Four-in-a-Row

Generated by Doxygen 1.8.17

1 Four-in-a-row 1

2 TODO 1

3 Hierarchical Index 1

3.1 Class Hierarchy . . . 1

4 Data Structure Index 2

4.1 Data Structures . . . 2

5 File Index 4

5.1 File List . . . 4

6 Data Structure Documentation 6

6.1 Args Class Reference . . . 6

6.1.1 Detailed Description . . . 7

6.1.2 Member Function Documentation . . . 7

6.1.3 Friends And Related Function Documentation . . . 7

6.2 CertificateMessage Class Reference . . . 7

6.2.1 Detailed Description . . . 8

6.2.2 Member Function Documentation . . . 8

6.3 CertificateRequestMessage Class Reference . . . 8

6.3.1 Detailed Description . . . 9

6.3.2 Member Function Documentation . . . 9

6.4 ChallengeForwardMessage Class Reference . . . 9

6.4.1 Detailed Description . . . 10

6.4.2 Member Function Documentation . . . 10

6.5 ChallengeMessage Class Reference . . . 11

6.5.1 Detailed Description . . . 11

6.5.2 Member Function Documentation . . . 11

6.6 ChallengeResponseMessage Class Reference . . . 12

6.6.1 Detailed Description . . . 12

6.6.2 Member Function Documentation . . . 12

6.7 ClientHelloMessage Class Reference . . . 13

6.7.1 Detailed Description . . . 14

6.7.2 Member Function Documentation . . . 14

6.8 ClientSecureSocketWrapper Class Reference . . . 14

6.8.1 Detailed Description . . . 15

6.8.2 Constructor & Destructor Documentation . . . 15

6.8.3 Member Function Documentation . . . 15

6.9 ClientSocketWrapper Class Reference . . . 16

6.9.1 Detailed Description . . . 16

6.9.2 Member Function Documentation . . . 16

6.10 ClientVerifyMessage Class Reference . . . 16

6.10.1 Detailed Description . . . 17
6.10.2 Member Function Documentation . . . 17
6.11 Connect4 Class Reference . . . 18
6.11.1 Detailed Description . . . 18
6.11.2 Constructor & Destructor Documentation . . . 18
6.11.3 Member Function Documentation . . . 19
6.12 ConnectionMode Struct Reference . . . 21
6.12.1 Detailed Description . . . 21
6.13 GameCancelMessage Class Reference . . . 22
6.13.1 Detailed Description . . . 22
6.13.2 Member Function Documentation . . . 22
6.14 GameEndMessage Class Reference . . . 23
6.14.1 Detailed Description . . . 23
6.14.2 Member Function Documentation . . . 23
6.15 GameStartMessage Class Reference . . . 24
6.15.1 Detailed Description . . . 25
6.15.2 Member Function Documentation . . . 25
6.16 Host Class Reference . . . 25
6.16.1 Detailed Description . . . 26
6.16.2 Constructor & Destructor Documentation . . . 26
6.17 Message Class Reference . . . 27
6.17.1 Detailed Description . . . 27
6.17.2 Member Function Documentation . . . 27
6.18 MessageQueue< T, MAX_SIZE > Class Template Reference . . . 28
6.18.1 Detailed Description . . . 28
6.18.2 Member Function Documentation . . . 29
6.19 MoveMessage Class Reference . . . 30
6.19.1 Detailed Description . . . 30
6.19.2 Member Function Documentation . . . 30
6.20 RegisterMessage Class Reference . . . 31
6.20.1 Detailed Description . . . 32
6.20.2 Member Function Documentation . . . 32
6.21 SecureHost Class Reference . . . 32
6.21.1 Detailed Description . . . 33
6.21.2 Constructor & Destructor Documentation . . . 33
6.22 SecureMessage Class Reference . . . 34
6.22.1 Detailed Description . . . 34
6.22.2 Member Function Documentation . . . 34
6.23 SecureSocketWrapper Class Reference . . . 35
6.23.1 Detailed Description . . . 37
6.23.2 Member Function Documentation . . . 37
6.24 Server Class Reference . . . 42

6.24.1 Detailed Description . . . 43
6.24.2 Member Function Documentation . . . 43
6.25 ServerHelloMessage Class Reference . . . 45
6.25.1 Detailed Description . . . 45
6.25.2 Member Function Documentation . . . 45
6.26 ServerSecureSocketWrapper Class Reference . . . 46
6.26.1 Detailed Description . . . 47
6.26.2 Constructor & Destructor Documentation . . . 47
6.26.3 Member Function Documentation . . . 47
6.27 ServerSocketWrapper Class Reference . . . 48
6.27.1 Detailed Description . . . 48
6.27.2 Member Function Documentation . . . 48
6.28 SocketWrapper Class Reference . . . 49
6.28.1 Detailed Description . . . 50
6.28.2 Member Function Documentation . . . 50
6.29 StartGameMessage Class Reference . . . 52
6.29.1 Detailed Description . . . 53
6.29.2 Member Function Documentation . . . 53
6.30 User Class Reference . . . 54
6.30.1 Detailed Description . . . 54
6.30.2 Constructor & Destructor Documentation . . . 54
6.31 UserList Class Reference . . . 55
6.31.1 Detailed Description . . . 56
6.31.2 Member Function Documentation . . . 56
6.32 UsersListMessage Class Reference . . . 58
6.32.1 Detailed Description . . . 59
6.32.2 Member Function Documentation . . . 59
6.33 UsersListRequestMessage Class Reference . . . 60
6.33.1 Detailed Description . . . 60
6.33.2 Member Function Documentation . . . 60

7 File Documentation **61**

7.1 args.cpp File Reference . . . 61
7.1.1 Detailed Description . . . 61
7.2 args.cpp . . . 62
7.3 args.h File Reference . . . 62
7.3.1 Detailed Description . . . 63
7.4 args.h . . . 63
7.5 client.cpp File Reference . . . 63
7.5.1 Detailed Description . . . 64
7.6 client.cpp . . . 64
7.7 connect4.cpp File Reference . . . 66

7.7.1 Detailed Description . . . 67
7.8 connect4.cpp . . . 67
7.9 connect4.h File Reference . . . 69
7.9.1 Detailed Description . . . 69
7.10 connect4.h . . . 70
7.11 connection_mode.h File Reference . . . 70
7.11.1 Detailed Description . . . 71
7.11.2 Enumeration Type Documentation . . . 71
7.12 connection_mode.h . . . 72
7.13 crypto.h File Reference . . . 72
7.13.1 Detailed Description . . . 73
7.13.2 Macro Definition Documentation . . . 74
7.13.3 Function Documentation . . . 74
7.14 crypto.h . . . 82
7.15 dump_buffer.cpp File Reference . . . 83
7.15.1 Detailed Description . . . 83
7.15.2 Function Documentation . . . 83
7.16 dump_buffer.cpp . . . 85
7.17 dump_buffer.h File Reference . . . 85
7.17.1 Detailed Description . . . 86
7.17.2 Function Documentation . . . 86
7.18 dump_buffer.h . . . 87
7.19 host.cpp File Reference . . . 87
7.19.1 Detailed Description . . . 87
7.20 host.cpp . . . 88
7.21 host.h File Reference . . . 88
7.21.1 Detailed Description . . . 88
7.22 host.h . . . 89
7.23 inet_utils.cpp File Reference . . . 89
7.23.1 Detailed Description . . . 90
7.23.2 Function Documentation . . . 90
7.24 inet_utils.cpp . . . 93
7.25 inet_utils.h File Reference . . . 95
7.25.1 Detailed Description . . . 96
7.25.2 Function Documentation . . . 96
7.26 inet_utils.h . . . 99
7.27 logging.h File Reference . . . 99
7.27.1 Detailed Description . . . 100
7.28 logging.h . . . 100
7.29 message_queue.h File Reference . . . 101
7.29.1 Detailed Description . . . 102
7.30 message_queue.h . . . 102

7.31 messages.cpp File Reference . . . 103
7.31.1 Detailed Description . . . 104
7.31.2 Function Documentation . . . 104
7.32 messages.cpp . . . 104
7.33 messages.h File Reference . . . 111
7.33.1 Detailed Description . . . 113
7.33.2 Enumeration Type Documentation . . . 113
7.33.3 Function Documentation . . . 114
7.34 messages.h . . . 115
7.35 multi_player.h File Reference . . . 120
7.35.1 Detailed Description . . . 120
7.36 multi_player.h . . . 121
7.37 secure_host.h File Reference . . . 121
7.37.1 Detailed Description . . . 121
7.38 secure_host.h . . . 122
7.39 secure_socket_wrapper.h File Reference . . . 122
7.39.1 Detailed Description . . . 123
7.40 secure_socket_wrapper.h . . . 123
7.41 server.cpp File Reference . . . 125
7.41.1 Detailed Description . . . 126
7.42 server/server.cpp . . . 126
7.43 server.h File Reference . . . 132
7.43.1 Detailed Description . . . 133
7.44 server.h . . . 133
7.45 server_lobby.cpp File Reference . . . 134
7.45.1 Detailed Description . . . 134
7.45.2 Function Documentation . . . 135
7.46 server_lobby.cpp . . . 135
7.47 server_lobby.h File Reference . . . 137
7.47.1 Detailed Description . . . 138
7.47.2 Function Documentation . . . 138
7.48 server_lobby.h . . . 138
7.49 single_player.h File Reference . . . 139
7.49.1 Detailed Description . . . 139
7.50 single_player.h . . . 139
7.51 socket_wrapper.cpp File Reference . . . 139
7.51.1 Detailed Description . . . 140
7.52 socket_wrapper.cpp . . . 140
7.53 socket_wrapper.h File Reference . . . 143
7.53.1 Detailed Description . . . 143
7.54 socket_wrapper.h . . . 144
7.55 user.h File Reference . . . 144

7.55.1 Detailed Description . . . 145
7.55.2 Enumeration Type Documentation . . . 145
7.56 user.h . . . 146
7.57 user_list.h File Reference . . . 146
7.57.1 Detailed Description . . . 147
7.58 user_list.h . . . 147

Index **149**

1 Four-in-a-row

Four-in-a-row game for the course of Foundations of Cyber Security at University of Pisa

2 TODO

- [x] Prompt user password to open key file
- [x] AAD
- [x] Use different format in place of PEM for certificate and keys over network
- [x] Client MUST request certificate from server
- [ ] Check possible security flaws
- [x] Hide username length by padding
- [ ] REPORT!!
- [x] variable size DS
- [x] check whether poll is required in RAND -> it's not
- [ ] check memory with Valgrind

3 Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Args **6**
Connect4 **18**
ConnectionMode **21**
Host **25**
SecureHost **32**

Message 27

CertificateMessage 7

CertificateRequestMessage 8

ChallengeForwardMessage 9

ChallengeMessage 11

ChallengeResponseMessage 12

ClientHelloMessage 13

ClientVerifyMessage 16

GameCancelMessage 22

GameEndMessage 23

GameStartMessage 24

MoveMessage 30

RegisterMessage 31

SecureMessage 34

ServerHelloMessage 45

StartGameMessage 52

UsersListMessage 58

UsersListRequestMessage 60

MessageQueue< T, MAX_SIZE > 28

SecureSocketWrapper 35

ClientSecureSocketWrapper 14

ServerSecureSocketWrapper 46

Server 42

SocketWrapper 49

ClientSocketWrapper 16

ServerSocketWrapper 48

User 54

UserList 55

4 Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Args
Utility class that parses an input line into a list of arguments (argc, argv) 6

CertificateMessage 7

CertificateRequestMessage 8

ChallengeForwardMessage
Message with which the server forwards a challenge 9

ChallengeMessage
Message that permits the client to challenge another client through the server 11

ChallengeResponseMessage
Message with which the client replies to a challenge 12

ClientHelloMessage 13

ClientSecureSocketWrapper
SocketWrapper for a TCP client 14

ClientSocketWrapper
SocketWrapper for a TCP client 16

ClientVerifyMessage 16

Connect4 18

ConnectionMode
Structure holding information about the connection requested by the user 21

GameCancelMessage
Message with which the server forwards a challenge rejectal or another event that caused the game to be canceled 22

GameEndMessage
Message that signals the server that the client is available 23

GameStartMessage
Message with which the server makes a new game start between clients 24

Host
Class that holds a host information 25

Message
Abstract class for Messages 27

MessageQueue< T, MAX_SIZE >
Thread-safe message queue template 28

MoveMessage
Message that signals a move 30

RegisterMessage
Message that permits the client to register to server 31

SecureHost
Class that holds a host information with certificate 32

SecureMessage 34

SecureSocketWrapper 35

Server
Utility class for interacting with the server 42

ServerHelloMessage 45

ServerSecureSocketWrapper
SocketWrapper for a TCP server 46

ServerSocketWrapper
SocketWrapper for a TCP server 48

SocketWrapper
Wrapper class around sockaddr_in and socket descriptor 49

StartGameMessage
Message that signals to start a new game 52

User
Class representing a user 54

UserList
Class that manages the users 55

UsersListMessage
Message that the server sends the client with the list of users 58

UsersListRequestMessage
Message with which the client asks for the list of connected users 60

5 File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

args.cpp
Implementation of the Args class 61

args.h
Definition of the Args class 62

buffer_io.cpp ??

buffer_io.h ??

client.cpp
Implementation of a 4-in-a-row game 63

config.h ??

connect4.cpp
Implementation of connect4.h 66

connect4.h
Header file for the class responsible of handling the board of a Connect4 game 69

connection_mode.h
Header file for utility structure ConnectionMode 70

src/security/crypto.cpp ??

tests/security/crypto.cpp ??

crypto.h
Header for crypto algorithms 72

crypto_utils.cpp ??

crypto_utils.h ??

dump_buffer.cpp
Implementation of dump_buffer.h 83

dump_buffer.h
Utility functions for writing and reading data from a buffer 85

host.cpp
Implementation of host.h 87

host.h
Definition of the helper class "Host" 88

inet_utils.cpp
Implementation of inet_utils.h 89

inet_utils.h
Utility funcions for managing inet addresses 95

logging.h
Logging macro 99

message_queue.h
Definition and implementation of the MessageQueue class 101

messages.cpp
Implementation of messages.h 103

messages.h
Definition of messages 111

multi_player.cpp ??

multi_player.h
Implementation of the multi player game main function and connection with peer functions 120

secure_host.h
Definition of the helper class "SecureHost" 121

secure_socket_wrapper.cpp ??

secure_socket_wrapper.h
Header file for SecureSocketWrapper 122

client/server.cpp ??

server/server.cpp
Implementation of a 4-in-a-row online server 125

server.h
Implementation of the utility class used to communicate with the server 132

server_lobby.cpp
Implementation of the function that handles user and network input while the user is in the server lobby waiting for a game to start 134

server_lobby.h
Definition of the function that handles user and network input while the user is in the server lobby waiting for a game to start 137

single_player.cpp ??

single_player.h
Implementation of the single player game main function 139

socket_wrapper.cpp
Implementation of socket_wrapper.h 139

socket_wrapper.h
Definition of the helper class "SocketWrapper" and derivatives 143

user.h
Definition of the User class 144

user_list.cpp ??

user_list.h
Implementation of the UserList class 146

6 Data Structure Documentation

6.1 Args Class Reference

Utility class that parses an input line into a list of arguments (argc, argv)

`#include <args.h>`

Public Member Functions

- Args ()
  Default constructor.
- Args (char *line)
  Constructor that parses the given input line.
- Args (std::istream &is)
  Constructor that reads from the given input stream.
- int getArgc ()
  Returns argument count.
- const char * getArgv (unsigned int i)
  Returns nth argument.
- const char * c_str ()
  Returns user friendly content as a C string.

Friends

- std::ostream & operator<< (std::ostream &os, const Args &b)
  Operator overload for printing the arguments with cout.

6.1.1 Detailed Description

Utility class that parses an input line into a list of arguments (argc, argv)

Definition at line 24 of file args.h.

6.1.2 Member Function Documentation

6.1.2.1 getArgc() `int Args::getArgc ( ) [inline]`

Returns argument count.

Returns

>=0 argument count @retuns -1 error reading stream (may be caused by EOF)

Definition at line 59 of file args.h.

6.1.3 Friends And Related Function Documentation

6.1.3.1 operator<< `std::ostream& operator<< ( std::ostream & os, const Args & b ) [friend]`

Operator overload for printing the arguments with cout.

Format: ["arg1", "arg2"]

6.2 CertificateMessage Class Reference

Inherits Message.

Public Member Functions

- **CertificateMessage** (X509 *cert)
- MessageType **getType** ()
- string getName ()

  Get message name (for debug purposes)
- X509 * **getCert** ()
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.

6.2.1 Detailed Description

Definition at line 494 of file messages.h.

6.2.2 Member Function Documentation

6.2.2.1 read() msglen_t CertificateMessage::read (char * buffer, msglen_t len) [virtual]

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 636 of file messages.cpp.

6.2.2.2 write() msglen_t CertificateMessage::write (char * buffer) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 620 of file messages.cpp.

6.3 CertificateRequestMessage Class Reference

Inherits Message.

Public Member Functions

- MessageType **getType** ()
- string getName ()

  Get message name (for debug purposes)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.

6.3.1 Detailed Description

Definition at line 482 of file messages.h.

6.3.2 Member Function Documentation

6.3.2.1 read()

```
msglen_t CertificateRequestMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 616 of file messages.cpp.

6.3.2.2 write()

```
msglen_t CertificateRequestMessage::write (
        char * buffer )  [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 604 of file messages.cpp.

6.4 ChallengeForwardMessage Class Reference

Message with which the server forwards a challenge.

```
#include <messages.h>
```

Inherits Message.

Public Member Functions

- **ChallengeForwardMessage** (string username)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.
- string getName ()

  Get message name (for debug purposes)
- string **getUsername** ()
- MessageType **getType** ()

6.4.1 Detailed Description

Message with which the server forwards a challenge.

Definition at line 280 of file messages.h.

6.4.2 Member Function Documentation

6.4.2.1 read()

```
msglen_t ChallengeForwardMessage::read (
        char * buffer,
        msglen_t len ) [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 269 of file messages.cpp.

6.4.2.2 write()

```
msglen_t ChallengeForwardMessage::write (
        char * buffer ) [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 255 of file messages.cpp.

6.5 ChallengeMessage Class Reference

Message that permits the client to challenge another client through the server.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **ChallengeMessage** (string username)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.
- string getName ()

  Get message name (for debug purposes)
- string **getUsername** ()
- MessageType **getType** ()

6.5.1 Detailed Description

Message that permits the client to challenge another client through the server.

Definition at line 194 of file messages.h.

6.5.2 Member Function Documentation

6.5.2.1 read()

```
msglen_t ChallengeMessage::read (
        char * buffer,
        msglen_t len ) [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 173 of file messages.cpp.

6.5.2.2 write() `msglen_t ChallengeMessage::write ( char * buffer ) [virtual]`

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 159 of file messages.cpp.

6.6 ChallengeResponseMessage Class Reference

Message with which the client replies to a challenge.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **ChallengeResponseMessage** (string username, bool response, uint16_t port)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.
- string getName ()

  Get message name (for debug purposes)
- string **getUsername** ()
- bool **getResponse** ()
- uint16_t **getListenPort** ()
- MessageType **getType** ()

6.6.1 Detailed Description

Message with which the client replies to a challenge.

Definition at line 303 of file messages.h.

6.6.2 Member Function Documentation

6.6.2.1 read()

```
msglen_t ChallengeResponseMessage::read (
        char * buffer,
        msglen_t len ) [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 302 of file messages.cpp.

6.6.2.2 write()

```
msglen_t ChallengeResponseMessage::write (
        char * buffer ) [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 280 of file messages.cpp.

6.7 ClientHelloMessage Class Reference

Inherits Message.

Public Member Functions

- **ClientHelloMessage** (EVP_PKEY *eph_key, nonce_t nonce, string my_id, string other_id)
- MessageType **getType** ()
- string getName ()

  Get message name (for debug purposes)
- nonce_t **getNonce** ()
- EVP_PKEY * **getEphKey** ()
- void **setEphKey** (EVP_PKEY *eph_key)
- string **getMyId** ()
- string **getOtherId** ()
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.

6.7.1 Detailed Description

Definition at line 404 of file messages.h.

6.7.2 Member Function Documentation

6.7.2.1 read() msglen_t ClientHelloMessage::read (char * buffer, msglen_t len) [virtual]

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 457 of file messages.cpp.

6.7.2.2 write() msglen_t ClientHelloMessage::write (char * buffer) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 430 of file messages.cpp.

6.8 ClientSecureSocketWrapper Class Reference

SocketWrapper for a TCP client.

```
#include <secure_socket_wrapper.h>
```

Inherits SecureSocketWrapper.

Public Member Functions

- ClientSecureSocketWrapper (X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store)

  Initialize a new socket on a random port.
- int connectServer (SecureHost host)

  Connects to a remote server.

Additional Inherited Members

6.8.1 Detailed Description

SocketWrapper for a TCP client.

It provides a new function to connect to server.

Definition at line 277 of file secure_socket_wrapper.h.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 ClientSecureSocketWrapper() `ClientSecureSocketWrapper::ClientSecureSocketWrapper (`

```
X509 * cert,
EVP_PKEY * my_priv_key,
X509_STORE * store )
```

Initialize a new socket on a random port.

Parameters

port	the port you want to bind on

Definition at line 560 of file secure_socket_wrapper.cpp.

6.8.3 Member Function Documentation

6.8.3.1 connectServer() `int ClientSecureSocketWrapper::connectServer (`

```
SecureHost host )
```

Connects to a remote server.

Returns

0 in case of success, something else otherwise

Definition at line 566 of file secure_socket_wrapper.cpp.

6.9 ClientSocketWrapper Class Reference

SocketWrapper for a TCP client.

`#include <socket_wrapper.h>`

Inherits SocketWrapper.

Public Member Functions

- int connectServer (Host host)
  Connects to a remote server.

Additional Inherited Members

6.9.1 Detailed Description

SocketWrapper for a TCP client.

It provides a new function to connect to server.

Definition at line 138 of file socket_wrapper.h.

6.9.2 Member Function Documentation

6.9.2.1 connectServer() `int ClientSocketWrapper::connectServer ( Host host )`

Connects to a remote server.

Returns

0 in case of success, something else otherwise

Definition at line 184 of file socket_wrapper.cpp.

6.10 ClientVerifyMessage Class Reference

Inherits Message.

Public Member Functions

- **ClientVerifyMessage** (char *ds, uint32_t ds_size)
- MessageType **getType** ()
- string getName ()

  Get message name (for debug purposes)
- char * **getDs** ()
- uint32_t **getDsSize** ()
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.

6.10.1 Detailed Description

Definition at line 430 of file messages.h.

6.10.2 Member Function Documentation

6.10.2.1 read()

```
msglen_t ClientVerifyMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 583 of file messages.cpp.

6.10.2.2 write()

```
msglen_t ClientVerifyMessage::write (
        char * buffer )  [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 564 of file messages.cpp.

6.11 Connect4 Class Reference

Public Member Functions

- Connect4 (int rows=6, int columns=7)

  Construct a new Connect 4 object.
- int getNumCols ()

  Get the number of columns of the board.
- int8_t play (int column, char player=0)

  Inserts a token.
- bool checkWin (int starting_row, int starting_col, char player=0)

  Checks if an inserted token causes a win.
- bool setPlayer (char player)

  Sets the default player.
- char getPlayer ()

  Get the default player.
- char getAdv ()

  Get the adversary, when a default player is set.
- void print (std::ostream &os)

  Prints the board.

Friends

- std::ostream & **operator**<< (std::ostream &os, const Connect4 &b)

6.11.1 Detailed Description

Definition at line 18 of file connect4.h.

6.11.2 Constructor & Destructor Documentation

6.11.2.1 Connect4() `Connect4::Connect4 (`

```
int rows = 6,
int columns = 7 )
```

Construct a new Connect 4 object.

Parameters

rows	Number of rows
columns	Number of columns

Definition at line 19 of file connect4.cpp.

6.11.3 Member Function Documentation

6.11.3.1 checkWin() `bool Connect4::checkWin (`

```
int starting_row,
int starting_col,
char player = 0 )
```

Checks if an inserted token causes a win.

Parameters

starting_row	row of the token
starting_col	col of the token
player	marker of the player inserting the token

Returns

true if winning, false otherwise

Definition at line 83 of file connect4.cpp.

6.11.3.2 getAdv() `char Connect4::getAdv ( )`

Get the adversary, when a default player is set.

Returns

enemy marker

Definition at line 146 of file connect4.cpp.

6.11.3.3 getNumCols() `int Connect4::getNumCols ( )`

Get the number of columns of the board.

Returns

number of columns

Definition at line 128 of file connect4.cpp.

6.11.3.4 getPlayer() `char Connect4::getPlayer ( )`

Get the default player.

Returns

player marker

Definition at line 142 of file connect4.cpp.

6.11.3.5 play() `int8_t Connect4::play ( int column, char player = 0 )`

Inserts a token.

Parameters

column	target column where the token should be added
player	player who is making the move

Return values

1	Success with win
0	Success without win
-1	Failure for full column
-2	Board is full, it could be so before or after the move takes place

Definition at line 31 of file connect4.cpp.

6.11.3.6 print() `void Connect4::print ( std::ostream & os )`

Prints the board.

Parameters

os	Output stream where the board has to be printed

Definition at line 15 of file connect4.cpp.

6.11.3.7 setPlayer() `bool Connect4::setPlayer ( char player )`

Sets the default player.

Parameters

player	player to be set

Returns

true if a valid player was supplied and set

false otherwise

Definition at line 132 of file connect4.cpp.

6.12 ConnectionMode Struct Reference

Structure holding information about the connection requested by the user.

`#include <connection_mode.h>`

Public Member Functions

- **ConnectionMode** (enum ConnectionType connection_type, const char *ip, int port, X509 *cert, uint16_t listen_port)
- **ConnectionMode** (enum ConnectionType connection_type, SecureHost host, uint16_t listen_port)
- **ConnectionMode** (enum ConnectionType connection_type, enum ExitCode exit_code)
- **ConnectionMode** (enum ConnectionType connection_type)

Data Fields

- enum ConnectionType **connection_type**
- SecureHost **host**
- union {
  uint16_t **listen_port**
  enum ExitCode **exit_code**
  };

6.12.1 Detailed Description

Structure holding information about the connection requested by the user.

See also

ConnectionType

Definition at line 34 of file connection_mode.h.

6.13 GameCancelMessage Class Reference

Message with which the server forwards a challenge rejectal or another event that caused the game to be canceled.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **GameCancelMessage** (string username)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.
- string getName ()

  Get message name (for debug purposes)
- string **getUsername** ()
- MessageType **getType** ()

6.13.1 Detailed Description

Message with which the server forwards a challenge rejectal or another event that caused the game to be canceled.

Definition at line 332 of file messages.h.

6.13.2 Member Function Documentation

6.13.2.1 read()

```
msglen_t GameCancelMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 335 of file messages.cpp.

6.13.2.2 write() msglen_t GameCancelMessage::write (char * *buffer*) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 321 of file messages.cpp.

6.14 GameEndMessage Class Reference

Message that signals the server that the client is available.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- msglen_t write (char *buffer)
  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)
  Read message from buffer.
- string getName ()
  Get message name (for debug purposes)
- MessageType **getType** ()

6.14.1 Detailed Description

Message that signals the server that the client is available.

Definition at line 217 of file messages.h.

6.14.2 Member Function Documentation

6.14.2.1 read() msglen_t GameEndMessage::read (char * buffer, msglen_t len) [virtual]

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 195 of file messages.cpp.

6.14.2.2 write() msglen_t GameEndMessage::write (char * buffer) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 184 of file messages.cpp.

6.15 GameStartMessage Class Reference

Message with which the server makes a new game start between clients.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **GameStartMessage** (string username, struct sockaddr_in addr, X509 *opp_cert)
- msglen_t write (char *buffer)
  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)
  Read message from buffer.
- string getName ()
  Get message name (for debug purposes)
- string **getUsername** ()
- struct sockaddr_in **getAddr** ()
- SecureHost **getHost** ()
- X509 * **getCert** ()
- MessageType **getType** ()

6.15.1 Detailed Description

Message with which the server makes a new game start between clients.

Definition at line 355 of file messages.h.

6.15.2 Member Function Documentation

6.15.2.1 read()

```
msglen_t GameStartMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 369 of file messages.cpp.

6.15.2.2 write()

```
msglen_t GameStartMessage::write (
        char * buffer )  [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 346 of file messages.cpp.

6.16 Host Class Reference

Class that holds a host information.

`#include <host.h>`

Inherited by SecureHost.

Public Member Functions

- Host ()
  Constructs new empty instance.
- Host (struct sockaddr_in addr)
  Constructs new instance from given inet address.
- Host (const char *ip, int port)
  Constructs new instance from IP/port pair.
- struct sockaddr_in getAddress ()
  Returns the inet address of the host.
- string toString ()
  Returns the inet address of the host.

6.16.1 Detailed Description

Class that holds a host information.

OpenSSL certificates are held in SecureHost class.

Definition at line 25 of file host.h.

6.16.2 Constructor & Destructor Documentation

6.16.2.1 Host() [1/2] Host::Host (struct sockaddr_in *addr*) [inline]

Constructs new instance from given inet address.

Parameters

addr	the inet address of the remote host

Definition at line 40 of file host.h.

6.16.2.2 Host() [2/2] Host::Host (const char * *ip*, int *port*)

Constructs new instance from IP/port pair.

Parameters

ip	the IP address the remote host
port	the port the remote host

Definition at line 15 of file host.cpp.

6.17 Message Class Reference

Abstract class for Messages.

`#include <messages.h>`

Inherited by CertificateMessage, CertificateRequestMessage, ChallengeForwardMessage, ChallengeMessage, ChallengeResponseMessage, ClientHelloMessage, ClientVerifyMessage, GameCancelMessage, GameEndMessage, GameStartMessage, MoveMessage, RegisterMessage, SecureMessage, ServerHelloMessage, StartGameMessage, UsersListMessage, and UsersListRequestMessage.

Public Member Functions

- virtual msglen_t write (char *buffer)=0
  Write message to buffer.
- virtual msglen_t read (char *buffer, msglen_t len)=0
  Read message from buffer.
- virtual string getName ()=0
  Get message name (for debug purposes)
- virtual MessageType **getType** ()=0

6.17.1 Detailed Description

Abstract class for Messages.

Definition at line 99 of file messages.h.

6.17.2 Member Function Documentation

6.17.2.1 read()

```
virtual msglen_t Message::read (
        char * buffer,
        msglen_t len )  [pure virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implemented in CertificateMessage, CertificateRequestMessage, ServerHelloMessage, ClientVerifyMessage, ClientHelloMessage, SecureMessage, GameStartMessage, GameCancelMessage, ChallengeResponseMessage, ChallengeForwardMessage, UsersListRequestMessage, UsersListMessage, GameEndMessage, ChallengeMessage, RegisterMessage, MoveMessage, and StartGameMessage.

6.17.2.2 write() `virtual msglen_t Message::write ( char * buffer ) [pure virtual]`

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implemented in CertificateMessage, CertificateRequestMessage, ServerHelloMessage, ClientVerifyMessage, ClientHelloMessage, SecureMessage, GameStartMessage, GameCancelMessage, ChallengeResponseMessage, ChallengeForwardMessage, UsersListRequestMessage, UsersListMessage, GameEndMessage, ChallengeMessage, RegisterMessage, MoveMessage, and StartGameMessage.

6.18 MessageQueue< T, MAX_SIZE > Class Template Reference

Thread-safe message queue template.

`#include <message_queue.h>`

Public Member Functions

- MessageQueue ()
  Default constructor that creates an empty queue and initializes both mutex and cond.
- bool push (T e)
  Insert a new element to the back of the queue WITHOUT SIGNALING on cond.
- bool pushSignal (T e)
  Insert a new element to the back of the queue, signaling any blocked thread that new items are available.
- T pull ()
  Retrieves and pops the first element from the queue, if any.
- T pullWait ()
  Retrieves and pops the first element from the queue.
- size_t size ()
  Returns the number of elements in queue.
- size_t empty ()
  Returns true if the queue is empty, false otherwise.

6.18.1 Detailed Description

template<typename T, int MAX_SIZE>
class MessageQueue< T, MAX_SIZE >

Thread-safe message queue template.

Threads are blocked on a pthread_cond if no message is available and are awaken by a pthread_cond_signal when a new item is added. Access to the class is regulated by a mutex.

Definition at line 32 of file message_queue.h.

6.18.2 Member Function Documentation

6.18.2.1 pull() template<typename T , int MAX_SIZE>

```
T MessageQueue< T, MAX_SIZE >::pull
```

Retrieves and pops the first element from the queue, if any.

Returns

the first item, if any, undefined behaviour otherwise.

Definition at line 105 of file message_queue.h.

6.18.2.2 pullWait() template<typename T , int MAX_SIZE>

```
T MessageQueue< T, MAX_SIZE >::pullWait
```

Retrieves and pops the first element from the queue.

If no item is in the queue, the thread is blocked waiting for new items to be inserted.

Returns

the first item.

Definition at line 115 of file message_queue.h.

6.18.2.3 push() template<typename T , int MAX_SIZE>

```
bool MessageQueue< T, MAX_SIZE >::push (
            T e )
```

Insert a new element to the back of the queue WITHOUT SIGNALING on cond.

Parameters

T	the element to be inserted

Returns

true if insertion was successfull, false otherwise

Definition at line 95 of file message_queue.h.

6.18.2.4 pushSignal() template<typename T , int MAX_SIZE>

```
bool MessageQueue< T, MAX_SIZE >::pushSignal (
            T e )
```

Insert a new element to the back of the queue, signaling any blocked thread that new items are available.

Parameters

T	the element to be inserted

Returns

true if insertion was successfull, false otherwise

Definition at line 127 of file message_queue.h.

6.19 MoveMessage Class Reference

Message that signals a move.

```
#include <messages.h>
```

Inherits Message.

Public Member Functions

- **MoveMessage** (char col)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.
- string getName ()

  Get message name (for debug purposes)
- char **getColumn** ()
- MessageType **getType** ()

6.19.1 Detailed Description

Message that signals a move.

Definition at line 147 of file messages.h.

6.19.2 Member Function Documentation

6.19.2.1 read() msglen_t MoveMessage::read (char * buffer, msglen_t len) [virtual]

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 122 of file messages.cpp.

6.19.2.2 write() msglen_t MoveMessage::write (char * buffer) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 107 of file messages.cpp.

6.20 RegisterMessage Class Reference

Message that permits the client to register to server.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **RegisterMessage** (string username)
- msglen_t write (char *buffer)
  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)
  Read message from buffer.
- string getName ()
  Get message name (for debug purposes)
- string **getUsername** ()
- MessageType **getType** ()

6.20.1 Detailed Description

Message that permits the client to register to server.

Definition at line 170 of file messages.h.

6.20.2 Member Function Documentation

6.20.2.1 read()

```
msglen_t RegisterMessage::read (
        char * buffer,
        msglen_t len ) [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 148 of file messages.cpp.

6.20.2.2 write()

```
msglen_t RegisterMessage::write (
        char * buffer ) [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 133 of file messages.cpp.

6.21 SecureHost Class Reference

Class that holds a host information with certificate.

`#include <secure_host.h>`

Inherits Host.

Public Member Functions

- SecureHost ()

  Constructs new empty instance.
- SecureHost (struct sockaddr_in addr, X509 ∗cert)

  Constructs new instance from given inet address and X509 certificate.
- SecureHost (const char ∗ip, int port, X509 ∗cert)

  Constructs new instance from IP/port pair.
- X509 ∗ getCert ()

  Returns the X509 certificate.

6.21.1 Detailed Description

Class that holds a host information with certificate.

Definition at line 25 of file secure_host.h.

6.21.2 Constructor & Destructor Documentation

6.21.2.1 SecureHost() [1/2]

```
SecureHost::SecureHost (
        struct sockaddr_in addr,
        X509 * cert ) [inline]
```

Constructs new instance from given inet address and X509 certificate.

Parameters

addr	the inet address of the remote host
cert	X509 certificate

Definition at line 40 of file secure_host.h.

6.21.2.2 SecureHost() [2/2]

```
SecureHost::SecureHost (
        const char * ip,
        int port,
        X509 * cert ) [inline]
```

Constructs new instance from IP/port pair.

Parameters

ip	the IP address the remote host
port	the port the remote host

Definition at line 48 of file secure_host.h.

6.22 SecureMessage Class Reference

Inherits Message.

Public Member Functions

- **SecureMessage** (char *ct, msglen_t ct_size, char *tag)
- MessageType **getType** ()
- string getName ()

  Get message name (for debug purposes)
- void **setCtSize** (msglen_t s)
- size_t **getCtSize** ()
- char * **getCt** ()
- char * **getTag** ()
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.

6.22.1 Detailed Description

Definition at line 380 of file messages.h.

6.22.2 Member Function Documentation

6.22.2.1 read()

```
msglen_t SecureMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 407 of file messages.cpp.

6.22.2.2 write() `msglen_t SecureMessage::write ( char * buffer ) [virtual]`

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 388 of file messages.cpp.

6.23 SecureSocketWrapper Class Reference

Inherited by ClientSecureSocketWrapper, and ServerSecureSocketWrapper.

Public Member Functions

- SecureSocketWrapper (X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store)
  Initialize on a new socket.
- SecureSocketWrapper (X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store, int sd)
  Initialize using existing socket.
- SecureSocketWrapper (X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store, SocketWrapper *sw)
  Constructor to generate connection socket wrappers.
- ~SecureSocketWrapper ()
  Destructor.
- Message * readPartMsg ()
  Read any new data from the socket but does not wait for the whole message to be ready.
- Message * receiveAnyMsg ()
  Receive any new message from the socket.
- Message * receiveMsg (MessageType type)
  Receive a new message of the given type from the socket.
- Message * receiveMsg (MessageType type[ ], int n_types)
  Receive a new message of any of the given types from the socket.
- Message * **handleMsg** (Message *msg)
- int **sendCertRequest** ()
- int **handleCertResponse** (CertificateMessage *cm)
- int **handleClientHello** (ClientHelloMessage *chm)
- int **handleServerHello** (ServerHelloMessage *shm)
- int **handleClientVerify** (ClientVerifyMessage *cvm)
- int **sendClientHello** ()
- int **sendServerHello** ()
- int **sendClientVerify** ()
- int **sendPlain** (Message *msg)
- int sendMsg (Message *msg)
  Sends the given message to the peer host through the socket.
- int handshakeServer ()
  Estiblishes a secure connection over the already specified socket.

- int handshakeClient ()

  Estiblishes a secure connection over the already specified socket.
- bool setOtherCert (X509 ∗other_cert)

  Sets the peer certificate.
- int getDescriptor ()

  Returns current socket file descriptor.
- void closeSocket ()

  Closes the socket.
- void setOtherAddr (struct sockaddr_in addr)

  Sets the address of the other host.
- sockaddr_in ∗ **getOtherAddr** ()
- SecureHost getConnectedHost ()

  Returns connected host.
- X509 ∗ getCert ()

  Returns the certificate of this host.

Protected Member Functions

- SecureSocketWrapper ()

  Empty constructor to use in child classes.
- void generateKeys (const char ∗role)

  Derives the key.
- void updateSendIV ()

  Calculates the IV to use when sending the next message.
- void updateRecvIV ()

  Calculates the IV to use when receiving a new message.
- void init (X509 ∗cert, EVP_PKEY ∗my_priv_key, X509_STORE ∗store)

  Internal initialization.
- Message ∗ decryptMsg (SecureMessage ∗sm)

  Decrypts a Secure Message into a Message.
- SecureMessage ∗ encryptMsg (Message ∗m)

  Encrypts a Message into a SecureMessage.
- int makeSignature (const char ∗role, char ∗∗ds)

  Make the signature for the handshake protocol.
- bool checkSignature (char ∗ds, size_t ds_size, const char ∗role)

  Checks the signature for the handshake protocol.
- int buildMsgToSign (const char ∗role, char ∗msg)

  Builds the message to be signed.
- void makeAAD (MessageType msg_type, msglen_t len, char ∗aad)

  Builds the aad of a message.

Protected Attributes

- SocketWrapper ∗ **sw**
- char **send_key** [KEY_SIZE]
- char **recv_key** [KEY_SIZE]
- char **send_iv_static** [IV_SIZE]
- char **recv_iv_static** [IV_SIZE]
- char **send_iv** [IV_SIZE]
- char **recv_iv** [IV_SIZE]

- uint64_t **send_seq_num**
- uint64_t **recv_seq_num**
- string **my_id**
- string **other_id**
- nonce_t **sv_nonce**
- nonce_t **cl_nonce**
- EVP_PKEY * **my_eph_key**
- EVP_PKEY * **other_eph_key**
- X509 * **my_cert**
- X509 * **other_cert**
- X509_STORE * **store**
- EVP_PKEY * **my_priv_key**
- bool **peer_authenticated**
- char **msg_to_sign_buf** [MAX_MSG_TO_SIGN_SIZE]

6.23.1 Detailed Description

Definition at line 28 of file secure_socket_wrapper.h.

6.23.2 Member Function Documentation

6.23.2.1 buildMsgToSign()

```
int SecureSocketWrapper::buildMsgToSign (
        const char * role,
        char * msg )  [protected]
```

Builds the message to be signed.

Parameters

role	the role of this peer
msg	the buffer to write the message to

Returns

number of written bytes

Definition at line 387 of file secure_socket_wrapper.cpp.

6.23.2.2 decryptMsg()

```
Message * SecureSocketWrapper::decryptMsg (
        SecureMessage * sm )  [protected]
```

Decrypts a Secure Message into a Message.

Parameters

sm	Secure message ptr

Returns

Message* Read message

Definition at line 50 of file secure_socket_wrapper.cpp.

6.23.2.3 encryptMsg() `SecureMessage * SecureSocketWrapper::encryptMsg ( Message * m ) [protected]`

Encrypts a Message into a SecureMessage.

Parameters

m	Message to encrypt

Returns

SecureMessage* Encrypted SecureMessage

Definition at line 114 of file secure_socket_wrapper.cpp.

6.23.2.4 generateKeys() `void SecureSocketWrapper::generateKeys ( const char * role ) [protected]`

Derives the key.

Parameters

role	Role in the communication

Definition at line 340 of file secure_socket_wrapper.cpp.

6.23.2.5 handshakeClient() `int SecureSocketWrapper::handshakeClient ( )`

Estiblishes a secure connection over the already specified socket.

To be run client-side.

Returns

int 0 in case of success, something else otherwise

Definition at line 490 of file secure_socket_wrapper.cpp.

6.23.2.6 handshakeServer() `int SecureSocketWrapper::handshakeServer ( )`

Estiblishes a secure connection over the already specified socket.

To be run server-side.

Returns

int 0 in case of success, something else otherwise

Definition at line 506 of file secure_socket_wrapper.cpp.

6.23.2.7 makeAAD() `void SecureSocketWrapper::makeAAD ( MessageType msg_type, msglen_t len, char * aad ) [protected]`

Builds the aad of a message.

I.e. this function builds the message header as SocketWrapper would.

Parameters

msg_type	the type of the message
len	the length of the message
aad	the aad buffer to write to (it must be AAD_SIZE long)

Returns

number of written bytes

See also

AAD_SIZE

Definition at line 170 of file secure_socket_wrapper.cpp.

6.23.2.8 readPartMsg() `Message * SecureSocketWrapper::readPartMsg ( )`

Read any new data from the socket but does not wait for the whole message to be ready.

This does not decrypt the message!

This API is blocking iff socket was not ready.

Parameters

size	the size of the temporary buffer

Returns

the received message or null if an error occurred

Definition at line 175 of file secure_socket_wrapper.cpp.

6.23.2.9 receiveAnyMsg() `Message * SecureSocketWrapper::receiveAnyMsg ( )`

Receive any new message from the socket.

This API is blocking.

Returns

the received message or null if an error occurred

Definition at line 180 of file secure_socket_wrapper.cpp.

6.23.2.10 receiveMsg() [1/2] `Message * SecureSocketWrapper::receiveMsg ( MessageType type )`

Receive a new message of the given type from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

Parameters

type	the type to keep

Returns

the received message or null if an error occurred

Definition at line 534 of file secure_socket_wrapper.cpp.

6.23.2.11 receiveMsg() [2/2]

```
Message * SecureSocketWrapper::receiveMsg (
    MessageType type[],
    int n_types )
```

Receive a new message of any of the given types from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

Parameters

type	the types to keep (array)
n_types	the number of types to keep (array length)

Returns

the received message or null if an error occurred

Definition at line 538 of file secure_socket_wrapper.cpp.

6.23.2.12 sendMsg()

```
int SecureSocketWrapper::sendMsg (
    Message * msg )
```

Sends the given message to the peer host through the socket.

Parameters

msg	the message to be sent

Returns

0 in case of success, something else otherwise

Definition at line 282 of file secure_socket_wrapper.cpp.

6.23.2.13 setOtherAddr()

```
void SecureSocketWrapper::setOtherAddr (
    struct sockaddr_in addr )  [inline]
```

Sets the address of the other host.

This is used when initializing a new SocketWrapper for a newly accepter connection.

Definition at line 257 of file secure_socket_wrapper.h.

6.23.2.14 updateRecvIV() `void SecureSocketWrapper::updateRecvIV ( ) [protected]`

Calculates the IV to use when receiving a new message.

Definition at line 486 of file secure_socket_wrapper.cpp.

6.23.2.15 updateSendIV() `void SecureSocketWrapper::updateSendIV ( ) [protected]`

Calculates the IV to use when sending the next message.

Definition at line 482 of file secure_socket_wrapper.cpp.

6.24 Server Class Reference

Utility class for interacting with the server.

`#include <server.h>`

Public Member Functions

- Server (SecureHost host, X509 *cert, EVP_PKEY *key, X509_STORE *store)
  Constructor.
- ~Server ()
  Destructor.
- int getServerCert ()
  Get and set the server certificate through a CERTIFICATE REQUEST (and then waits a CERTIFICATE)
- int registerToServer ()
  Registers the user in the server.
- string getUserList ()
  Returns the list of available users in the server as a comma separated list.
- int challengePeer (string username, SecureHost *peerHost)
  Challenges the given peer and wait for a reply.
- int replyPeerChallenge (string username, bool response, SecureHost *peerHost, uint16_t *listen_port)
  Replies to the challenge of another user.
- int signalGameEnd ()
  Signals the server that the user finished his game.
- void disconnect ()
  Disconnects from the server.
- SecureSocketWrapper * getSocketWrapper ()
  Returns the internal SocketWrapper.
- SecureHost getHost ()
  Returns the internal Host.
- string getPlayerUsername ()
  Returns the player username from his certificate.
- bool isConnected ()
  Returns whether server is connected.

6.24.1 Detailed Description

Utility class for interacting with the server.

Definition at line 22 of file server.h.

6.24.2 Member Function Documentation

6.24.2.1 challengePeer() `int Server::challengePeer ( string username, SecureHost * peerHost )`

Challenges the given peer and wait for a reply.

TODO: add timeout and possibility to interrupt ?

Parameters

username	the username of the peer to challenge
peerHost	a pointer to the structure that will be filled with the peer connection parameters in case the challenge is accepted

Returns

0 in case of accepted challenge

-1 in case of refused challenge

1 in case of connection failures

Definition at line 98 of file client/server.cpp.

6.24.2.2 getServerCert() `int Server::getServerCert ( )`

Get and set the server certificate through a CERTIFICATE REQUEST (and then waits a CERTIFICATE)

Returns

0 in case of success, 1 in case of error

Definition at line 24 of file client/server.cpp.

6.24.2.3 getUserList() `string Server::getUserList ( )`

Returns the list of available users in the server as a comma separated list.

Returns

the list of users.

Definition at line 65 of file client/server.cpp.

6.24.2.4 registerToServer() `int Server::registerToServer ( )`

Registers the user in the server.

This function also connects the socket if not already done.

Username is inferred from the certificate

Returns

0 in case of success, 1 in case of error

Definition at line 35 of file client/server.cpp.

6.24.2.5 replyPeerChallenge()

```
int Server::replyPeerChallenge (
        string username,
        bool response,
        SecureHost * peerHost,
        uint16_t * listen_port )
```

Replies to the challenge of another user.

Parameters

username	the username of the user that sent the challenge
response	the reply (true => accept)
peerHost	a pointer to the structure that will be filled with the peer connection parameters in case the challenge is accepted

Returns

0 in case of accepted challenge

-1 in case of refused challenge

1 in case of connection failures

Definition at line 140 of file client/server.cpp.

6.24.2.6 signalGameEnd() `int Server::signalGameEnd ( )`

Signals the server that the user finished his game.

Returns

0 in case of success

1 in case message could not be delivered

Definition at line 191 of file client/server.cpp.

6.25 ServerHelloMessage Class Reference

Inherits Message.

Public Member Functions

- **ServerHelloMessage** (EVP_PKEY *eph_key, nonce_t nonce, string my_id, string other_id, char *ds, uint32_t ds_size)
- MessageType **getType** ()
- string getName ()

  Get message name (for debug purposes)
- nonce_t **getNonce** ()
- EVP_PKEY * **getEphKey** ()
- void **setEphKey** (EVP_PKEY *eph_key)
- string **getMyId** ()
- string **getOtherId** ()
- char * **getDs** ()
- uint32_t **getDsSize** ()
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.

6.25.1 Detailed Description

Definition at line 451 of file messages.h.

6.25.2 Member Function Documentation

6.25.2.1 read()

```
msglen_t ServerHelloMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 521 of file messages.cpp.

6.25.2.2 write()

```
msglen_t ServerHelloMessage::write (
        char * buffer )  [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 486 of file messages.cpp.

6.26 ServerSecureSocketWrapper Class Reference

SocketWrapper for a TCP server.

`#include <secure_socket_wrapper.h>`

Inherits SecureSocketWrapper.

Public Member Functions

- ServerSecureSocketWrapper (X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store)
  Initialize a new socket on a random port.
- int bindPort (int port)
  Binds the socket to the requested port.
- int bindPort ()
  Binds the socket to a random port.
- SecureSocketWrapper * acceptClient ()
  Accepts any incoming connection and returns the related SocketWrapper.
- SecureSocketWrapper * acceptClient (X509 *other_cert)
  Accepts any incoming connection and returns the related SocketWrapper.
- int getPort ()
  Returns port the server is listening new connections on.

Additional Inherited Members

6.26.1 Detailed Description

SocketWrapper for a TCP server.

It provides a new function to accept clients. Constructor also set listen mode.

Definition at line 305 of file secure_socket_wrapper.h.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 ServerSecureSocketWrapper()

```
ServerSecureSocketWrapper::ServerSecureSocketWrapper (
        X509 * cert,
        EVP_PKEY * my_priv_key,
        X509_STORE * store )
```

Initialize a new socket on a random port.

Parameters

port	the port you want to bind on

Definition at line 575 of file secure_socket_wrapper.cpp.

6.26.3 Member Function Documentation

6.26.3.1 acceptClient()

```
SecureSocketWrapper * ServerSecureSocketWrapper::acceptClient (
        X509 * other_cert )
```

Accepts any incoming connection and returns the related SocketWrapper.

The certificate is set as the expected certificate of the peer.

Definition at line 586 of file secure_socket_wrapper.cpp.

6.26.3.2 bindPort() [1/2]

```
int ServerSecureSocketWrapper::bindPort ( )  [inline]
```

Binds the socket to a random port.

Returns

0 in case of success

1 otherwise

Definition at line 333 of file secure_socket_wrapper.h.

6.26.3.3 bindPort() [2/2] int ServerSecureSocketWrapper::bindPort (int *port*) [inline]

Binds the socket to the requested port.

Parameters

port	the port you want to bind on

Returns

0 in case of success

1 otherwise

Definition at line 325 of file secure_socket_wrapper.h.

6.27 ServerSocketWrapper Class Reference

SocketWrapper for a TCP server.

`#include <socket_wrapper.h>`

Inherits SocketWrapper.

Public Member Functions

- int bindPort (int port)
  Binds the socket to the requested port.
- int bindPort ()
  Binds the socket to a random port.
- SocketWrapper * acceptClient ()
  Accepts any incoming connection and returns the related SocketWrapper.
- int getPort ()
  Returns port the server is listening new connections on.

Additional Inherited Members

6.27.1 Detailed Description

SocketWrapper for a TCP server.

It provides a new function to accept clients. Constructor also set listen mode.

Definition at line 154 of file socket_wrapper.h.

6.27.2 Member Function Documentation

6.27.2.1 bindPort() [1/2] int ServerSocketWrapper::bindPort ( )

Binds the socket to a random port.

Returns

0 in case of success

1 otherwise

Definition at line 204 of file socket_wrapper.cpp.

6.27.2.2 bindPort() [2/2] int ServerSocketWrapper::bindPort (int *port*)

Binds the socket to the requested port.

Parameters

port	the port you want to bind on

Returns

0 in case of success

1 otherwise

Definition at line 220 of file socket_wrapper.cpp.

6.28 SocketWrapper Class Reference

Wrapper class around sockaddr_in and socket descriptor.

`#include <socket_wrapper.h>`

Inherited by ClientSocketWrapper, and ServerSocketWrapper.

Public Member Functions

- SocketWrapper ()
  Initialize on a new socket.
- SocketWrapper (int sd)
  Initialize using existing socket.
- int getDescriptor ()
  Returns current socket file descriptor.
- Message * readPartMsg ()
  Read any new data from the socket but does not wait for the whole message to be ready.
- Message * receiveAnyMsg ()
  Receive any new message from the socket.

- Message * receiveMsg (MessageType type)

  Receive a new message of the given type from the socket.
- Message * receiveMsg (MessageType type[ ], int n_types)

  Receive a new message of any of the given types from the socket.
- int sendMsg (Message *msg)

  Sends the given message to the peer host through the socket.
- void closeSocket ()

  Closes the socket.
- void setOtherAddr (struct sockaddr_in addr)

  Sets the address of the other host.
- sockaddr_in * **getOtherAddr** ()
- Host getConnectedHost ()

  Returns connected host.

Protected Attributes

- struct sockaddr_in other_addr

  Other host inet socket.
- int socket_fd

  Socket file descriptor.
- char buffer_in [MAX_MSG_SIZE]

  Pre-allocated buffer for incoming messages.
- char buffer_out [MAX_MSG_SIZE]

  Pre-allocated buffer for outgoing messages.
- msglen_t buf_idx

  Index in the buffer that has been read up to now.

6.28.1 Detailed Description

Wrapper class around sockaddr_in and socket descriptor.

It provides a more simple interface saving a lot of boiler-plate code. There are two subclasses: ClientSocketWrapper and ServerSocketWrapper.

Definition at line 25 of file socket_wrapper.h.

6.28.2 Member Function Documentation

6.28.2.1 readPartMsg() `Message * SocketWrapper::readPartMsg ( )`

Read any new data from the socket but does not wait for the whole message to be ready.

This API is blocking iff socket was not ready.

Parameters

size	the size of the temporary buffer

Returns

the received message or null if an error occurred

Definition at line 24 of file socket_wrapper.cpp.

6.28.2.2 receiveAnyMsg() `Message * SocketWrapper::receiveAnyMsg ( )`

Receive any new message from the socket.

This API is blocking.

Returns

the received message or null if an error occurred

Definition at line 82 of file socket_wrapper.cpp.

6.28.2.3 receiveMsg() [1/2] `Message * SocketWrapper::receiveMsg ( MessageType type )`

Receive a new message of the given type from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

Parameters

type	the type to keep

Returns

the received message or null if an error occurred

Definition at line 123 of file socket_wrapper.cpp.

6.28.2.4 receiveMsg() [2/2] `Message * SocketWrapper::receiveMsg ( MessageType type[], int n_types )`

Receive a new message of any of the given types from the socket.

When a message of the wrong type is received it is simply ignored.

This API is blocking.

Parameters

type	the types to keep (array)
n_types	the number of types to keep (array length)

Returns

the received message or null if an error occurred

Definition at line 127 of file socket_wrapper.cpp.

6.28.2.5 sendMsg() `int SocketWrapper::sendMsg ( Message * msg )`

Sends the given message to the peer host through the socket.

Parameters

msg	the message to be sent

Returns

0 in case of success, something else otherwise

Definition at line 150 of file socket_wrapper.cpp.

6.28.2.6 setOtherAddr() `void SocketWrapper::setOtherAddr ( struct sockaddr_in addr ) [inline]`

Sets the address of the other host.

This is used when initializing a new SocketWrapper for a newly accepter connection.

Definition at line 123 of file socket_wrapper.h.

6.29 StartGameMessage Class Reference

Message that signals to start a new game.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- msglen_t write (char *buffer)
  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)
  Read message from buffer.
- string getName ()
  Get message name (for debug purposes)
- MessageType **getType** ()

6.29.1 Detailed Description

Message that signals to start a new game.

Definition at line 130 of file messages.h.

6.29.2 Member Function Documentation

6.29.2.1 read() msglen_t StartGameMessage::read (char * *buffer*, msglen_t *len*) [virtual]

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 103 of file messages.cpp.

6.29.2.2 write() msglen_t StartGameMessage::write (char * *buffer*) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 92 of file messages.cpp.

6.30 User Class Reference

Class representing a user.

`#include <user.h>`

Public Member Functions

- User (SecureSocketWrapper *sw)
  Contructor.
- ~User ()
  Destructor.
- void lock ()
  Locks the user instance unsing the internal mutex.
- void unlock ()
  Unlocks the user instance unsing the internal mutex.
- string getUsername ()
  Returns the username.
- void setUsername (string username)
  Sets the username.
- SecureSocketWrapper * getSocketWrapper ()
  Returns the socket wrapper.
- UserState getState ()
  Returns the current state of the user.
- void setState (UserState state)
  Sets the current state of the user.
- string getOpponent ()
  Returns the username of the opponent.
- void setOpponent (string opponent)
  Sets the username of the opponent.
- int countRefs ()
  Returns the reference count.

6.30.1 Detailed Description

Class representing a user.

Always lock() before using an instance and unlock() afterwards. In order to prevent deadlocks, take locks in alphabetical order of username.

Limitations:

- a user may be challenged by only another user at a time

Definition at line 46 of file user.h.

6.30.2 Constructor & Destructor Documentation

6.30.2.1 User() `User::User ( SecureSocketWrapper * sw ) [inline]`

Contructor.

The user is put in the JUST_CONNECTED STATE, username is set to empty string.

Definition at line 82 of file user.h.

6.30.2.2 ~User() `User::~User ( ) [inline]`

Destructor.

The socket_wrapper is deleted.

Definition at line 94 of file user.h.

6.31 UserList Class Reference

Class that manages the users.

`#include <user_list.h>`

Public Member Functions

- UserList ()
  Initializes an empty list.
- bool add (User *u)
  Inserts a new user to the internal hash maps.
- User * get (string username)
  Returns the user matching the given username.
- User * get (int fd)
  Returns the user matching the given file descriptor.
- bool exists (string username)
  Checks whether there is a user matching the given username.
- bool exists (int fd)
  Checks whether there is a user matching the given file descriptor.
- void yield (User *u)
  Signals that the given user is no longer being used.
- string listAvailableFromTo (int from)
  Returns a comma separated list of users in the AVAILABLE state, starting from the given offset.
- int size ()
  Returns the number of all users in the list.

6.31.1 Detailed Description

Class that manages the users.

1) keeps track of connected users through two hash maps (one by username) and one by file descriptor of the socket the user is connected to. 2) updates reference count of the users in order to safely delete disconnected users only once no thread holds a reference to it. 3) disposes of disconnected users that are no longer referenced by any
thread.

Every method in this class is protected against concurrent modifications by a mutex.

The maximum number of users is defined by MAX_USERS.

Definition at line 38 of file user_list.h.

6.31.2 Member Function Documentation

6.31.2.1 add() `bool UserList::add (User * u )`

Inserts a new user to the internal hash maps.

The user may not have a username but must have a file descriptor.

Calling this function does not increase the reference count of user since it's assumed that either the reference is already held or that the user will no longer be needed by the calling thread. In case this is not true, call also one of the get functions to correctly update the reference count.

Parameters

u	the user to be added

Returns

true in case of success, false otherwise (e.g. full)

Definition at line 26 of file user_list.cpp.

6.31.2.2 exists() [1/2] `bool UserList::exists (int fd )`

Checks whether there is a user matching the given file descriptor.

Parameters

fd	the file descriptor to be checked

Returns

true if exists, false otherwise

Definition at line 80 of file user_list.cpp.

6.31.2.3 exists() [2/2] bool UserList::exists (string *username*)

Checks whether there is a user matching the given username.

Parameters

username	the username to be checked

Returns

true if exists, false otherwise

Definition at line 72 of file user_list.cpp.

6.31.2.4 get() [1/2] User * UserList::get (int *fd*)

Returns the user matching the given file descriptor.

The reference count of the user is increased.

Parameters

fd	the file descriptor of the user to be retrieved

Returns

a pointer to the requested user or NULL in case it is not found.

Definition at line 58 of file user_list.cpp.

6.31.2.5 get() [2/2] User * UserList::get (string *username*)

Returns the user matching the given username.

The reference count of the user is increased.

Parameters

username	the username of the user to be retrieved

Returns

a pointer to the requested user or NULL in case it is not found.

Definition at line 44 of file user_list.cpp.

6.31.2.6 listAvailableFromTo() `string UserList::listAvailableFromTo ( int from )`

Returns a comma separated list of users in the AVAILABLE state, starting from the given offset.

Parameters

from	the offset

Returns

the comma separated list of users as a string

Definition at line 110 of file user_list.cpp.

6.31.2.7 yield() `void UserList::yield ( User * u )`

Signals that the given user is no longer being used.

The reference count of the user is decreased. If the reference count is 0 and the user is disconnected, the user is deleted.

NB: the deletion of a user may put sockets in an inconsistent state that must be fixed by the user of this class

Parameters

u	the user whose reference is being marked as no longer used

Definition at line 88 of file user_list.cpp.

6.32 UsersListMessage Class Reference

Message that the server sends the client with the list of users.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **UsersListMessage** (string usernames)
- msglen_t write (char *buffer)
  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)
  Read message from buffer.
- string getName ()
  Get message name (for debug purposes)
- string **getUsernames** ()
- MessageType **getType** ()

6.32.1 Detailed Description

Message that the server sends the client with the list of users.

Definition at line 234 of file messages.h.

6.32.2 Member Function Documentation

6.32.2.1 read()

```
msglen_t UsersListMessage::read (
        char * buffer,
        msglen_t len ) [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 219 of file messages.cpp.

6.32.2.2 write()

```
msglen_t UsersListMessage::write (
        char * buffer ) [virtual]
```

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 199 of file messages.cpp.

6.33 UsersListRequestMessage Class Reference

Message with which the client asks for the list of connected users.

`#include <messages.h>`

Inherits Message.

Public Member Functions

- **UsersListRequestMessage** (unsigned int offset)
- msglen_t write (char *buffer)

  Write message to buffer.
- msglen_t read (char *buffer, msglen_t len)

  Read message from buffer.
- string getName ()

  Get message name (for debug purposes)
- uint32_t **getOffset** ()
- MessageType **getType** ()

6.33.1 Detailed Description

Message with which the client asks for the list of connected users.

Definition at line 257 of file messages.h.

6.33.2 Member Function Documentation

6.33.2.1 read()

```
msglen_t UsersListRequestMessage::read (
        char * buffer,
        msglen_t len )  [virtual]
```

Read message from buffer.

Returns

0 in case of success, something else in case of errors. Refer to the implementation for details

Implements Message.

Definition at line 244 of file messages.cpp.

6.33.2.2 write() msglen_t UsersListRequestMessage::write (char * *buffer*) [virtual]

Write message to buffer.

Returns

>0 number of bytes written

0 in case of errors

Implements Message.

Definition at line 229 of file messages.cpp.

7 File Documentation

7.1 args.cpp File Reference

Implementation of the Args class.

```
#include "utils/args.h"
#include <sstream>
```

Functions

- ostream & **operator**<< (ostream &os, const Args &a)

7.1.1 Detailed Description

Implementation of the Args class.

Author

Riccardo Mancini

Adapted from https://stackoverflow.com/a/14266139

Date

2020-05-27

Definition in file args.cpp.

7.2 args.cpp

```
00001
00012 #include "utils/args.h"
00013 #include <sstream>
00014
00015 using namespace std;
00016
00017 void Args::parseLine(string s){
00018     string delimiter = " ";
00019     size_t pos = 0;
00020     string token;
00021
00022     while ((pos = s.find(delimiter)) != string::npos) {
00023         token = s.substr(0, pos);
00024         argv.push_back(token);
00025         s.erase(0, pos + delimiter.length());
00026     }
00027
00028     if (!s.empty()){
00029         argv.push_back(s);
00030     }
00031 }
00032
00033 Args::Args(char* line){
00034     parseLine(string(line));
00035 }
00036
00037 Args::Args(istream &is){
00038     string line;
00039     getline(is, line);
00040     if (!is){
00041         status = 1;
00042     } else {
00043         status = 0;
00044         parseLine(line);
00045     }
00046 }
00047
00048 ostream& operator«(ostream& os, const Args& a){
00049     os « "[";
00050
00051     for (vector<string>::const_iterator it = a.argv.begin(); it != a.argv.end(); it++){
00052         os « *it;
00053         if (it != a.argv.end()-1)
00054             os « ",";
00055     }
00056
00057     os « "]";
00058     return os;
00059 }
00060
00061 const char* Args::c_str(){
00062     ostringstream os;
00063     os « *this;
00064     return os.str().c_str();
00065 }
```

7.3 args.h File Reference

Definition of the Args class.

```
#include <cstring>
#include <string>
#include <list>
#include <iostream>
#include <vector>
```

Data Structures

- class Args

  Utility class that parses an input line into a list of arguments (argc, argv)

7.3.1 Detailed Description

Definition of the Args class.

Author

Riccardo Mancini

Date

2020-05-27

Definition in file args.h.

7.4 args.h

```
00001
00010 #ifndef ARGS_H
00011 #define ARGS_H
00012
00013 #include <cstring>
00014 #include <string>
00015 #include <list>
00016 #include <iostream>
00017 #include <vector>
00018
00019 using namespace std;
00020
00024 class Args {
00025 private:
00026     int status;
00027     vector<string> argv;
00028
00029     void parseLine(string s);
00030 public:
00034     Args() : status(0) {}
00035
00039     Args(char* line);
00040
00044     Args(std::istream &is);
00045
00051     friend std::ostream& operator<<(std::ostream& os, const Args& b);
00052
00059     int getArgc(){ return status == 0 ? argv.size() : -1;}
00060
00064     const char* getArgv(unsigned int i) {
00065         if (status == 0 && i < argv.size())
00066             return argv.at(i).c_str();
00067         else
00068             return NULL;
00069     }
00070
00074     const char* c_str();
00075 };
00076
00077 #endif //PARSE_ARGS_H
```

7.5 client.cpp File Reference

Implementation of a 4-in-a-row game.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include "connect4.h"
#include "logging.h"
#include "network/socket_wrapper.h"
```

```
#include "network/host.h"
#include "utils/args.h"
#include "single_player.h"
#include "multi_player.h"
#include "connection_mode.h"
#include "server_lobby.h"
#include "security/crypto.h"
#include "server.h"
```

Functions

- void print_help (char *argv0)

  Prints command usage information.
- void **printWelcome** ()
- struct ConnectionMode **promptChooseConnection** ()
- int **main** (int argc, char **argv)

7.5.1 Detailed Description

Implementation of a 4-in-a-row game.

Author

Mirko Laruina

Date

2020-05-14

Definition in file client.cpp.

7.6 client.cpp

```
00001
00009 #include <iostream>
00010 #include <cstdlib>
00011 #include <ctime>
00012 #include "connect4.h"
00013 #include "logging.h"
00014 #include "network/socket_wrapper.h"
00015 #include "network/host.h"
00016 #include "utils/args.h"
00017 #include "single_player.h"
00018 #include "multi_player.h"
00019 #include "connection_mode.h"
00020 #include "server_lobby.h"
00021 #include "security/crypto.h"
00022 #include "server.h"
00023
00024 using namespace std;
00025
00026 static const char players[] = {'X', 'O'};
00027
00031 void print_help(char* argv0){
00032   cout«"Usage: "«argv0«" cert.pem key.pem cacert.pem crl.pem [other_cert.pem]"«endl;
00033 }
00034
00035 void printWelcome(){
00036     cout«"*****************************************************************\n"
00037         «"*        __ __     _                                            *\n"
00038         «"*       / // /    (_)___     ____ _   _________ _      __      *\n"
```

```
00039         «"*      / // /_   / / __ \\   / __ `/  / ___/ __ \\ | /| / /      *\n"
00040         «"*     /__  __/  / / / / /  / /_/ /  / /  / /_/ / |/ |/ /       *\n"
00041         «"*       /_/    /_/_/ /_/   \\__,_/  /_/   \\____/|__/|__/        *\n"
00042         «"*                                                               *\n"
00043         «"*****************************************************************"
00044         «endl;
00045 }
00046
00047 struct ConnectionMode promptChooseConnection(){
00048     cout«"You can connect to a server, wait for a peer or connect to a peer"« endl;
00049     cout«"To connect to a server type: `server host port [path/to/server_cert.pem]`"« endl;
00050     cout«"To connect to a peer type: `peer host port path/to/peer_cert.pem`"« endl;
00051     cout«"To wait for a peer type: `peer listen_port path/to/peer_cert.pem`"« endl;
00052     cout«"To play offline type: `offline`"« endl;
00053     cout«"To exit type: `exit`"« endl;
00054
00055     do {
00056         cout«"> "«flush;
00057         Args args(cin);
00058         if (args.getArgc() == 3 && strcmp(args.getArgv(0), "peer") == 0){
00059             X509* cert = load_cert_file(args.getArgv(2));
00060             char dummy_ip[] = "127.0.0.1";
00061             return ConnectionMode(WAIT_FOR_PEER, dummy_ip,
00062                                   0, cert, atoi(args.getArgv(1)));
00063
00064         } else if (args.getArgc() == 4 && strcmp(args.getArgv(0), "peer") == 0){
00065             X509* cert = load_cert_file(args.getArgv(3));
00066             return ConnectionMode(CONNECT_TO_PEER, args.getArgv(1),
00067                                   atoi(args.getArgv(2)), cert, 0);
00068
00069         } else if (args.getArgc() >= 3 && strcmp(args.getArgv(0), "server") == 0){
00070             X509* cert;
00071             if(args.getArgc() == 4)
00072                 cert = load_cert_file(args.getArgv(3));
00073             else
00074                 cert = NULL;
00075             return ConnectionMode(CONNECT_TO_SERVER, args.getArgv(1),
00076                                   atoi(args.getArgv(2)), cert, 0);
00077
00078         } else if (args.getArgc() == 1 && strcmp(args.getArgv(0), "offline") == 0){
00079             return ConnectionMode(SINGLE_PLAYER);
00080
00081         } else if (args.getArgc() == 1 && strcmp(args.getArgv(0), "exit") == 0){
00082             cout « "Bye" « endl;
00083             return ConnectionMode(EXIT, OK);
00084         } else if (args.getArgc() == 0){
00085             return ConnectionMode(CONTINUE);
00086         } else if (args.getArgc() == -1){ // EOF
00087             cout « "Bye" « endl;
00088             return ConnectionMode(EXIT, OK);
00089         }else{
00090             cout « "Could not parse arguments: "« args « endl;
00091         }
00092     } while (true);
00093 }
00094
00095
00096 int main(int argc, char** argv){
00097     SecureSocketWrapper *sw = NULL;
00098     Server* server = NULL;
00099
00100     if (argc < 5){
00101         print_help(argv[0]);
00102         return 1;
00103     }
00104
00105     X509* cert = load_cert_file(argv[1]);
00106     X509* cacert = load_cert_file(argv[3]);
00107     X509_CRL* crl = load_crl_file(argv[4]);
00108     X509_STORE* store = build_store(cacert, crl);
00109
00110     srand(time(NULL));
00111
00112     int ret;
00113
00114     printWelcome();
00115     cout«endl«"Welcome to 4-in-a-row!"«endl;
00116     cout«"The rules of the game are simple: you win when you have 4 connected tokens along any
       direction."«endl;
00117
00118     EVP_PKEY* key;
00119     do {
00120         key = load_key_file(argv[2], NULL);
00121         if( key == NULL ){
00122             cout«"Wrong password"«endl;
00123         }
00124     } while(!key);
```

```
00125     cout<<endl;
00126
00127     do{
00128         struct ConnectionMode ucc = promptChooseConnection();
00129
00130         if (ucc.connection_type == EXIT && ucc.exit_code == OK){
00131             exit(0); // Bye
00132         }
00133
00134         if (ucc.connection_type == CONNECT_TO_SERVER){
00135             server = new Server(ucc.host, cert, key, store);
00136         }
00137
00138         bool loopLobby = true;
00139
00140         do{
00141             try{
00142                 if (server != NULL){
00143                     ucc = serverLobby(server);
00144                 }
00145
00146                 switch(ucc.connection_type){
00147                     case WAIT_FOR_PEER:
00148                         sw = waitForPeer(ucc.listen_port, ucc.host, cert, key, store);
00149                         if (sw != NULL)
00150                             ret = playWithPlayer(MY_TURN, sw);
00151                         else
00152                             ret = CONNECTION_ERROR;
00153
00154                         loopLobby = true;
00155                         break;
00156                     case CONNECT_TO_PEER:
00157                         sw = connectToPeer(ucc.host, cert, key, store);
00158                         if (sw != NULL)
00159                             ret = playWithPlayer(THEIR_TURN, sw);
00160                         else
00161                             ret = CONNECTION_ERROR;
00162
00163                         loopLobby = true;
00164                         break;
00165                     case SINGLE_PLAYER:
00166                         ret = playSinglePlayer();
00167                         loopLobby = false;
00168                         break;
00169                     case EXIT:
00170                         ret = ucc.exit_code;
00171                         loopLobby = false;
00172                         break;
00173                     case CONNECT_TO_SERVER:
00174                         ret = FATAL_ERROR;
00175                         loopLobby = false;
00176                         break;
00177                     case CONTINUE:
00178                         ret = OK;
00179                         loopLobby = true;
00180                         break;
00181                 }
00182
00183                 if (loopLobby && ret == OK &&
00184                     server != NULL && server->isConnected()
00185                 ){
00186                     server->signalGameEnd();
00187                 }
00188             } catch(const char* error_msg){
00189                 LOG(LOG_ERR, "Caught error: %s", error_msg);
00190                 ret = GENERIC_ERROR;
00191                 loopLobby = false;
00192             }
00193
00194         } while (loopLobby && server != NULL && server->isConnected());
00195         if (server != NULL){
00196             delete server;
00197             server = NULL;
00198         }
00199     } while(ret != FATAL_ERROR);
00200
00201     return ret;
00202 }
```

7.7 connect4.cpp File Reference

Implementation of connect4.h.

```
#include "connect4.h"
```

Functions

- ostream & **operator**<< (ostream &os, const Connect4 &c)

7.7.1 Detailed Description

Implementation of connect4.h.

Author

Mirko Laruina

Date

2020-05-14

See also

connect4.h

Definition in file connect4.cpp.

7.8 connect4.cpp

```
00001
00012 #include "connect4.h"
00013 using namespace std;
00014
00015 void Connect4::print(ostream& os){
00016     os«*this;
00017 }
00018
00019 Connect4::Connect4(int rows /* = 6 */, int columns /* = 7 */){
00020     rows_ = rows;
00021     cols_ = columns;
00022     size_ = rows*columns;
00023     full_ = false;
00024
00025     //Maybe check for overflow if we will use different board values
00026
00027     cells_ = new char[size_];
00028     memset(cells_, 0, size_);
00029 }
00030
00031 int8_t Connect4::play(int col, char player){
00032     // bool col_full = true;
00033     if(player == 0){
00034         player = player_;
00035     }
00036
00037     //Trying to play with a full board
00038     if(full_){
00039         return -2;
00040     }
00041
00042     for(int i = rows_-1; i>=0; --i){
00043         if(cells_[i*cols_+col] == 0){
00044             // col_full = false;
00045             cells_[i*cols_+col] = player;
00046             if( checkWin(i, col, player) ){
00047                 return 1;
00048             } else {
```

```
00049                 //All the board could be full now
00050                 if(i == 0 && checkFullTopRow()){
00051                     full_ = true;
00052                     return -2;
00053                 } else {
00054                     return 0;
00055                 }
00056             }
00057         }
00058     }
00059
00060     //We are sure the board is not full, otherwise we would have already exited
00061     //If a play was possible, we would have already exited too
00062     //Only possible case is full column
00063     return -1;
00064 }
00065
00066 int Connect4::countNexts(char player, int row, int col, int di, int dj){
00067     int count = 0;
00068     for(
00069         int i = row+di, j = col+dj;
00070         i >= 0 && j >= 0 && i < rows_ && j < cols_;
00071         i+=di, j+=dj)
00072     {
00073         if(cells_[i*cols_+j] != player){
00074             break;
00075         } else {
00076             LOG(LOG_DEBUG, "%d %d", i, j);
00077             count++;
00078         }
00079     }
00080     return count;
00081 }
00082
00083 bool Connect4::checkWin(int row, int col, char player){
00084     /*
00085         Take any of the 4 possible directions
00086         count how many token of the same player there are
00087         before and after the new one
00088         if more than 4, declare win
00089     */
00090
00091     LOG(LOG_DEBUG, "Checking (%d, %d)", row, col);
00092     if(player == 0){
00093         player = player_;
00094     }
00095
00096     for(int di = 1; di >= 0 && di != -1; --di){
00097         for(int dj = 1; dj >= 0 && di != -1; --dj){
00098             // direction (0, 0) is useless, since we would miss diagonal (-1, 1)
00099             // we can exploit the loop to iterate over that
00100             if(di == 0 && dj == 0){
00101                 di = -1;
00102                 dj = 1;
00103             }
00104
00105             int count_forward = Connect4::countNexts(player, row, col, di, dj);
00106             int count_backward = Connect4::countNexts(player, row, col, -di, -dj);
00107
00108             // N_IN_A_ROW minus 1 since the token just inserted is excluded
00109             if(count_forward + count_backward >= (N_IN_A_ROW - 1)){
00110                 return true;
00111             }
00112         }
00113     }
00114
00115     return false;
00116
00117 }
00118
00119 bool Connect4::checkFullTopRow(){
00120     for(int j = 0; j<cols_; ++j){
00121         if(cells_[j] == 0){
00122             return false;
00123         }
00124     }
00125     return true;
00126 }
00127
00128 int Connect4::getNumCols(){
00129     return cols_;
00130 }
00131
00132 bool Connect4::setPlayer(char player){
00133     if(player == 'X' || player == 'x'
00134         || player == 'O' || player == 'o'){
00135         player_ = toupper(player);
```

```
00136         adversary_ = player_ == 'X' ? 'O' : 'X';
00137         return true;
00138     }
00139     return false;
00140 }
00141
00142 char Connect4::getPlayer(){
00143     return player_;
00144 }
00145
00146 char Connect4::getAdv(){
00147     return adversary_;
00148 }
00149
00150 ostream& operator<<(ostream& os, const Connect4& c){
00151     int width = 2+3*(c.rows_+1);
00152     for(int i = 0; i<width; ++i){
00153         os<<'*';
00154     }
00155     os<<endl;
00156
00157     for(int i = 0; i<c.rows_; ++i){
00158         os<<"*";
00159         for(int j = 0; j<c.cols_; ++j){
00160             if(c.cells_[i*c.cols_+j] == 0){
00161                 os<<"   ";
00162             } else {
00163                 os<< " " << (c.cells_[i*c.cols_+j] == 'X' ? "\033[31mX" : "\033[34mO") <<" ";
00164             }
00165         }
00166         os<<"\033[0m*"<<endl;
00167     }
00168
00169     for(int i = 0; i<width; ++i){
00170         os<<'*';
00171     }
00172     os<<endl;
00173
00174     for(int i = 0; i<width; ++i){
00175         if( (i+1)%3 == 0  ){
00176             os<<(i+1)/3;
00177         } else {
00178             os<<" ";
00179         }
00180     }
00181     os<<endl;
00182     return os;
00183 }
```

7.9 connect4.h File Reference

Header file for the class responsible of handling the board of a Connect4 game.

```
#include <iostream>
#include <cstring>
#include "config.h"
#include "logging.h"
```

Data Structures

- class Connect4

7.9.1 Detailed Description

Header file for the class responsible of handling the board of a Connect4 game.

Author

Mirko Laruina

Date

2020-05-14

Definition in file connect4.h.

7.10 connect4.h

```
00001
00010 #ifndef CONNECT4_H
00011 #define CONNECT4_H
00012 #include <iostream>
00013 #include <cstring>
00014 #include "config.h"
00015 #include "logging.h"
00016
00017
00018 class Connect4 {
00020     int rows_, cols_, size_;
00021
00023     bool full_;
00024
00026     char* cells_;
00027
00029     char player_;
00030
00032     char adversary_;
00033
00046     int countNexts(char player, int row, int col, int di, int dj);
00047
00054     bool checkFullTopRow();
00055     public:
00056
00063     Connect4(int rows = 6, int columns = 7);
00064
00070     int getNumCols();
00071
00083     int8_t play(int column, char player = 0);
00084
00094     bool checkWin(int starting_row, int starting_col, char player = 0);
00095
00103     bool setPlayer(char player);
00104
00110     char getPlayer();
00111
00117     char getAdv();
00118
00124     void print(std::ostream& os);
00125
00126     friend std::ostream& operator<<(std::ostream& os, const Connect4& b);
00127 };
00128 #endif //CONNECT4_H
```

7.11 connection_mode.h File Reference

Header file for utility structure ConnectionMode.

```
#include "security/secure_host.h"
```

Data Structures

- struct ConnectionMode

  Structure holding information about the connection requested by the user.

Enumerations

- enum ConnectionType {
  CONNECT_TO_SERVER, **CONNECT_TO_PEER**, **WAIT_FOR_PEER**, **SINGLE_PLAYER**,
  EXIT, **CONTINUE** }

  Type of gmae connection requested by the user: CONNECT_TO_SERVER: the user connects to the given server (Host) that manages users and forwards challenges between users.
- enum **ExitCode** { **OK**, **CONNECTION_ERROR**, **GENERIC_ERROR**, **FATAL_ERROR** }

7.11.1 Detailed Description

Header file for utility structure ConnectionMode.

Author

Riccardo Mancini

Date

2020-05-29

Definition in file connection_mode.h.

7.11.2 Enumeration Type Documentation

7.11.2.1 ConnectionType `enum ConnectionType`

Type of gmae connection requested by the user: CONNECT_TO_SERVER: the user connects to the given server (Host) that manages users and forwards challenges between users.

CONNECT_TO_PEER: the user directly connects to the given peer (Host) for a game. WAIT_FOR_PEER: the user waits for requests from other peers on the given port and accepts any incoming game. SINGLE_PLAYER: the player plays against an AI (tbf it's random). EXIT: (used internally) exit the game with the given return code.

Definition at line 25 of file connection_mode.h.

7.12 connection_mode.h

```
00001
00009 #ifndef CONNECTION_MODE_H
00010 #define CONNECTION_MODE_H
00011
00012 #include "security/secure_host.h"
00013
00025 enum ConnectionType {CONNECT_TO_SERVER, CONNECT_TO_PEER, WAIT_FOR_PEER, SINGLE_PLAYER, EXIT,
       CONTINUE};
00026
00027 enum ExitCode {OK, CONNECTION_ERROR, GENERIC_ERROR, FATAL_ERROR};
00028
00034 struct ConnectionMode {
00035     enum ConnectionType connection_type;
00036     SecureHost host;
00037     union{
00038         uint16_t listen_port;
00039         enum ExitCode exit_code;
00040     };
00041     ConnectionMode(enum ConnectionType connection_type,
00042                    const char* ip, int port, X509* cert, uint16_t listen_port)
00043             : connection_type(connection_type), host(SecureHost(ip, port, cert)),
       listen_port(listen_port) {}
00044     ConnectionMode(enum ConnectionType connection_type, SecureHost host, uint16_t listen_port)
00045             : connection_type(connection_type), host(host), listen_port(listen_port) {}
00046
00047     ConnectionMode(enum ConnectionType connection_type, enum ExitCode exit_code)
00048             : connection_type(connection_type), exit_code(exit_code) {}
00049
00050     ConnectionMode(enum ConnectionType connection_type)
00051             : connection_type(connection_type) {}
00052
00053 };
00054
00055 #endif //CONNECTION_MODE_H
```

7.13 crypto.h File Reference

Header for crypto algorithms.

```
#include <openssl/conf.h>
#include <openssl/evp.h>
#include <openssl/err.h>
#include <openssl/rand.h>
#include <openssl/pem.h>
#include <openssl/hmac.h>
#include <openssl/kdf.h>
#include <string.h>
#include "logging.h"
```

Macros

- #define TAG_SIZE 16

  AES-256 GCM.
- #define **IV_SIZE** 12
- #define **KEY_SIZE** 16
- #define handleErrorsNoException(level)

  Print OpenSSL errors.
- #define handleErrors()

  Print OpenSSL errors and throw exception.

Typedefs

- typedef uint32_t **nonce_t**

Functions

- int aes_gcm_encrypt (char *plaintext, int plaintext_len, char *aad, int aad_len, char *key, char *iv, char *ciphertext, char *tag)

  Encrypts using AES in GCM mode.
- int aes_gcm_decrypt (char *ciphertext, int ciphertext_len, char *aad, int aad_len, char *key, char *iv, char *plaintext, char *tag)

  Decrypts using AES in GCM mode.
- int get_ecdh_key (EVP_PKEY **key)

  Generate a ECDH key.
- int dhke (EVP_PKEY *my_key, EVP_PKEY *peer_pubkey, char **shared_key)

  Apply the DHKE to derive a shared secret.
- nonce_t get_rand ()

  Get a random number.
- void get_rand (char *buffer, int bytes)

  Fills a buffer with a random value.
- X509 * load_cert_file (const char *file_name)

  Load a certificate from file.
- X509_CRL * load_crl_file (const char *file_name)

  Load certificate revocation list from file.
- EVP_PKEY * load_key_file (const char *file_name, const char *password)

  Load a key from file.
- X509_STORE * build_store (X509 *cacert, X509_CRL *crl)

  Build a CA store from CA certificate and CRL.
- bool verify_peer_cert (X509_STORE *store, X509 *cert)
- int hmac (char *msg, int msg_len, char *key, unsigned int keylen, char *hmac)

  Calculate HMAC of the msg.
- bool compare_hmac (char *hmac_expected, char *hmac_rcv, unsigned int len)

  Compare two HMAC in a secure way.
- void hkdf_one_info (char *key, size_t key_len, char *info, size_t info_len, char *out, size_t outlen)

  Apply HKDF, takes only one info field.
- void hkdf (char *key, size_t key_len, nonce_t nonce1, nonce_t nonce2, char *label, char *out, size_t outlen)

  Apply HKDF, takes two nonces and a label field.
- int dsa_sign (char *msg, int msglen, char **signature, EVP_PKEY *prvkey)

  Signs the given message.
- bool dsa_verify (char *msg, int msglen, char *signature, int sign_len, EVP_PKEY *pkey)

  Checks the given signature on the given message.

7.13.1 Detailed Description

Header for crypto algorithms.

Header for crypto utilities.

Author

Mirko Laruina

Date

2020-06-07

Author

Riccardo Mancini

Date

2020-06-07

Definition in file crypto.h.

7.13.2 Macro Definition Documentation

7.13.2.1 handleErrors `#define handleErrors( )`

Value:

```
{ \
handleErrorsNoException(LOG_ERR); \
throw "OpenSSL Error"; \
}
```

Print OpenSSL errors and throw exception.

Definition at line 43 of file crypto.h.

7.13.2.2 handleErrorsNoException `#define handleErrorsNoException( level )`

Value:

```
{ \
LOG((level), "OpenSSL Exception"); \
FILE* stream; \
if (level < LOG_ERR) \
    stream = stdout; \
else \
    stream = stderr; \
ERR_print_errors_fp(stream); \
}
```

Print OpenSSL errors.

Definition at line 30 of file crypto.h.

7.13.3 Function Documentation

7.13.3.1 aes_gcm_decrypt() `int aes_gcm_decrypt (`

```
char * ciphertext,
int ciphertext_len,
char * aad,
int aad_len,
char * key,
char * iv,
char * plaintext,
char * tag )
```

Decrypts using AES in GCM mode.

Parameters

ciphertext	buffer where the ciphertext is stored
ciphertext_len	length of said buffer
aad	additional authenticated data buffer
aad_len	length of said buffer
key	decryption key
iv	initialization vector
plaintext	buffer (already allocated) where the pt will be stored
tag	tag buffer

Return values

-1	on error
n	number of written bytes

Definition at line 58 of file src/security/crypto.cpp.

7.13.3.2 aes_gcm_encrypt()

```
int aes_gcm_encrypt (
        char * plaintext,
        int plaintext_len,
        char * aad,
        int aad_len,
        char * key,
        char * iv,
        char * ciphertext,
        char * tag )
```

Encrypts using AES in GCM mode.

Parameters

plaintext	buffer where the plaintext is stored
plaintext_len	length of said buffer
aad	additional authenticated data buffer
aad_len	length of said buffer
key	encryption key
iv	initialization vector
ciphertext	buffer (already allocated) where the ct will be stored
tag	tag buffer

Returns

number of written bytes

Definition at line 4 of file src/security/crypto.cpp.

7.13.3.3 build_store()

```
X509_STORE* build_store (
        X509 * cacert,
        X509_CRL * crl )
```

Build a CA store from CA certificate and CRL.

Parameters

cacert	CA certificate
crl	CRL

Returns

X509_STORE* the store

Definition at line 320 of file src/security/crypto.cpp.

7.13.3.4 compare_hmac()

```
bool compare_hmac (
        char * hmac_expected,
        char * hmac_rcv,
        unsigned int len )
```

Compare two HMAC in a secure way.

Parameters

hmac_expected	Expected HMAC
hmac_rcv	Received HMAC
len	Length of the buffers to compare

Returns

true if they are the same

false otherwise

Definition at line 398 of file src/security/crypto.cpp.

7.13.3.5 dhke()

```
int dhke (
        EVP_PKEY * my_key,
        EVP_PKEY * peer_pubkey,
        char ** shared_key )
```

Apply the DHKE to derive a shared secret.

Parameters

my_key	first key
peer_pubkey	second key
shared_key	output buffer location (unallocated), it will contained the shared key

Returns

int shared_key length

Parameters

my_key	first key
peer_pubkey	second key
shared_key	output buffer location (unallocated), it will contained the shared key

Returns

int ???

Definition at line 197 of file src/security/crypto.cpp.

7.13.3.6 dsa_sign()

```
int dsa_sign (
        char * msg,
        int msglen,
        char ** signature,
        EVP_PKEY * prvkey )
```

Signs the given message.

Parameters

msg	the message to be signed
msglen	the length of the message to be signed
signature	pointer to the output signature
prvkey	the private key

Returns

the length of the signature

Definition at line 467 of file src/security/crypto.cpp.

7.13.3.7 dsa_verify()

```
bool dsa_verify (
        char * msg,
        int msglen,
        char * signature,
        int sign_len,
        EVP_PKEY * pkey )
```

Checks the given signature on the given message.

Parameters

msg	the message to be signed
msglen	the length of the message to be signed
signature	the signature
prvkey	the public key

Returns

true if message is authentic, false otherwise

Definition at line 499 of file src/security/crypto.cpp.

7.13.3.8 get_ecdh_key() `int get_ecdh_key ( EVP_PKEY ** key )`

Generate a ECDH key.

Example of usage: EVP_PKEY *key=NULL; int ret = get_ecdh_key(&key);

Parameters

key	the generated key

Returns

int ???

Definition at line 120 of file src/security/crypto.cpp.

7.13.3.9 get_rand() [1/2] `nonce_t get_rand ( )`

Get a random number.

Returns

nonce_t the random number

Definition at line 241 of file src/security/crypto.cpp.

7.13.3.10 get_rand() [2/2] `void get_rand ( char * buffer, int bytes )`

Fills a buffer with a random value.

Parameters

char	buffer to fill
bytes	number of bytes (buffer length)

Definition at line 254 of file src/security/crypto.cpp.

7.13.3.11 hkdf()

```
void hkdf (
    char * key,
    size_t key_len,
    nonce_t nonce1,
    nonce_t nonce2,
    char * label,
    char * out,
    size_t outlen )
```

Apply HKDF, takes two nonces and a label field.

Parameters

key	Key to use
key_len	Size of said key
nonce1	First nonce
nonce2	Second nonce
label	Label field
out	Output buffer (allocated)
outlen	Output len

Definition at line 444 of file src/security/crypto.cpp.

7.13.3.12 hkdf_one_info()

```
void hkdf_one_info (
    char * key,
    size_t key_len,
    char * info,
    size_t info_len,
    char * out,
    size_t outlen )
```

Apply HKDF, takes only one info field.

Parameters

key	Key to use
key_len	Size of said key
info	Info field to use
info_len	Size of said info field
out	Output buffer (allocated)
outlen	Output len

Definition at line 410 of file src/security/crypto.cpp.

7.13.3.13 hmac()

```
int hmac (
    char * msg,
    int msg_len,
    char * key,
    unsigned int keylen,
    char * hmac )
```

Calculate HMAC of the msg.

Parameters

msg	message of which we need the HMAC
msg_len	size of said message
key	key to use for the HMAC
keylen	size of said key
hmac	output buffer (uninitialized, it will be allocated)

Returns

int size of the HMAC

Definition at line 375 of file src/security/crypto.cpp.

7.13.3.14 load_cert_file()

```
X509* load_cert_file (
    const char * file_name )
```

Load a certificate from file.

Parameters

file_name	file name of the certificate

Returns

X509* the certificate ptr, NULL if not read correctly

Definition at line 264 of file src/security/crypto.cpp.

7.13.3.15 load_crl_file()

```
X509_CRL* load_crl_file (
    const char * file_name )
```

Load certificate revocation list from file.

Parameters

file_name	file name

Returns

X509_CRL* the CRL, NULL if not read correctly

Definition at line 301 of file src/security/crypto.cpp.

7.13.3.16 load_key_file()

```
EVP_PKEY* load_key_file (
    const char * file_name,
    const char * password )
```

Load a key from file.

Parameters

file_name	file name of the key file
password	key password

Returns

X509* the key ptr, NULL if not read correctly

Definition at line 283 of file src/security/crypto.cpp.

7.13.3.17 verify_peer_cert()

```
bool verify_peer_cert (
    X509_STORE * store,
    X509 * cert )
```

Parameters

store	Certificate store
cert	Certificate

Returns

true if validation is successful

false otherwise

Parameters

store	
cert	

Returns

true

false

Definition at line 351 of file src/security/crypto.cpp.

7.14 crypto.h

```
00001
00008 #ifndef CRYPTO_H
00009 #define CRYPTO_H
00010 #include <openssl/conf.h>
00011 #include <openssl/evp.h>
00012 #include <openssl/err.h>
00013 #include <openssl/rand.h>
00014 #include <openssl/pem.h>
00015 #include <openssl/hmac.h>
00016 #include <openssl/kdf.h>
00017 #include <string.h>
00018 #include "logging.h"
00019
00021 #define TAG_SIZE     16
00022 #define IV_SIZE      12
00023 #define KEY_SIZE     16
00024
00025 typedef uint32_t nonce_t;
00026
00030 #define handleErrorsNoException(level) { \
00031     LOG((level), "OpenSSL Exception"); \
00032     FILE* stream; \
00033     if (level < LOG_ERR) \
00034         stream = stdout; \
00035     else \
00036         stream = stderr; \
00037     ERR_print_errors_fp(stream); \
00038 }
00039
00043 #define handleErrors() { \
00044     handleErrorsNoException(LOG_ERR); \
00045     throw "OpenSSL Error"; \
00046 }
00047
00062 int aes_gcm_encrypt(char *plaintext, int plaintext_len,
00063                     char *aad, int aad_len,
00064                     char *key, char *iv,
00065                     char *ciphertext,
00066                     char *tag);
00067
00083 int aes_gcm_decrypt(char *ciphertext, int ciphertext_len,
00084                     char *aad, int aad_len,
00085                     char *key,
00086                     char *iv,
00087                     char *plaintext,
00088                     char *tag);
00089
00100 int get_ecdh_key(EVP_PKEY **key);
00101
00110 int dhke(EVP_PKEY *my_key, EVP_PKEY *peer_pubkey, char **shared_key);
00111
00117 nonce_t get_rand();
00118
00125 void get_rand(char* buffer, int bytes);
00126
00133 X509 *load_cert_file(const char *file_name);
00134
00141 X509_CRL *load_crl_file(const char *file_name);
00142
00150 EVP_PKEY *load_key_file(const char *file_name, const char* password);
00151
00159 X509_STORE *build_store(X509 *cacert, X509_CRL *crl);
00160
00161
00170 bool verify_peer_cert(X509_STORE *store, X509 *cert);
00171
00182 int hmac(char *msg, int msg_len, char *key, unsigned int keylen,
00183          char *hmac);
00184
00194 bool compare_hmac(char *hmac_expected, char *hmac_rcv, unsigned int len);
00195
00206 void hkdf_one_info(char *key, size_t key_len,
```

```
00207                   char *info, size_t info_len,
00208                   char *out, size_t outlen);
00209
00221 void hkdf(char *key, size_t key_len,
00222           nonce_t nonce1, nonce_t nonce2,
00223           char *label,
00224           char *out, size_t outlen);
00225
00235 int dsa_sign(char* msg, int msglen, char** signature,
00236              EVP_PKEY *prvkey);
00237
00247 bool dsa_verify(char* msg, int msglen,
00248             char* signature, int sign_len,
00249             EVP_PKEY *pkey);
00250
00251 #endif
```

7.15 dump_buffer.cpp File Reference

Implementation of dump_buffer.h.

```
#include "utils/dump_buffer.h"
#include "logging.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <cctype>
```

Macros

- #define **ROW** 32

Functions

- void dump_buffer_hex (char *buffer, int len, int log_level, const char *name)

  Prints content of buffer to stdout, showing it as hex values.

7.15.1 Detailed Description

Implementation of dump_buffer.h.

Author

Riccardo Mancini

See also

dump_buffer.h

Definition in file dump_buffer.cpp.

7.15.2 Function Documentation

7.15.2.1 dump_buffer_hex()

```
void dump_buffer_hex (
        char * buffer,
        int len,
        int log_level,
        const char * name )
```

Prints content of buffer to stdout, showing it as hex values.

It uses the logging infrastructure to print.

Parameters

buffer	pointer to the buffer to be printed
len	the length (in bytes) of the buffer

Definition at line 20 of file dump_buffer.cpp.

7.16 dump_buffer.cpp

```
00001
00010 #include "utils/dump_buffer.h"
00011 #include "logging.h"
00012 #include <stdio.h>
00013 #include <stdlib.h>
00014 #include <string.h>
00015 #include <cctype>
00016
00017 #define ROW 32
00018
00019
00020 void dump_buffer_hex(char* buffer, int len, int log_level, const char* name){
00021   char *str, tmp3[4], tmp1[2];
00022   int i, j;
00023   int n_rows = (len+ROW-1)/ROW;
00024
00025   str = (char*) malloc(n_rows*(3*ROW + 4 + ROW + 1)+1);
00026   if (!str){
00027     LOG_PERROR(LOG_ERR, "Malloc failed: %s");
00028     return;
00029   }
00030
00031   const char* col_sep = "    ";
00032   const char* row_sep = "\n";
00033
00034   str[0] = '\0';
00035   for (i=0; i<n_rows; i++){
00036     for (j=0; j<ROW; j++){
00037       int idx = i*ROW+j;
00038       if (idx < len){
00039         sprintf(tmp3, "%02x ", (unsigned char) buffer[idx]);
00040       } else {
00041         sprintf(tmp3, "   ");
00042       }
00043       strcat(str, tmp3);
00044     }
00045
00046     strcat(str, col_sep);
00047
00048     for (j=0; j<ROW; j++){
00049       int idx = i*ROW+j;
00050       if (idx >= len)
00051         break;
00052
00053       if (isprint(buffer[idx])){
00054         sprintf(tmp1, "%c", buffer[idx]);
00055       } else{
00056         sprintf(tmp1, ".");
00057       }
00058       strcat(str, tmp1);
00059     }
00060     if (i != n_rows - 1)
00061       strcat(str, row_sep);
00062   }
00063   LOG(log_level, "Dumping %s", name);
00064   if (log_level >= LOG_LEVEL)
00065     printf("%s%s\033[0m\n", logColor(log_level), str);
00066   free(str);
00067 }
```

7.17 dump_buffer.h File Reference

Utility functions for writing and reading data from a buffer.

Macros

- #define **DUMP_BUFFER_HEX_DEBUG**(buffer, len) dump_buffer_hex(buffer, len, LOG_DEBUG, #buffer)

Functions

- void dump_buffer_hex (char *buffer, int len, int log_level, const char *name)

  Prints content of buffer to stdout, showing it as hex values.

7.17.1 Detailed Description

Utility functions for writing and reading data from a buffer.

Utility function for dumping a buffer as hex string.

Author

Riccardo Mancini

These functions are buffer-overflow-safe, i.e. they check the remaining buffer length before writing/reading. The return is -1 in case of errors, the written/read size otherwise.

Date

2020-06-16

Author

Riccardo Mancini

Date

2020-05-17

Author

Riccardo Mancini

Date

2020-06-16

Definition in file dump_buffer.h.

7.17.2 Function Documentation

7.17.2.1 dump_buffer_hex()

```
void dump_buffer_hex (
        char * buffer,
        int len,
        int log_level,
        const char * name )
```

Prints content of buffer to stdout, showing it as hex values.

It uses the logging infrastructure to print.

Parameters

buffer	pointer to the buffer to be printed
len	the length (in bytes) of the buffer

Definition at line 20 of file dump_buffer.cpp.

7.18 dump_buffer.h

```
00001
00010 #ifndef DUMP_BUFFER_H
00011 #define DUMP_BUFFER_H
00012
00013
00022 void dump_buffer_hex(char* buffer, int len, int log_level, const char* name);
00023
00024 #if LOG_LEVEL == LOG_DEBUG
00025 #define DUMP_BUFFER_HEX_DEBUG(buffer, len) dump_buffer_hex(buffer, len, LOG_DEBUG, #buffer)
00026 #else
00027 #define DUMP_BUFFER_HEX_DEBUG(buffer, len)
00028 #endif
00029
00030
00031 #endif // DUMP_BUFFER_H
```

7.19 host.cpp File Reference

Implementation of host.h.

```
#include "network/host.h"
#include "network/inet_utils.h"
```

7.19.1 Detailed Description

Implementation of host.h.

Author

Riccardo Mancini

See also

host.h

Date

2020-05-20

Definition in file host.cpp.

7.20 host.cpp

```
00001
00012 #include "network/host.h"
00013 #include "network/inet_utils.h"
00014
00015 Host::Host(const char* ip, int port){
00016     addr = make_sv_sockaddr_in(ip, port);
00017 }
00018 string Host::toString(){
00019     return sockaddr_in_to_string(addr);
00020 }
```

7.21 host.h File Reference

Definition of the helper class "Host".

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string>
#include "logging.h"
```

Data Structures

- class Host

  Class that holds a host information.

7.21.1 Detailed Description

Definition of the helper class "Host".

Author

Riccardo Mancini

Date

2020-05-17

Definition in file host.h.

7.22 host.h

```
00001
00010 #ifndef HOST_H
00011 #define HOST_H
00012
00013 #include <sys/socket.h>
00014 #include <netinet/in.h>
00015 #include <string>
00016 #include "logging.h"
00017
00018 using namespace std;
00019
00025 class Host{
00026 private:
00027     struct sockaddr_in addr;
00028
00029 public:
00033     Host() {}
00034
00040     Host(struct sockaddr_in addr)
00041         : addr(addr) {}
00042
00049     Host(const char* ip, int port);
00050
00052     struct sockaddr_in getAddress(){return addr;}
00053
00055     string toString();
00056
00057 };
00058
00059 #endif // HOST_H
```

7.23 inet_utils.cpp File Reference

Implementation of inet_utils.h.

```
#include <stdlib.h>
#include <string>
#include <string.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <stdint.h>
#include "logging.h"
#include "network/inet_utils.h"
```

Functions

- int bind_random_port (int socket, struct sockaddr_in *addr)

  Binds socket to a random port.
- struct sockaddr_in make_sv_sockaddr_in (const char *ip, int port)

  Makes sockaddr_in structure given ip string and port of server.
- struct sockaddr_in make_my_sockaddr_in (int port)

  Makes sockaddr_in structure of this host.
- int sockaddr_in_cmp (struct sockaddr_in sai1, struct sockaddr_in sai2)

  Compares INET addresses, returning 0 in case they're equal.
- string sockaddr_in_to_string (struct sockaddr_in src)

  Converts sockaddr_in structure to string to be printed.
- int writeSockAddrIn (char *buffer, size_t buf_len, struct sockaddr_in src)

  Serializes sockaddr_in structure to given buffer.
- int readSockAddrIn (struct sockaddr_in *dst, char *buffer, size_t buf_len)

  Deerializes sockaddr_in structure from given buffer.

7.23.1 Detailed Description

Implementation of inet_utils.h.

Author

Riccardo Mancini

See also

inet_utils.h

Date

2020-05-17

Definition in file inet_utils.cpp.

7.23.2 Function Documentation

7.23.2.1 bind_random_port()

```
int bind_random_port (
        int socket,
        struct sockaddr_in * addr )
```

Binds socket to a random port.

Parameters

socket	socket ID
addr	inet addr structure

Returns

0 in case of failure, port it could bind to otherwise

See also

FROM_PORT

TO_PORT

MAX_TRIES

Definition at line 26 of file inet_utils.cpp.

7.23.2.2 make_my_sockaddr_in() `struct sockaddr_in make_my_sockaddr_in ( int port )`

Makes sockaddr_in structure of this host.

INADDR_ANY is used as IP address.

Parameters

port	port of the server

Returns

sockaddr_in structure this host on given port

Definition at line 55 of file inet_utils.cpp.

7.23.2.3 make_sv_sockaddr_in()

```
struct sockaddr_in make_sv_sockaddr_in (
    const char * ip,
    int port )
```

Makes sockaddr_in structure given ip string and port of server.

Parameters

ip	ip address of server
port	port of the server

Returns

sockaddr_in structure for the given server

Definition at line 46 of file inet_utils.cpp.

7.23.2.4 readSockAddrIn()

```
int readSockAddrIn (
    struct sockaddr_in * src,
    char * buffer,
    size_t buf_len )
```

Deerializes sockaddr_in structure from given buffer.

Parameters

buffer	the buffer

Returns

the built sockaddr_in struct

Definition at line 105 of file inet_utils.cpp.

7.23.2.5 sockaddr_in_cmp()

```
int sockaddr_in_cmp (
        struct sockaddr_in sai1,
        struct sockaddr_in sai2 )
```

Compares INET addresses, returning 0 in case they're equal.

Parameters

sai1	first address
sai2	second address

Returns

0 if thery're equal, 1 otherwise

Definition at line 64 of file inet_utils.cpp.

7.23.2.6 sockaddr_in_to_string()

```
string sockaddr_in_to_string (
        struct sockaddr_in src )
```

Converts sockaddr_in structure to string to be printed.

Parameters

src	the input address

Definition at line 72 of file inet_utils.cpp.

7.23.2.7 writeSockAddrIn()

```
int writeSockAddrIn (
        char * buffer,
        size_t buf_len,
        struct sockaddr_in src )
```

Serializes sockaddr_in structure to given buffer.

Parameters

src	the input address
buffer	the buffer

Definition at line 92 of file inet_utils.cpp.

7.24 inet_utils.cpp

```
00001
00012 #include <stdlib.h>
```

```
00013 #include <string>
00014 #include <string.h>
00015 #include <sys/socket.h>
00016 #include <netinet/in.h>
00017 #include <arpa/inet.h>
00018 #include <stdint.h>
00019
00020 #include "logging.h"
00021
00022 #include "network/inet_utils.h"
00023
00024 using namespace std;
00025
00026 int bind_random_port(int socket, struct sockaddr_in *addr){
00027     int port, ret, i;
00028     for (i = 0; i < MAX_TRIES; i++){
00029         if (i == 0) // first I generate a random one
00030             port = rand() % (TO_PORT - FROM_PORT + 1) + FROM_PORT;
00031         else //if it's not free I scan the next one
00032             port = (port - FROM_PORT + 1) % (TO_PORT - FROM_PORT + 1) + FROM_PORT;
00033
00034         LOG(LOG_DEBUG, "Trying port %d...", port);
00035
00036         addr->sin_port = htons(port);
00037         ret = bind(socket, (struct sockaddr *)addr, sizeof(*addr));
00038         if (ret != -1)
00039             return port;
00040         // consider only some errors?
00041     }
00042     LOG(LOG_ERR, "Could not bind to random port after %d attempts", MAX_TRIES);
00043     return 0;
00044 }
00045
00046 struct sockaddr_in make_sv_sockaddr_in(const char *ip, int port){
00047     struct sockaddr_in addr;
00048     memset(&addr, 0, sizeof(addr));
00049     addr.sin_family = AF_INET;
00050     addr.sin_port = htons(port);
00051     inet_pton(AF_INET, ip, &addr.sin_addr);
00052     return addr;
00053 }
00054
00055 struct sockaddr_in make_my_sockaddr_in(int port){
00056     struct sockaddr_in addr;
00057     memset(&addr, 0, sizeof(addr));
00058     addr.sin_family = AF_INET;
00059     addr.sin_port = htons(port);
00060     addr.sin_addr.s_addr = htonl(INADDR_ANY);
00061     return addr;
00062 }
00063
00064 int sockaddr_in_cmp(struct sockaddr_in sai1, struct sockaddr_in sai2){
00065     if (sai1.sin_port == sai2.sin_port &&
00066         sai1.sin_addr.s_addr == sai2.sin_addr.s_addr)
00067         return 0;
00068     else
00069         return 1;
00070 }
00071
00072 string sockaddr_in_to_string(struct sockaddr_in src){
00073     char dst[MAX_SOCKADDR_STR_LEN];
00074     char port_str[6];
00075     const char *ret;
00076
00077     sprintf(port_str, "%d", ntohs(src.sin_port));
00078
00079     ret = inet_ntop(AF_INET, (void *)&src.sin_addr, dst, MAX_SOCKADDR_STR_LEN);
00080     if (ret != NULL){
00081         strcat(dst, ":");
00082         strcat(dst, port_str);
00083     } else {
00084         strcpy(dst, "ERROR");
00085     }
00086
00087     string s = dst;
00088
00089     return s;
00090 }
00091
00092 int writeSockAddrIn(char* buffer, size_t buf_len, struct sockaddr_in src){
00093     if (buf_len < SERIALIZED_SOCKADDR_IN_LEN)
00094         return -1;
00095
00096     memcpy(buffer,
00097             &src.sin_addr.s_addr,
00098             sizeof(src.sin_addr.s_addr));
00099     memcpy(buffer+sizeof(src.sin_addr.s_addr),
```

```
00100             &src.sin_port,
00101             sizeof(src.sin_port));
00102     return SERIALIZED_SOCKADDR_IN_LEN;
00103 }
00104
00105 int readSockAddrIn(struct sockaddr_in *dst, char* buffer, size_t buf_len){
00106     if (buf_len < SERIALIZED_SOCKADDR_IN_LEN){
00107         return -1;
00108     }
00109
00110     memset(dst, 0, sizeof(*dst));
00111     dst->sin_family = AF_INET;
00112     memcpy(&(dst->sin_addr.s_addr),
00113         buffer,
00114         sizeof(dst->sin_addr.s_addr));
00115     memcpy(&(dst->sin_port),
00116         buffer+sizeof(dst->sin_addr.s_addr),
00117         sizeof(dst->sin_port));
00118     return SERIALIZED_SOCKADDR_IN_LEN;
00119 }
```

7.25 inet_utils.h File Reference

Utility funcions for managing inet addresses.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string>
```

Macros

- #define FROM_PORT 49152

  Random port will be greater or equal to FROM_PORT.
- #define TO_PORT 65535

  Random port will be lower or equal to TO_PORT.
- #define MAX_TRIES 256

  Maximum number of trials before giving up opening a random port.
- #define MAX_SOCKADDR_STR_LEN 22

  Maximum number of characters of INET address to string (eg 123.156.189.123:45678).
- #define SERIALIZED_SOCKADDR_IN_LEN 6

  Size of a serialized sockaddr_in 32 bit address + 16 bit port = 6 bytes.

Functions

- int bind_random_port (int socket, struct sockaddr_in *addr)

  Binds socket to a random port.
- struct sockaddr_in make_sv_sockaddr_in (const char *ip, int port)

  Makes sockaddr_in structure given ip string and port of server.
- struct sockaddr_in make_my_sockaddr_in (int port)

  Makes sockaddr_in structure of this host.
- int sockaddr_in_cmp (struct sockaddr_in sai1, struct sockaddr_in sai2)

  Compares INET addresses, returning 0 in case they're equal.
- string sockaddr_in_to_string (struct sockaddr_in src)

  Converts sockaddr_in structure to string to be printed.
- int writeSockAddrIn (char *buffer, size_t buf_len, struct sockaddr_in src)

  Serializes sockaddr_in structure to given buffer.
- int readSockAddrIn (struct sockaddr_in *src, char *buffer, size_t buf_len)

  Deerializes sockaddr_in structure from given buffer.

7.25.1 Detailed Description

Utility funcions for managing inet addresses.

Author

Riccardo Mancini

This library provides functions for creating sockaddr_in structures from IP address string and integer port number and for binding to a random port (chosen using rand() builtin C function).

Date

2020-05-17

See also

sockaddr_in

rand

Definition in file inet_utils.h.

7.25.2 Function Documentation

7.25.2.1 bind_random_port()

```
int bind_random_port (
        int socket,
        struct sockaddr_in * addr )
```

Binds socket to a random port.

Parameters

socket	socket ID
addr	inet addr structure

Returns

0 in case of failure, port it could bind to otherwise

See also

FROM_PORT

TO_PORT

MAX_TRIES

Definition at line 26 of file inet_utils.cpp.

7.25.2.2 make_my_sockaddr_in()

```
struct sockaddr_in make_my_sockaddr_in (
        int port )
```

Makes sockaddr_in structure of this host.

INADDR_ANY is used as IP address.

Parameters

port	port of the server

Returns

sockaddr_in structure this host on given port

Definition at line 55 of file inet_utils.cpp.

7.25.2.3 make_sv_sockaddr_in()

```
struct sockaddr_in make_sv_sockaddr_in (
        const char * ip,
        int port )
```

Makes sockaddr_in structure given ip string and port of server.

Parameters

ip	ip address of server
port	port of the server

Returns

sockaddr_in structure for the given server

Definition at line 46 of file inet_utils.cpp.

7.25.2.4 readSockAddrIn()

```
int readSockAddrIn (
        struct sockaddr_in * src,
        char * buffer,
        size_t buf_len )
```

Deerializes sockaddr_in structure from given buffer.

Parameters

buffer	the buffer

Returns

the built sockaddr_in struct

Definition at line 105 of file inet_utils.cpp.

7.25.2.5 sockaddr_in_cmp()

```
int sockaddr_in_cmp (
        struct sockaddr_in sai1,
        struct sockaddr_in sai2 )
```

Compares INET addresses, returning 0 in case they're equal.

Parameters

sai1	first address
sai2	second address

Returns

0 if thery're equal, 1 otherwise

Definition at line 64 of file inet_utils.cpp.

7.25.2.6 sockaddr_in_to_string()

```
string sockaddr_in_to_string (
        struct sockaddr_in src )
```

Converts sockaddr_in structure to string to be printed.

Parameters

src	the input address

Definition at line 72 of file inet_utils.cpp.

7.25.2.7 writeSockAddrIn()

```
int writeSockAddrIn (
        char * buffer,
        size_t buf_len,
        struct sockaddr_in src )
```

Serializes sockaddr_in structure to given buffer.

Parameters

src	the input address
buffer	the buffer

Definition at line 92 of file inet_utils.cpp.

7.26 inet_utils.h

```
00001
00017 #ifndef INET_UTILS
00018 #define INET_UTILS
00019
00020
00021 #include <sys/socket.h>
00022 #include <netinet/in.h>
00023 #include <string>
00024
00025 using namespace std;
00026
00028 #define FROM_PORT 49152
00029
00031 #define TO_PORT   65535
00032
00034 #define MAX_TRIES 256
00035
00040 #define MAX_SOCKADDR_STR_LEN 22
00041
00046 #define SERIALIZED_SOCKADDR_IN_LEN 6
00047
00048
00060 int bind_random_port(int socket, struct sockaddr_in *addr);
00061
00069 struct sockaddr_in make_sv_sockaddr_in(const char* ip, int port);
00070
00079 struct sockaddr_in make_my_sockaddr_in(int port);
00080
00088 int sockaddr_in_cmp(struct sockaddr_in sai1, struct sockaddr_in sai2);
00089
00095 string sockaddr_in_to_string(struct sockaddr_in src);
00096
00103 int writeSockAddrIn(char* buffer, size_t buf_len, struct sockaddr_in src);
00104
00111 int readSockAddrIn(struct sockaddr_in *src, char* buffer, size_t buf_len);
00112
00113 #endif
```

7.27 logging.h File Reference

Logging macro.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/time.h>
#include <errno.h>
```

Macros

- #define **TZ_OFFSET** (2)
- #define **LOG_FATAL** (1)
- #define **LOG_ERR** (2)
- #define **LOG_WARN** (3)
- #define **LOG_INFO** (4)
- #define **LOG_DEBUG** (5)
- #define **LOG_LEVEL** LOG_DEBUG
- #define **LOG**(level, ...)
- #define **LOG_PERROR**(level, ...) LOG(level, __VA_ARGS__, strerror(errno))

Functions

- const char * **logColor** (int level)
- void **printtime** (FILE *dbgstream)

7.27.1 Detailed Description

Logging macro.

Author

Riccardo Mancini

This file contains a macro for logging in different levels.

There are 5 levels of logging:

- fatal (LOG_FATAL)
- error (LOG_ERROR)
- warning (LOG_WARN)
- information (LOG_INFO)
- debug (LOG_DEBUG)

The first three will be outputted to stderr, the latter two to stdout.

You can define the LOG_LEVEL macro to one of the available levels for hiding some of the logging messages (default: debug).

TODO: logging may overlap in a concurrent environment

Adapted from https://stackoverflow.com/a/328660

Definition in file logging.h.

7.28 logging.h

```
00001
00026 #ifndef LOGGING
00027 #define LOGGING
00028
00029
00030 #include <stdio.h>
00031 #include <sys/types.h>
00032 #include <unistd.h>
00033 #include <sys/time.h>
00034 #include <errno.h>
00035
00036 #define TZ_OFFSET (2)
00037
00038
00039 #define LOG_FATAL    (1)
00040 #define LOG_ERR      (2)
00041 #define LOG_WARN     (3)
00042 #define LOG_INFO     (4)
00043 #define LOG_DEBUG    (5)
00044
00045 #ifndef LOG_LEVEL
00046 #define LOG_LEVEL LOG_DEBUG
00047 #endif
```

```
00048
00049 inline const char* logColor(int level){
00050   switch(level){
00051     case LOG_FATAL:
00052       return "\033[31m";
00053     case LOG_ERR:
00054       return "\033[91m";
00055     case LOG_WARN:
00056       return "\033[33m";
00057     case LOG_INFO:
00058       return "\033[32m";
00059     case LOG_DEBUG:
00060       return "\033[94m";
00061     default:
00062       return "\033[0m";
00063   }
00064 }
00065
00066 inline void printtime(FILE* dbgstream){
00067   timeval tv;
00068   int ret = gettimeofday(&tv, NULL);
00069   if(ret == -1){
00070     return;
00071   }
00072
00073   unsigned int hour = tv.tv_sec % (24*60*60) / (60*60);
00074   hour += TZ_OFFSET;
00075   hour %= 24;
00076   unsigned int min = tv.tv_sec % (60*60) / 60;
00077   unsigned int sec = tv.tv_sec % (60);
00078   unsigned int msec = tv.tv_usec / 1000;
00079   fprintf(dbgstream, "[%02u:%02u:%02u.%03u]", hour, min, sec, msec );
00080 }
00081
00082 #define LOG(level, ...) do {  \
00083                           if (level <= LOG_LEVEL) { \
00084                             FILE *dbgstream; \
00085                             char where[50]; \
00086                             switch(level){ \
00087                               case LOG_FATAL: \
00088                                 dbgstream = stderr; \
00089                                 fprintf(dbgstream, "%s[FATAL]", logColor(LOG_FATAL)); \
00090                                 break; \
00091                               case LOG_ERR: \
00092                                 dbgstream = stderr; \
00093                                 fprintf(dbgstream, "%s[ERROR]", logColor(LOG_ERR)); \
00094                                 break; \
00095                               case LOG_WARN: \
00096                                 dbgstream = stderr; \
00097                                 fprintf(dbgstream, "%s[WARN ]", logColor(LOG_WARN)); \
00098                                 break; \
00099                               case LOG_INFO: \
00100                                 dbgstream = stdout; \
00101                                 fprintf(dbgstream, "%s[INFO ]", logColor(LOG_INFO)); \
00102                                 break; \
00103                               case LOG_DEBUG: \
00104                                 dbgstream = stdout; \
00105                                 fprintf(dbgstream, "%s[DEBUG]", logColor(LOG_DEBUG)); \
00106                                 break; \
00107                             } \
00108                             fprintf(dbgstream, "[%-5d]", (int) getpid()); \
00109                             printtime(dbgstream); \
00110                             snprintf(where, 50, "[%s:%d]", __FILE__, __LINE__); \
00111                             fprintf(dbgstream, "%-25s ", where); \
00112                             fprintf(dbgstream, __VA_ARGS__); \
00113                             fprintf(dbgstream, "\033[0m\n"); \
00114                             fflush(dbgstream); \
00115                           } \
00116                         } while (0)
00117
00118 #define LOG_PERROR(level, ...) LOG(level, __VA_ARGS__, strerror(errno))
00119
00120 #endif
```

7.29 message_queue.h File Reference

Definition and implementation of the MessageQueue class.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
```

```
#include <pthread.h>
#include <queue>
#include <utility>
#include "config.h"
```

Data Structures

- class MessageQueue< T, MAX_SIZE >
  Thread-safe message queue template.

7.29.1 Detailed Description

Definition and implementation of the MessageQueue class.

Author

Riccardo Mancini

Date

2020-05-23

Definition in file message_queue.h.

7.30 message_queue.h

```
00001
00010 #ifndef MESSAGE_QUEUE_H
00011 #define MESSAGE_QUEUE_H
00012
00013 #include <iostream>
00014 #include <cstdlib>
00015 #include <ctime>
00016 #include <pthread.h>
00017 #include <queue>
00018 #include <utility>
00019
00020 #include "config.h"
00021
00022 using namespace std;
00023
00031 template <typename T, int MAX_SIZE>
00032 class MessageQueue{
00033 private:
00034     queue<T> msg_queue;
00035     pthread_mutex_t mutex;
00036     pthread_cond_t available_messages;
00037 public:
00042     MessageQueue();
00043
00050     bool push(T e);
00051
00059     bool pushSignal(T e);
00060
00066     T pull();
00067
00074     T pullWait();
00075
00079     size_t size(){return msg_queue.size();}
00080
00084     size_t empty(){return msg_queue.empty();}
00085 };
00086
00087 // implementation must stay in header since I've used a template
00088
```

```
00089 template <typename T, int MAX_SIZE>
00090 MessageQueue<T,MAX_SIZE>::MessageQueue(){
00091     pthread_mutex_init(&mutex, NULL);
00092 }
00093
00094 template <typename T, int MAX_SIZE>
00095 bool MessageQueue<T,MAX_SIZE>::push(T e){
00096     bool success;
00097     pthread_mutex_lock(&mutex);
00098     if ((success = msg_queue.size() < MAX_SIZE))
00099         msg_queue.push(e);
00100     pthread_mutex_unlock(&mutex);
00101     return success;
00102 }
00103
00104 template <typename T, int MAX_SIZE>
00105 T MessageQueue<T,MAX_SIZE>::pull(){
00106     T e;
00107     pthread_mutex_lock(&mutex);
00108     e = msg_queue.front();
00109     msg_queue.pop();
00110     pthread_mutex_unlock(&mutex);
00111     return e;
00112 }
00113
00114 template <typename T, int MAX_SIZE>
00115 T MessageQueue<T,MAX_SIZE>::pullWait(){
00116     T e;
00117     pthread_mutex_lock(&mutex);
00118     while(msg_queue.empty())
00119         pthread_cond_wait(&available_messages, &mutex);
00120     e = msg_queue.front();
00121     msg_queue.pop();
00122     pthread_mutex_unlock(&mutex);
00123     return e;
00124 }
00125
00126 template <typename T, int MAX_SIZE>
00127 bool MessageQueue<T,MAX_SIZE>::pushSignal(T e){
00128     bool success;
00129     pthread_mutex_lock(&mutex);
00130     if ((success = msg_queue.size() < MAX_SIZE)){
00131         msg_queue.push(e);
00132         if (msg_queue.size() == 1)
00133             pthread_cond_signal(&available_messages);
00134     }
00135     pthread_mutex_unlock(&mutex);
00136     return success;
00137
00138 }
00139
00140 #endif // MESSAGE_QUEUE_H
```

7.31 messages.cpp File Reference

Implementation of messages.h.

```
#include <cstdlib>
#include <cstring>
#include <cmath>
#include "network/messages.h"
#include "network/inet_utils.h"
#include "security/crypto_utils.h"
#include "utils/buffer_io.h"
```

Functions

- Message * readMessage (char *buffer, msglen_t len)

  Reads the message using the correct class and returns a pointer to it.

7.31.1 Detailed Description

Implementation of messages.h.

Author

Riccardo Mancini

See also

messages.h

Definition in file messages.cpp.

7.31.2 Function Documentation

7.31.2.1 readMessage() `Message* readMessage (`

```
char * buffer,
msglen_t len )
```

Reads the message using the correct class and returns a pointer to it.

NB: remeber to dispose of the created Message when you are done with it.

Definition at line 20 of file messages.cpp.

7.32 messages.cpp

```
00001
00010 #include <cstdlib>
00011 #include <cstring>
00012 #include <cmath>
00013
00014 #include "network/messages.h"
00015 #include "network/inet_utils.h"
00016
00017 #include "security/crypto_utils.h"
00018 #include "utils/buffer_io.h"
00019
00020 Message* readMessage(char *buffer, msglen_t len){
00021     Message *m;
00022     int ret;
00023
00024     switch(buffer[0]){
00025         case SECURE_MESSAGE:
00026             m = new SecureMessage;
00027             break;
00028         case CLIENT_HELLO:
00029             m = new ClientHelloMessage;
00030             break;
00031         case SERVER_HELLO:
00032             m = new ServerHelloMessage;
00033             break;
00034         case CLIENT_VERIFY:
00035             m = new ClientVerifyMessage;
00036             break;
00037         case START_GAME_PEER:
00038             m = new StartGameMessage;
00039             break;
00040         case MOVE:
00041             m = new MoveMessage;
00042             break;
```

```
00043         case REGISTER:
00044             m = new RegisterMessage;
00045             break;
00046         case CHALLENGE:
00047             m = new ChallengeMessage;
00048             break;
00049         case GAME_END:
00050             m = new GameEndMessage;
00051             break;
00052         case USERS_LIST:
00053             m = new UsersListMessage;
00054             break;
00055         case USERS_LIST_REQ:
00056             m = new UsersListRequestMessage;
00057             break;
00058         case CHALLENGE_FWD:
00059             m = new ChallengeForwardMessage;
00060             break;
00061         case CHALLENGE_RESP:
00062             m = new ChallengeResponseMessage;
00063             break;
00064         case GAME_START:
00065             m = new GameStartMessage;
00066             break;
00067         case GAME_CANCEL:
00068             m = new GameCancelMessage;
00069             break;
00070         case CERT_REQ:
00071             m = new CertificateRequestMessage;
00072             break;
00073         case CERTIFICATE:
00074             m = new CertificateMessage;
00075             break;
00076         default:
00077             m = NULL;
00078             LOG(LOG_ERR, "Unrecognized message type %d", buffer[0]);
00079             return NULL;
00080     };
00081
00082     ret = m->read(buffer, len);
00083
00084     if (ret != 0){
00085         LOG(LOG_ERR, "Error reading message of type %d: %d", buffer[0], ret);
00086         return NULL;
00087     } else{
00088         return m;
00089     }
00090 }
00091
00092 msglen_t StartGameMessage::write(char *buffer){
00093     int i = 0;
00094     int ret;
00095
00096     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) START_GAME_PEER)) < 0)
00097         return 0;
00098     i += ret;
00099
00100     return i;
00101 }
00102
00103 msglen_t StartGameMessage::read(char *buffer, msglen_t len){
00104     return 0;
00105 }
00106
00107 msglen_t MoveMessage::write(char *buffer){
00108     int i = 0;
00109     int ret;
00110
00111     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) MOVE)) < 0)
00112         return 0;
00113     i += ret;
00114
00115     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, col)) < 0)
00116         return 0;
00117     i += ret;
00118
00119     return i;
00120 }
00121
00122 msglen_t MoveMessage::read(char *buffer, msglen_t len){
00123     int i = 1;
00124     int ret;
00125
00126     if ((ret = readUInt8((uint8_t*)&col, &buffer[i], len-i)) < 0)
00127         return 1;
00128     i += ret;
00129
```

```
00130     return 0;
00131 }
00132
00133 msglen_t RegisterMessage::write(char *buffer){
00134     int i = 0;
00135     int ret;
00136
00137     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) REGISTER)) < 0)
00138         return 0;
00139     i += ret;
00140
00141     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, username)) < 0)
00142         return 0;
00143     i += ret;
00144
00145     return i;
00146 }
00147
00148 msglen_t RegisterMessage::read(char *buffer, msglen_t len){
00149     int i = 1;
00150     int ret;
00151
00152     if ((ret = readUsername(&username, &buffer[i], len-i)) < 0)
00153         return 1;
00154     i += ret;
00155
00156     return 0;
00157 }
00158
00159 msglen_t ChallengeMessage::write(char *buffer){
00160     int i = 0;
00161     int ret;
00162
00163     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CHALLENGE)) < 0)
00164         return 0;
00165     i += ret;
00166
00167     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, username)) < 0)
00168         return 0;
00169     i += ret;
00170
00171     return i;
00172 }
00173 msglen_t ChallengeMessage::read(char *buffer, msglen_t len){
00174     int i = 1;
00175     int ret;
00176
00177     if ((ret = readUsername(&username, &buffer[i], len-i)) < 0)
00178         return 1;
00179     i += ret;
00180
00181     return 0;
00182 }
00183
00184 msglen_t GameEndMessage::write(char *buffer){
00185     int i = 0;
00186     int ret;
00187
00188     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) GAME_END)) < 0)
00189         return 0;
00190
00191     i += ret;
00192
00193     return i;
00194 }
00195 msglen_t GameEndMessage::read(char *buffer, msglen_t len){
00196     return 0;
00197 }
00198
00199 msglen_t UsersListMessage::write(char *buffer){
00200     int i = 0;
00201     int ret;
00202
00203     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) USERS_LIST)) < 0)
00204         return 0;
00205     i += ret;
00206
00207     size_t strsize = min(usernames.size(),
00208                          (size_t) ((MAX_USERNAME_LENGTH+1)*MAX_USERS));
00209     size_t padded_size =
    (strsize+MAX_USERNAME_LENGTH)/(MAX_USERNAME_LENGTH+1)*(MAX_USERNAME_LENGTH+1);
00210     if ((int)padded_size > MAX_MSG_SIZE-i)
00211         return 0;
00212     strncpy(&buffer[i], usernames.c_str(), strsize);
00213     memset(&buffer[1+strsize], 0, padded_size-strsize+1);
00214     i += padded_size+1;
00215
```

```
00216     return i;
00217 }
00218
00219 msglen_t UsersListMessage::read(char *buffer, msglen_t len){
00220     int maxsize = min((MAX_USERNAME_LENGTH+1)*MAX_USERS, len-1);
00221     if (maxsize <= 0){
00222         return 1;
00223     }
00224
00225     usernames = string(&buffer[1], maxsize);
00226     return 0;
00227 }
00228
00229 msglen_t UsersListRequestMessage::write(char *buffer){
00230         int i = 0;
00231     int ret;
00232
00233     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) USERS_LIST_REQ)) < 0)
00234         return 0;
00235     i += ret;
00236
00237     if ((ret = writeUInt32(&buffer[i], MAX_MSG_SIZE-i, offset)) < 0)
00238         return 0;
00239     i += ret;
00240
00241     return i;
00242 }
00243
00244 msglen_t UsersListRequestMessage::read(char *buffer, msglen_t len){
00245     int i = 1;
00246     int ret;
00247
00248     if ((ret = readUInt32(&offset, &buffer[i], len-i)) < 0)
00249         return 1;
00250     i += ret;
00251
00252     return 0;
00253 }
00254
00255 msglen_t ChallengeForwardMessage::write(char *buffer){
00256     int i = 0;
00257     int ret;
00258
00259     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CHALLENGE_FWD)) < 0)
00260         return 0;
00261     i += ret;
00262
00263     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, username)) < 0)
00264         return 0;
00265     i += ret;
00266
00267     return i;
00268 }
00269 msglen_t ChallengeForwardMessage::read(char *buffer, msglen_t len){
00270     int i = 1;
00271     int ret;
00272
00273     if ((ret = readUsername(&username, &buffer[i], len-i)) < 0)
00274         return 1;
00275     i += ret;
00276
00277     return 0;
00278 }
00279
00280 msglen_t ChallengeResponseMessage::write(char *buffer){
00281     int i = 0;
00282     int ret;
00283
00284     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CHALLENGE_RESP)) < 0)
00285         return 0;
00286     i += ret;
00287
00288     if ((ret = writeBool(&buffer[i], MAX_MSG_SIZE-i, response)) < 0)
00289         return 0;
00290     i += ret;
00291
00292     if ((ret = writeUInt16(&buffer[i], MAX_MSG_SIZE-i, listen_port)) < 0)
00293         return 0;
00294     i += ret;
00295
00296     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, username)) < 0)
00297         return 0;
00298     i += ret;
00299
00300     return i;
00301 }
00302 msglen_t ChallengeResponseMessage::read(char *buffer, msglen_t len){
```

```
00303     int i = 1;
00304     int ret;
00305
00306     if ((ret = readBool(&response, &buffer[i], len-i)) < 0)
00307         return 1;
00308     i += ret;
00309
00310     if ((ret = readUInt16(&listen_port, &buffer[i], len-i)) < 0)
00311         return 1;
00312     i += ret;
00313
00314     if ((ret = readUsername(&username, &buffer[i], len-i)) < 0)
00315         return 1;
00316     i += ret;
00317
00318     return 0;
00319 }
00320
00321 msglen_t GameCancelMessage::write(char *buffer){
00322     int i = 0;
00323     int ret;
00324
00325     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) GAME_CANCEL)) < 0)
00326         return 0;
00327     i += ret;
00328
00329     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, username)) < 0)
00330         return 0;
00331     i += ret;
00332
00333     return i;
00334 }
00335 msglen_t GameCancelMessage::read(char *buffer, msglen_t len){
00336     int i = 1;
00337     int ret;
00338
00339     if ((ret = readUsername(&username, &buffer[i], len-i)) < 0)
00340         return 1;
00341     i += ret;
00342
00343     return 0;
00344 }
00345
00346 msglen_t GameStartMessage::write(char *buffer){
00347     int i = 0;
00348     int ret;
00349
00350     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) GAME_START)) < 0)
00351         return 0;
00352     i += ret;
00353
00354     if ((ret = writeSockAddrIn(&buffer[i], MAX_MSG_SIZE-i, addr)) < 0)
00355         return 0;
00356     i += ret;
00357
00358     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, username)) < 0)
00359         return 0;
00360     i += ret;
00361
00362     if ((ret = cert2buf(cert, &buffer[i], MAX_MSG_SIZE - i)) < 0)
00363         return 0;
00364     i += ret;
00365
00366     return i;
00367 }
00368
00369 msglen_t GameStartMessage::read(char *buffer, msglen_t len){
00370     int i = 1;
00371     int ret;
00372
00373     if ((ret = readSockAddrIn(&addr, &buffer[i], len-i)) < 0)
00374         return 1;
00375     i += ret;
00376
00377     if ((ret = readUsername(&username, &buffer[i], len-i)) < 0)
00378         return 1;
00379     i += ret;
00380
00381     if ((ret = buf2cert(&buffer[i], len - i, &cert)) < 0)
00382         return 1;
00383     i += ret;
00384
00385     return 0;
00386 }
00387
00388 msglen_t SecureMessage::write(char* buffer){
00389     int i = 0;
```

```
00390     int ret;
00391
00392     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) SECURE_MESSAGE)) < 0)
00393         return 0;
00394     i += ret;
00395
00396     if ((ret = writeBuf(&buffer[i], MAX_MSG_SIZE-i, ct, ct_size)) < 0)
00397         return 0;
00398     i += ret;
00399
00400     if ((ret = writeBuf(&buffer[i], MAX_MSG_SIZE-i, tag, TAG_SIZE)) < 0)
00401         return 0;
00402     i += ret;
00403
00404     return i;
00405 }
00406
00407 msglen_t SecureMessage::read(char* buffer, msglen_t len){
00408     ct_size = len-1-TAG_SIZE;
00409     ct = (char*) malloc(ct_size);
00410     tag = (char*) malloc(TAG_SIZE);
00411     if(!ct || !tag){
00412         LOG(LOG_WARN, "Malloc failed for message of length %d", len);
00413         return -1;
00414     }
00415
00416     int i = 1;
00417     int ret;
00418
00419     if ((ret = readBuf(ct, ct_size, &buffer[i], len-i)) < 0)
00420         return 1;
00421     i += ret;
00422
00423     if ((ret = readBuf(tag, TAG_SIZE, &buffer[i], len-i)) < 0)
00424         return 1;
00425     i += ret;
00426
00427     return 0;
00428 }
00429
00430 msglen_t ClientHelloMessage::write(char* buffer){
00431     int i = 0;
00432     int ret;
00433
00434     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CLIENT_HELLO)) < 0)
00435         return 0;
00436     i += ret;
00437
00438     if ((ret = writeUInt32(&buffer[i], MAX_MSG_SIZE-i, nonce)) < 0)
00439         return 0;
00440     i += ret;
00441
00442     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, my_id)) < 0)
00443         return 0;
00444     i += ret;
00445
00446     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, other_id)) < 0)
00447         return 0;
00448     i += ret;
00449
00450     if ((ret = pkey2buf(eph_key, &buffer[i], MAX_MSG_SIZE-i)) < 0)
00451         return 0;
00452     i += ret;
00453
00454     return i;
00455 }
00456
00457 msglen_t ClientHelloMessage::read(char* buffer, msglen_t len){
00458     int i = 1;
00459     int ret;
00460
00461     if ((ret = readUInt32(&nonce, &buffer[i], len-i)) < 0)
00462         return 1;
00463     i += ret;
00464
00465     if ((ret = readUsername(&my_id, &buffer[i], len-i)) < 0)
00466         return 1;
00467     i += ret;
00468
00469     if ((ret = readUsername(&other_id, &buffer[i], len-i)) < 0)
00470         return 1;
00471     i += ret;
00472
00473     if((ret = buf2pkey(&buffer[i], len-i, &eph_key)) < 0)
00474         return 1;
00475     i += ret;
00476
```

```
00477     return 0;
00478 }
00479
00480 ServerHelloMessage::~ServerHelloMessage(){
00481     if (ds != NULL){
00482         free(ds);
00483     }
00484 }
00485
00486 msglen_t ServerHelloMessage::write(char* buffer){
00487     int i = 0;
00488     int ret;
00489
00490     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) SERVER_HELLO)) < 0)
00491         return 0;
00492     i += ret;
00493
00494     if ((ret = writeUInt32(&buffer[i], MAX_MSG_SIZE-i, nonce)) < 0)
00495         return 0;
00496     i += ret;
00497
00498     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, my_id)) < 0)
00499         return 0;
00500     i += ret;
00501
00502     if ((ret = writeUsername(&buffer[i], MAX_MSG_SIZE-i, other_id)) < 0)
00503         return 0;
00504     i += ret;
00505
00506     if ((ret = writeUInt32(&buffer[i], MAX_MSG_SIZE-i, ds_size)) < 0)
00507         return 0;
00508     i += ret;
00509
00510     if ((ret = writeBuf(&buffer[i], MAX_MSG_SIZE-i, ds, ds_size)) < 0)
00511         return 0;
00512     i += ret;
00513
00514     if ((ret = pkey2buf(eph_key, &buffer[i], MAX_MSG_SIZE-i)) < 0)
00515         return 0;
00516     i += ret;
00517
00518     return i;
00519 }
00520
00521 msglen_t ServerHelloMessage::read(char* buffer, msglen_t len){
00522     int i = 1;
00523     int ret;
00524
00525     if ((ret = readUInt32(&nonce, &buffer[i], len-i)) < 0)
00526         return 1;
00527     i += ret;
00528
00529     if ((ret = readUsername(&my_id, &buffer[i], len-i)) < 0)
00530         return 1;
00531     i += ret;
00532
00533     if ((ret = readUsername(&other_id, &buffer[i], len-i)) < 0)
00534         return 1;
00535     i += ret;
00536
00537     if ((ret = readUInt32(&ds_size, &buffer[i], len-i)) < 0)
00538         return 1;
00539     i += ret;
00540
00541     ds = (char*) malloc(ds_size);
00542     if (!ds){
00543         LOG_PERROR(LOG_ERR, "Malloc failed: %s");
00544         return 1;
00545     }
00546
00547     if ((ret = readBuf(ds, ds_size, &buffer[i], len-i)) < 0)
00548         return 1;
00549     i += ret;
00550
00551     if((ret = buf2pkey(&buffer[i], len-i, &eph_key)) < 0)
00552         return 1;
00553     i += ret;
00554
00555     return 0;
00556 }
00557
00558 ClientVerifyMessage::~ClientVerifyMessage(){
00559     if (ds != NULL){
00560         free(ds);
00561     }
00562 }
00563
```

```
00564 msglen_t ClientVerifyMessage::write(char* buffer){
00565     int i = 0;
00566     int ret;
00567 
00568     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CLIENT_VERIFY)) < 0)
00569         return 0;
00570     i += ret;
00571 
00572     if ((ret = writeUInt32(&buffer[i], MAX_MSG_SIZE-i, ds_size)) < 0)
00573         return 0;
00574     i += ret;
00575 
00576     if ((ret = writeBuf(&buffer[i], MAX_MSG_SIZE-i, ds, ds_size)) < 0)
00577         return 0;
00578     i += ret;
00579 
00580     return i;
00581 }
00582 
00583 msglen_t ClientVerifyMessage::read(char* buffer, msglen_t len){
00584    int i = 1;
00585     int ret;
00586 
00587     if ((ret = readUInt32(&ds_size, &buffer[i], len-i)) < 0)
00588         return 1;
00589     i += ret;
00590 
00591     ds = (char*) malloc(ds_size);
00592     if (!ds){
00593         LOG_PERROR(LOG_ERR, "Malloc failed: %s");
00594         return 1;
00595     }
00596 
00597     if ((ret = readBuf(ds, ds_size, &buffer[i], len-i)) < 0)
00598         return 1;
00599     i += ret;
00600 
00601     return 0;
00602 }
00603 
00604 msglen_t CertificateRequestMessage::write(char* buffer){
00605     int i = 0;
00606     int ret;
00607 
00608     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CERT_REQ)) < 0)
00609         return 0;
00610 
00611     i += ret;
00612 
00613     return i;
00614 }
00615 
00616 msglen_t CertificateRequestMessage::read(char* buffer, msglen_t len){
00617     return 0;
00618 }
00619 
00620 msglen_t CertificateMessage::write(char* buffer){
00621     int i = 0;
00622     int ret;
00623 
00624     if ((ret = writeUInt8(&buffer[i], MAX_MSG_SIZE-i, (char) CERTIFICATE)) < 0)
00625         return 0;
00626 
00627     i += ret;
00628 
00629     if ((ret = cert2buf(cert, &buffer[i], MAX_MSG_SIZE-1)) < 0)
00630         return 0;
00631     i += ret;
00632 
00633     return i;
00634 }
00635 
00636 msglen_t CertificateMessage::read(char* buffer, msglen_t len){
00637     int ret = buf2cert(&buffer[1], len-1, &cert);
00638     return ret > 0 ? 0 : 1;
00639 }
```

7.33 messages.h File Reference

Definition of messages.

```
#include <string>
#include <stdint.h>
```

```
#include <sys/socket.h>
#include <netinet/in.h>
#include "logging.h"
#include "config.h"
#include "network/inet_utils.h"
#include "network/host.h"
#include "security/crypto.h"
#include "security/secure_host.h"
```

Data Structures

- class Message

  Abstract class for Messages.
- class StartGameMessage

  Message that signals to start a new game.
- class MoveMessage

  Message that signals a move.
- class RegisterMessage

  Message that permits the client to register to server.
- class ChallengeMessage

  Message that permits the client to challenge another client through the server.
- class GameEndMessage

  Message that signals the server that the client is available.
- class UsersListMessage

  Message that the server sends the client with the list of users.
- class UsersListRequestMessage

  Message with which the client asks for the list of connected users.
- class ChallengeForwardMessage

  Message with which the server forwards a challenge.
- class ChallengeResponseMessage

  Message with which the client replies to a challenge.
- class GameCancelMessage

  Message with which the server forwards a challenge rejectal or another event that caused the game to be canceled.
- class GameStartMessage

  Message with which the server makes a new game start between clients.
- class SecureMessage
- class ClientHelloMessage
- class ClientVerifyMessage
- class ServerHelloMessage
- class CertificateRequestMessage
- class CertificateMessage

Macros

- #define **MSGLEN_HTON**(x) htons((x))
- #define **MSGLEN_NTOH**(x) ntohs((x))

Typedefs

- typedef uint16_t msglen_t

  Type of message length (first N bytes of packet)

Enumerations

- enum MessageType {
  SECURE_MESSAGE, **CLIENT_HELLO**, **SERVER_HELLO**, **CLIENT_VERIFY**,
  START_GAME_PEER, **MOVE**, **REGISTER**, **CHALLENGE**,
  GAME_END, **USERS_LIST**, **USERS_LIST_REQ**, **CHALLENGE_FWD**,
  CHALLENGE_RESP, **GAME_START**, **GAME_CANCEL**, **CERT_REQ**,
  CERTIFICATE }

  Possible type of messages.

Functions

- size_t usernameLength (string s)

  Utility for getting username length (excluding '\0')
- int readUsername (string *username, char *buf, size_t buf_len)

  Utility for getting username from buffer.
- int writeUsername (char *buf, size_t buf_len, string s)

  Utility for writing username to buffer.
- Message * readMessage (char *buffer, msglen_t len)

  Reads the message using the correct class and returns a pointer to it.

7.33.1 Detailed Description

Definition of messages.

Author

Riccardo Mancini

Date

2020-05-17

Definition in file messages.h.

7.33.2 Enumeration Type Documentation

7.33.2.1 MessageType `enum MessageType`

Possible type of messages.

When adding a new message class, add a related type here and set its getType method to return it.

Definition at line 75 of file messages.h.

7.33.3 Function Documentation

7.33.3.1 readMessage() `Message* readMessage (`

```
char * buffer,
msglen_t len )
```

Reads the message using the correct class and returns a pointer to it.

NB: remeber to dispose of the created Message when you are done with it.

Definition at line 20 of file messages.cpp.

7.33.3.2 readUsername() `int readUsername (`

```
string * username,
char * buf,
size_t buf_len )  [inline]
```

Utility for getting username from buffer.

Parameters

buf	the buffer to read the string from
buflen	the size of the buffer

Returns

the read string

Definition at line 44 of file messages.h.

7.33.3.3 writeUsername() `int writeUsername (`

```
char * buf,
size_t buf_len,
string s )  [inline]
```

Utility for writing username to buffer.

NB: buffer must be large enough

Parameters

s	the username string to be written on buffer
buf	the buffer to write the string to

Returns

number of written bytes

Definition at line 59 of file messages.h.

7.34 messages.h

```
00001
00010 #ifndef MESSAGES_H
00011 #define MESSAGES_H
00012
00013 #include <string>
00014 #include <stdint.h>
00015 #include <sys/socket.h>
00016 #include <netinet/in.h>
00017
00018 #include "logging.h"
00019 #include "config.h"
00020 #include "network/inet_utils.h"
00021 #include "network/host.h"
00022 #include "security/crypto.h"
00023 #include "security/secure_host.h"
00024
00025 using namespace std;
00026
00028 typedef uint16_t msglen_t;
00029
00030 #define MSGLEN_HTON(x) htons((x))
00031 #define MSGLEN_NTOH(x) ntohs((x))
00032
00034 inline size_t usernameLength(string s){
00035     return min(strlen(s.c_str()), (size_t) MAX_USERNAME_LENGTH);
00036 }
00037
00044 inline int readUsername(string *username, char* buf, size_t buf_len){
00045     size_t size = min(strnlen(buf, buf_len-1), (size_t) MAX_USERNAME_LENGTH);
00046     *username = string(buf, size);
00047     return MAX_USERNAME_LENGTH+1;
00048 }
00049
00059 inline int writeUsername(char* buf, size_t buf_len, string s){
00060     if (buf_len < MAX_USERNAME_LENGTH+1){
00061         return -1;
00062     }
00063     size_t strsize = usernameLength(s);
00064     strncpy(buf, s.c_str(), strsize);
00065     memset(&buf[strsize], 0, MAX_USERNAME_LENGTH-strsize+1);
00066     return MAX_USERNAME_LENGTH+1;
00067 }
00068
00075 enum MessageType
00076 {
00077     SECURE_MESSAGE,
00078     CLIENT_HELLO,
00079     SERVER_HELLO,
00080     CLIENT_VERIFY,
00081     START_GAME_PEER,
00082     MOVE,
00083     REGISTER,
00084     CHALLENGE,
00085     GAME_END,
00086     USERS_LIST,
00087     USERS_LIST_REQ,
00088     CHALLENGE_FWD,
00089     CHALLENGE_RESP,
00090     GAME_START,
00091     GAME_CANCEL,
00092     CERT_REQ,
00093     CERTIFICATE,
00094 };
00095
00099 class Message
00100 {
00101 public:
00102     virtual ~Message(){};
00109     virtual msglen_t write(char *buffer) = 0;
00110
00117     virtual msglen_t read(char *buffer, msglen_t len) = 0;
00118
00122     virtual string getName() = 0;
00123
```

```
00124     virtual MessageType getType() = 0;
00125 };
00126
00130 class StartGameMessage : public Message
00131 {
00132 public:
00133     StartGameMessage() {}
00134     ~StartGameMessage() {}
00135
00136     msglen_t write(char *buffer);
00137     msglen_t read(char *buffer, msglen_t len);
00138
00139     string getName() { return "StartGame"; }
00140
00141     MessageType getType() { return START_GAME_PEER; }
00142 };
00143
00147 class MoveMessage : public Message
00148 {
00149 private:
00150     char col;
00151
00152 public:
00153     MoveMessage() {}
00154     MoveMessage(char col) : col(col) {}
00155     ~MoveMessage() {}
00156
00157     msglen_t write(char *buffer);
00158     msglen_t read(char *buffer, msglen_t len);
00159
00160     string getName() { return "Move"; }
00161
00162     char getColumn() { return col; }
00163
00164     MessageType getType() { return MOVE; }
00165 };
00166
00170 class RegisterMessage : public Message
00171 {
00172 private:
00173     string username;
00174
00175 public:
00176     RegisterMessage() {}
00177     RegisterMessage(string username) : username(username) {}
00178     ~RegisterMessage() {}
00179
00180     msglen_t write(char *buffer);
00181     msglen_t read(char *buffer, msglen_t len);
00182
00183     string getName() { return "Register"; }
00184
00185     string getUsername() { return username; }
00186
00187     MessageType getType() { return REGISTER; }
00188 };
00189
00194 class ChallengeMessage : public Message
00195 {
00196 private:
00197     string username;
00198
00199 public:
00200     ChallengeMessage() {}
00201     ChallengeMessage(string username) : username(username) {}
00202     ~ChallengeMessage() {}
00203
00204     msglen_t write(char *buffer);
00205     msglen_t read(char *buffer, msglen_t len);
00206
00207     string getName() { return "Challenge"; }
00208
00209     string getUsername() { return username; }
00210
00211     MessageType getType() { return CHALLENGE; }
00212 };
00213
00217 class GameEndMessage : public Message
00218 {
00219 public:
00220     GameEndMessage() {}
00221     ~GameEndMessage() {}
00222
00223     msglen_t write(char *buffer);
00224     msglen_t read(char *buffer, msglen_t len);
00225
00226     string getName() { return "Game End"; }
```

```
00227
00228     MessageType getType() { return GAME_END; }
00229 };
00230
00234 class UsersListMessage : public Message
00235 {
00236 private:
00237     string usernames;
00238
00239 public:
00240     UsersListMessage() {}
00241     UsersListMessage(string usernames) : usernames(usernames) {}
00242     ~UsersListMessage() {}
00243
00244     msglen_t write(char *buffer);
00245     msglen_t read(char *buffer, msglen_t len);
00246
00247     string getName() { return "User list"; }
00248
00249     string getUsernames() { return usernames; }
00250
00251     MessageType getType() { return USERS_LIST; }
00252 };
00253
00257 class UsersListRequestMessage : public Message
00258 {
00259 private:
00260     uint32_t offset;
00261
00262 public:
00263     UsersListRequestMessage() : offset(0) {}
00264     UsersListRequestMessage(unsigned int offset) : offset(offset) {}
00265     ~UsersListRequestMessage() {}
00266
00267     msglen_t write(char *buffer);
00268     msglen_t read(char *buffer, msglen_t len);
00269
00270     string getName() { return "Users list request"; }
00271
00272     uint32_t getOffset() { return offset; }
00273
00274     MessageType getType() { return USERS_LIST_REQ; }
00275 };
00276
00280 class ChallengeForwardMessage : public Message
00281 {
00282 private:
00283     string username;
00284
00285 public:
00286     ChallengeForwardMessage() {}
00287     ChallengeForwardMessage(string username) : username(username) {}
00288     ~ChallengeForwardMessage() {}
00289
00290     msglen_t write(char *buffer);
00291     msglen_t read(char *buffer, msglen_t len);
00292
00293     string getName() { return "Challenge forward"; }
00294
00295     string getUsername() { return username; }
00296
00297     MessageType getType() { return CHALLENGE_FWD; }
00298 };
00299
00303 class ChallengeResponseMessage : public Message
00304 {
00305 private:
00306     string username;
00307     bool response;
00308     uint16_t listen_port;
00309
00310 public:
00311     ChallengeResponseMessage() {}
00312     ChallengeResponseMessage(string username, bool response, uint16_t port)
00313         : username(username), response(response), listen_port(port) {}
00314     ~ChallengeResponseMessage() {}
00315
00316     msglen_t write(char *buffer);
00317     msglen_t read(char *buffer, msglen_t len);
00318
00319     string getName() { return "Challenge response"; }
00320
00321     string getUsername() { return username; }
00322     bool getResponse() { return response; }
00323     uint16_t getListenPort() { return listen_port; }
00324
00325     MessageType getType() { return CHALLENGE_RESP; }
```

```
00326 };
00327
00332 class GameCancelMessage : public Message
00333 {
00334 private:
00335     string username;
00336
00337 public:
00338     GameCancelMessage() {}
00339     GameCancelMessage(string username) : username(username) {}
00340     ~GameCancelMessage() {}
00341
00342     msglen_t write(char *buffer);
00343     msglen_t read(char *buffer, msglen_t len);
00344
00345     string getName() { return "Game cancel"; }
00346
00347     string getUsername() { return username; }
00348
00349     MessageType getType() { return GAME_CANCEL; }
00350 };
00351
00355 class GameStartMessage : public Message
00356 {
00357 private:
00358     string username;
00359     struct sockaddr_in addr;
00360     X509* cert;
00361
00362 public:
00363     GameStartMessage() {}
00364     GameStartMessage(string username, struct sockaddr_in addr, X509* opp_cert)
00365         : username(username), addr(addr), cert(opp_cert) {} //TODO cert
00366     ~GameStartMessage() {}
00367
00368     msglen_t write(char *buffer);
00369     msglen_t read(char *buffer, msglen_t len);
00370
00371     string getName() { return "Game start"; }
00372
00373     string getUsername() { return username; }
00374     struct sockaddr_in getAddr() { return addr; }
00375     SecureHost getHost() { return SecureHost(addr, cert); }
00376     X509*      getCert() { return cert; }
00377     MessageType getType() { return GAME_START; }
00378 };
00379
00380 class SecureMessage : public Message
00381 {
00382 private:
00383     char *ct;
00384     msglen_t ct_size;
00385     char* tag;
00386
00387 public:
00388     SecureMessage() : ct(NULL), ct_size(0), tag(NULL){}
00389     SecureMessage(char* ct, msglen_t ct_size, char* tag) : ct(ct), ct_size(ct_size), tag(tag){}
00390     ~SecureMessage(){ if (ct != NULL) free(ct); if (tag != NULL) free(tag); }
00391
00392     MessageType getType() { return SECURE_MESSAGE; }
00393     string getName() { return "Secure message"; }
00394
00395     void setCtSize(msglen_t s) { ct_size = s; }
00396     size_t getCtSize() { return ct_size; }
00397     char* getCt() { return ct; }
00398     char* getTag() { return tag; }
00399
00400     msglen_t write(char *buffer);
00401     msglen_t read(char *buffer, msglen_t len);
00402 };
00403
00404 class ClientHelloMessage: public Message
00405 {
00406 private:
00407     EVP_PKEY* eph_key;
00408     nonce_t nonce;
00409     string my_id;
00410     string other_id;
00411
00412 public:
00413     ClientHelloMessage() : eph_key(NULL) {}
00414     ClientHelloMessage(EVP_PKEY* eph_key, nonce_t nonce, string my_id, string other_id)
00415         : eph_key(eph_key), nonce(nonce), my_id(my_id), other_id(other_id) {}
00416
00417     MessageType getType() {return CLIENT_HELLO; }
00418     string getName() { return "Client Hello message"; }
00419
```

```
00420     nonce_t getNonce() { return nonce; }
00421     EVP_PKEY* getEphKey() { return eph_key; }
00422     void setEphKey(EVP_PKEY* eph_key) { this->eph_key=eph_key; }
00423     string getMyId() { return my_id; }
00424     string getOtherId() { return other_id; }
00425 
00426     msglen_t write(char* buffer);
00427     msglen_t read(char* buffer, msglen_t len);
00428 };
00429 
00430 class ClientVerifyMessage: public Message
00431 {
00432 private:
00433     char* ds;
00434     uint32_t ds_size;
00435 
00436 public:
00437     ClientVerifyMessage() : ds(NULL), ds_size(0) {}
00438     ClientVerifyMessage(char* ds, uint32_t ds_size) : ds(ds), ds_size(ds_size) {}
00439     ~ClientVerifyMessage();
00440 
00441     MessageType getType() {return CLIENT_VERIFY; }
00442     string getName() { return "Client Verify message"; }
00443 
00444     char* getDs() { return ds; }
00445     uint32_t getDsSize() { return ds_size; }
00446 
00447     msglen_t write(char* buffer);
00448     msglen_t read(char* buffer, msglen_t len);
00449 };
00450 
00451 class ServerHelloMessage: public Message
00452 {
00453 private:
00454     EVP_PKEY* eph_key;
00455     nonce_t nonce;
00456     string my_id;
00457     string other_id;
00458     char* ds;
00459     uint32_t ds_size;
00460 
00461 public:
00462     ServerHelloMessage() : eph_key(NULL), ds(NULL), ds_size(0) {}
00463     ServerHelloMessage(EVP_PKEY* eph_key, nonce_t nonce, string my_id, string other_id, char* ds,
      uint32_t ds_size)
00464         : eph_key(eph_key), nonce(nonce), my_id(my_id), other_id(other_id), ds(ds), ds_size(ds_size)
      {}
00465     ~ServerHelloMessage();
00466 
00467     MessageType getType() {return SERVER_HELLO; }
00468     string getName() { return "Server Hello message"; }
00469 
00470     nonce_t getNonce() { return nonce; }
00471     EVP_PKEY* getEphKey() { return eph_key; }
00472     void setEphKey(EVP_PKEY* eph_key) { this->eph_key=eph_key; }
00473     string getMyId() { return my_id; }
00474     string getOtherId() { return other_id; }
00475     char* getDs() { return ds; }
00476     uint32_t getDsSize() { return ds_size; }
00477 
00478     msglen_t write(char* buffer);
00479     msglen_t read(char* buffer, msglen_t len);
00480 };
00481 
00482 class CertificateRequestMessage: public Message
00483 {
00484 public:
00485     CertificateRequestMessage(){}
00486 
00487     MessageType getType() {return CERT_REQ; }
00488     string getName() { return "Certificate Request message"; }
00489 
00490     msglen_t write(char* buffer);
00491     msglen_t read(char* buffer, msglen_t len);
00492 };
00493 
00494 class CertificateMessage: public Message
00495 {
00496 private:
00497     X509* cert;
00498 public:
00499     CertificateMessage() : cert(NULL) {}
00500     CertificateMessage(X509* cert) : cert(cert) {}
00501 
00502     MessageType getType() {return CERTIFICATE; }
00503     string getName() { return "Certificate message"; }
00504     X509* getCert() { return cert; }
```

```
00505
00506     msglen_t write(char* buffer);
00507     msglen_t read(char* buffer, msglen_t len);
00508 };
00509
00515 Message *readMessage(char *buffer, msglen_t len);
00516
00517 #endif // MESSAGES_H
```

7.35 multi_player.h File Reference

Implementation of the multi player game main function and connection with peer functions.

```
#include "security/secure_socket_wrapper.h"
#include "network/host.h"
```

Macros

- #define **MY_TURN** (0)
- #define **THEIR_TURN** (1)

Functions

- int playWithPlayer (int turn, SecureSocketWrapper *sw)

  Play against the player at the given socket.
- SecureSocketWrapper * waitForPeer (int port, SecureHost peer, X509 *cert, EVP_PKEY *key, X509_STORE *store)

  Starts a server on the given port waiting for peers to connect.
- SecureSocketWrapper * connectToPeer (SecureHost peer, X509 *cert, EVP_PKEY *key, X509_STORE *store)

  Connects to the server of another peer.

7.35.1 Detailed Description

Implementation of the multi player game main function and connection with peer functions.

Definition of the multi player game main function and connection with peer functions.

Author

Riccardo Mancini

Date

2020-05-29

Definition in file multi_player.h.

7.36 multi_player.h

```
00001
00011 #ifndef MULTI_PLAYER_H
00012 #define MULTI_PLAYER_H
00013
00014 #include "security/secure_socket_wrapper.h"
00015 #include "network/host.h"
00016
00017 #define MY_TURN    (0)
00018 #define THEIR_TURN (1)
00019
00023 int playWithPlayer(int turn, SecureSocketWrapper *sw);
00024
00028 SecureSocketWrapper* waitForPeer(int port, SecureHost peer, X509* cert, EVP_PKEY* key, X509_STORE*
       store);
00029
00033 SecureSocketWrapper* connectToPeer(SecureHost peer, X509* cert, EVP_PKEY* key, X509_STORE* store);
00034
00035 #endif // MULTI_PLAYER_H
```

7.37 secure_host.h File Reference

Definition of the helper class "SecureHost".

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <string>
#include "logging.h"
#include "network/host.h"
#include "security/crypto.h"
```

Data Structures

- class SecureHost

  Class that holds a host information with certificate.

7.37.1 Detailed Description

Definition of the helper class "SecureHost".

Author

Riccardo Mancini

Date

2020-06-11

Definition in file secure_host.h.

7.38 secure_host.h

```
00001
00010 #ifndef SECURE_HOST_H
00011 #define SECURE_HOST_H
00012
00013 #include <sys/socket.h>
00014 #include <netinet/in.h>
00015 #include <string>
00016 #include "logging.h"
00017 #include "network/host.h"
00018 #include "security/crypto.h"
00019
00020 using namespace std;
00021
00025 class SecureHost : public Host{
00026 private:
00027     X509* cert;
00028 public:
00032     SecureHost() {}
00033
00040     SecureHost(struct sockaddr_in addr, X509* cert) : Host(addr), cert(cert) {}
00041
00048     SecureHost(const char* ip, int port, X509* cert) : Host(ip, port), cert(cert) {}
00049
00051     X509* getCert(){return cert;}
00052 };
00053
00054 #endif // SECURE_HOST_H
```

7.39 secure_socket_wrapper.h File Reference

Header file for SecureSocketWrapper.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include "logging.h"
#include "network/messages.h"
#include "network/socket_wrapper.h"
#include "security/crypto.h"
#include "security/crypto_utils.h"
#include "security/secure_host.h"
#include "utils/dump_buffer.h"
```

Data Structures

- class SecureSocketWrapper
- class ClientSecureSocketWrapper

  SocketWrapper for a TCP client.
- class ServerSecureSocketWrapper

  SocketWrapper for a TCP server.

Macros

- #define **MAX_MSG_TO_SIGN_SIZE** (2∗MAX_USERNAME_LENGTH + 2 ∗ sizeof(nonce_t) + 2 ∗ KEY_B↩ IO_MAX_SIZE)
- #define **MAX_SEC_MSG_SIZE** (MAX_MSG_SIZE - TAG_SIZE - sizeof(msglen_t) - 1)
- #define **AAD_SIZE** (sizeof(msglen_t) + 1)

7.39.1 Detailed Description

Header file for SecureSocketWrapper.

Author

Mirko Laruina

Date

2020-06-09

Definition in file secure_socket_wrapper.h.

7.40 secure_socket_wrapper.h

```
00001
00010 #ifndef SECURE_SOCKET_WRAPPER_H
00011 #define SECURE_SOCKET_WRAPPER_H
00012
00013 #include <sys/socket.h>
00014 #include <netinet/in.h>
00015 #include "logging.h"
00016 #include "network/messages.h"
00017 #include "network/socket_wrapper.h"
00018 #include "security/crypto.h"
00019 #include "security/crypto_utils.h"
00020 #include "security/secure_host.h"
00021 #include "utils/dump_buffer.h"
00022
00023 #define MAX_MSG_TO_SIGN_SIZE (2*MAX_USERNAME_LENGTH + 2 * sizeof(nonce_t) + 2 * KEY_BIO_MAX_SIZE )
00024 #define MAX_SEC_MSG_SIZE (MAX_MSG_SIZE - TAG_SIZE - sizeof(msglen_t) - 1)
00025
00026 #define AAD_SIZE (sizeof(msglen_t) + 1)
00027
00028 class SecureSocketWrapper
00029 {
00030 protected:
00031     SocketWrapper *sw;
00032
00033     char send_key[KEY_SIZE];
00034     char recv_key[KEY_SIZE];
00035     char send_iv_static[IV_SIZE];
00036     char recv_iv_static[IV_SIZE];
00037     char send_iv[IV_SIZE];
00038     char recv_iv[IV_SIZE];
00039     uint64_t send_seq_num;
00040     uint64_t recv_seq_num;
00041     string my_id;
00042     string other_id;
00043     nonce_t sv_nonce;
00044     nonce_t cl_nonce;
00045     EVP_PKEY *my_eph_key;
00046     EVP_PKEY *other_eph_key;
00047     X509 *my_cert;
00048     X509 *other_cert;
00049     X509_STORE *store;
00050     EVP_PKEY *my_priv_key;
00051
00052     bool peer_authenticated;
00053
00054     char msg_to_sign_buf[MAX_MSG_TO_SIGN_SIZE];
00055
00059     SecureSocketWrapper(){};
00060
00066     void generateKeys(const char *role);
00067
00072     void updateSendIV();
00073
00078     void updateRecvIV();
00079
00081     void init(X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store);
00082
00089     Message *decryptMsg(SecureMessage *sm);
00090
```

```
00097     SecureMessage *encryptMsg(Message *m);
00098
00102     int makeSignature(const char *role, char** ds);
00103
00107     bool checkSignature(char *ds, size_t ds_size, const char *role);
00108
00116     int buildMsgToSign(const char *role, char *msg);
00117
00130     void makeAAD(MessageType msg_type, msglen_t len, char* aad);
00131
00132 public:
00136     SecureSocketWrapper(X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store);
00137
00141     SecureSocketWrapper(X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store, int sd);
00142
00146     SecureSocketWrapper(X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store, SocketWrapper *sw);
00147
00151     ~SecureSocketWrapper();
00152
00162     Message *readPartMsg();
00163
00171     Message *receiveAnyMsg();
00172
00183     Message *receiveMsg(MessageType type);
00184
00196     Message *receiveMsg(MessageType type[], int n_types);
00197
00198     Message *handleMsg(Message *msg);
00199
00200
00201     int sendCertRequest();
00202
00203     int handleCertResponse(CertificateMessage* cm);
00204
00205     int handleClientHello(ClientHelloMessage *chm);
00206     int handleServerHello(ServerHelloMessage *shm);
00207     int handleClientVerify(ClientVerifyMessage *cvm);
00208
00209     int sendClientHello();
00210     int sendServerHello();
00211     int sendClientVerify();
00212
00213     int sendPlain(Message *msg);
00220     int sendMsg(Message *msg);
00221
00227     int handshakeServer();
00228
00234     int handshakeClient();
00235
00239     bool setOtherCert(X509 *other_cert);
00240
00244     int getDescriptor() { return sw->getDescriptor(); };
00245
00249     void closeSocket() { sw->closeSocket(); }
00250
00257     void setOtherAddr(struct sockaddr_in addr) { sw->setOtherAddr(addr); }
00258
00259     sockaddr_in *getOtherAddr() { return sw->getOtherAddr(); }
00260
00264     SecureHost getConnectedHost() { return SecureHost(*getOtherAddr(), other_cert); }
00265
00269     X509* getCert(){ return my_cert;}
00270 };
00271
00277 class ClientSecureSocketWrapper : public SecureSocketWrapper
00278 {
00279 private:
00280     ClientSocketWrapper *csw;
00281
00282 public:
00288     ClientSecureSocketWrapper(X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store);
00289
00295     int connectServer(SecureHost host);
00296
00297 };
00298
00305 class ServerSecureSocketWrapper : public SecureSocketWrapper
00306 {
00307 private:
00308     ServerSocketWrapper *ssw;
00309
00310 public:
00316     ServerSecureSocketWrapper(X509 *cert, EVP_PKEY *my_priv_key, X509_STORE *store);
00317
00325     int bindPort(int port) { return ssw->bindPort(port); }
00326
00333     int bindPort() { return ssw->bindPort(); }
```

```
00334
00338     SecureSocketWrapper *acceptClient();
00339
00345     SecureSocketWrapper *acceptClient(X509 *other_cert);
00346
00350     int getPort() { return ssw->getPort(); }
00351 };
00352
00353 #endif
```

7.41 server.cpp File Reference

Implementation of a 4-in-a-row online server.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <pthread.h>
#include <map>
#include <queue>
#include <utility>
#include <stdio.h>
#include <errno.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include "logging.h"
#include "config.h"
#include "network/socket_wrapper.h"
#include "network/host.h"
#include "user.h"
#include "user_list.h"
#include "utils/message_queue.h"
#include "security/crypto_utils.h"
```

Typedefs

- typedef pair< int, Message * > **msgqueue_t**
- typedef map< string, X509 * > **cert_map_t**

Functions

- void **logUnexpectedMessage** (User *u, Message *m)
- void **doubleLock** (User *u_with_lock, User *u_without_lock)
- void **doubleUnlock** (User *u_keep_lock, User *u_unlock)
- bool **handleRegisterMessage** (User *u, RegisterMessage *msg)
- bool **handleChallengeMessage** (User *u, ChallengeMessage *msg)
- bool **handleGameEndMessage** (User *u, GameEndMessage *msg)
- bool **handleUsersListRequestMessage** (User *u, UsersListRequestMessage *msg)
- bool **handleChallengeResponseMessage** (User *u, ChallengeResponseMessage *msg)
- bool **handleClientHelloMessage** (User *u, ClientHelloMessage *chm)
- bool **handleClientVerifyMessage** (User *u, ClientVerifyMessage *cvm)

- bool **handleCertificateRequestMessage** (User *u, CertificateRequestMessage *crm)
- bool **handleMessage** (User *user, Message *raw_msg)
- void * **worker** (void *args)
- void **init_threads** ()
- bool **checkCertsInCertMap** (X509_STORE *store, cert_map_t cert_map)
- int **main** (int argc, char **argv)

7.41.1 Detailed Description

Implementation of a 4-in-a-row online server.

Author

Riccardo Mancini

The select implementation was inspired by https://www.gnu.org/software/libc/manual/html_node/Server-Example.html

Date

2020-05-23

Definition in file server/server.cpp.

7.42 server/server.cpp

```
00001
00012 #include <iostream>
00013 #include <cstdlib>
00014 #include <ctime>
00015 #include <pthread.h>
00016 #include <map>
00017 #include <queue>
00018 #include <utility>
00019 #include <stdio.h>
00020 #include <errno.h>
00021 #include <stdlib.h>
00022 #include <unistd.h>
00023 #include <sys/types.h>
00024 #include <sys/socket.h>
00025 #include <netinet/in.h>
00026 #include <netdb.h>
00027
00028 #include "logging.h"
00029 #include "config.h"
00030 #include "network/socket_wrapper.h"
00031 #include "network/host.h"
00032
00033 #include "user.h"
00034 #include "user_list.h"
00035 #include "utils/message_queue.h"
00036
00037 #include "security/crypto_utils.h"
00038
00039 using namespace std;
00040
00041 typedef pair<int,Message*> msgqueue_t;
00042 typedef map<string,X509*> cert_map_t;
00043
00044 static UserList user_list;
00045 static MessageQueue<msgqueue_t,MAX_QUEUE_LENGTH> message_queue;
00046 static pthread_t threads[N_THREADS];
00047 static cert_map_t cert_map;
00048 static X509* cert;
00049
00050 void logUnexpectedMessage(User* u, Message* m){
00051     LOG(LOG_WARN, "User %s (state %d) was not expecting a message of type %d",
00052         u->getUsername().c_str(), (int)u->getState(), (int)m->getType());
```

```
00053 }
00054
00055 void doubleLock(User* u_with_lock, User* u_without_lock){
00056     // prevent deadlocks
00057     if (u_without_lock->getUsername() < u_with_lock->getUsername()){
00058         u_with_lock->unlock();
00059         u_without_lock->lock();
00060         u_with_lock->lock();
00061     } else{
00062         u_without_lock->lock();
00063     }
00064 }
00065
00066 void doubleUnlock(User* u_keep_lock, User* u_unlock){
00067     // prevent deadlocks
00068     if (u_unlock->getUsername() < u_keep_lock->getUsername()){
00069         u_keep_lock->unlock();
00070         u_unlock->unlock();
00071         u_keep_lock->lock();
00072     } else{
00073         u_unlock->unlock();
00074     }
00075 }
00076
00077 bool handleRegisterMessage(User* u, RegisterMessage* msg){
00078     string username = msg->getUsername();
00079     string usernameCert = usernameFromCert(u->getSocketWrapper()->getCert());
00080     if (username.compare(usernameCert) == 0){
00081         LOG(LOG_WARN, "Malicious operation: %s tried to register as %s",
00082                 usernameCert.c_str(), username.c_str());
00083         return false;
00084     }
00085     u->setUsername(username);
00086
00087     if (!user_list.exists(username)){
00088         u->setState(AVAILABLE);
00089         // readd with username
00090         user_list.add(u);
00091         return true;
00092     } else {
00093         LOG(LOG_WARN, "User %s already registered!", username.c_str());
00094         //TODO send error
00095         u->setState(DISCONNECTED);
00096         return false;
00097     }
00098 }
00099
00100 bool handleChallengeMessage(User* u, ChallengeMessage* msg){
00101     bool res;
00102     string chlg_username = msg->getUsername();
00103     User* challenged = user_list.get(chlg_username);
00104     if (challenged == NULL || challenged == u){
00105         GameCancelMessage cancel_msg(chlg_username);
00106         return u->getSocketWrapper()->sendMsg(&cancel_msg) == 0;
00107     }
00108
00109     doubleLock(u, challenged);
00110
00111     if (u->getState() != AVAILABLE){
00112         // someother thing concurrently happened, ignore
00113         doubleUnlock(u, challenged);
00114         user_list.yield(challenged);
00115         return true;
00116     }
00117
00118     if (challenged->getState() != AVAILABLE){
00119         // someother thing concurrently happened, abort
00120         GameCancelMessage cancel_msg(chlg_username);
00121         res = u->getSocketWrapper()->sendMsg(&cancel_msg) == 0;
00122
00123         doubleUnlock(u, challenged);
00124         user_list.yield(challenged);
00125         return res;
00126     }
00127
00128     // both are available, send challenge
00129     ChallengeForwardMessage fwd_msg(u->getUsername());
00130     if (challenged->getSocketWrapper()->sendMsg(&fwd_msg) == 0){
00131         // challenge sent, mark them as playing until I receive a response
00132         u->setState(CHALLENGED);
00133         u->setOpponent(challenged->getUsername());
00134         challenged->setState(CHALLENGED);
00135         challenged->setOpponent(u->getUsername());
00136         res = true;
00137     } else{
00138         // connection error -> assume disconnected and notify u
00139         GameCancelMessage cancel_msg(chlg_username);
```

```
00140         challenged->setState(DISCONNECTED);
00141         res = u->getSocketWrapper()->sendMsg(&cancel_msg) == 0;
00142     }
00143
00144     doubleUnlock(u, challenged);
00145     user_list.yield(challenged);
00146
00147     return res;
00148 }
00149
00150 bool handleGameEndMessage(User* u, GameEndMessage* msg){
00151     u->setState(AVAILABLE);
00152     return true;
00153 }
00154
00155 bool handleUsersListRequestMessage(User* u, UsersListRequestMessage* msg){
00156     UsersListMessage ul_msg(user_list.listAvailableFromTo(msg->getOffset()));
00157     return u->getSocketWrapper()->sendMsg(&ul_msg) == 0;
00158 }
00159
00160 bool handleChallengeResponseMessage(User* u, ChallengeResponseMessage* msg){
00161     bool res;
00162     User *opponent = user_list.get(u->getOpponent());
00163     if (opponent == NULL || opponent == u){
00164         // opponent disconnected or invalid opponent -> cancel
00165         u->setState(AVAILABLE);
00166         GameCancelMessage cancel_msg(u->getOpponent());
00167         return u->getSocketWrapper()->sendMsg(&cancel_msg) == 0;
00168     }
00169
00170     doubleLock(u, opponent);
00171
00172     if (msg->getResponse()){ // accepted
00173         if (u->getState() != CHALLENGED){
00174             // someother thing concurrently happened, abort
00175             // maybe this refers to old challenge
00176             doubleUnlock(u, opponent);
00177             user_list.yield(opponent);
00178             return true;
00179         }
00180
00181         if (opponent->getState() != CHALLENGED){
00182             // opponent is in wrong state...
00183             // maybe this refers to old challenge
00184             // notify u of opponent not ready
00185             GameCancelMessage cancel_msg(opponent->getUsername());
00186             res = u->getSocketWrapper()->sendMsg(&cancel_msg) == 0;
00187             doubleUnlock(u, opponent);
00188             user_list.yield(opponent);
00189             return res;
00190         }
00191
00192         struct sockaddr_in opp_addr = opponent->getSocketWrapper()
00193                                       ->getConnectedHost().getAddress();
00194         opp_addr.sin_port = 0;
00195         cert_map_t::iterator opp_pair = cert_map.find(opponent->getUsername());
00196         if(opp_pair == cert_map.end()) {
00197             doubleUnlock(u, opponent);
00198             user_list.yield(opponent);
00199             return false;
00200         }
00201         GameStartMessage msg_to_u(opponent->getUsername(), opp_addr, opp_pair->second);
00202
00203         struct sockaddr_in u_addr = u->getSocketWrapper()
00204                                     ->getConnectedHost().getAddress();
00205         u_addr.sin_port = htons(msg->getListenPort());
00206         cert_map_t::iterator u_pair = cert_map.find(u->getUsername());
00207         if(u_pair == cert_map.end()) {
00208             doubleUnlock(u, opponent);
00209             user_list.yield(opponent);
00210             return false;
00211         }
00212         GameStartMessage msg_to_opp(u->getUsername(), u_addr, u_pair->second);
00213
00214         int res_u = u->getSocketWrapper()->sendMsg(&msg_to_u);
00215         int res_opp = opponent->getSocketWrapper()->sendMsg(&msg_to_opp);
00216
00217         if (res_u == 0 && res_opp == 0){
00218             //success
00219             u->setState(PLAYING);
00220             opponent->setState(PLAYING);
00221
00222             res = true;
00223         } else if (res_u != 0 && res_opp != 0){
00224             // both disconnected
00225             opponent->setState(DISCONNECTED);
00226             u->setState(DISCONNECTED);
```

```
00227             res = false;
00228         } else {
00229             if (res_u != 0){ // just u disconnected => notify opp
00230                 GameCancelMessage cancel_msg(u->getUsername());
00231                 if (opponent->getSocketWrapper()->sendMsg(&cancel_msg) == 0){
00232                     opponent->setState(AVAILABLE);
00233                 } else {
00234                     opponent->setState(DISCONNECTED);
00235                 }
00236             } else if (res_opp != 0){ // just opp disconnected => notify u
00237                 GameCancelMessage cancel_msg(opponent->getUsername());
00238                 if (u->getSocketWrapper()->sendMsg(&cancel_msg) == 0){
00239                     u->setState(AVAILABLE);
00240                     res = true;
00241                 } else {
00242                     u->setState(DISCONNECTED);
00243                     res = false;
00244                 }
00245             }
00246         }
00247     } else{ // rejected
00248         u->setState(AVAILABLE);
00249         GameCancelMessage cancel_msg(u->getUsername());
00250         if (opponent->getSocketWrapper()->sendMsg(&cancel_msg) == 0){
00251             opponent->setState(AVAILABLE);
00252         } else{
00253             opponent->setState(DISCONNECTED);
00254         }
00255     }
00256
00257     doubleUnlock(u, opponent);
00258     user_list.yield(opponent);
00259
00260     return res;
00261 }
00262
00263 bool handleClientHelloMessage(User* u, ClientHelloMessage* chm){
00264     string username = chm->getMyId();
00265     cert_map_t::iterator res;
00266     SecureSocketWrapper *sw = u->getSocketWrapper();
00267
00268     if ((res = cert_map.find(username)) != cert_map.end()){
00269         sw->setOtherCert(res->second);
00270         int ret = u->getSocketWrapper()->handleClientHello(chm);
00271         return ret == 0;
00272     } else{
00273         LOG(LOG_WARN, "User %s not found in cert_map", username.c_str());
00274         return false;
00275     }
00276 }
00277
00278 bool handleClientVerifyMessage(User* u, ClientVerifyMessage* cvm){
00279     int ret = u->getSocketWrapper()->handleClientVerify(cvm);
00280     if(ret == 0){
00281         u->setState(SECURELY_CONNECTED);
00282         return true;
00283     } else {
00284         LOG(LOG_ERR, "Client Verify failed!");
00285         u->setState(DISCONNECTED);
00286         return false;
00287     }
00288 }
00289
00290 bool handleCertificateRequestMessage(User* u, CertificateRequestMessage* crm){
00291     CertificateMessage cm(cert);
00292     int ret = u->getSocketWrapper()->sendPlain(&cm);
00293     if(ret == 0){
00294         return true;
00295     } else {
00296         LOG(LOG_ERR, "Error sending certificate to client! Error %d", ret);
00297         u->setState(DISCONNECTED);
00298         return false;
00299     }
00300 }
00301
00302 bool handleMessage(User* user, Message* raw_msg){
00303     bool res = true;
00304
00305     user->lock();
00306
00307     try{
00308
00309         Message* msg = user->getSocketWrapper()->handleMsg(raw_msg);
00310
00311         if (msg == NULL)
00312             return false;
00313
```

```
00314         LOG(LOG_INFO, "User %s (state %d) received a message of type %s",
00315             user->getUsername().c_str(), (int) user->getState(), msg->getName().c_str());
00316 
00317         switch(user->getState()){
00318             case JUST_CONNECTED:
00319                 switch(msg->getType()){
00320                     case CLIENT_HELLO:
00321                         res = handleClientHelloMessage(user,
00322                             dynamic_cast<ClientHelloMessage*>(msg));
00323                         break;
00324                     case CLIENT_VERIFY:
00325                         res = handleClientVerifyMessage(user,
00326                             dynamic_cast<ClientVerifyMessage*>(msg));
00327                         break;
00328                     case CERT_REQ:
00329                         res = handleCertificateRequestMessage(user,
00330                             dynamic_cast<CertificateRequestMessage*>(msg));
00331                     // TODO: handle cert request
00332                     default:
00333                         logUnexpectedMessage(user, msg);
00334                 }
00335                 break;
00336             case SECURELY_CONNECTED:
00337                 switch(msg->getType()){
00338                     case REGISTER:
00339                         res = handleRegisterMessage(user,
00340                             dynamic_cast<RegisterMessage*>(msg));
00341                         break;
00342                     default:
00343                         logUnexpectedMessage(user, msg);
00344                 }
00345                 break;
00346             case AVAILABLE:
00347                 switch(msg->getType()){
00348                     case CHALLENGE:
00349                         res = handleChallengeMessage(user,
00350                             dynamic_cast<ChallengeMessage*>(msg));
00351                         break;
00352                     case USERS_LIST_REQ:
00353                         res = handleUsersListRequestMessage(user,
00354                             dynamic_cast<UsersListRequestMessage*>(msg));
00355                         break;
00356                     default:
00357                         logUnexpectedMessage(user, msg);
00358                 }
00359                 break;
00360             case CHALLENGED:
00361                 switch(msg->getType()){
00362                     case CHALLENGE_RESP:
00363                         res = handleChallengeResponseMessage(user,
00364                             dynamic_cast<ChallengeResponseMessage*>(msg));
00365                         break;
00366                     default:
00367                         logUnexpectedMessage(user, msg);
00368                 }
00369                 break;
00370             case PLAYING:
00371                 switch(msg->getType()){
00372                     case GAME_END:
00373                         res = handleGameEndMessage(user,
00374                             dynamic_cast<GameEndMessage*>(msg));
00375                         break;
00376                     default:
00377                         logUnexpectedMessage(user, msg);
00378                 }
00379                 break;
00380             default:
00381                 LOG(LOG_ERR, "User %s is in unrecognized state %d",
00382                     user->getUsername().c_str(), (int) user->getState());
00383         }
00384 
00385         delete msg;
00386     } catch(const char* error_msg){
00387         LOG(LOG_ERR, "Caught error: %s", error_msg);
00388         res = false;
00389     }
00390 
00391     user->unlock();
00392     return res;
00393 }
00394 
00395 void* worker(void *args){
00396     while (1){
00397         msgqueue_t p = message_queue.pullWait();
00398         User* u = user_list.get(p.first);
00399         if (u != NULL){
00400             if (!handleMessage(u, p.second)){
```

```
00401                 // Connection error -> assume disconnected
00402                 u->setState(DISCONNECTED);
00403             }
00404             user_list.yield(u);
00405         }
00406     }
00407
00408 }
00409
00410 void init_threads(){
00411     for (int i=0; i < N_THREADS; i++){
00412         pthread_create(&threads[i], NULL, worker, NULL);
00413     }
00414 }
00415
00416 bool checkCertsInCertMap(X509_STORE* store, cert_map_t cert_map){
00417     for (cert_map_t::iterator it = cert_map.begin();
00418         it != cert_map.end();
00419         ++it
00420     ){
00421         if (!verify_peer_cert(store, it->second)){
00422             LOG(LOG_ERR, "Validation failed for certificate in directory: %s",
00423                     it->first.c_str());
00424             return false;
00425         }
00426     }
00427     return true;
00428
00429 }
00430
00431 int main(int argc, char** argv){
00432     fd_set active_fd_set, read_fd_set;
00433
00434     if (argc < 7){
00435         cout«"Usage: "«argv[0]«" port cert.pem key.pem cacert.pem crl.pem"«endl;
00436         exit(1);
00437     }
00438
00439     int port = atoi(argv[1]);
00440     cert = load_cert_file(argv[2]);
00441     EVP_PKEY* key = load_key_file(argv[3], NULL);
00442     X509* cacert = load_cert_file(argv[4]);
00443     X509_CRL* crl = load_crl_file(argv[5]);
00444     X509_STORE* store = build_store(cacert, crl);
00445     cert_map = buildCertMapFromDirectory(argv[6]);
00446
00447     if (cert_map.size() == 0){
00448         LOG(LOG_ERR, "No certificates found in directory");
00449         return 1;
00450     }
00451
00452     if (!checkCertsInCertMap(store, cert_map)){
00453         return 1;
00454     }
00455
00456     LOG(LOG_INFO, "Loaded certificates from %s", argv[6]);
00457
00458     ServerSecureSocketWrapper server_sw(cert, key, store);
00459
00460     int ret = server_sw.bindPort(port);
00461     if (ret != 0){
00462         LOG(LOG_FATAL, "Error binding to port %d", port);
00463         exit(1);
00464     }
00465
00466     LOG(LOG_INFO, "Binded to port %d", port);
00467
00468     init_threads();
00469
00470     LOG(LOG_INFO, "Started %d worker threads", N_THREADS);
00471
00472     /* Initialize the set of active sockets. */
00473     FD_ZERO(&active_fd_set);
00474     FD_SET(server_sw.getDescriptor(), &active_fd_set);
00475
00476     LOG(LOG_INFO, "Polling open sockets");
00477
00478     while (1){
00479         /* Block until input arrives on one or more active sockets. */
00480         read_fd_set = active_fd_set;
00481         if (select(FD_SETSIZE, &read_fd_set, NULL, NULL, NULL) < 0) {
00482             if (errno == EBADF){ // clean closed sockets
00483                 LOG(LOG_DEBUG, "Bad file descriptor");
00484                 for (int i = 0; i < FD_SETSIZE; ++i){
00485                     if (FD_ISSET(i, &active_fd_set)){
00486                         if (i != server_sw.getDescriptor()
00487                                 && !user_list.exists(i)
```

```
00488                           ){
00489                           // user was disconnected but I still need to clear it
00490                           LOG(LOG_DEBUG, "Cleared fd %d", i);
00491                           FD_CLR(i, &active_fd_set);
00492                       }
00493                   }
00494               }
00495               continue;
00496           }
00497
00498           LOG_PERROR(LOG_FATAL, "Error in select: %s");
00499           exit(1);
00500       }
00501
00502       /* Service all the sockets with input pending. */
00503       for (int i = 0; i < FD_SETSIZE; ++i){
00504           if (FD_ISSET(i, &read_fd_set)){
00505               if (i == server_sw.getDescriptor()) {
00506                   /* Connection request on original socket. */
00507                   SecureSocketWrapper* sw = server_sw.acceptClient();
00508
00509                   LOG(LOG_INFO, "New connection from %s",
00510                       sw->getConnectedHost().toString().c_str());
00511
00512                   FD_SET(sw->getDescriptor(), &active_fd_set);
00513
00514                   User *u = new User(sw);
00515                   user_list.add(u);
00516               } else {
00517                   User *u = user_list.get(i);
00518                   if (u->getState() == DISCONNECTED){
00519                       LOG(LOG_DEBUG, "Received message from disconnected user with countRefs = %d",
      u->countRefs());
00520                       user_list.yield(u);
00521                       FD_CLR(i, &active_fd_set); // ignore him
00522                       continue;
00523                   }
00524                   const char* u_addr_str = u->getSocketWrapper()
00525                           ->getConnectedHost().toString().c_str();
00526                   LOG(LOG_INFO, "Available message from %s (%s)",
00527                       u->getUsername().c_str(), u_addr_str);
00528                   try{
00529                       Message* m = u->getSocketWrapper()->readPartMsg();
00530                       if (m != NULL)
00531                           message_queue.pushSignal(msgqueue_t(i, m));
00532                   } catch(const char* msg){
00533                       LOG(LOG_WARN, "Client %s disconnected: %s",
00534                           u_addr_str, msg);
00535                       u->setState(DISCONNECTED);
00536                   }
00537                   user_list.yield(u);
00538                   if (!user_list.exists(i)){
00539                       // user was disconnected -> clear it
00540                       FD_CLR(i, &active_fd_set);
00541                   }
00542               }
00543           } else if (FD_ISSET(i, &active_fd_set)){
00544               if (i != server_sw.getDescriptor()
00545                           && !user_list.exists(i)
00546               ){
00547                   LOG(LOG_DEBUG, "Cleared fd %d", i);
00548                   // user was disconnected but I still need to clear it
00549                   FD_CLR(i, &active_fd_set);
00550               }
00551           }
00552       }
00553   }
00554 }
00555
00556
```

7.43 server.h File Reference

Implementation of the utility class used to communicate with the server.

```
#include "security/secure_socket_wrapper.h"
#include "security/secure_host.h"
#include "security/crypto_utils.h"
```

Data Structures

- class Server

  Utility class for interacting with the server.

7.43.1 Detailed Description

Implementation of the utility class used to communicate with the server.

Definition of the utility class used to communicate with the server.

Author

Riccardo Mancini

Date

2020-05-29

Definition in file server.h.

7.44 server.h

```
00001
00010 #ifndef SERVER_H
00011 #define SERVER_H
00012
00013 #include "security/secure_socket_wrapper.h"
00014 #include "security/secure_host.h"
00015 #include "security/crypto_utils.h"
00016
00017 using namespace std;
00018
00022 class Server{
00023 private:
00024     SecureHost host;
00025     ClientSecureSocketWrapper* sw;
00026     bool connected;
00027 public:
00031     Server(SecureHost host, X509* cert, EVP_PKEY* key, X509_STORE* store) : host(host),
       connected(false) {sw = new ClientSecureSocketWrapper(cert, key, store);}
00032
00036     ~Server();
00037
00043     int getServerCert();
00044
00054     int registerToServer();
00055
00062     string getUserList();
00063
00077     int challengePeer(string username, SecureHost* peerHost);
00078
00091     int replyPeerChallenge(string username, bool response, SecureHost* peerHost, uint16_t
       *listen_port);
00092
00099     int signalGameEnd();
00100
00104     void disconnect();
00105
00109     SecureSocketWrapper* getSocketWrapper(){return sw;}
00110
00114     SecureHost getHost(){return host;}
00115
00119     string getPlayerUsername(){ return usernameFromCert(sw->getCert());}
00120
00124     bool isConnected(){ return connected; }
00125 };
00126
00127 #endif // SERVER_H
```

7.45 server_lobby.cpp File Reference

Implementation of the function that handles user and network input while the user is in the server lobby waiting for a game to start.

```
#include <sys/select.h>
#include <stdio.h>
#include <cstring>
#include <iostream>
#include <sstream>
#include "utils/args.h"
#include "security/secure_socket_wrapper.h"
#include "security/crypto_utils.h"
#include "server_lobby.h"
#include "server.h"
```

Macros

- #define STDIN (0)

  stdin has file descriptor 0 in Unix

Functions

- void **printAvailableActions** ()
- int **doAction** (Args args, Server ∗server, SecureHost ∗peer_host)
- int **handleReceivedChallenge** (Server ∗server, ChallengeForwardMessage ∗msg, SecureHost ∗peer_host, uint16_t ∗listen_port)
- ConnectionMode **handleMessage** (Message ∗msg, Server ∗server)
- ConnectionMode **handleStdin** (Server ∗server)
- ConnectionMode serverLobby (Server ∗server)

  Handles interaction among the user, the client and the remote server.

7.45.1 Detailed Description

Implementation of the function that handles user and network input while the user is in the server lobby waiting for a game to start.

Author

Riccardo Mancini

The select implementation was inspired by https://www.gnu.org/software/libc/manual/html_node/Server-Example.html

Date

2020-05-29

Definition in file server_lobby.cpp.

7.45.2 Function Documentation

7.45.2.1 serverLobby() ConnectionMode serverLobby (Server * server)

Handles interaction among the user, the client and the remote server.

While in the lobby, users can receive challenges from other users through the server, request the list of users in the server and challenge other users.

Once started, this function spawns a new connection to the given host and registers to it.

Definition at line 167 of file server_lobby.cpp.

7.46 server_lobby.cpp

```
00001
00014 #include <sys/select.h>
00015 #include <stdio.h>
00016 #include <cstring>
00017 #include <iostream>
00018 #include <sstream>
00019
00020 #include "utils/args.h"
00021 #include "security/secure_socket_wrapper.h"
00022 #include "security/crypto_utils.h"
00023
00024 #include "server_lobby.h"
00025 #include "server.h"
00026
00027 using namespace std;
00028
00030 #define STDIN (0)
00031
00032 void printAvailableActions(){
00033     cout«"You can list users, challenge a user, exit or simply wait for other users to challenge
       you."« endl;
00034     cout«"To list users type: `list`"« endl;
00035     cout«"To challenge a user type: `challenge username`"« endl;
00036     cout«"To disconnect type: `exit`"« endl;
00037     cout«"NB: you cannot receive challenges if you are challenging another user"« endl;
00038 }
00039
00040 int doAction(Args args, Server *server, SecureHost* peer_host){
00041     LOG(LOG_DEBUG, "Args: %s", args.c_str());
00042     if (args.getArgc() == 1 && strcmp(args.getArgv(0), "exit") == 0){
00043         return -2;
00044     } else if (args.getArgc() == 1 && strcmp(args.getArgv(0), "list") == 0){
00045         cout«"Retrieving the list of users..."«endl;
00046         string userlist = server->getUserList();
00047         if (userlist.empty()){
00048             return 1;
00049         }
00050         cout«"Online users: "«userlist«endl;
00051         return 0;
00052     } else if (args.getArgc() == 2 && strcmp(args.getArgv(0), "challenge") == 0){
00053         cout«"Sending challenge to "«args.getArgv(1)«" and waiting for response..."«endl;
00054         string username(args.getArgv(1));
00055         int ret = server->challengePeer(username, peer_host);
00056         switch (ret){
00057             case -1: // refused
00058                 cout«username«" refused your challenge"«endl;
00059                 return 0;
00060             case 0: //accepted
00061                 cout«username«" accepted your challenge"«endl;
00062                 return -1;
00063             default:
00064                 cout«"Error connecting to server!"«endl;
00065                 return 1;
00066         }
00067     } else if (args.getArgc() < 0) {
00068         return -2; //exit
00069     } else {
```

```
00070         return 0;
00071     }
00072
00073 }
00074
00075 int handleReceivedChallenge(Server *server,
00076                             ChallengeForwardMessage* msg,
00077                             SecureHost* peer_host,
00078                             uint16_t* listen_port){
00079     cout«endl«"You received a challenge from "«msg->getUsername()«endl;
00080     cout«"Do you want to accept? (y/n)";
00081
00082     bool response;
00083
00084     do{
00085         cout«"> "«flush;
00086         Args args(cin);
00087         LOG(LOG_DEBUG, "Args: %s", args.c_str());
00088         if (args.getArgc() == 1 && strcmp(args.getArgv(0), "y") == 0){
00089             response = true;
00090             break;
00091         } else if (args.getArgc() == 1 && strcmp(args.getArgv(0), "n") == 0){
00092             response = false;
00093             break;
00094         } else if (args.getArgc() < 0){ // EOF
00095             response = false;
00096             break;
00097         } else{
00098             continue;
00099         }
00100     } while(1);
00101
00102
00103     return server->replyPeerChallenge(msg->getUsername(), response, peer_host, listen_port);
00104 }
00105
00106 ConnectionMode handleMessage(Message* msg, Server* server){
00107     ChallengeForwardMessage* cfm;
00108     SecureHost peer_host;
00109     uint16_t listen_port;
00110
00111     int ret;
00112     LOG(LOG_INFO, "Server sent message %s", msg->getName().c_str());
00113     switch(msg->getType()){
00114         case CHALLENGE_FWD:
00115             cfm = dynamic_cast<ChallengeForwardMessage*>(msg);
00116             ret = handleReceivedChallenge(server, cfm, &peer_host, &listen_port);
00117             switch (ret){
00118                 case -1: // game canceled
00119                     cout«"Game was canceled"«endl;
00120                     return ConnectionMode(CONTINUE);
00121                 case 0:
00122                     cout«"Starting game..."«endl;
00123                     return ConnectionMode(WAIT_FOR_PEER, peer_host, listen_port);
00124                 default:
00125                     cout«"Error"«endl;
00126                     return ConnectionMode(EXIT, CONNECTION_ERROR);
00127             }
00128             break;
00129         default:
00130             // other messages are handled internally to
00131             // Server since they require the user to wait
00132             LOG(LOG_WARN, "Received unexpected message %s", msg->getName().c_str());
00133             return ConnectionMode(CONTINUE);
00134     }
00135 }
00136
00137 ConnectionMode handleStdin(Server* server){
00138     SecureHost peer_host;
00139
00140     int ret;
00141     // Input from user
00142     Args args(cin);
00143     if (args.getArgc() < 0){
00144         ret = -2; // received EOF
00145     } else{
00146         ret = doAction(args, server, &peer_host);
00147     }
00148     switch (ret){
00149         case 0: // do nothing
00150             LOG(LOG_DEBUG, "No action");
00151             return ConnectionMode(CONTINUE);
00152         case 1: // error
00153             cout«"Error!"«endl;
00154             return ConnectionMode(EXIT, CONNECTION_ERROR);
00155         case -1: // challenge accepted
00156             cout«"Starting game..."«endl;
```

```
00157             return ConnectionMode(CONNECT_TO_PEER, peer_host, 0);
00158         case -2:
00159             cout«"Bye"«endl;
00160             return ConnectionMode(EXIT, OK);
00161     default:
00162         return ConnectionMode(CONTINUE);
00163     }
00164 }
00165
00166
00167 ConnectionMode serverLobby(Server* server){
00168     fd_set active_fd_set, read_fd_set;
00169
00170     string username = server->getPlayerUsername();
00171
00172     if (!server->isConnected()){
00173         cout«"Registering to "«server->getHost().toString()«" as "«username«endl;
00174         if (server->registerToServer() != 0){
00175             cout«"Connection to "«server->getHost().toString()«" failed!"«endl;
00176             return ConnectionMode(EXIT, CONNECTION_ERROR);
00177         }
00178         LOG(LOG_INFO, "Server %s is now connected",
00179                         server->getHost().toString().c_str());
00180     } else {
00181         LOG(LOG_INFO, "Server %s was already connected",
00182                         server->getHost().toString().c_str());
00183     }
00184
00185     SecureHost peer_host;
00186
00187     /* Initialize the set of active sockets. */
00188     FD_ZERO(&active_fd_set);
00189     FD_SET(server->getSocketWrapper()->getDescriptor(), &active_fd_set);
00190     FD_SET(STDIN, &active_fd_set);
00191
00192     printAvailableActions();
00193
00194     while (1){
00195         cout«endl«"> "«flush;
00196
00197         /* Block until input arrives on one or more active sockets. */
00198         read_fd_set = active_fd_set;
00199         if (select(FD_SETSIZE, &read_fd_set, NULL, NULL, NULL) < 0) {
00200             LOG_PERROR(LOG_ERR, "Error in select: %s");
00201             return ConnectionMode(EXIT, GENERIC_ERROR);
00202         }
00203
00204         /* Service all the socketsexit(1); with input pending. */
00205         for (int i = 0; i < FD_SETSIZE; ++i){
00206             if (FD_ISSET(i, &read_fd_set)){
00207                 if (i == server->getSocketWrapper()->getDescriptor()){
00208                     // Message from server.
00209                     Message* msg;
00210                     try{
00211                         msg = server->getSocketWrapper()->receiveAnyMsg();
00212                     } catch(const char* msg){
00213                         LOG(LOG_ERR, "Error: %s", msg);
00214                         return ConnectionMode(EXIT, CONNECTION_ERROR);
00215                     }
00216
00217                     ConnectionMode m = handleMessage(msg, server);
00218                     if (m.connection_type != CONTINUE){
00219                         return m;
00220                     }
00221                 } else if (i == STDIN){
00222                     ConnectionMode m = handleStdin(server);
00223                     if (m.connection_type != CONTINUE){
00224                         return m;
00225                     }
00226                 }
00227             }
00228         }
00229     }
00230 }
00231
```

7.47 server_lobby.h File Reference

Definition of the function that handles user and network input while the user is in the server lobby waiting for a game to start.

```
#include "connection_mode.h"
#include "security/secure_host.h"
#include "security/crypto.h"
#include "server.h"
```

Functions

- ConnectionMode serverLobby (Server *server)

  Handles interaction among the user, the client and the remote server.

7.47.1 Detailed Description

Definition of the function that handles user and network input while the user is in the server lobby waiting for a game to start.

Author

Riccardo Mancini

Date

2020-05-29

Definition in file server_lobby.h.

7.47.2 Function Documentation

7.47.2.1 serverLobby() `ConnectionMode serverLobby ( Server * server )`

Handles interaction among the user, the client and the remote server.

While in the lobby, users can receive challenges from other users through the server, request the list of users in the server and challenge other users.

Once started, this function spawns a new connection to the given host and registers to it.

Definition at line 167 of file server_lobby.cpp.

7.48 server_lobby.h

```
00001
00011 #ifndef SERVER_LOBBY_H
00012 #define SERVER_LOBBY_H
00013
00014 #include "connection_mode.h"
00015 #include "security/secure_host.h"
00016 #include "security/crypto.h"
00017 #include "server.h"
00018
00029 ConnectionMode serverLobby(Server* server);
00030
00031 #endif // SERVER_LOBBY_H
```

7.49 single_player.h File Reference

Implementation of the single player game main function.

Functions

- int playSinglePlayer ()
  Starts a game against a random-playing opponent.

7.49.1 Detailed Description

Implementation of the single player game main function.

Definition of the single player game main function.

Author

Mirko Laruina
Riccardo Mancini

Date

2020-05-27

Author

Riccardo Mancini

Date

2020-05-27

Definition in file single_player.h.

7.50 single_player.h

```
00001
00010 #ifndef SINGLE_PLAYER_H
00011 #define SINGLE_PLAYER_H
00012
00016 int playSinglePlayer();
00017
00018 #endif // SINGLE_PLAYER_H
```

7.51 socket_wrapper.cpp File Reference

Implementation of socket_wrapper.h.

```
#include <assert.h>
#include "logging.h"
#include "network/socket_wrapper.h"
#include "utils/dump_buffer.h"
#include "network/inet_utils.h"
```

7.51.1 Detailed Description

Implementation of socket_wrapper.h.

Author

Riccardo Mancini

See also

socket_wrapper.h

Definition in file socket_wrapper.cpp.

7.52 socket_wrapper.cpp

```
00001
00010 #include <assert.h>
00011 #include "logging.h"
00012 #include "network/socket_wrapper.h"
00013 #include "utils/dump_buffer.h"
00014 #include "network/inet_utils.h"
00015
00016 SocketWrapper::SocketWrapper() {
00017     socket_fd = socket(AF_INET, SOCK_STREAM, 0);
00018     if (socket_fd < 0){
00019         LOG_PERROR(LOG_ERR, "Error creating socket: %s");
00020         return;
00021     }
00022     buf_idx = 0;
00023 }
00024 Message* SocketWrapper::readPartMsg(){
00025     int len;
00026     msglen_t msglen = 0;
00027
00028     if (buf_idx < sizeof(msglen)){ // I first need to read msglen
00029         // read available message
00030         len = read(socket_fd, buffer_in+buf_idx, sizeof(msglen)-buf_idx);
00031         // I will read rest of it in another moment since I do not know
00032         // whether other data is available.
00033         // TODO: find a way to tell whether socket has other data
00034     } else{
00035         // read msg length
00036         msglen = MSGLEN_NTOH(*((msglen_t*)buffer_in));
00037         assert(msglen <= MAX_MSG_SIZE); // must not happen
00038
00039         // read up to msg length
00040         len = read(socket_fd, buffer_in+buf_idx, msglen-buf_idx);
00041     }
00042
00043     DUMP_BUFFER_HEX_DEBUG(buffer_in, len);
00044
00045     if (len < 0){
00046         LOG_PERROR(LOG_ERR, "Error reading from socket: %s");
00047         throw "Error reading from socket";
00048     } else if (len == 0){
00049         throw "Connection lost";
00050     }
00051
00052     // buf_idx is also the number of read bytes up to now
00053     buf_idx += len;
00054
00055     if (buf_idx < sizeof(msglen)){
00056         LOG(LOG_DEBUG, "Too few bytes recevied from socket: %d < %lu",
00057             buf_idx, sizeof(msglen));
00058         return NULL;
00059     }
00060
00061     // read msg length
00062     msglen = MSGLEN_NTOH(*((msglen_t*)buffer_in));
00063
00064     if (msglen > MAX_MSG_SIZE){
00065         throw("Message is too big");
00066     }
00067
```

```
00068     if (buf_idx != msglen){
00069         LOG(LOG_DEBUG, "Too few bytes received from socket: %d < %d",
00070             buf_idx, msglen);
00071         return NULL;
00072     }
00073
00074     Message *m = readMessage(buffer_in+sizeof(msglen), msglen-sizeof(msglen));
00075
00076     // reset buffer
00077     buf_idx = 0;
00078
00079     return m;
00080 }
00081
00082 Message* SocketWrapper::receiveAnyMsg(){
00083     int len;
00084     msglen_t msglen;
00085
00086     // read msg length
00087     len = recv(socket_fd, buffer_in, sizeof(msglen), MSG_WAITALL);
00088
00089     DUMP_BUFFER_HEX_DEBUG(buffer_in, len);
00090
00091     if (len == 0){
00092         throw "Connection lost";
00093     } else if (len != sizeof(msglen)){
00094         LOG(LOG_ERR, "Too few bytes recevied from socket: %d < %lu",
00095             len, sizeof(msglen));
00096         return NULL;
00097     }
00098
00099     // read msg payload
00100     msglen = MSGLEN_NTOH(*((msglen_t*)buffer_in));
00101
00102     if (msglen > MAX_MSG_SIZE){
00103         throw("Message is too big");
00104     }
00105
00106     len += recv(socket_fd, buffer_in+len, msglen-len, MSG_WAITALL);
00107
00108     DUMP_BUFFER_HEX_DEBUG(buffer_in, len);
00109
00110     if (len == 0){
00111         throw "Received EOF";
00112     } else if (len != msglen){
00113         LOG(LOG_ERR, "Too few bytes recevied from socket: %d < %d",
00114             len, msglen);
00115         return NULL;
00116     }
00117
00118     Message *m = readMessage(buffer_in+sizeof(msglen), msglen-sizeof(msglen));
00119
00120     return m;
00121 }
00122
00123 Message* SocketWrapper::receiveMsg(MessageType type){
00124     return this->receiveMsg(&type, 1);
00125 }
00126
00127 Message* SocketWrapper::receiveMsg(MessageType type[], int n_types){
00128     Message *m = NULL;
00129     while (m == NULL){
00130         try{
00131             m = receiveAnyMsg();
00132         } catch(const char* msg){
00133             LOG(LOG_ERR, "%s", msg);
00134             return NULL;
00135         }
00136         if (m != NULL){
00137             for (int i = 0; i < n_types; i++){
00138                 if (m->getType() == type[i]){
00139                     return m;
00140                 }
00141             }
00142             LOG(LOG_WARN, "Received unexpected message of type %s",  m->getName().c_str());
00143         }
00144     }
00145     //TODO: add timeout?
00146     return NULL;
00147 }
00148
00149
00150 int SocketWrapper::sendMsg(Message *msg){
00151     msglen_t msglen, pktlen;
00152     int len;
00153
00154     msglen = msg->write(buffer_out+sizeof(msglen));
```

```
00155     if (msglen == 0)
00156         return 1;
00157 
00158     pktlen = msglen + sizeof(msglen);
00159     *((msglen_t*)buffer_out) = MSGLEN_HTON(pktlen);
00160 
00161     LOG(LOG_DEBUG, "Sending %s", msg->getName().c_str());
00162 
00163     DUMP_BUFFER_HEX_DEBUG(buffer_out, pktlen);
00164 
00165     len = send(socket_fd, buffer_out, pktlen, 0);
00166     if (len != pktlen){
00167         LOG(LOG_ERR, "Error sending %s: len (%d) != msglen (%d)",
00168             msg->getName().c_str(),
00169             len,
00170             msglen
00171         );
00172         return 1;
00173     }
00174 
00175     LOG(LOG_DEBUG, "Sent message %s", msg->getName().c_str());
00176 
00177     return 0;
00178 }
00179 
00180 void SocketWrapper::closeSocket(){
00181     close(socket_fd);
00182 }
00183 
00184 int ClientSocketWrapper::connectServer(Host host){
00185     int ret;
00186 
00187     other_addr = host.getAddress();
00188 
00189     ret = connect(
00190         socket_fd,
00191         (struct sockaddr*) &other_addr,
00192         sizeof(other_addr)
00193     );
00194 
00195     if (ret != 0){
00196         LOG_PERROR(LOG_ERR, "Error connecting to %s: %s",
00197             sockaddr_in_to_string(host.getAddress()).c_str());
00198         return ret;
00199     }
00200 
00201     return ret;
00202 }
00203 
00204 int ServerSocketWrapper::bindPort(){
00205     my_addr = make_my_sockaddr_in(0);
00206     int ret = bind_random_port(socket_fd, &my_addr);
00207     if (ret <= 0){
00208         LOG_PERROR(LOG_ERR, "Error in binding: %s");
00209         return ret;
00210     }
00211 
00212     ret = listen(socket_fd, 10);
00213     if (ret != 0){
00214         LOG_PERROR(LOG_ERR, "Error in setting socket to listen mode: %s");
00215     }
00216 
00217     return ret;
00218 }
00219 
00220 int ServerSocketWrapper::bindPort(int port){
00221     my_addr = make_my_sockaddr_in(port);
00222     int ret = bind(socket_fd, (struct sockaddr*) &my_addr, sizeof(my_addr));
00223     if (ret != 0){
00224         LOG_PERROR(LOG_ERR, "Error in binding: %s");
00225         return ret;
00226     }
00227 
00228     ret = listen(socket_fd, 10);
00229     if (ret != 0){
00230         LOG_PERROR(LOG_ERR, "Error in setting socket to listen mode: %s");
00231     }
00232 
00233     return ret;
00234 }
00235 
00236 SocketWrapper* ServerSocketWrapper::acceptClient(){
00237     socklen_t len = sizeof(other_addr);
00238     int new_sd = accept(
00239         socket_fd,
00240         (struct sockaddr*) &other_addr,
00241         &len
```

```
00242     );
00243
00244     SocketWrapper *sw = new SocketWrapper(new_sd);
00245     sw->setOtherAddr(other_addr);
00246     return sw;
00247 }
```

7.53 socket_wrapper.h File Reference

Definition of the helper class "SocketWrapper" and derivatives.

```
#include <sys/socket.h>
#include <netinet/in.h>
#include "logging.h"
#include "network/messages.h"
#include "network/host.h"
```

Data Structures

- class SocketWrapper

  Wrapper class around sockaddr_in and socket descriptor.
- class ClientSocketWrapper

  SocketWrapper for a TCP client.
- class ServerSocketWrapper

  SocketWrapper for a TCP server.

7.53.1 Detailed Description

Definition of the helper class "SocketWrapper" and derivatives.

Author

Riccardo Mancini

Date

2020-05-17

Definition in file socket_wrapper.h.

7.54 socket_wrapper.h

```
00001
00010 #ifndef SOCKET_WRAPPER_H
00011 #define SOCKET_WRAPPER_H
00012
00013 #include <sys/socket.h>
00014 #include <netinet/in.h>
00015 #include "logging.h"
00016 #include "network/messages.h"
00017 #include "network/host.h"
00018
00025 class SocketWrapper{
00026 protected:
00028     struct sockaddr_in other_addr;
00029
00031     int socket_fd;
00032
00034     char buffer_in[MAX_MSG_SIZE];
00035
00037     char buffer_out[MAX_MSG_SIZE];
00038
00040     msglen_t buf_idx;
00041 public:
00045     SocketWrapper();
00046
00050     SocketWrapper(int sd) : socket_fd(sd), buf_idx(0) {}
00051
00052     ~SocketWrapper(){closeSocket();}
00053
00057     int getDescriptor(){return socket_fd;};
00058
00068     Message* readPartMsg();
00069
00077     Message* receiveAnyMsg();
00078
00089     Message* receiveMsg(MessageType type);
00090
00102     Message* receiveMsg(MessageType type[], int n_types);
00103
00110     int sendMsg(Message *msg);
00111
00115     void closeSocket();
00116
00123     void setOtherAddr(struct sockaddr_in addr){other_addr = addr;}
00124
00125     sockaddr_in* getOtherAddr() { return &other_addr;}
00126
00130     Host getConnectedHost(){return Host(other_addr);}
00131 };
00132
00138 class ClientSocketWrapper : public SocketWrapper{
00139 public:
00145     int connectServer(Host host);
00146 };
00147
00154 class ServerSocketWrapper : public SocketWrapper{
00155 private:
00157     struct sockaddr_in my_addr;
00158 public:
00166     int bindPort(int port);
00167
00174     int bindPort();
00175
00179     SocketWrapper* acceptClient();
00180
00184     int getPort(){return ntohs(my_addr.sin_port);}
00185 };
00186
00187 #endif // SOCKET_WRAPPER_Hln
```

7.55 user.h File Reference

Definition of the User class.

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <pthread.h>
#include <map>
```

```
#include "logging.h"
#include "security/secure_socket_wrapper.h"
#include "network/host.h"
```

Data Structures

- class User

  Class representing a user.

Enumerations

- enum UserState {
  JUST_CONNECTED, **SECURELY_CONNECTED**, **AVAILABLE**, **CHALLENGED**,
  PLAYING, **DISCONNECTED** }

  Definition of the available states the user may be in.

7.55.1 Detailed Description

Definition of the User class.

Author

Riccardo Mancini

Date

2020-05-23

Definition in file user.h.

7.55.2 Enumeration Type Documentation

7.55.2.1 UserState `enum UserState`

Definition of the available states the user may be in.

JUST_CONNECTED: the user has just_connected but not yet registered. AVAILABLE: the user is registered and available for challenges. CHALLENGED: the user is being challenged by or has challenged another player. PLA↩ YING: the user is currently playing with another user. DISCONNECTED: the user is disconnected.

Definition at line 35 of file user.h.

7.56 user.h

```
00001
00010 #ifndef USER_H
00011 #define USER_H
00012
00013 #include <iostream>
00014 #include <cstdlib>
00015 #include <ctime>
00016 #include <pthread.h>
00017 #include <map>
00018
00019 #include "logging.h"
00020 #include "security/secure_socket_wrapper.h"
00021 #include "network/host.h"
00022
00024 class UserList;
00025
00035 enum UserState {JUST_CONNECTED, SECURELY_CONNECTED, AVAILABLE, CHALLENGED, PLAYING, DISCONNECTED};
00036
00046 class User{
00047     friend UserList;
00048 private:
00049     SecureSocketWrapper *sw;
00050     UserState state;
00051     string username;
00052     string opponent_username;
00053     pthread_mutex_t mutex;
00054
00060     unsigned int references;
00061
00067     void increaseRefs(){references++;}
00068
00074     void decreaseRefs(){references--;}
00075 public:
00082     User(SecureSocketWrapper *sw)
00083             : sw(sw), state(JUST_CONNECTED),
00084                 username(""), opponent_username(""),
00085                 references(0) {
00086         pthread_mutex_init(&mutex, NULL);
00087     }
00088
00094     ~User(){
00095         pthread_mutex_destroy(&mutex);
00096         delete sw;
00097     }
00098
00102     void lock(){pthread_mutex_lock(&mutex);}
00103
00107     void unlock(){pthread_mutex_unlock(&mutex);}
00108
00112     string getUsername(){return username;}
00113
00117     void setUsername(string username){this->username=username;}
00118
00122     SecureSocketWrapper* getSocketWrapper(){return sw;}
00123
00127     UserState getState(){return state;}
00128
00132     void setState(UserState state){
00133         LOG(LOG_DEBUG, "User %s (%d) is now in state %d",
00134                 username.c_str(), sw->getDescriptor(), (int)state);
00135         this->state=state;
00136     }
00137
00141     string getOpponent(){return opponent_username;}
00142
00146     void setOpponent(string opponent){this->opponent_username=opponent;}
00147
00151     int countRefs(){return references;}
00152 };
00153
00154 #endif // USER_H
```

7.57 user_list.h File Reference

Implementation of the UserList class.

```
#include <pthread.h>
#include <map>
#include <cstring>
```

```
#include <list>
#include "config.h"
#include "user.h"
```

Data Structures

- class UserList

  Class that manages the users.

7.57.1 Detailed Description

Implementation of the UserList class.

Definition of the UserList class.

Author

Riccardo Mancini

Date

2020-05-23

Definition in file user_list.h.

7.58 user_list.h

```
00001
00010 #ifndef USER_LIST_H
00011 #define USER_LIST_H
00012
00013 #include <pthread.h>
00014 #include <map>
00015 #include <cstring>
00016 #include <list>
00017
00018 #include "config.h"
00019 #include "user.h"
00020
00021 using namespace std;
00022
00038 class UserList{
00039 private:
00040     map<string,User*> user_map_by_username;
00041     map<int,User*> user_map_by_fd;
00042     pthread_mutex_t mutex;
00043 public:
00047     UserList();
00048
00063     bool add(User *u);
00064
00073     User* get(string username);
00074
00083     User* get(int fd);
00084
00091     bool exists(string username);
00092
00099     bool exists(int fd);
00100
00112     void yield(User *u);
00113
00121     string listAvailableFromTo(int from);
00122
00126     int size();
00127 };
00128
00129 #endif // USER_LIST_H
```

Index

~User
    User, 55

acceptClient
    ServerSecureSocketWrapper, 47
add
    UserList, 56
aes_gcm_decrypt
    crypto.h, 74
aes_gcm_encrypt
    crypto.h, 75
Args, 6
    getArgc, 7
    operator<<, 7
args.cpp, 61, 62
args.h, 62, 63

bind_random_port
    inet_utils.cpp, 90
    inet_utils.h, 96
bindPort
    ServerSecureSocketWrapper, 47
    ServerSocketWrapper, 48, 49
build_store
    crypto.h, 75
buildMsgToSign
    SecureSocketWrapper, 37

CertificateMessage, 7
    read, 8
    write, 8
CertificateRequestMessage, 8
    read, 9
    write, 9
ChallengeForwardMessage, 9
    read, 10
    write, 10
ChallengeMessage, 11
    read, 11
    write, 11
challengePeer
    Server, 43
ChallengeResponseMessage, 12
    read, 12
    write, 13
checkWin
    Connect4, 19
client.cpp, 63, 64
ClientHelloMessage, 13
    read, 14
    write, 14
ClientSecureSocketWrapper, 14
    ClientSecureSocketWrapper, 15
    connectServer, 15
ClientSocketWrapper, 16
    connectServer, 16
ClientVerifyMessage, 16
    read, 17
    write, 17
compare_hmac
    crypto.h, 76
Connect4, 18
    checkWin, 19
    Connect4, 18
    getAdv, 19
    getNumCols, 19
    getPlayer, 19
    play, 20
    print, 20
    setPlayer, 20
connect4.cpp, 66, 67
connect4.h, 69, 70
connection_mode.h, 70, 72
    ConnectionType, 71
ConnectionMode, 21
ConnectionType
    connection_mode.h, 71
connectServer
    ClientSecureSocketWrapper, 15
    ClientSocketWrapper, 16
crypto.h, 72, 82
    aes_gcm_decrypt, 74
    aes_gcm_encrypt, 75
    build_store, 75
    compare_hmac, 76
    dhke, 76
    dsa_sign, 77
    dsa_verify, 77
    get_ecdh_key, 78
    get_rand, 78
    handleErrors, 74
    handleErrorsNoException, 74
    hkdf, 79
    hkdf_one_info, 79
    hmac, 80
    load_cert_file, 80
    load_crl_file, 80
    load_key_file, 81
    verify_peer_cert, 81

decryptMsg
    SecureSocketWrapper, 37
dhke
    crypto.h, 76
dsa_sign
    crypto.h, 77
dsa_verify
    crypto.h, 77
dump_buffer.cpp, 83, 85
    dump_buffer_hex, 83
dump_buffer.h, 85, 87

dump_buffer_hex, 86
dump_buffer_hex
dump_buffer.cpp, 83
dump_buffer.h, 86

encryptMsg
SecureSocketWrapper, 38
exists
UserList, 56, 57

GameCancelMessage, 22
read, 22
write, 22
GameEndMessage, 23
read, 23
write, 24
GameStartMessage, 24
read, 25
write, 25
generateKeys
SecureSocketWrapper, 38
get
UserList, 57
get_ecdh_key
crypto.h, 78
get_rand
crypto.h, 78
getAdv
Connect4, 19
getArgc
Args, 7
getNumCols
Connect4, 19
getPlayer
Connect4, 19
getServerCert
Server, 43
getUserList
Server, 43

handleErrors
crypto.h, 74
handleErrorsNoException
crypto.h, 74
handshakeClient
SecureSocketWrapper, 38
handshakeServer
SecureSocketWrapper, 39
hkdf
crypto.h, 79
hkdf_one_info
crypto.h, 79
hmac
crypto.h, 80
Host, 25
Host, 26
host.cpp, 87, 88
host.h, 88, 89

inet_utils.cpp, 89, 93
bind_random_port, 90
make_my_sockaddr_in, 90
make_sv_sockaddr_in, 92
readSockAddrIn, 92
sockaddr_in_cmp, 92
sockaddr_in_to_string, 93
writeSockAddrIn, 93
inet_utils.h, 95, 99
bind_random_port, 96
make_my_sockaddr_in, 96
make_sv_sockaddr_in, 97
readSockAddrIn, 97
sockaddr_in_cmp, 98
sockaddr_in_to_string, 98
writeSockAddrIn, 98

listAvailableFromTo
UserList, 58
load_cert_file
crypto.h, 80
load_crl_file
crypto.h, 80
load_key_file
crypto.h, 81
logging.h, 99, 100

make_my_sockaddr_in
inet_utils.cpp, 90
inet_utils.h, 96
make_sv_sockaddr_in
inet_utils.cpp, 92
inet_utils.h, 97
makeAAD
SecureSocketWrapper, 39
Message, 27
read, 27
write, 27
message_queue.h, 101, 102
MessageQueue< T, MAX_SIZE >, 28
pull, 29
pullWait, 29
push, 29
pushSignal, 29
messages.cpp, 103, 104
readMessage, 104
messages.h, 111, 115
MessageType, 113
readMessage, 114
readUsername, 114
writeUsername, 114
MessageType
messages.h, 113
MoveMessage, 30
read, 30
write, 31
multi_player.h, 120, 121

operator<<

Args, 7

play
Connect4, 20
print
Connect4, 20
pull
MessageQueue< T, MAX_SIZE >, 29
pullWait
MessageQueue< T, MAX_SIZE >, 29
push
MessageQueue< T, MAX_SIZE >, 29
pushSignal
MessageQueue< T, MAX_SIZE >, 29

read
CertificateMessage, 8
CertificateRequestMessage, 9
ChallengeForwardMessage, 10
ChallengeMessage, 11
ChallengeResponseMessage, 12
ClientHelloMessage, 14
ClientVerifyMessage, 17
GameCancelMessage, 22
GameEndMessage, 23
GameStartMessage, 25
Message, 27
MoveMessage, 30
RegisterMessage, 32
SecureMessage, 34
ServerHelloMessage, 45
StartGameMessage, 53
UsersListMessage, 59
UsersListRequestMessage, 60
readMessage
messages.cpp, 104
messages.h, 114
readPartMsg
SecureSocketWrapper, 39
SocketWrapper, 50
readSockAddrIn
inet_utils.cpp, 92
inet_utils.h, 97
readUsername
messages.h, 114
receiveAnyMsg
SecureSocketWrapper, 40
SocketWrapper, 51
receiveMsg
SecureSocketWrapper, 40
SocketWrapper, 51
RegisterMessage, 31
read, 32
write, 32
registerToServer
Server, 44
replyPeerChallenge
Server, 44
secure_host.h, 121, 122
secure_socket_wrapper.h, 122, 123
SecureHost, 32
SecureHost, 33
SecureMessage, 34
read, 34
write, 34
SecureSocketWrapper, 35
buildMsgToSign, 37
decryptMsg, 37
encryptMsg, 38
generateKeys, 38
handshakeClient, 38
handshakeServer, 39
makeAAD, 39
readPartMsg, 39
receiveAnyMsg, 40
receiveMsg, 40
sendMsg, 41
setOtherAddr, 41
updateRecvIV, 41
updateSendIV, 42
sendMsg
SecureSocketWrapper, 41
SocketWrapper, 52
Server, 42
challengePeer, 43
getServerCert, 43
getUserList, 43
registerToServer, 44
replyPeerChallenge, 44
signalGameEnd, 44
server.cpp, 125, 126
server.h, 132, 133
server_lobby.cpp, 134, 135
serverLobby, 135
server_lobby.h, 137, 138
serverLobby, 138
ServerHelloMessage, 45
read, 45
write, 46
serverLobby
server_lobby.cpp, 135
server_lobby.h, 138
ServerSecureSocketWrapper, 46
acceptClient, 47
bindPort, 47
ServerSecureSocketWrapper, 47
ServerSocketWrapper, 48
bindPort, 48, 49
setOtherAddr
SecureSocketWrapper, 41
SocketWrapper, 52
setPlayer
Connect4, 20
signalGameEnd
Server, 44
single_player.h, 139

sockaddr_in_cmp
inet_utils.cpp, 92
inet_utils.h, 98
sockaddr_in_to_string
inet_utils.cpp, 93
inet_utils.h, 98
socket_wrapper.cpp, 139, 140
socket_wrapper.h, 143, 144
SocketWrapper, 49
readPartMsg, 50
receiveAnyMsg, 51
receiveMsg, 51
sendMsg, 52
setOtherAddr, 52
StartGameMessage, 52
read, 53
write, 53

updateRecvIV
SecureSocketWrapper, 41
updateSendIV
SecureSocketWrapper, 42
User, 54
∼User, 55
User, 54
user.h, 144, 146
UserState, 145
user_list.h, 146, 147
UserList, 55
add, 56
exists, 56, 57
get, 57
listAvailableFromTo, 58
yield, 58
UsersListMessage, 58
read, 59
write, 59
UsersListRequestMessage, 60
read, 60
write, 60
UserState
user.h, 145

verify_peer_cert
crypto.h, 81

write
CertificateMessage, 8
CertificateRequestMessage, 9
ChallengeForwardMessage, 10
ChallengeMessage, 11
ChallengeResponseMessage, 13
ClientHelloMessage, 14
ClientVerifyMessage, 17
GameCancelMessage, 22
GameEndMessage, 24
GameStartMessage, 25
Message, 27
MoveMessage, 31
RegisterMessage, 32
SecureMessage, 34
ServerHelloMessage, 46
StartGameMessage, 53
UsersListMessage, 59
UsersListRequestMessage, 60
writeSockAddrIn
inet_utils.cpp, 93
inet_utils.h, 98
writeUsername
messages.h, 114

yield
UserList, 58