Mandatory Assignment 2

Implementation

I have implemented a simple version of additive sharing, with the use of a TLS-protocol using OpenSSL.

This project consist of two elements: a server (the hospital) and a group of clients (Alice, Bob and Charlie).

Each client has a secret number that has to be summed up with the other two clients. To enable each client from sharing their secret number with the hospital, the clients mix their numbers before sending the result to the hospital. This is done, by each client splitting their number up into 3 uneven parts (x, y, z) before send two of the 3 part to each of the other peers repectively. After recieving two numbers from the two other clients, each sums up the 3 numbers (one part of own secret, and one part from each of client's secret), and sends that result to the hospital. The hospital sums up the final 3 numbers, into the final answer.

Regarding certificates

For this assignment, I have only generated a single key and certificate into one `.pem` file. The reason behind this decision, was that if each parcipant (both server and clients) should have their own certificate, each certificate should be signed by a Certificate Authority. For that Certificate Authority to be trusted, it would need to be trusted by my computer. This means, that for each computer my program is run on, the Certificate Authority would need to be trusted. This process would be cumbersome to say the least. Therefore, I have only generated a single key and certificate, that all parties use.

Generating 3 random parts of a single number

To split the secret number into 3 ueven parts, I have made a method called `split_in_three_uneven`. This method generates a `number1`, `number2` and `number3`, where only the last two will be send out to the other clients, while the first is kept by the client generating the numbers. `number1` is generated by selecting a random-number between 0 and the secret number `x` (`y`), and afterwards substracting `y` from `x`. `number2` is generated by selecting a random-number between 0 and `number1` (y), and afterwards substracting `y` from `x`. Lastly, `number3` is generated by taking `x - number2 - number1`.

If I should have changed anything, it would be to randomize which number would be kept, and which numbers would be send out, since this would make the entire procedure even more safe. I decided that for this project, it wouldn't be necessary.