Advanced Topics in Security and Privacy

Project 13: Secure Multi-Party Computation
Team 16

Ayça Avcı (s4505972) Fabian Prins (s3460509) Robbin de Groot (s3376508)

October 29, 2020

1 Problem Statement

Secure multi-party computation is a subfield of cryptography with the goal of creating methods for parties to jointly compute a function over their inputs while keeping those inputs private.
In this project, we are tasked to implement a simple, secure multi-party computation protocol. In this protocol, computation parties (CP) should be connected to each other and exchange messages through the network.

2 Requirements

There have to be at least three parties connected in a ring. The relationship between these parties is illustrated in figure 1. The parties should be able to receive data from the incoming connection and send data to the outgoing connection. The implementation needs to allow changes in the total computation parties that are part of the ring ($n \geq 3$). An additional computation party should be added at the end of the ring, such that CP_{n-1} now connects to CP_n, and CP_n connects to CP_1. When removing a party the neighbouring parties are simply connected such that the first-most neighbour in the ring has an outgoing connection to the other neighbour.

3 Methods

3.1 Protocol Definition

We define a protocol defined over the ring network as elaborated on in section 2. The protocol allows a multi-party computation of individual shares u_i of a multi-party object u. In this case, we have a shared integer number that gets randomly split and distributed over the computation parties as

$$u = \sum_i u_i.$$

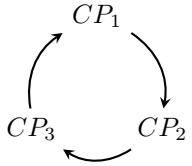

Figure 1: Smallest multi-party computation ring.

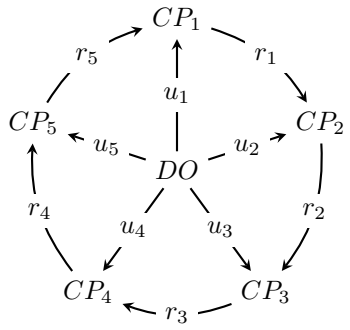

Figure 2: A ring of 5, showing off how the shares (u_i) are distributed by the data owner (DO) and the parts (r_i) are communicated from one client (CP_i) to the next.

Each party may share a part r_i of their share to the next in the ring. When every computation party does this, we may update each share to w_i as

$$w_i = u_i + r_{sent} - r_{received}.$$

The updated share w will have an equal value to u using this protocol, because

$$w = \sum_i u_i + \sum_i r_{sent,i} - \sum_i r_{received,i} = u.$$

We display the configuration of the CP_is, alongside the distribution of shares by DO and the exchanged parts in figure 2.

3.2 Implementation

To define the network and distribute the shares, we set up a temporary client-server architecture. We may view the data owner as the secret holder, the server as the ring initiator and the data owner's source of the client ports. This means that after the server has initiated the ring of clients, it will provide the data owner with the ports of all clients. We may view the clients as the possessors of the individual shares. This means that the clients should have access only to their own share, yet allow computations over those shares with their neighbours in the ring. The server sets up this relationship in 6 steps.

1. The server awaits the data owner to connect such that it can be instructed to distribute shares later.
2. The server defines how many clients it expects to join and awaits their request to connect.
3. Once sufficient clients are connected, the server assigns them each a unique port to bind to (assuming a local network, so the IP is the same for every client). The clients send a confirmation back as soon as the port is successfully bound.
4. When all confirmations are in, the server sends the ports to the data owner such that they can distribute the shares among the clients.
5. The server then sends the port of the client next in the ring for every client. The clients do not yet connect to a port as this causes a deadlock.
6. To prevent a deadlock, the server sends 2 different commands to the clients. Every client but the first ($CP_2, \ldots, CP_n$) receives a command to first accept any incoming request to connect, then attempt to connect to their given port in the previous step. The first client (CP_1) gets the inverse order of operations. This starts a connection daisy-chain starting and ending at CP_1.

The clients now send each other their r_i values in a daisy-chain (once again started at CP_1). This allows every CP_i to compute their updated share which is valid as soon as the chain returns at CP_1. In practice, this would be done out of interest of the clients, who will be represented by actual individuals.

4 Demonstration

We will not demonstrate the values handled by each client. The output below is raw output of 3 cycles of messages passed, provided by a 3 client ring.

```
Initial share:  48
3001 3002
Received:  67
Updated share:  27 ,
    r value used:  46
Received:  30
Updated share:  35 ,
    r value used:  38
Received:  91
Updated share:  -20 ,
    r value used:  36
```

```
Initial share:  88
3002 3003
Received:  46
Updated share:  74 ,
    r value used:  32
Received:  38
Updated share:  135 ,
    r value used:  99
Received:  36
Updated share:  131 ,
    r value used:  32
```

```
Initial share:  164
3003 3001
Received:  32
Updated share:  199 ,
    r value used:  67
Received:  99
Updated share:  130 ,
    r value used:  30
Received:  32
Updated share:  189 ,
    r value used:  91
```

From top to bottom, we first display the initial share provided by the data owner. Afterwards, we print the port this client is hosted at, followed by the port it connects to. Observe how these are structured in a ring like required. After that, we start receiving r_{i-1}. Combined with this value r_i, we update the share, which are the two values printed next. Observe how the updated share values all still sum to the original sum of 300, which is the hard-coded secret we choose. Of course, this value is easily randomised or hand-picked according to the application.
We show that using our protocol, the shared secret is maintained if parts of the shares are exchanged according to the definition of the protocol.

5 Discussion

The program presented in this report is not perfect. As no encryption is used, the messaged passed are still vulnerable to man-in-the-middle attacks and attacks of similar nature. Encryption of the packets using systems like PGP will correct this, but introduces drawbacks that come with these procedures.
Additionally, there is currently little control and limitations surrounding the generation of the random r_i values exchanged. It may be possible to obtain negative shares which may not be realistic according to the application. Additionally, their range is known and predictable. If the source of these values is to be random, further improvements can be made to better constrain and obfuscate them.

6 Conclusion

The protocol presented in this report maintains an MPC secret among a predefined amount of parties of at least 3 computation parties. The parties are connected to a single, unknown other computation party to which they can send part of their share. They will also receive such a part in order to finish the computation. This method maintains the shared secret while not revealing the shares to other parties. The architecture and protocol presented are relatively flexible and allow more complex algorithms to be defined upon them.