CS-523 Project 1 Report

Furkan Karakaş, Onur Veyisoğlu

***Abstract*—In this report, we present the details and performance evaluations of a secure multi-party computation (SMC) engine implemented in the programming language `Python`. In this computational model, a group of participants executes basic arithmetic operations on secret data without revealing their own secret information to the other participants. Finally, we propose a real-world application where this computational model would be useful.**

I. Introduction

The aim of this project is to implement a Secure Multiparty Computation engine in Python and evaluate its performance in different settings. The engine that is implemented is capable of executing generic arithmetic operations in an N-party setting. Specifically, the engine can execute addition operations with secrets, as well as scalars, and multiplication operation between secrets by means of a Beaver Triplet Protocol using a trusted third party (TTP), and the usual multiplication operation of scalars.

We note that we assume the existence of a TTP if the arithmetical operations involve the multiplication of at least two secret values.

II. Threat model

We assume the existence of a *passive adversary*, i.e., the adversary does not alter the content of the messages in the network, in a *semi-honest* setting. If the adversary can control at least $N-1$ nodes, then the adversary can reconstruct the secret information of the remaining participant. So, we assume that the adversary controls at most $N-2$ participants, and the adversary can eavesdrop the messages in the entire network where a message is broadcast. Note that we assume that the adversary cannot read the private messages between participants. Also, we need to have the assumption that the TTP is indeed trustworthy, and it cannot be easily corrupted by the adversary.

III. Implementation details

In our implementation, each party processes the incoming expression recursively by means of a function called "process_expression". If the expression contains only scalar values, then we call another function called "processScalars" to carry out the multiplication, subtraction and addition operations with scalars. At the beginning of the protocol, each party broadcasts the identifier of the secret information that it possesses so that they can successfully map the ID of a secret expression to the secret share. Later, secret shares are generated and sent to the corresponding parties *privately*. The TTP uses a dictionary to record the Beaver triplet-secret pairs so that it can give the correct information to the parties who request a Beaver triplet from the TTP. At the end of the protocol, each party reconstructs the secret shares into the final result and hence, the SMC protocol is complete.

IV. Performance evaluation

Performance of our engine is evaluated in terms of both communication and computational costs. Specifically, the effect of number of parties, addition operations and multiplication operations on the costs are measured. The unit of computational costs is in seconds. The bytes received and sent are without units.

As the sample size, we use 25 iterations. At the end of the algorithms, we combine the measurements of different parties and we compute the statistics on the combined data.

As the underlying hardware, we use Intel(R) Core(TM) i5-10210U CPU @ 1.60 GHz (8 CPUs), 2.1 GHz and 8192MB RAM. The measurements are not directly conducted in the host operating system. Instead, we use Windows Subsystem for Linux (WSL) environment embedded into Windows 10.

A. Evaluation of Costs in Terms of Number of Parties

In this evaluation we fixed the circuit, which is simply $f(a,b) = a+b$ and we increase the number of participants.

TABLE I: Computational costs w.r.t. the number of parties

Participants	Max	Min	Mean	Median	Std.
5	0.52	0.14	0.23	0.17	0.09
9	0.67	0.41	0.46	0.45	0.05
50	21.56	12.7	17.72	17.98	1.53
100	83.22	66.01	69.3	68.515	3.19

TABLE II: Bytes received w.r.t. the number of parties

Participants	Max	Min	Mean	Median	Std.
5	60	57	59.62	60	0.67
9	108	104	107.24	108	0.99
50	600	591	596.62	597	1.76
100	1200	1184	1192.82	1193	2.86

TABLE III: Bytes sent w.r.t. the number of parties

Participants	Max	Min	Mean	Median	Std.
5	20	18	19.78	20	0.47
9	28	26	27.66	28	0.57
50	110	102	108.34	109	1.31
100	210	199	206.71	207	1.81

B. Evaluation of Costs in Terms of Number of Additions

In this part only additions with secrets are considered since addition with scalar takes constant time and requires no message exchange. We consider two participants in the SMC protocol and they compute $f(a,b) = \Sigma_{i=1}^{\frac{N+1}{2}}(a+b)$, where N is the number of additions that we want to test our system with. As convenience, we choose N to be an odd integer.

TABLE IV: Computational costs w.r.t. addition

#Add.	Max	Min	Mean	Median	Std.
5	0.06	0.03	0.04	0.04	0.01
49	0.05	0.03	0.04	0.04	0.0
499	0.07	0.03	0.04	0.04	0.01
999	0.06	0.03	0.04	0.04	0.01

TABLE V: Bytes received w.r.t. addition

#Add.	Max	Min	Mean	Median	Std.
5	24	23	23.82	24.0	0.38
49	24	23	23.94	24.0	0.24
499	24	23	23.88	24.0	0.32
999	24	23	23.9	24.0	0.3

TABLE VI: Bytes sent w.r.t. addition

#Add.	Max	Min	Mean	Median	Std.
5	14	13	13.86	14.0	0.35
49	14	13	13.96	14.0	0.2
499	14	13	13.94	14.0	0.24
999	14	13	13.92	14.0	0.27

C. Evaluation of Costs in Terms of Number of Multiplications

In this part only multiplications with secrets are considered since multiplication with scalar takes constant time and requires no message exchange. We consider two participants in the SMC protocol and they compute $f(a,b) = \prod_{i=1}^{\frac{N+1}{2}} (a \cdot b)$, where N is the number of multiplications that we want to test our system with. As convenience, we choose N to be an odd integer.

TABLE VII: Computational costs w.r.t. multiplication

#Mult.	Max	Min	Mean	Median	Std.
5	0.34	0.13	0.16	0.15	0.04
49	3.07	1.03	1.79	1.85	0.56
499	30.32	15.04	20.97	19.81	4.58
999	52.07	30.6	38.73	36.84	5.46

TABLE VIII: Bytes received w.r.t. multiplication

#Mult.	Max	Min	Mean	Median	Std.
5	124	120	123.12	124	1.26
49	1003	973	997.96	1000.0	7.54
499	9981	9715	9951.1	9970.5	69.47
999	19950	19428	19877.28	19938	164.64

TABLE IX: Bytes sent w.r.t. multiplication

#Mult.	Max	Min	Mean	Median	Std.
5	34	30	33.54	34	0.9
49	210	180	206.96	208.0	5.39
499	2001	1739	1983.54	1993	49.83
999	3987	3468	3946.62	3976.5	119.59

V. Discussion of Results

In the Table I, as the number of participants increases, the cost of computation increases non-linearly, which implies that as the number of participants gets close to 50 or more, there is a non-negligible loss of efficiency. The reason for that is probably due to server overloading and congestion. As there are more participants in the network, they all want to contact the server simultaneously. This phenomenon can be seen in Tables II and III where the number of bytes received and sent increases linearly.

Tables IV, V and VI demonstrate that increase in the number of additions does not cause a significant loss of efficiency since the number of secrets to be exchanged does not change with the number of addition operations, i.e., there is an exchange of secret IDs in the beginning and the exchange of final results in the end. Hence, the computational as well as communication costs are $\Theta(1)$.

As Table VII shows, computational costs increase proportional with number of multiplication operation. This is an expected result since the increase in number of multiplications results in linear increase in bytes exchanged as shown in Tables VIII and IX. Since we need to receive Beaver triplets and exchange additional messages for each multiplication operation, the costs are linear in that case.

VI. Application

Suppose that a group of college friends, namely Alice, Bob, Charlie, David, Eve, Frank and Gerald, want to take a vacation. They want to know how many people they will be in total so that they can do the proper arrangements beforehand. However, it turns out that Eve really finds it annoying that Frank is sometimes too cringe about his jokes, especially if they talk for a long time. For that reason, if Frank joins the group, then Eve will not join. Note that she might still not join regardless of Frank's situation, she has not decided yet. She does not want Frank to know this, so she wants to do the computation in a privacy-preserving manner. On the other hand, David is really into Charlie and he really wants to take the chance to be together with her. So, he participates if and only if Charlie participates. Alice and Bob do not really have such complicated feelings and they are chill about it. It is already known that Gerald will participate no matter what even if he is going alone. He definitely does not want to miss this vacation, it will be awesome!

The result of this secure multi-party computation can be represented with the following circuit:

$$f(a,b,c,d,e,f) = a + b + (c \cdot d \cdot 2) + f + e \cdot (1 - f) + 1$$

Note that +1 comes from Gerald since he certainly takes it. Also, scalar multiplication of 2 for Charlie and David is needed since they either participate together or neither of them participates.

The computation of such a circuit is given in the file called `test_application.py`. The real case scenario is tested for various situations, and the result is computed by each person individually. Each of the tests passes with success.

There are various possible privacy leakages in this system. For example, if they compute the final result to be 1, then they immediately conclude that only Gerald will go, and neither of the remaining people will participate in the group. On the other hand, if the result is 2, then David already knows that Charlie will not participate (by the way, David's secret is always 1 in this case). If she did, then the result would have been *at least 3*. But still, David cannot determine who else participates, i.e., he cannot distinguish the remaining people from one another. One more scenario would be this – assume that 6 people are participating. Then the person who did not want to participate, whoever he/she is, knows for sure that the remaining people will participate in the vacation. In the case of Eve, if she did not want to participate, then she learns that Frank was in the group. So, she can rest assured since she did not want to come in the first place!

Unfortunately, we could not come up with a counter-measure against the privacy-leakage problems described above. The circuit above could be extended to be made more complex by adding randomness, which is coming from a dummy participant called "Zed". But in that case, that will introduce additional problems since they might end up booking more or fewer places for the vacation. This randomness can be always positive so that everyone will be guaranteed to get his or her resources, but in that case, they will end up spending more than necessary, which is a wastage of resources.