SEC MH2

lakl

October 2022

1 How to run

This is copied from the readme:

This is meant to be a "simulation" between Alice and Bob. For showing the protocol in effect, it is not possible in the current implementation to interact yourself between the two nodes. Proper example output as been placed in the folder **output**.

To run the program, please open two terminals and have python 3 installed.

In terminal one:

python Bob.py

In terminal two:

python Alice.py

2 Protocol

We must make sure that Alice and Bob can play the virtual dice game over an insecure network, where they do not trust each other, and do not want anybody else to see that they are playing dice.

To do this we must incorporate authenticity, integrity and confidentiality in our protocol. This can be done using the TLS[1] protocol and using certificates. I am using self signed certificates in my protocol.

2.1 Authentication

In standard TLS, it is only required for client to authenticate the servers certificate. However, to ensure authentication for both the client and server we must generate a certificate and a secret key for both Alice and Bob.

[1] `https://www.ibm.com/docs/en/ibm-mq/9.1?topic=tls-how-provides-identification-authentication-confidentiality-i`

The server verifies the client's identity by verifying the clients digital certificate with the public key for the CA that issued the personal certificate, and vice versa for the client verifying the servers identity. In this case the server is Bob and the client is Alice.

2.2 Confidentiality

TLS uses a combination of symmetric and asymmetric encryption to ensure message privacy. During the TLS handshake, the client and server agrees on a algorithm to encrypt the message, and a shared secret key to be used on the session. TLS uses asymmetric encryption when transporting the shared secret key.

2.3 Integrity

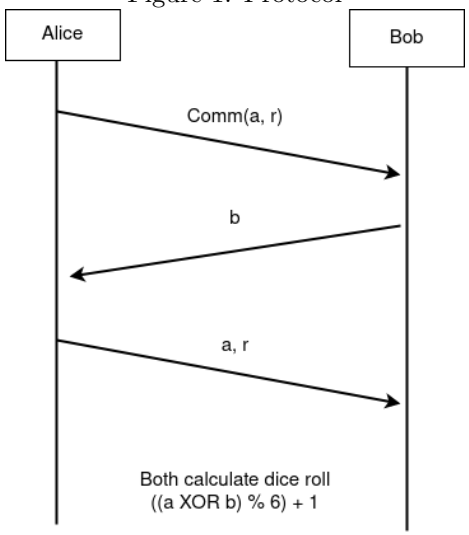
TLS provides data integrity by calculating a message digest. Message digest is a hash of data, which the receiver calculates after decrypting the original message, thus ensuring integrity of the message. Furthermore, integrity is also ensured via my commitment.

2.4 Commitment

I am using the hashing commitment, more specifically the SHA-256 hashing algorithm is used for calculating the commitment.

2.5 Protocol in action

Figure 1: Protocol



In figure 1, we see how the protocol works in action. First Alice sends her Commitment(a, r) to Bob. The a is her die roll, while r is a random 256-bit string.

Then Bob sends b, his dice roll, to Alice.

After this Alice sends her original a and r to Bob. Using this, Bob can use these arguments to calculate the commitment, and ensure that Alice's original message is the same. This means she is trustworthy, and now they can both calculate the dice roll using XOR.

3 Implementation

I have implemented the protocol in Python using the socket with ssl library[2]. Bob acts as a server, while Alice is the client in the socket setup. All code that is meant to be shared between them has been put in a Utility file. This is e.g. hashing commitment, calculating dice, etc.

With Bob acting as a server, it is ensured that he must also authenticate the clients certificate.

In both Alice and Bob, I make sure the TLS connection have been established correctly. The code can be seen in Figure 2, while the output in the terminal can be seen in Figure 3.

Figure 2: Protocol

```python
# Establish TSL is configured correctly
clientCert = conn.getpeercert()
print(f"client have cert: {clientCert != None}")

shake = conn.do_handshake()
print(f"handshake done: {shake == None}")

a, b, c = conn.cipher()
print(f"TLS cipher: {a, b, c}")
```

Figure 3: Protocol

```
client have cert: True
handshake done: True
TLS cipher: ('TLS_AES_256_GCM_SHA384', 'TLSv1.3', 256)
```

[2] https://docs.python.org/3/library/ssl.html