CS523 Project 1 Report

Pavliv Valentyna, Lion Clement

***Abstract*—During this project, we had to design a protocol that could compute secure-multiparty communications[1]. Our programming language is Lattigo, for it offers the opportunity to create goroutines. These are very appropriate to simulate multiple parties trying to communicate.**
During our project, we used the library Lattigo[2]. It offers homomorphic encryption which is particularly useful for the exchange of Beaver triplets, an invaluable element to do secure communications.

I. Introduction

During this project, our aim is to design a N-party multiplayer computation engine in a passive adversarial setting. Multiple peers all hold an input, the engine should be able to read a circuit asking to do a combination of the five following operations : addition, subtraction, addition by a constant, multiplication and multiplication by a constant. All these operations have to be carried out in a privacy-preserving way, all of the peers must ignore the values put as inputs except for their own values.

In order to fill in these constraints, our protocol follows three major steps:

- The peers split their inputs in shares and broadcast a share to all of the other peers.
- Each peers runs the circuit with the shares he received.
- Each peer broadcasts the result he obtained, the sum of all the results gives the final result to the computation.

During the first three weeks, we focused on designing the method that would split the shares of the peers and implementing the easier gates. We spent the next two weeks focusing on the multiplication gate in the trusted third party context. Finally, we modified the multiplication gate and removed the necessity of having a trusted third party by using homomorphic encryption.

II. Part I

A. Threat model

In the first part of our project, we have a passive adversary in the form of the other peers. They can just observe the data that is communicated to them and try to deduce information from it. We don't want to give away any knowledge susceptible to help them guess the initial input of the circuit. We make the assumption that there exists a trusted-third party who produces the Beaver Triplets when we need to do a multiplication.

B. Implementation details

1) Basic Protocol

In our implementation, each peer is represented as a dummyprotocol. It is a structure which has an ID, a channel to communicate with the other dummyprotocols, a list of all the peers with whom he will communicate, an input and an output where he will store the result of the multiparty computation.
At the beginning of our communication, we put the dummyprotocols in relationship with one another and check that they can communicate without any error. The dummyprotocols then proceed to splitting their values into shares and sending one of these shares to each of his peers. For this part, we decided to take the values uniformly at random.

Once this is done, we ask each dummyprotocol to run the circuit (represented as a list of operations) in a goroutine. They start by checking that they can run the operation and evaluate it if possible.

The verification consists of checking if the wires of the circuit supposed to hold a value do indeed hold a value and if the wire where we are supposed to write our computation is empty.

We have the following algorithm:

Algorithm 1 Dummyprotocol

```
1:  We bind the dummyprotocols together
2:  Each dummyprotocol splits his input and send a share to
    its peers
3:  for operations in circuit: do
4:      Check if you can evaluate the operation
5:      if You can carry out the operation  then
6:  ——Evaluate
7:      end if
8:  end for
9:  Broadcast the result to the other peers
10: Reconstruct result
```

In order to synchronize our dummyprotocols, we use a waitgroup structure (which can be seen as a semaphore).

Most of the operations are pretty straightforward to implement. Two of them are however a little more complicated: adding a constant and the multiplication.

- Adding a constant demands that only one of the peers the addition, or else, the computation will be false. We solve this issue by choosing that only the peer whose ID is 0 does the addition. Since our IDs start at 0, we are guaranteed to always have a peer with the 0 ID.

- For the multiplication, we use Beaver triplets, generated by a trusted third-party.

2) Beaver Protocol

We want to carry out multiplications. In order to do so, we need to create and exchange Beaver Triplets. In order to do so, we assign at the beginning of the computation a beaverprotocol to each of our dummyprotocols. A beaverprotocol can communicate with other beaverprotocols through a dedicated channel. They hold a triplet of a[i], b[i] and c[i] so that:

$$\sum_{i=0}^{i=n} a[i] * \sum_{i=0}^{i=n} b[i] = \sum_{i=0}^{i=n} c[i]$$

Every time we encounter a multiplication, we ask the beaverprotocol whose ID is 0 to generate n beaver coefficients uniformly between 0 and a threshold (we used 100 for our algorithm. This value should at least be as big as the biggest input of the peers in order to be hiding).
The beaver algorithm is

Algorithm 2 Dummyprotocol with a multiplication

```
1: We bind the dummyprotocols together
2: We attach a beaverprotocol to each dummyprotocol
3: We bind the beaverprotocols together
4: Each dummyprotocol splits his input and send a share to its peers.
5: if Dummyprotocol.ID is equal to 0  then
6: ——Generate beaver coefficients and send them to the other beaverprotocols
7: end if
8: Each dummyprotocol computes [x-a] and [y-b]
9: Each dummyprotocol broadcasts [x-a] and [y-b]
10: Each dummyprotocol computes [z] = [c] + [x] * (y-b) + [y] * (x-a)
11: if Dummyprotocol.ID is equal to 0  then
12: ——Do [z] = [z] - (x-a) * (y-b)
13: end if
14: Broadcast the result to the other peers
15: Reconstruct result
```

III. Part II

A. Threat model

In the second part of the project, our adversaries are still the other peers and they have a similar objective to part one. However, we remove the presence of the trusted-third party, and we therefore have to implement a protocol to exchange the beaver triplets without giving away their value. Indeed, if any of the peers guesses the value of the triplets, it can probably guess the value of one other peer. In order to do so, we will use homomorphic encryption. This will let the peers do calculations on the triplets without having to reveal them.

B. Implementation details

Give implementation details.

IV. Evaluation

From the description of the algorithms, we can see that the most computationally heavy operation is the multiplication. Therefore, while circuits involving only additions or operations with constants take only half a second to complete, circuits with many multiplications can become long. Moreover, the part when we bind our dummyprotocol network can take time and the more peer, the longer it takes.

A first difference in the two circuits is when we create our Beaver triplets. In our first implementation, we used to run the protocol to create a new triplet every time we needed one. In our second implementation, we create a large number of triplets at the beginning of the whole communication. As a result, for circuits with few multiplications, the algorithm from the first part performed better.

- Give a comprehensive comparison and evaluation about Part1 and Part2 of the project including performance results. Feel free to use charts, tables, plots...

- What affects the efficiency of the executions? Be specific, which types of operations/circuits are directly linked to performance?
- Is there any difference in terms of performance between Part I and Part II? Why?

V. Discussion

If we are in a situation with a lot of peers, it seems complicated to find a third party that everybody would trust. Therefore, the second algorithm should be the most suited for this type of situation (example: an online test where we want to compute the mean and the variance without anybody having to reveal their grade). However, when we have a lot of peers, the best would be to avoid complicated circuits and multiplications. If we have few peers, we can imagine that can agree on a trusted third party which could provide the beaver triplets (example: three organizations holding their own database and wanting to compute statistics from the three databases.). We can therefore think about doing complicated circuits.
Since the first case seems the more likely, we understand how important it is to develop powerful encryption schemes.

In order to go further, the next step would have made the communication fully-encrypted. During the parts where the peers exchange their inputs, we could encrypt the shares that are sent. All of the gates in our circuit would also have to support operations on encrypted data. We should however be careful not to do too many operations, for the bfv type of encryption only supports a limited number of them. In order to avoid computing false results, we should check the number of operations in the circuit. If the circuit is too long, we would have to divide it in multiple smaller sub-circuits and have the peers follow these circuits. This would however lead to the reveal of intermediary results, which could be insecure. The problem that we would also have with this approach is the complexity which would be much higher.

- Comment on your findings, discuss different outcomes for each part.
- Discuss outcomes from different circuits including your own circuit.
- In your opinion, which model is appropriate to use under which conditions/threat model? Why? Discuss.
- Come up with a scenario for each part of the implementation, discuss why it makes sense to use homomorphic encryption based generation of Beaver triplets.

VI. Conclusion

This project was first of all the opportunity to discover the go language and to have a glimpse of the possibilities it offers. It was also the chance to put in practice the notions we have studied during the course.
An important outcome that out of this project was the importance of synchronization. Many times, our algorithm would fail because one of the peers communicating took a little too much time and would be late to send his inputs. These issues relate more to telecommunications and it was interesting to see how this domain overlapped with information security.
Finally, this project was the occasion to put into practice some of the elements of the course Advanced Topics on PETS. Notions that appeared at first vague (like the beaver Triplets) took a new dimension. It can however be a little disheartening to see that a project that took place on two months gives only limited results.

- Assess your learning outcomes for this project.
- What did you do? What did you learn? Any interesting design ideas?

References

[1] O. Goldreich, S. Micali, and A. Wigderson, "A completeness theorem for protocols with honest majority," *Conference Proceedings of the Annual ACM Symposium on Theory of Computing*, 01 1987.

[2] "Lattigo 1.3.1," feb 2020, ePFL-LDS. [Online]. Available: http://github.com/ldsec/lattigo