

Task 2 Descriptions and Examples

This task consists of securing the connection between a client (the sender of messages) and a server (the receiver of messages). For the purpose of this document, Alice will be known as the server and Bob will be known as the client.

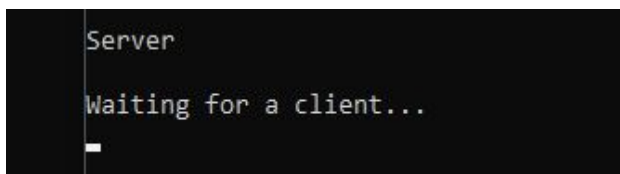
The program begins by establishing a connection between Alice and Bob, given that they are both running. If Bob starts and Alice has not yet started, Bob will wait until Alice starts before proceeding (as seen in *Figure 1*). If Alice starts and Bob has not yet started, Alice will also wait until Bob has started before proceeding (as seen in *Figure 2*).

Figure 1



```
Client
Waiting for server...
_
```

Figure 2



```
Server
Waiting for a client...
_
```

Alice finds a free port from 1024 and above. Once a free port is found, its value is stored in a file named “port” (see *Figure 3*). When Bob tries to connect, Bob looks for the server port in this file to ensure a connection is always available.

Figure 3



Since the point of task 2 is not key distribution, once a connection has been made, Alice generates a key and simply sends it over the network unencrypted to Bob. Should a message sending application be built, it would use a key distribution method as outlined in task 1, hence combining the task 1 and task 2 implementations.

Once a connection is made, Bob will be prompted to send messages to Alice as shown in *Figure 4* and Alice commits a log saying that Bob has connected (shown in *Figure 5*). This log is saved in a log file dedicated to the session, which tracks all interactions between Bob and Alice.

Figure 4

```
Client
Waiting for server...
Connected to server.
You can now send messages to the server:
```

Figure 5

```
Server
Waiting for a client...
Connected: ('127.0.0.1', 56723)
Logging messages in: 20181022185312550764.log
```

Traffic Analysis Attacks

To prevent an attack by traffic analysis, all messages sent are encrypted using a Fernet encryption. Fernet is an encryption method that implements AES symmetric authenticated cryptography. It guarantees that message encrypted by it cannot be read without the key. Since task 2 assumes that only Alice and Bob have the key, this ensures protection against traffic analysis. Bob encrypts the message and sends it as a byte array to Alice, who then receives the message and decrypts it. The key used is stored in the aforementioned log file, as shown in *Figure 6*.

Figure 6

```
Connected: ('127.0.0.1', 54033)
Key: 8Y2DiZruYqpsQxaBD5Wl8FSALsj3MdXo4AVJO21wd0o=
```

Modification Attacks

To protect against modification attacks, each message uses a SHA-2 512 bit cryptographic hash function. Once a hash value is created from the message, it is appended to the the front of the byte array of the message being sent. Since Alice knows how long this hash value is, it can separate it out from the actual text message in the incoming data. The text message is then re-hashed to see if the new value matches the incoming hash value. If it does not match or there was no hash value to begin with the user is notified of a potential modification attack, as shown in the code in *Figure 7*. The hash value for a particular message is also stored in the log file, as shown in *Figure 8*. Furthermore, a possible modification attack does not necessarily mean that the text of the message has been tampered with, it could also mean that data was simply lost through transfer - through this we also ensure message integrity.

Figure 7

```
# check if hash values match
if (test_hash_value != hash_value):
    print ("Incorrect hash value.")
    with os.fdopen(os.open(LOG_FILE, FLAG, PERM), 'a') as fout:
        fout.write("\nIncorrect hash value.")
    hacked("modification")
```

Figure 8

```
Message ID: 0
Hash value: 7\x1c%\xce\xe5\xd0T\x86\x0c*\xd9]\xcc\xd7\x1a\x84ba\xcapJP!\x89\xa2\x1c
\xb4\x88\x8d\xfc\xe3\x9d\xe8\xa5\xf3\x8eR\x97\xb4\x80\xf8F\xb9\xfaJ\x05\xd3\xf6\x0c
=\xf5\x91\xe6\xef\n\x9e/\x0e\x13\x07A\xeb\x00M
```

Replay Attacks

To prevent replay attacks, a timestamp for every message is created by Bob. The timestamp includes the year, month, day, hour, minute, second and microsecond that the message was sent. The timestamp is attached to the front of the message (before the hash value). Since Alice knows that there is a timestamp and where to expect it, it can be easily extracted from the received message. If the timestamp is older than 3 seconds, the user is notified of a possible replay attack, as shown in the code in figure 9. If there is no timestamp, then there is a possible modification attack. The timestamp of the message is also stored in the log file as shown in figure 10.

Figure 9

```
# if the timestamp is older than the most recent timestamp - possible attack
print(current_time)
if (current_time - timestamp >= timedelta(microseconds = 0)):
    hacked("replay")
else:
    current_time = timestamp
```

Figure 10

```
Timestamp: 2018-10-22 15:49:27.776152
```

Interruption Attacks

The interruption of messages is mitigated by having Alice confirm receipt of each message sent by Bob. If Bob does not receive this confirmation of receipt, a continuous loop is executed and the same message is resent every 10 seconds. The loop will stop once a confirmation is received or once the connection is severed - we do not allow this to be ignored for the sake of the connections integrity. The confirmation, sent by Alice, contains the initial hash value, encrypted with the private key, so as to ensure its validity, i.e. no fake confirmations are accepted (e.g. those coming from an attacker on the outside). An example of this interaction can be seen in *Figure 11*, below.

Figure 11

```
Client
Waiting for server...
Connected to server.
You can now send messages to the server:

Hey
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
No confirmation received from the server. Resending message.
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
Waiting for receival confirmation from server....
No confirmation received from the server. Resending message.
^Z
[3]+  Stopped                  python3 client.py
```

Repudiation Attacks

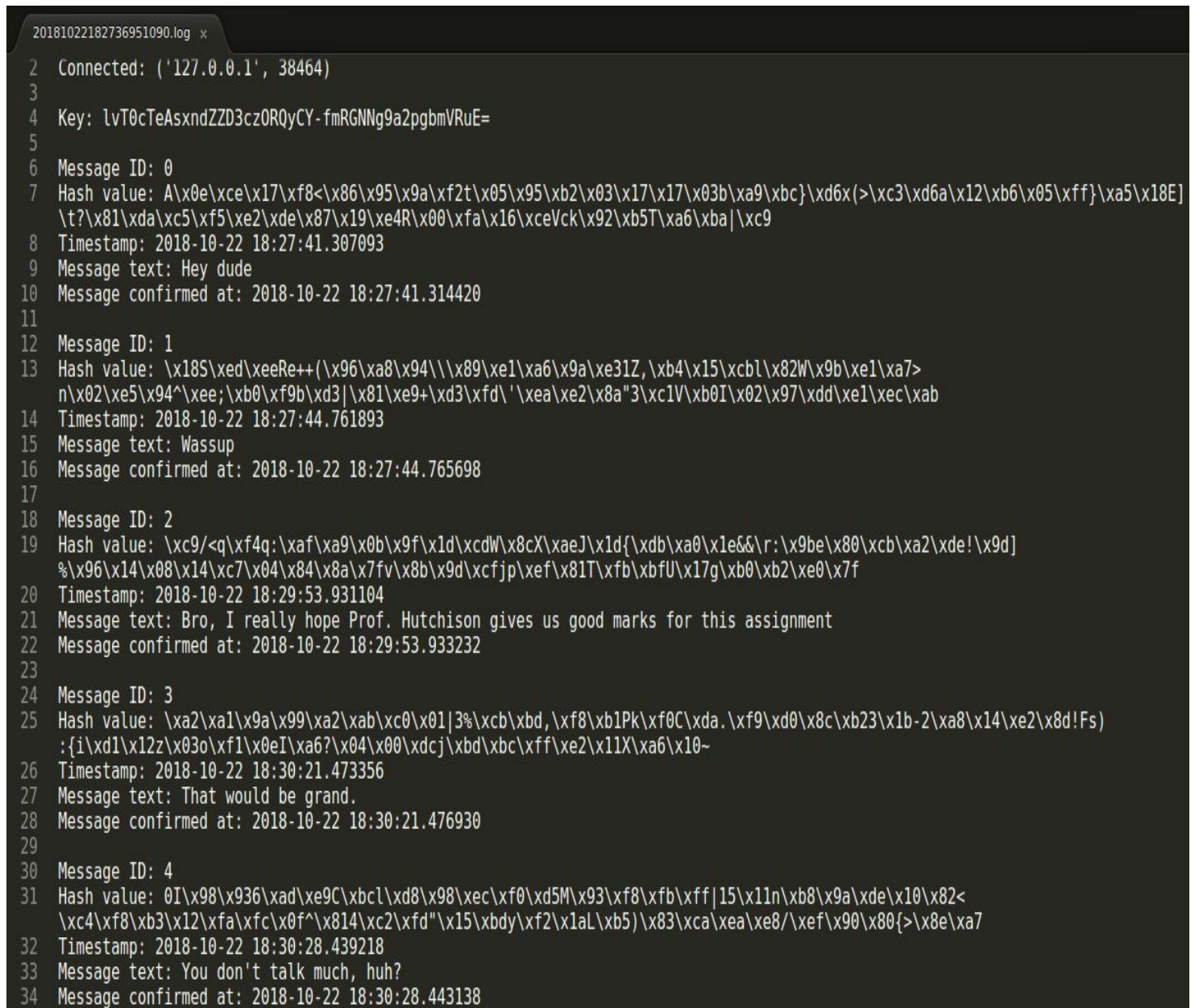
We decided to not only prevent attacks from external attackers, but from the user as well. To prevent the user (i.e. Bob) from denying that s/he/it initiated a particular connection or participated in certain interactions, we implemented repudiation countermeasures. This was done by logging all connections made and messages sent between Bob and Alice. Included in these logs is the private key used in the connection, the port and host over which the connection takes place, the consecutive ID of each message sent, its hash value, the contents of the message itself and finally the confirmation of receipt of said message.

To prevent a repudiation attack - in our scenario, since only Bob can send messages - it is only necessary to log the timestamp of messages sent, as well as the confirmation of receipt. Any other information would put the privacy of the connection at risk. However, we added the extra information purely for our own learning purposes - it would not be implemented in a proper client-server connection.

Note also that the naming convention for each log file is based on the time of connection establishment, down to the microsecond. E.g. *20181022173702342515.log* would contain the logging information of a

connection between Alice and Bob, established on the 22nd of October 2018 at 17:37:02.342515 (based on Alice's time zone). An example of these log files can be found in *Figure 12*, below.

Figure 12



```
20181022182736951090.log x
2 Connected: ('127.0.0.1', 38464)
3
4 Key: lvT0cTeAsxndZZD3cz0RQyCY-fmRGNNg9a2pgbmVRuE=
5
6 Message ID: 0
7 Hash value: A\x0e\xce\x17\xf8<\x86\x95\x9a\xf2t\x05\x95\xb2\x03\x17\x17\x03b\xa9\xbc}\xd6x(>\xc3\xd6a\x12\xb6\x05\xff}\xa5\x18E]
  \t?\x81\xda\xc5\xf5\xe2\xde\x87\x19\xe4R\x00\xfa\x16\xceVck\x92\xb5T\xa6\xba|\xc9
8 Timestamp: 2018-10-22 18:27:41.307093
9 Message text: Hey dude
10 Message confirmed at: 2018-10-22 18:27:41.314420
11
12 Message ID: 1
13 Hash value: \x185\xed\xeeRe++(\x96\xa8\x94\\\x89\xe1\xa6\x9a\xe31Z,\xb4\x15\xcb1\x82W\x9b\xe1\xa7>
  n\x02\xe5\x94^\xee;\xb0\xf9b\xd3|\x81\xe9+\xd3\xfd\' \xea\xe2\x8a"3\xc1V\xb0I\x02\x97\xdd\xe1\xec\xab
14 Timestamp: 2018-10-22 18:27:44.761893
15 Message text: Wassup
16 Message confirmed at: 2018-10-22 18:27:44.765698
17
18 Message ID: 2
19 Hash value: \xc9/<q\xf4q:\xaf\xa9\x0b\x9f\x1d\xcdW\x8cX\xaeJ\x1d{\xdb\xa0\x1e&&r:\x9be\x80\xcb\xa2\xde!\x9d]
  %\x96\x14\x08\x14\xc7\x04\x84\x8a\x7fv\x8b\x9d\xcfjp\xef\x81T\xfb\xbfU\x17g\xb0\xb2\xe0\x7f
20 Timestamp: 2018-10-22 18:29:53.931104
21 Message text: Bro, I really hope Prof. Hutchison gives us good marks for this assignment
22 Message confirmed at: 2018-10-22 18:29:53.933232
23
24 Message ID: 3
25 Hash value: \xa2\xa1\x9a\x99\xa2\xab\xc0\x01|3%\xcb\xbd,\xf8\xb1Pk\xf0C\xda.\xf9\xd0\x8c\xb23\x1b-2\xa8\x14\xe2\x8d!Fs)
  :{i\xd1\x12z\x03o\xf1\x0eI\xa6?\x04\x00\xdcj\xbd\xbc\xff\xe2\x11X\xa6\x10~
26 Timestamp: 2018-10-22 18:30:21.473356
27 Message text: That would be grand.
28 Message confirmed at: 2018-10-22 18:30:21.476930
29
30 Message ID: 4
31 Hash value: 0I\x98\x936\xad\xe9C\xbc1\xd8\x98\xec\xf0\xd5M\x93\xf8\xfb\xff|15\x11n\xb8\x9a\xde\x10\x82<
  \xc4\xf8\xb3\x12\xfa\xfc\x0f^\x814\xc2\xfd"\x15\xbdy\xf2\x1aL\xb5)\x83\xca\xea\xe8/\xef\x90\x80{>\x8e\xa7
32 Timestamp: 2018-10-22 18:30:28.439218
33 Message text: You don't talk much, huh?
34 Message confirmed at: 2018-10-22 18:30:28.443138
```