩🟩 ||     ||1
||     ||     ||     ||     ||     ||🟩🟩 ||     ||
===================================================
||     ||     ||     ||     ||     ||     ||     ||
||     ||     ||     ||     ||     ||     ||     ||2
||     ||     ||     ||     ||     ||     ||     ||
===================================================
||     ||     ||🟥🟥 ||     ||     ||     ||     ||
||     ||     ||🟥🟥 ||     ||     ||     ||     ||3
||     ||     ||🟥🟥 ||     ||     ||     ||     ||
===================================================
||     ||     ||     ||     ||     ||     ||     ||
||     ||     ||     ||     ||     ||     ||     ||4
||     ||     ||     ||     ||     ||     ||     ||
===================================================
||     ||     ||     ||     ||     ||     ||     ||
||     ||     ||     ||     ||     ||     ||     ||5
||     ||     ||     ||     ||     ||     ||     ||
===================================================
||     ||     ||     ||     ||     ||     ||     ||
||     ||     ||     ||     ||     ||     ||     ||6
||     ||     ||     ||     ||     ||     ||     ||
===================================================
received coordinate X=3 Y= 3
Type the row number: choose a number between 1 and 6
```

Figure 1 Game grid with two moves form the player (green square) and one from the opponent (red square)

A player can play every game he wants, but every time a different key is being negotiated for security purposes.

2. PROTOCOL DESCRIPTION

2.1 C/S Secure Channel Establishment



1) The client sends a Hello Message in clear containing:
    - opcode;
    - username;
    - a 32-bits nonce.

2) The server replies with a Certificate Message in cleartext containing:
    - opcode;
    - its certificate released by a Certification Authority;
    - a 32-bits nonce.

3) The client sends an YC_Message containing:
    - opcode;
    - both nonces;
    - its Diffie Hellmann public key;
    - a 2048-bits RSA signature of the previous fields.

4) The server replies with the same kind of message containing its signature.

After verifying signatures and nonces, both parties are able to compute the shared secret and to obtain a session key by using the 128 least significant bits of the shared secret SHA256 digest.

Now on a secure channel is established:

- the messages are encrypted with a 128-bits AES-GCM algorithm and the integrity is ensured by means of a 256-bits tag;
- the freshness of each message is guaranteed by means of a client-server synchronous counter.

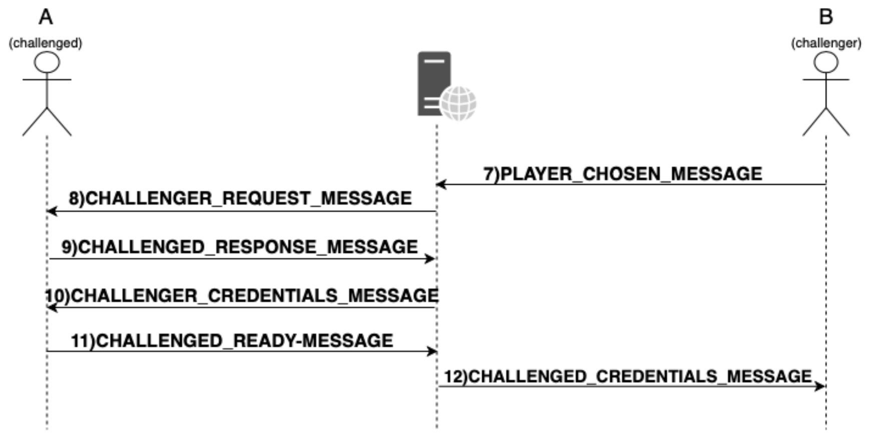
Once a secure channel is established, the player can start playing, namely:

5) The client sends a Players List Request message containing:
    - opcode, IV, counter authenticated by means of 256-bits tag;
    - encrypted payload containing its username;
    - AES-GCM tag.

6) The server replies by sending a Players List message containing:
    - additional authenticated data (opcode, IV, counter);
    - encrypted payload containing the username' list of the available players;
    - AES-GCM tag.

One the client has received this last message, it can either choose an adversary (if there is one) or decide to wait for a challenge request.

2.2 *C/S Exchange Player List and Credentials*


A
(challenged)
B
(challenger)
7)PLAYER_CHOSEN_MESSAGE
8)CHALLENGER_REQUEST_MESSAGE
9)CHALLENGED_RESPONSE_MESSAGE
10)CHALLENGER_CREDENTIALS_MESSAGE
11)CHALLENGED_READY-MESSAGE
12)CHALLENGED_CREDENTIALS_MESSAGE

7) The player (challenger) who has chosen the peer who wants to play against (challenged), sends to the server a Player Chosen message composed of:
    - opcode, IV, counter authenticated by means of a 256-bits tag;
    - encrypted payload containing the chosen adversary username;
    - AES-GCM tag.

8) The server, after having decrypted and verified the message, reads the opponent username received and sends to it a Challenger Request message containing:
    - opcode, IV, counter authenticated by means of a 256-bits tag;
    - encrypted challenger username;
    - AES-GCM tag.

The encryption/decryption and authentication are done with different session keys depending on the considered client.

9) The client, received the challenger request, can decide either to accept or refuse it; in both cases it sends a Challenged Response message to the server so formed:
    - opcode, IV, counter authenticated by means of a 256-bits tag;
    - encrypted payload containing the response (Y or N) and the challenger username;
    - AES-GCM tag.

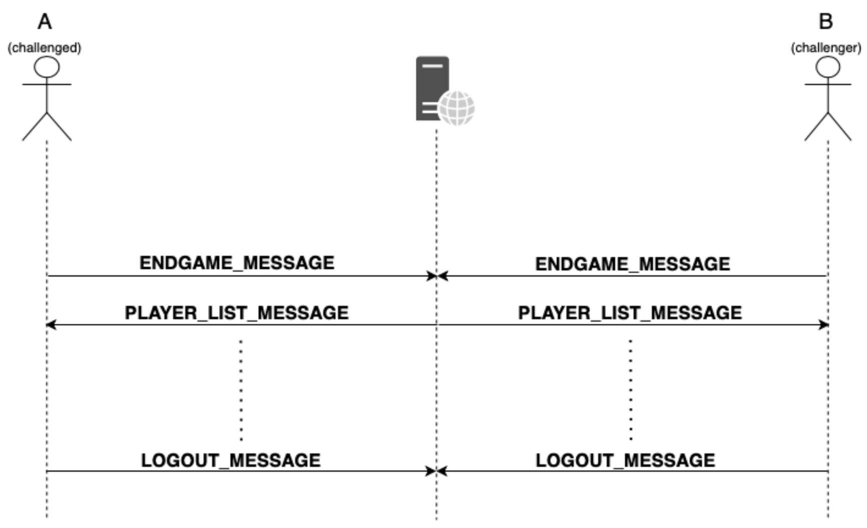
10) The server can now send two different messages depending on the response: in case of rejection it automatically sends again the Players List message, otherwise it sends a Challenger Credential Message composed by:
    - opcode, IV, counter authenticated by means of a 256-bits tag;
    - encrypted payload containing opponent IP, RSA public key and port (not used as explained later);
    - AES-GCM tag.

11) Once the challenged client is listening to new connection, it can send a Challenged Ready message to the server, composed by:
    - opcode, IV, counter authenticated by means of a 256-bits tag;

- encrypted payload containing: the port on which he is listening and its username;
- AES-GCM tag.

12) The server sends to challenger the Challenged Credentials Message which has the same structure of the Challenger Credential message but in this case the port will be used by the player to send a tcp connection request.

At this point the game can start; at the end of the game, or in case of errors other messages are exchanged between the clients and the server as shown below.



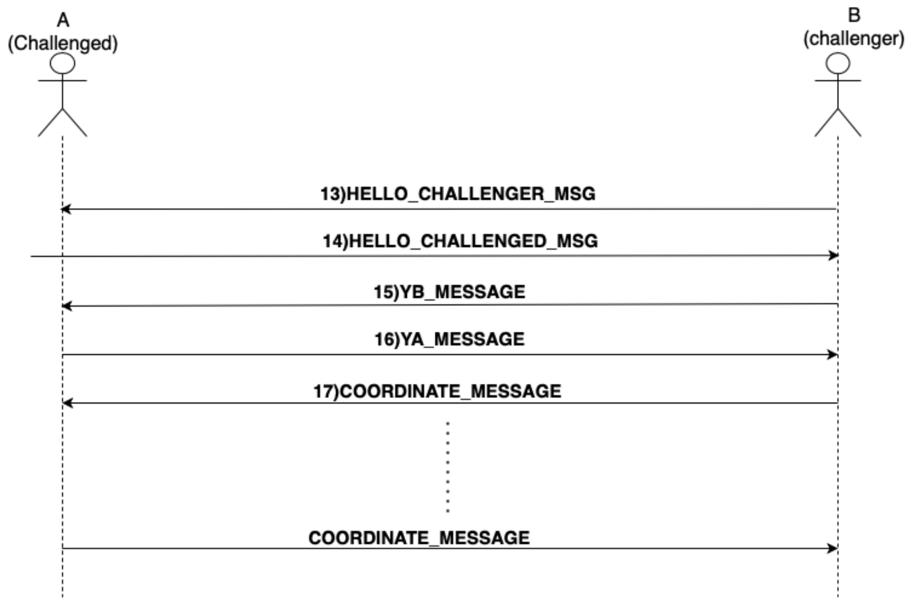
- The clients send an End Game message to the server when the challenge finishes or when some connection error occurs in order to be able to start a new match.
  The message is composed by:
  - opcode, IV, counter all authenticated by means of 256-bits tag;
  - encrypted payload containing the client username;
  - AES-GCM tag.

When the server receives an endgame message it sends back to the client the updated players list containing the players currently available (online and not engaged in connections with other clients).

- When the client wants to log out and so to stop playing it sends a Logout Message to the server, so formed:
  - opcode, IV, counter all authenticated by means of 256-bits tag;
  - encrypted payload containing the client username;
  - AES-GCM tag.

Logout and endgame messages can be distinguished from the opcode.

2.3 *C/S P2P Secure Channel Establishment*



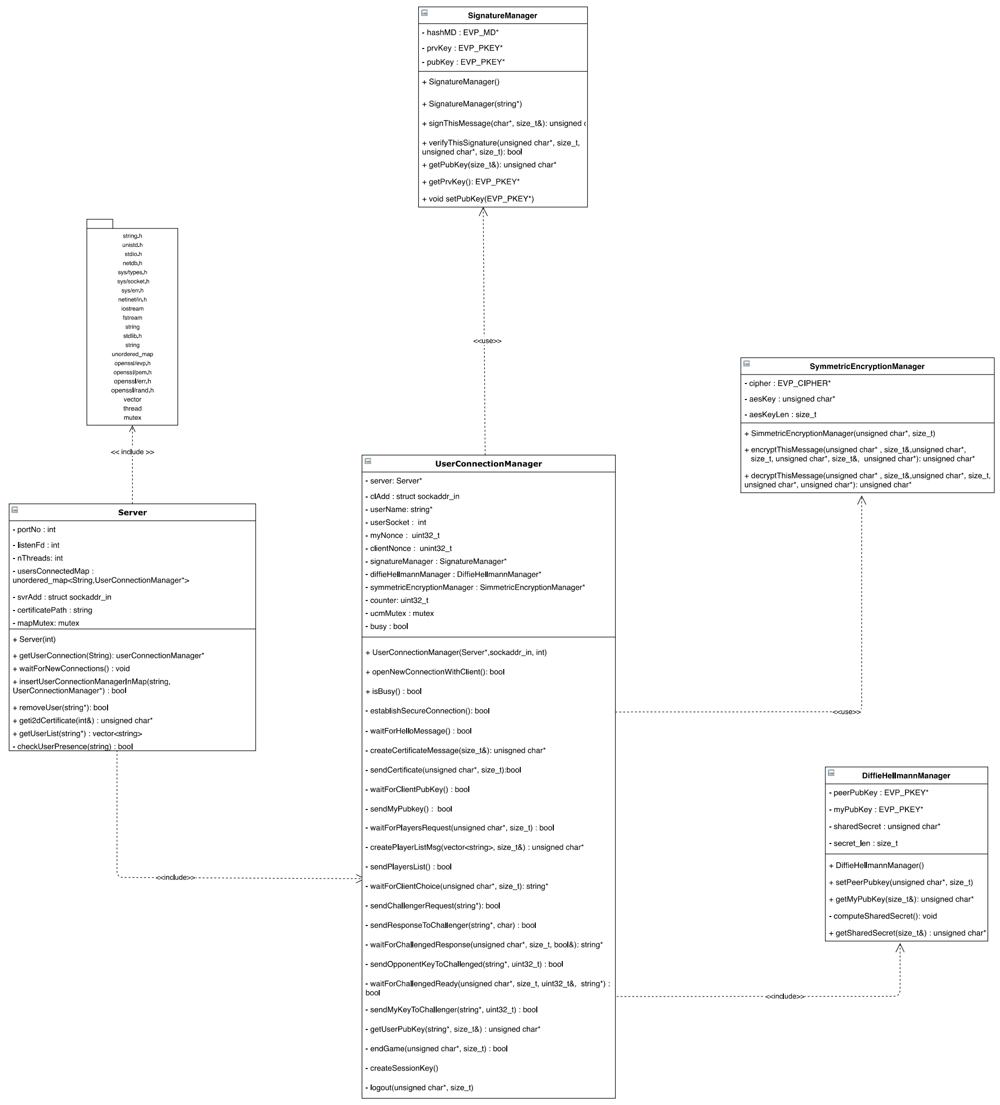
Once the server has sent to both clients their opponent credentials, clients themselves are able to establish a secure connection. The challenged player acts as a server and awaits a request of connection from the challenger. If the challenged accepts, a handshake for the creation of a secure channel begins. The handshake is pretty similar to the C/S one except from the fact that the RSA keys are supplied by the server itself to both opponents.
Peers can now start the game by exchanging the coordinates to be inserted in the game board. Only the new typed coordinate is sent through the channel, while the entire matrix is saved within each client.

- The message is formed as follow:
    - opcode, IV, counter all authenticated by means of 256-bits tag;
    - encrypted payload containing the row and column numbers;
    - AES-GCM tag.

If the typed coordinate is already present the game stops and a logout message is sent. At the end of the game each peer sends an End Game to the server in order to receive the new Players List.

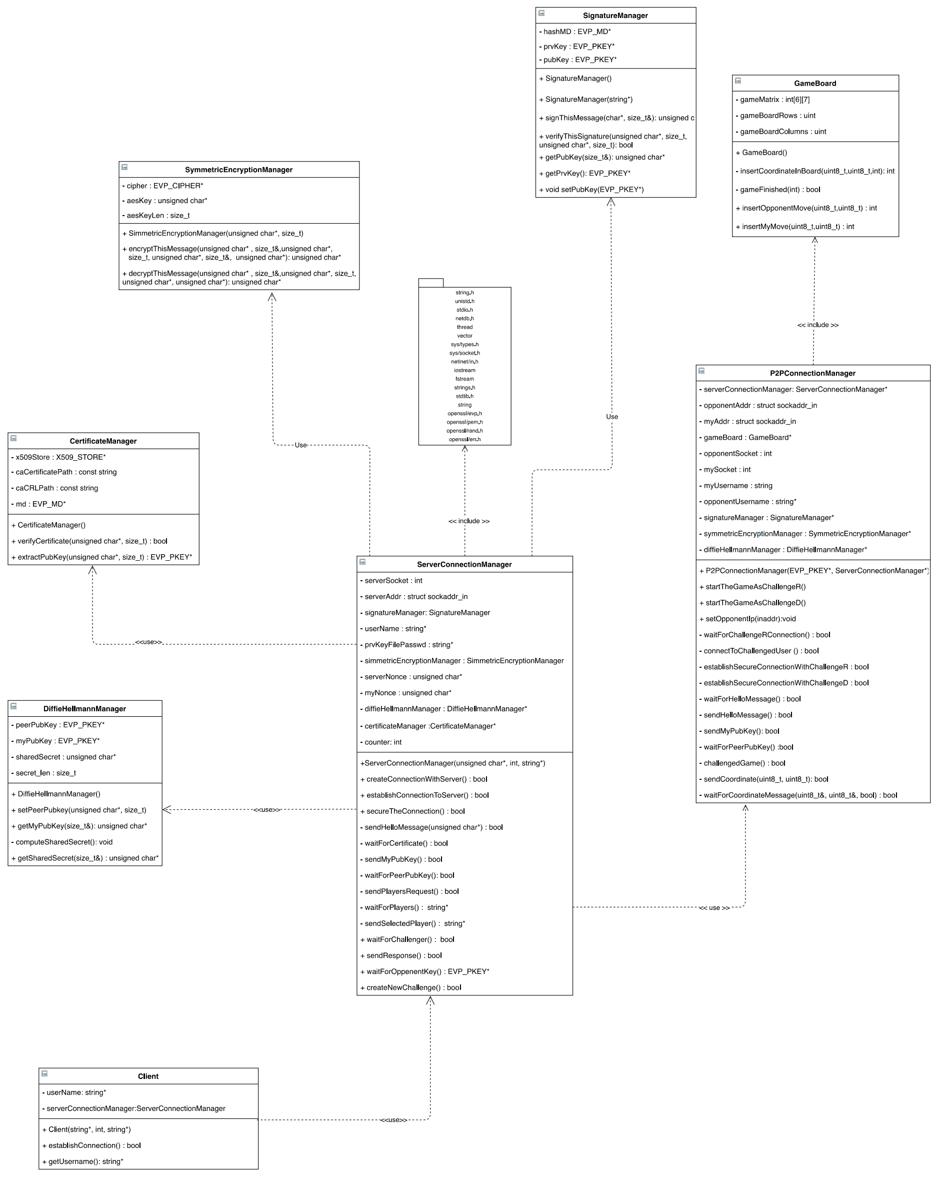
3. IMPLEMENTATION

3.1. Class Diagram


SignatureManager
- hashMD : EVP_MD*
- prvKey : EVP_PKEY*
- pubKey : EVP_PKEY*
+ SignatureManager()
+ SignatureManager(string*)
+ signThisMessage(char*, size_t&): unsigned c
+ verifyThisSignature(unsigned char*, size_t, unsigned char*, size_t): bool
+ getPubKey(size_t&): unsigned char*
+ getPrvKey(): EVP_PKEY*
+ void setPubKey(EVP_PKEY*)
<<use>>
string.h
unistd.h
stdio.h
netdb.h
sys/types.h
sys/socket.h
sys/err.h
netinet/in.h
iostream
fstream
string
stdlib.h
string
unordered_map
openssl/evp.h
openssl/pem.h
openssl/err.h
openssl/rand.h
vector
thread
mutex
<< include >>
Server
- portNo : int
- listenFd : int
- nThreads: int
- usersConnectedMap : unordered_map<String,UserConnectionManager*>
- svrAdd : struct sockaddr_in
- certificatePath : string
- mapMutex: mutex
+ Server(int)
+ getUserConnection(String): userConnectionManager*
+ waitForNewConnections() : void
+ insertUserConnectionManagerInMap(string, UserConnectionManager*) : bool
+ removeUser(string*): bool
+ geti2dCertificate(int&) : unsigned char*
+ getUserList(string*) : vector<string>
- checkUserPresence(string) : bool
<<include>>
SymmetricEncryptionManager
- cipher : EVP_CIPHER*
- aesKey : unsigned char*
- aesKeyLen : size_t
+ SimmetricEncryptionManager(unsigned char*, size_t)
+ encryptThisMessage(unsigned char* , size_t&,unsigned char*, size_t, unsigned char*, size_t&, unsigned char*): unsigned char*
+ decryptThisMessage(unsigned char* , size_t&,unsigned char*, size_t, unsigned char*, unsigned char*): unsigned char*
UserConnectionManager
- server: Server*
- clAdd : struct sockaddr_in
- userName: string*
- userSocket : int
- myNonce : uint32_t
- clientNonce : unint32_t
- signatureManager : SignatureManager*
- diffieHellmannManager : DiffieHellmannManager*
- symmetricEncryptionManager : SimmetricEncryptionManager*
- counter: uint32_t
- ucmMutex : mutex
- busy : bool
+ UserConnectionManager(Server*,sockaddr_in, int)
+ openNewConnectionWithClient(): bool
+ isBusy() : bool
- establishSecureConnection(): bool
- waitForHelloMessage() : bool
- createCertificateMessage(size_t&): unisgned char*
- sendCertificate(unsigned char*, size_t):bool
- waitForClientPubKey() : bool
- sendMyPubkey() : bool
- waitForPlayersRequest(unsigned char*, size_t) : bool
- createPlayerListMsg(vector<string>, size_t&) : unsigned char*
- sendPlayersList() : bool
- waitForClientChoice(unsigned char*, size_t): string*
- sendChallengerRequest(string*): bool
- sendResponseToChallenger(string*, char) : bool
- waitForChallengedResponse(unsigned char*, size_t, bool&): string*
- sendOpponentKeyToChallenged(string*, uint32_t) : bool
- waitForChallengedReady(unsigned char*, size_t, uint32_t&, string*) : bool
- sendMyKeyToChallenger(string*, uint32_t) : bool
- getUserPubKey(string*, size_t&) : unsigned char*
- endGame(unsigned char*, size_t) : bool
- createSessionKey()
- logout(unsigned char*, size_t)
<<use>>
DiffieHellmannManager
- peerPubKey : EVP_PKEY*
- myPubKey : EVP_PKEY*
- sharedSecret : unsigned char*
- secret_len : size_t
+ DiffieHellmannManager()
+ setPeerPubkey(unsigned char*, size_t)
+ getMyPubKey(size_t&): unsigned char*
- computeSharedSecret(): void
+ getSharedSecret(size_t&) : unsigned char*
<<include>>

On server side the classes are:

- Server: is the main thread, which opens a socket and keeps waiting for new connection requests. Once a connection is accepted it creates a new UserConnectionManager which fires a new thread that handles the connection with the client itself. The Server object maintains the list of the currently connected users.
- UserConnectionManager: is the class that manages the connection between the server and the correspondent client: it establishes a secure channel, it sends the peer the players list, the challenge requests and all the other messages destinated to him.


SignatureManager
- hashMD : EVP_MD*
- prvKey : EVP_PKEY*
- pubKey : EVP_PKEY*
+ SignatureManager()
+ SignatureManager(string*)
+ signThisMessage(char*, size_t&): unsigned c
+ verifyThisSignature(unsigned char*, size_t, unsigned char*, size_t): bool
+ getPubKey(size_t&): unsigned char*
+ getPrvKey(): EVP_PKEY*
+ void setPubKey(EVP_PKEY*)
GameBoard
- gameMatrix : int[6][7]
- gameBoardRows : uint
- gameBoardColumns : uint
+ GameBoard()
- insertCoordinateInBoard(uint8_t,uint8_t,int): int
- gameFinished(int) : bool
+ insertOpponentMove(uint8_t,uint8_t) : int
+ insertMyMove(uint8_t,uint8_t) : int
SymmetricEncryptionManager
- cipher : EVP_CIPHER*
- aesKey : unsigned char*
- aesKeyLen : size_t
+ SimmetricEncryptionManager(unsigned char*, size_t)
+ encryptThisMessage(unsigned char* , size_t&,unsigned char*, size_t, unsigned char*, size_t&, unsigned char*): unsigned char*
+ decryptThisMessage(unsigned char* , size_t&,unsigned char*, size_t, unsigned char*, unsigned char*): unsigned char*
string.h
unistd.h
stdio.h
netdb.h
thread
vector
sys/types.h
sys/socket.h
netinet/in.h
iostream
fstream
strings.h
stdlib.h
string
openssl/evp.h
openssl/pem.h
openssl/rand.h
openssl/err.h
<< include >>
Use
Use
<< include >>
P2PConnectionManager
- serverConnectionManager: ServerConnectionManager*
- opponentAddr : struct sockaddr_in
- myAddr : struct sockaddr_in
- gameBoard : GameBoard*
- opponentSocket : int
- mySocket : int
- myUsername : string
- opponentUsername : string*
- signatureManager : SignatureManager*
- symmetricEncryptionManager : SymmetricEncryptionManager*
- diffieHellmannManager : DiffieHellmannManager*
+ P2PConnectionManager(EVP_PKEY*, ServerConnectionManager*)
+ startTheGameAsChallengeR()
+ startTheGameAsChallengeD()
+ setOpponentIp(inaddr):void
- waitForChallengeRConnection() : bool
- connectToChallengedUser () : bool
- establishSecureConnectionWithChallengeR : bool
- establishSecureConnectionWithChallengeD : bool
- waitForHelloMessage() : bool
- sendHelloMessage() : bool
- sendMyPubKey(): bool
- waitForPeerPubKey() :bool
- challengedGame() : bool
- sendCoordinate(uint8_t, uint8_t): bool
- waitForCoordinateMessage(uint8_t&, uint8_t&, bool) : bool
CertificateManager
- x509Store : X509_STORE*
- caCertificatePath : const string
- caCRLPath : const string
- md : EVP_MD*
+ CertificateManager()
+ verifyCertificate(unsigned char*, size_t) : bool
+ extractPubKey(unsigned char*, size_t) : EVP_PKEY*
<<use>>
ServerConnectionManager
- serverSocket : int
- serverAddr : struct sockaddr_in
- signatureManager: SignatureManager
- userName : string*
- prvKeyFilePasswd : string*
- simmetricEncryptionManager : SimmetricEncryptionManager
- serverNonce : unsigned char*
- myNonce : unsigned char*
- diffieHellmannManager : DiffieHellmannManager*
- certificateManager :CertificateManager*
- counter: int
+ServerConnectionManager(unsigned char*, int, string*)
+ createConnectionWithServer() : bool
+ establishConnectionToServer() : bool
+ secureTheConnection() : bool
- sendHelloMessage(unsigned char*) : bool
- waitForCertificate() : bool
- sendMyPubKey() : bool
- waitForPeerPubKey(): bool
- sendPlayersRequest() : bool
- waitForPlayers() : string*
- sendSelectedPlayer() : string*
+ waitForChallenger() : bool
+ sendResponse() : bool
+ waitForOppenentKey() : EVP_PKEY*
+ createNewChallenge() : bool
DiffieHellmannManager
- peerPubKey : EVP_PKEY*
- myPubKey : EVP_PKEY*
- sharedSecret : unsigned char*
- secret_len : size_t
+ DiffieHellmannManager()
+ setPeerPubkey(unsigned char*, size_t)
+ getMyPubKey(size_t&): unsigned char*
- computeSharedSecret(): void
+ getSharedSecret(size_t&) : unsigned char*
<<use>>
<< use >>
Client
- userName: string*
- serverConnectionManager:ServerConnectionManager
+ Client(string*, int, string*)
+ establishConnection() : bool
+ getUsername(): string*
<<use>>

On client side we have:

- Client: is the class that takes from keyboard the user's credentials, server address and port. It creates a new ServerConnectionManager and open a new thread for that connection.
- ServerConnectionManager: is the class which handles the connection between client and server, it establishes a secure communication and manages the overall messages exchanged between them. Especially at the beginning of a game it's in charge of creating a new P2PConnectionManager and a new thread responsible for the game.
- P2PConnectionManager: is responsible for creating and managing the actual game. It handles the establishment of a secure connection and all the messages exchanged during

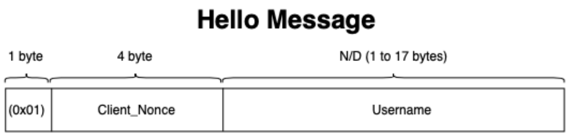
a game. When a secure channel has been created it sets up a new GameBoard to store the match data.
- GameBoard: is the board where the match takes place: it saves the moves of the current match; it controls the validity of the input data and it declares victory or defeat of the player.

Finally, we have a series of library classes, used by both sides, which deal with the management of the secure connection:

- SignatureManager: is the class in charge of sign messages and verify the signature.
- DiffieHellmannManager: is the class in charge of generating the peers' DH public keys, store them, generating the shared secret and finally the session key.
- SymmetricEncryptionManager: is the class that stores the AES symmetric key shared by the two peers. It has been created from the 128 least significant bits of the hashed DH shared secret.
- CertificateManager: this class is used by the ServerConnectionManager to deserialize the server certificate, verify it through the use of the CA's certificate and obtain the server's public key.

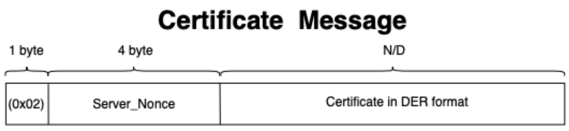
3.2. *Message Format*

- Hello Message



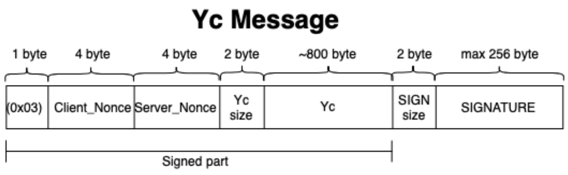
This message is used both for C/S and P2P communication. Opcode 0x01.

- Certificate Message



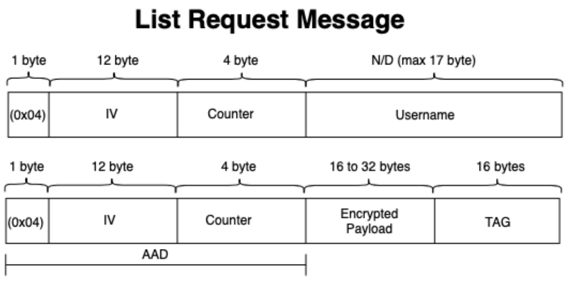
This is the message used by the server to send to the client its certificate. Opcode 0x02.

- Y Message



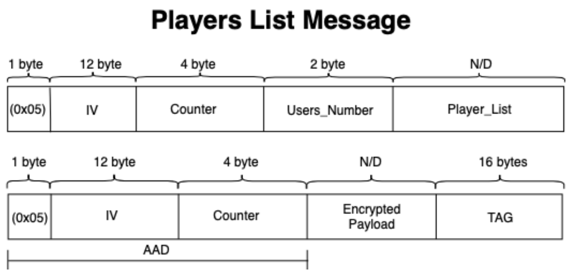
Used by both parties to send their DH public key; it contains the two nonces previously exchanged and it’s signed with 2048-bits RSA key to prove the authenticity. Opcode 0x03.

- List Request Message



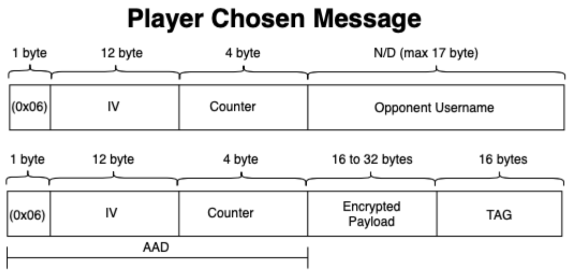
Sent by the client to obtain the players list. It contains the counter, generated randomly by the client. The AAD is composed of opcode, IV and counter authenticated with AES-GCM. Opcode 0x04.

- Players List Message



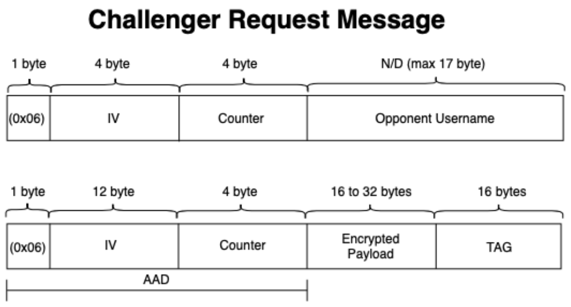
Sent by the server, it contains the list of the current available players and their number both encrypted by means of a 128-bits AES key. The message is authenticated using AES-GCM. Opcode 0x05.

- Player Chosen Message



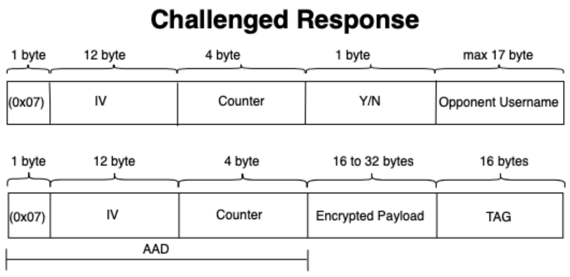
Sent by the client and contains the chosen player. Opcode 0x06.

- Challenger Request Message



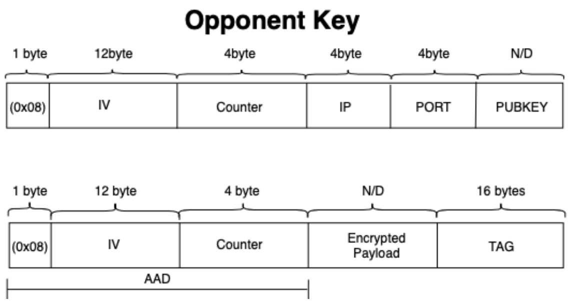
Sent by the server after receiving a Player Chosen Message, to the challenged previously chosen. It contains the challenger username. Opcode 0x06.

- Challenged Response Message



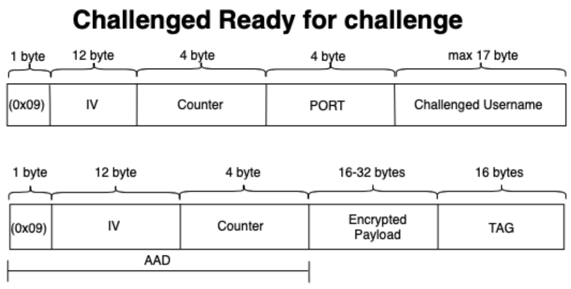
Sent by the challenged to decline or accept a challenge. It contains the response (Y or N) and the opponent username. It's identified with the opcode 0x07.

- Opponent Key Message



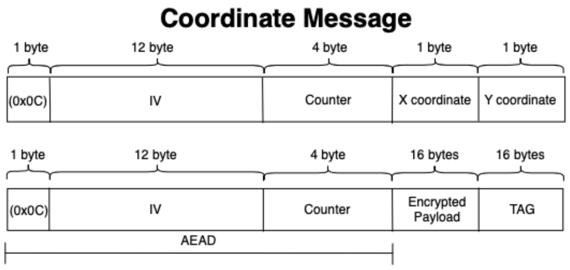
Sent by the server to both clients playing the match; it contains the opponent IP, port and RSA public key. Opcode 0x08.

- Challenged Ready Message



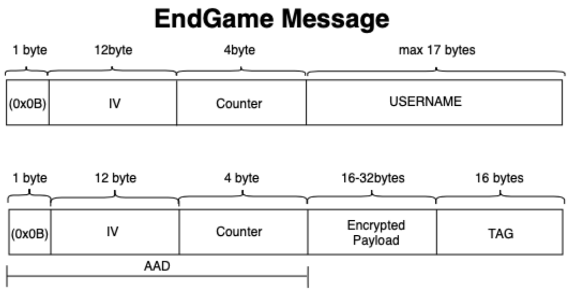
Sent by the challenged peer when it is ready to accept new connections. It communicates the port on which it's listening. Opcode 0x09.

- Coordinate Message



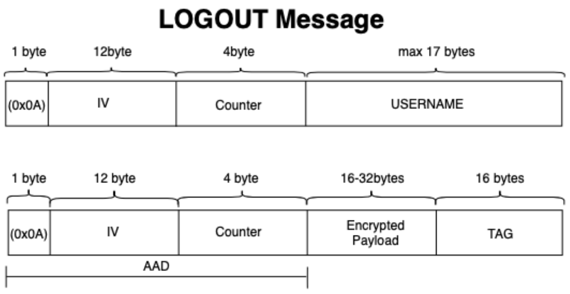
Sent during the P2P game to exchange the chosen move. It contains the row and columns numbers that identify the coordinate. This message is encrypted and authenticated with the players shared session key. Opcode 0x0C.

- EndGame Message



Sent by a client when the match finishes or when some connection error occurs. After this message it receives back the updated players list to start a new game. Opcode 0x0B.

- Logout Message



Sent by a client to exit the game.Opcode 0X0A.

4. BAN LOGIC

4.1. ASSUMPTIONS

4.1.1. POSTULATES

1) $$\frac{P \models Q \overset{K}{\leftrightarrow} P, P \triangleleft \{X\}_K}{P \models Q \mid\sim X}$$

If K is a shared key between P and Q, and P sees a message encrypted by K containing X (and P did not send that message), then P believes that X was sent by Q.

2) $$\frac{P \models \mid\xrightarrow{e_Q}, Q \triangleleft \{X\}_{e_Q^{-1}}}{P \models Q \mid\sim X}$$

If K is Q's public key, and P sees a message signed by con e_q^{-1}containing X, then P believes that X was sent by Q

3) $$\frac{P \models \#\{X\}, P \models Q \mid\sim X}{P \models Q \models X}$$

If P believes X was sent by Q, and P believes X is fresh, then P believes Q has sent X in this protocol execution instance

5) $$\frac{C \triangleleft \left\{\mid\xrightarrow{e_S} S\right\}_{e_T^{-1}}}{C \models \mid\xrightarrow{e_S} S}$$

If a client receives a certificate signed by T (Certification Authority) the client itself believes that that key is the server public key

6) $$\frac{P \models \#\{X\}}{P \models \#\{X, Y\}}$$

If P believes that message containing x is fresh, P believes that message x, y is fresh too.

4.1.2. KEYS

1) $\boldsymbol{S} \models \xrightarrow{\boldsymbol{e_c}} \boldsymbol{C}$

2) $\boldsymbol{S} \models \xrightarrow{\boldsymbol{e_T}} \boldsymbol{T}$

3) $\boldsymbol{C} \models \xrightarrow{\boldsymbol{e_T}} \boldsymbol{T}$

4.1.3. FRESHNESS

1) $\boldsymbol{S} \models \#(\boldsymbol{N_C})$

2) $\boldsymbol{C} \models \#(\boldsymbol{N_S})$

3) $\boldsymbol{C} \models \#(\boldsymbol{Counter_1})$

4) $\boldsymbol{S} \models \#(\boldsymbol{Counter_1})$

4.2.2.2 TRUST

1) $\boldsymbol{S} \models \boldsymbol{C} \Rightarrow \#(\boldsymbol{N_C})$

2) $\boldsymbol{C} \models \boldsymbol{S} \Rightarrow \#(\boldsymbol{N_S})$

3) $\boldsymbol{C} \models \boldsymbol{S} \Rightarrow \#(\boldsymbol{Y_S})$

4) $\boldsymbol{S} \models \boldsymbol{C} \Rightarrow \#(\boldsymbol{Y_C})$

5) $\boldsymbol{S} \models \boldsymbol{C} \Rightarrow \#(\boldsymbol{Counter_1})$

4.2. C/S SECURE CHANNEL ESTABLISHMENT

4.2.1. GOALS

1) $\boldsymbol{C} \models \boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}$

2) $\boldsymbol{S} \models \boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}$

3) $\boldsymbol{C} \models \boldsymbol{S} \models \boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}$

4) $\boldsymbol{S} \models \boldsymbol{C} \models \boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}$

5) $\boldsymbol{C} \models \#\left(\boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}\right)$

6) $\boldsymbol{S} \models \#\left(\boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}\right)$

4.2.2. IDEALIZED PROTOCOL

M2:

$\boldsymbol{C} \models |\xrightarrow{e_s} \boldsymbol{S}$ (applying the fifth postulate)

M3:

$\boldsymbol{S} \models \boldsymbol{C} |\sim \boldsymbol{M3}$ (applying the 2nd postulate)

$\boldsymbol{S} \models \mathbf{C} \models \mathbf{M3}$ (applying the 3rd postulate)

$\boldsymbol{S} \models \#\{\boldsymbol{Y_C}\}$ (applying the 6th postulate)

M4:

$\boldsymbol{C} \models \boldsymbol{S} |\sim \boldsymbol{M4}$ (applying the 2nd postulate)

$\boldsymbol{C} \models \mathbf{S} \models \mathbf{M4}$ (applying the 3rd postulate)

$\boldsymbol{C} \models \#\{\boldsymbol{Y_S}\}$ (applying the 6th postulate)

$\boldsymbol{C} \models \boldsymbol{S} \models \boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}$ (goal 3 achieved, C has K_{CS} on her hands)

After M4 goals 1 to 3 are being achieved.

M5:

$\boldsymbol{S} \models \#\{\boldsymbol{Counter_1}\}$

$\boldsymbol{S} \models \#\{\boldsymbol{M5}\}$ (applying the 6th postulate)

$\boldsymbol{S} \models \boldsymbol{C}|\sim\{\boldsymbol{M5}\}$ (applying the 1th postulate)

$\boldsymbol{S} \models \boldsymbol{C} \models \boldsymbol{C} \xleftrightarrow{\boldsymbol{K_{CS}}} \boldsymbol{S}$ (goal 4 achieved, S has K_{CS} on her hands)

M6,…,M12:

$\boldsymbol{S} \models \#\{\boldsymbol{Counter_i}\}$

$\boldsymbol{S} \models \#\{\boldsymbol{M6}\}$ (applying the 6th postulate)

$\boldsymbol{S} \models \boldsymbol{C}|\sim\{\boldsymbol{M6}\}$ (applying the 1th postulate)

4.3. P2P SECURE CHANNEL ESTABLISHMENT

4.3.1. GOALS

1) $\boldsymbol{CR} \models \boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}$

2) $\boldsymbol{CD} \models \boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}$

3) $\boldsymbol{CD} \models CR \models \boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}$

4) $\boldsymbol{CR} \models \boldsymbol{CD} \models \boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}$

5) $\boldsymbol{CR} \models \#\left(\boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}\right)$

6) $\boldsymbol{CD} \models \#\left(\boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}\right)$

4.3.2. IDEALIZED PROTOCOL

M15:

$\boldsymbol{CD} \models \boldsymbol{CR} \mid\sim \boldsymbol{M15}$ (applying the 2nd postulate)

$\boldsymbol{CD} \models \mathbf{CR} \models \mathbf{M15}$ (applying the 3rd postulate)

$\boldsymbol{CD} \models \#\{\boldsymbol{Y_{CR}}\}$ (applying the 6th postulate)

M16:

$\boldsymbol{CR} \models \boldsymbol{CD} \mid\sim \boldsymbol{M16}$ (applying the 2nd postulate)

$\boldsymbol{CR} \models \mathbf{CD} \models \mathbf{M16}$ (applying the 3rd postulate)

$\boldsymbol{CR} \models \#\{\boldsymbol{Y_{CD}}\}$ (applying the 6th postulate)

$\boldsymbol{CR} \models \boldsymbol{CD} \models \boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}$ (goal 3 achieved, C has K_{CS} on her hands)

After M16 goals 1 to 3 are being achieved.

M17:

$\boldsymbol{CD} \models \#\{\boldsymbol{Counter_1}\}$

$\boldsymbol{CD} \models \#\{\boldsymbol{M17}\}$ (applying the 6th postulate)

$\boldsymbol{CD} \models \boldsymbol{CR} \mid\sim \{\boldsymbol{M17}\}$ (applying the 1th postulate)

$\boldsymbol{CD} \models \boldsymbol{CR} \models \boldsymbol{CR} \xleftrightarrow{\boldsymbol{K_{CC}}} \boldsymbol{CD}$ (goal 4 achieved, S has K_{CS} on her hands)