IT UNIVERSITY OF COPENHAGEN

Mandatory assignment 2

Course:	Security 1 (fall 2022)
Course code:	BSSECU11KU
Date:	25. oktober 2022
Author:	Andreas Wachs
Student number:	19167

Contents

1 The protocol

The protocol to play a game of dice over an insecure network between distrustful parties is as following.

Each dice roll for each party (Alice and Bob) is constructed from two partial rolls. In this way, should just one of the partial rolls for each party be an honest roll of a dice, then the resulting complete roll for that party be random.

This effectively means that Alice rolls a dice twice and gets rolls: r_{1A} and r_{2A}. Bob also rolls a dice twice and gets the rolls: r_{1B} and r_{2B}. Alice and Bob exchange their second rolls, such that they can compute their final roll for them selves.

The protocol is based on hash based commitments and use hash function $H(m)$ to generate hashes from messages m.

Communications is done over TLS and are thus inherently secure. See more in the techincal description.

1.1 Detailed walkthrough

Here I will describe how a game of dice is conducted with this protocol:

1. Alice generates her two partial rolls: r_{1A} and r_{2A}. Alice generates bitstrings b_{1A} and b_{2A} band commitments c_{1A} and c_{2A} for those rolls. Alice sends the commitments r_{1A} and r_{2A} to Bob

2. Bob generates her two partial rolls: r_{1B} and r_{2B}. Bob generates bitstrings b_{1B} and b_{2B} and commitments c_{1B} and c_{2B} for those rolls. Bob sends the commitments r_{1A} and r_{2A} to Alice

3. Alice reveals her commitments by sending b_{1A}, b_{2A}, r_{1A} and r_{2A} to Bob

4. Bob reveals his commitments by sending b_{1B}, b_{2B}, r_{1B} and r_{2B} to Alice

5. Alice verifies Bob's commitments for both partial dice rolls received: $c_{1B} \overset{?}{=} H(b_{1B}|r_{1B}$ and $c_{2B} \overset{?}{=} H(b_{2B}|r_{2B}$

6. Bob verifies Alice's commitments for both partial dice rolls received: $c_{1A} \overset{?}{=} H(b_{1A}|r_{1A}$ and $c_{2A} \overset{?}{=} H(b_{2A}|r_{2A}$

7. Both parties computes their own and their opponent's roll: $r_A = (r_{1A} + r_{2B}) \mod 6 + 1$ and $r_B = (r_{1B} + r_{2A}) \mod 6 + 1$ and announce the winner of the game.

2 Guarantees for confidentiality, integrity and authenticity

In this section, I will provide arguments as to why this solution provides guarantees of *confidentiality, integrity and authenticity* for the communications of Alice and Bob over an insecure network.

The HTTPS communication is using *mutual TLS*, where both parties are verified upon communicating, instead of just verifying the receiver of requests (in this case Bob).

2.1 Confidentiality

Message confidentiality is guaranteed from the Handshake protocol part of TLS. Here, asymmetric encryption is used to generate a shared secret key. An asymmetric key exchange protocol is bootstrapping the transition from asymmetric encryption to symmetric encryption for performance reasons.

2.2 Integrity

Part of the Handshake protocol also defines a shared secret key for generating MAC to sign messages with, between Alice and Bob. With each message, the receiver can verify that the message contents has not been tampered with along the way.

2.3 Authenticity

Guarantees of authentication is given from the TLS protocol, where communication between Alice and Bob is encrypted using the other party's public keys, and decrypted using their own private key. This ensures that messages received and decrypted, especially with the conjunction of the integrity guarantee, can be trusted to be genuine.

3 Technical description

For a description of the programming language and tools used, please see the `README.md` file provided with the source code.

In this solution, Alice and Bob are two instances of the same program, communicating over RESTful service endpoints, exposed with https.

When started, Alice and Bob both spawn 3 notable processes. A *client*, a *server* and a *endpoints* process. *Client* processes are responsible for initializing and progress through a game of dice with another party's *server* process. The *server* process responds to requests from the other party's *client* process. The *endpoints* process is located right before the *server* process, taking care of exposing appropriate RESTful endpoints and delivering messages to the *server* process and returning responses to the *client* processes.

The communication *client* and *server* processes are implemented in such a way that *servers* can handle multiple concurrent games and *clients* always plays the correct game.

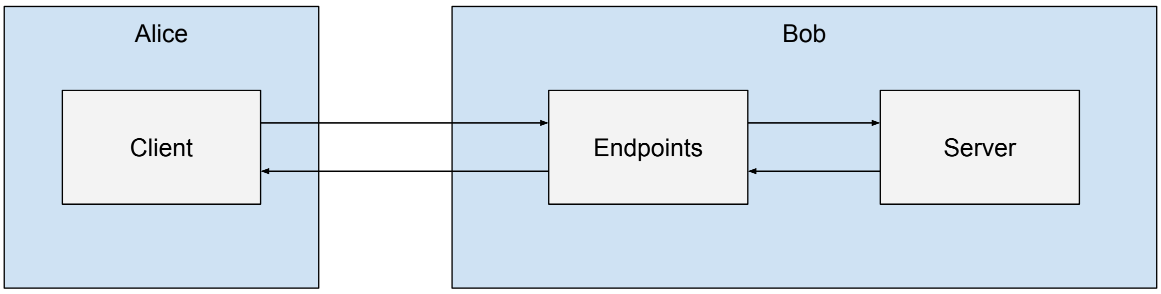

Figure 1: Visual representation of the communication flow between processes.

Communication from *client* processes to *endpoints* processes happen over the network, using the HTTPS protocol.

This solution uses hash based commitments, generated with the SHA3-512 algorithm. Random bitstrings generated are generated to with a length of 128 random chars, which should equal the length of the hashes from the SHA3-512 algorithm. This is symmetry is slightly ruined as I have to encode these both in base 16, as to avoid issues with the generated strings.