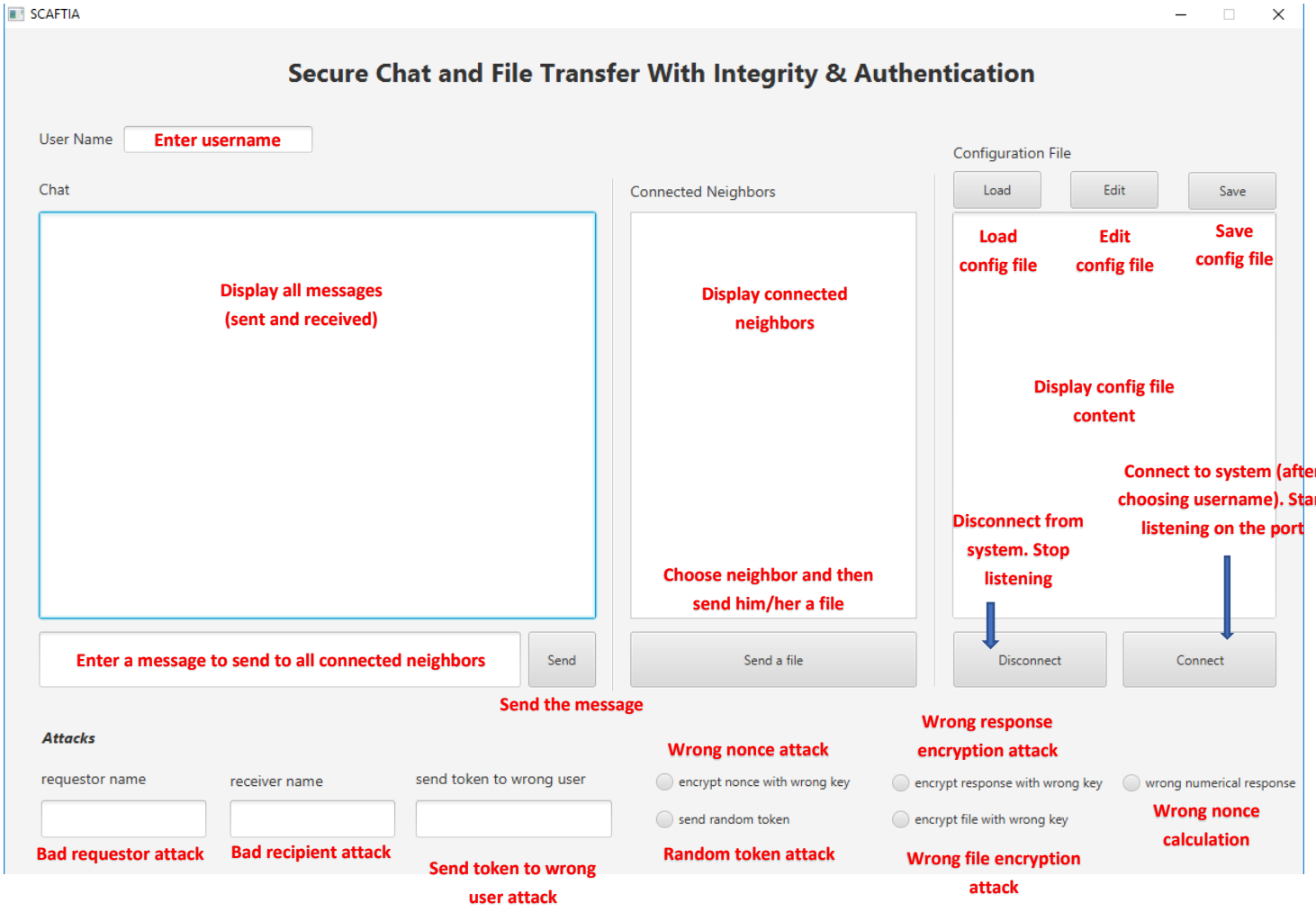
Instructions for compiling the source code for SCAFTIA:

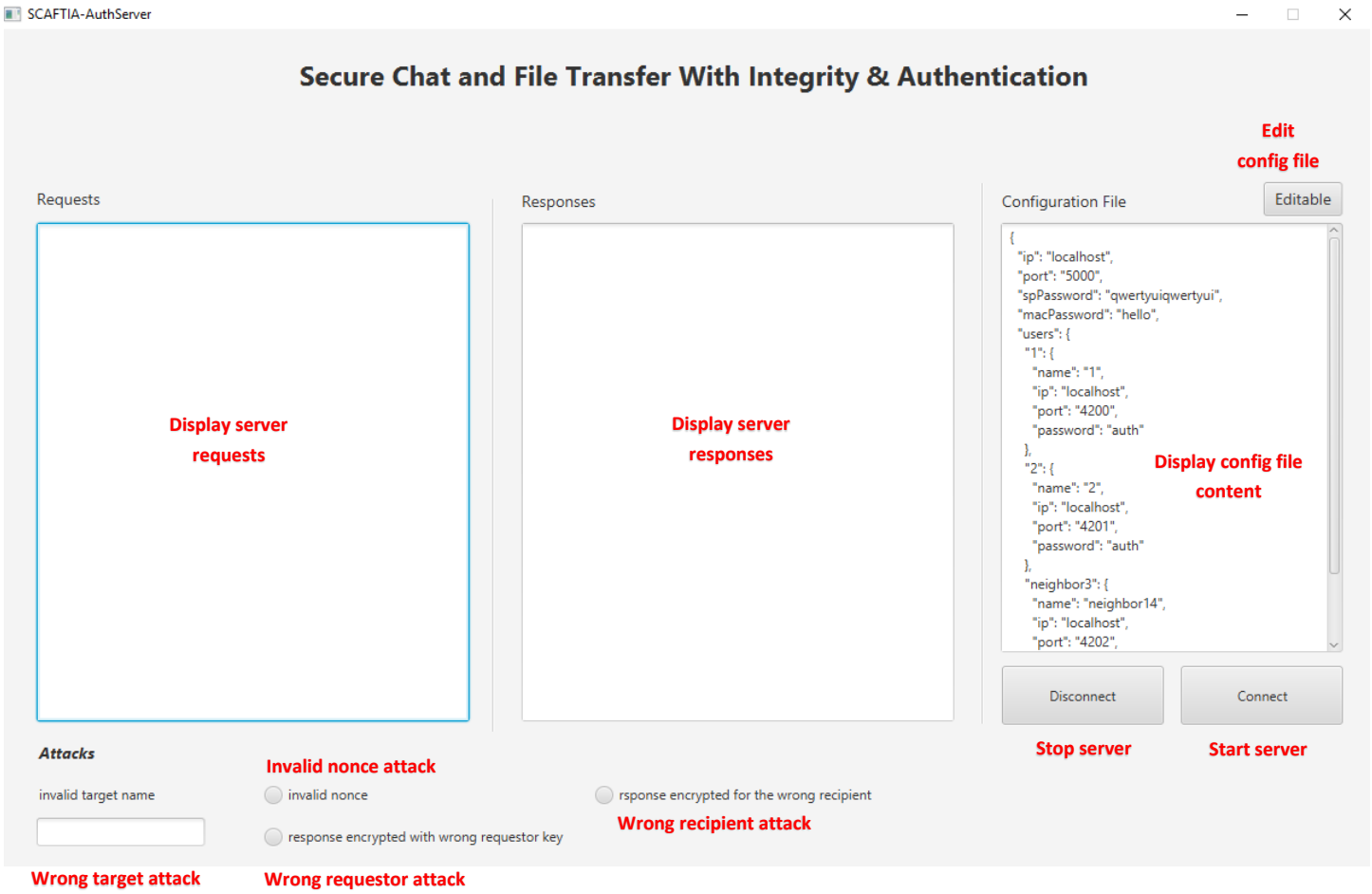
1. Open SCAFTIA project using IntelliJ.

2. Press on the Run icon (green triangle).

Instructions for running the SCAFTIA programs:



Instructions for running the SCAFTIA AuthServer:



AuthServer Notes:

- The server loads the configuration file automatically.
- the configuration file is saved automatically after every change.

Configuration files documentation:

- Our configuration files both for server and client were written in JSON format.
- We used GSON library for serialize and deserialize into objects.

AuthServer configuration file:

```
{
  "ip": "localhost",
  "port": "5000",
  "spPassword": "qwertyuiqwertyui",
  "macPassword": "hello",
  "users": {
    "1": {
      "name": "1",
      "ip": "localhost",
      "port": "4200",
      "password": "auth1"
    },
    "2": {
      "name": "2",
      "ip": "localhost",
      "port": "4201",
      "password": "auth2"
    },
    "3": {
      "name": "3",
      "ip": "localhost",
      "port": "4202",
      "password": "auth3"
    }
  }
}
```

Information:

- The IP and port the server is listening on.
- The server contains ALL the passwords in the system (shared, mac, private of every user).
- Map of users – the key is the name of the user, the value contains the following information about the user: name, IP, port, private password.

SCAFTIA configuration file:

```
{
  "password": "qwertyuiqwertyui",
  "macPassword": "hello",
  "authPassword": "auth1",
  "authServerIp": "10.0.201.19",
  "authServerPort": "5000",
  "ip": "10.0.201.22",
  "port": "4200",
  "neighbors": [
    {
      "name": "neighbor1",
      "ip": "10.0.201.23",
      "port": "4201"
    },
    {
      "name": "neighbor2",
      "ip": "10.0.201.24",
      "port": "4202"
    },
    {
      "name": "neighbor3",
      "ip": "10.0.201.25",
      "port": "4203"
    }
  ]
}
```

Information:

- The IP and port SCAFTIA user listening on.
- Passwords: shared, mac, own private.
- Server IP and port.
- List of neighbors: each neighbor contains name, IP and port.

Documentation of the communication protocol:

2. BYE message - BYE Username IP Port

3. OK message - OK Username IP Port MessageContent

4. NO message - NO Username IP Port MessageContent

5. MESSAGE message - MESSAGE Username IP Port MessageContent

6. SENDFILE message - SENDFILE Username IP Port FileName

7. ACK message – ACK Username IP Port Message

8. FAILED message – FAILED Username IP Port Message

Finally: we encrypt the message and send it with the corresponding IV and HMAC-SHA256 digest.

For example: Encrypt(HELLO Username IP Port)-IV-digest

4a28b97db2e4-fa45d46bafad4cd46e5b4a28b97db2e4

Encrypt/Decrypt messages and integrity explanation:

Encryption:

1. Generate key – get password from config file, encode it using UTF8 encoding and then hash it with SHA256 algorithm.
2. Generate random IV – each sending we create new random IV (128 bits).
3. Encryption using AES/CTR.

Decryption:

1. Parse the encrypted incoming message by splitting it by "-" character. First section is the message and the second is the IV.
2. Generate key – get password from config file, encode it using UTF8 encoding and then hash it with SHA256 algorithm.
3. Decryption using AES/CTR.

Integrity:

1. Generate key – get password from config file, encode it using UTF8 encoding and then hash it with SHA256 algorithm.
2. Calculate digest using HMAC-SHA256 algorithm.

How SCAFTIA parses the messages:

The SCAFTIA tool listens to incoming messages on the port from the config file. When received a new message, SCAFTIA calculate the digest of the IV concatenated to the encrypted message, and finally decrypt the message. SCAFTIA compare the calculated digest to the received digest -> if the comparison is false, i.e. the message is corrupted or the key is wrong, SCAFTIA does not show the incoming message, displays an error message about the corrupted message and writes it to the log.
SCAFTIA parse the decrypted output by splitting it by " "(space) character.
SCAFTIA takes the information from the parsed message according to the formats we described above.

How the SCAFTIA tool reacts to the different messages:

1. HELLO message – if the sender of the message isn't in the connected neighbors list add the sender to the connected neighbors list, and send HELLO message back. Displays an appropriate message

2. BYE message – delete the sender of the message from the connected neighbors list. Displays an appropriate message.

3. OK message - displays a message that the receiver accepted the file.

4. NO message - displays a message that the receiver refused to accept the file.

5. MESSAGE message – display the message content in the chat.

6. SENDFILE message – display a message to the user that someone wants to send a file to him, and wait for the user to answer. According to the user answer – send back OK or NO message.

7. ACK message – displays a message that the receiver received the file successfully.

8. FAILED message – displays a message that the receiver did not receive the file because the file is corrupted.

Documentation of how the SCAFTIA tool implements the Needham-Schroeder protocol and uses session keys in file transformation:

Message formats:

- Messages between sender and receiver on main port (these messages encrypted as explained above with shared key):
    1. SENDFILE message - SENDFILE Username IP Port FileName.
    2. OK message - OK Username IP Port MessageContent.
- Messages between sender and server:
    1. REQUEST message – SenderName RecieverName Nonce (this message is not encrypted, only integrity check with mac key).
    2. RESPONSE message – SessionKey Nonce Receiver Token (this message is encrypted with the sender private key).
    3. Token – SessionKey SenderName (the token is encrypted with the receiver private key).
- Messages between sender and receiver on random port:
    1. TOKEN message– SessionKey SenderName (the token is encrypted with the receiver private key).
    2. CHALLENGE message – Nonce (encrypted with session key).
    3. RESPONSE message – Nonce-1 (encrypted with session key).
    4. OK message – OK (encrypted with session key).
    5. FILE – the file to send (encrypted with session key).
    6. ACK message – message that the file received successfully (encrypted with session key).
    7. FAILED message – message that error occurred (encrypted with session key).

Explanation:

We used the Needham-Schroeder protocol in the file transfer process. The process starts with send file request on the main port to the intended receiver. If the receiver confirms the request, the sender asks the server for session key with the request message (explained above). The server responses with response message that contains new session key. The sender decrypts the response with his private key. The sender opens a new session with a random port which received from the receiver – all the next steps occurred in this session. The sender sends the token to the receiver. The receiver decrypts the token with his private key. The receiver sends a challenge (nonce) to the sender. The sender responses to the challenge (nonce -1) and send it. The receiver checks if the response is correct – if yes he sends ok message to the sender. The sender sends the file encrypted with the session key. After the file received successfully the receiver send ack to the sender.

Documentation of how the server works:

Message formats:

- Messages between sender and server:
    1. REQUEST message – SenderName RecieverName Nonce (this message is not encrypted, only integrity check with mac key).
    2. RESPONSE message – SessionKey Nonce Receiver Token (this message is encrypted with the sender private key).
    3. Token – SessionKey SenderName (the token is encrypted with the receiver private key).

User Information:

The server stores all user's information in the configuration file as explained above.

Generate session keys:

For every session the server generates a new byte array (32 bytes – 256 bits) and filling it using *secureRandom.nextBytes(byte[])* method.

SCAFTIA Log File:

- The name of the log file is *SCAFTIA-Logger.log.*
- The location of the log file is the same as the location you open the SCAFTIA tool.
- The format of the log file (auth/token error message), all in one line:
  [2019-06-17 15:11:37]
  ERROR Auth Error -
  sender's ip and port: 10.0.201.20:4201,
  sender's name: Kobi,
  received message or token: failed to decrypt token,
  decryption error - can't decrypt the received token,
  iv: decryption failed,
  hmac: decryption failed,
  error description: Received invalid token
- The format of the log file, all in one line:
  [2019-05-28 09:13:51]
  INFO - Received Message –
  ip: 10.0.201.22,
  port: 4200,
  name: Alice,
  type: MESSAGE,
  message: new message,
  iv: EB4328BE1F44E7FA1187B30B50CA7559,
  hmac: F61AE9B5C4D61D4C30CB40CCBFD3DAECDD323C127C861F95B634EFE1FA6BDF6A,
  valid: TRUE

AuthServer Log File:

- The name of the log file is SCAFTIA-AuthServer-Logger.*log.*
- The location of the log file is the same as the location you open the AuthServer tool.
- The format of the log file, all in one line:
  [2019-06-17 15:11:37]
  INFO –
  sender's ip and port: 10.0.201.20:4201,
  sender's name: Kobi,
  recipient's name: Hovav,
  nonce: 777E7DDD781215889F1E97BD83A04BA6,
  is request message encrypted: false,
  is HMAC valid: true,
  message: Send new auth response - sender: Kobi, recipient: Hovav,
  error message: no error occurred,
  is response sent back: true,
  is response sent back was intentionally incorrect: no, correct response