# Mandatory assignment 2

Created by: Andreas Nicolaj Tietgen (anti@itu.dk)

## Running the dice game

In order to run the project then you need to use the following tools and versions(to be sure that it works):

- Docker: Docker desktop 2.7.1
- OpenSSL: LibreSSL 2.8.3
- GoLang: v.1.19.2

If certificates has not yet been created then we need to do that before using docker. We do so by using the terminal:

- Go to the root of the project
- Type in the following `cd ./src/assets/certificates && sh gen_cert.sh`

Now that the certificates have been generated then we can start docker. We build and run the docker images by typing the following: `docker compose up --build`

### Output

When running the application through docker, the output will look similar to this(of course with a different result in terms of who wins):

```
sec-mandatory-exercise-02-bob-1    | Starting server...
sec-mandatory-exercise-02-bob-1    | Server listening on [::]:5001
sec-mandatory-exercise-02-alice-1  | Server endpoint address from flag: bob.dk:5001
sec-mandatory-exercise-02-alice-1  | Setting up client...
sec-mandatory-exercise-02-bob-1    | Alice Won!
sec-mandatory-exercise-02-alice-1  | Alice Won!
sec-mandatory-exercise-02-bob-1    | Alice Won!
sec-mandatory-exercise-02-alice-1  | Alice Won!
sec-mandatory-exercise-02-bob-1    | Alice Won!
sec-mandatory-exercise-02-alice-1  | Alice Won!
sec-mandatory-exercise-02-bob-1    | Bob Won!
sec-mandatory-exercise-02-alice-1  | Bob Won!
sec-mandatory-exercise-02-bob-1    | Alice Won!
sec-mandatory-exercise-02-alice-1  | Alice Won!
sec-mandatory-exercise-02-bob-1    | Bob Won!
sec-mandatory-exercise-02-alice-1  | Bob Won!
```

## Answer to the questions

Since this is an insecure network and Alice and Bob are paranoid who do not trust anybody, then we want the following security properties:

- **Confidentiality:** We don't want anybody who is not a part of the game to read our messages
- **Authenticity:** We need to be sure that the message actually comes from Alice or Bob
- **Integrity:** We need to be sure that the dice roll is not altered and that the message is the actual message being sent

In order to solve this we use Coin Tossing with hash based commitments, and TLS.

### Why do we use TLS?

We use TLS in order to provide *Confidentiality*, *Authenticity*, and *Integrity*. TLS provides

- Confidentiality by using symmetric encryption. This ensures that the message is not sent through the network in clear text
- Authenticity by signing the message.
- Integrity by creating a message authentication code(MAC).

Some of the key reasons to use TLS is that it handles the key exchange by using asymmetric encryption first. This is ensures that an adversary cannot see our keys in clear text. After the key exchange then we can begin to use the symmetric encryption which is faster than asymmetric encryption.

### Why do we use Coin Tossing with commitments?

The coin tossing scheme provides us with the ability to generate random values where we know that the other part cannot cheat by only roll a 6 in a 6 sided dice.

One roll is created by having two partial rolls. One from Bob and one from Alice. The roll is created by doing an XOR operation on the two partial rolls.

How can we be sure that the partial roll that they have sent is the actual roll? We do so by using *commitments*. We verify this by comparing the hash value of the random bit string and the message from the opening, and the commitment hash.

## How do Alice and Bob play the Dice game?

I have created a client/server solution. Alice is the client and Bob is the server. Alice is the one who initiates the game every time.

In order to communicate then we have created a GRPC service called Dice. This service contains two operations:

- Commit
- Reveal

When Alice sends a Commit message to Bob, then Bob replies with a Commit. The same goes for the reveal.

In order for them both to have a dice roll then the above happens twice. After that, they compare the rolls and decides who wins(Which you can see in the output).

The TLS is created with a self-signed CA Certificate which have signed the server certificate. The creation of certificates can be found in `assets/certificates/gen_cert.sh`.