CS-523 SMCompiler Report

Laval Maxime, Carles Victor

I. Introduction

The goal of this project was to implement and apply a secure multi-party computation engine (secure MPC, or SMC) in a semi-honest(passive) adversarial setting. We implemented an SMC framework that works for generic arithmetic circuits assuming the existence of a trusted third party and additive secret sharing. We considered arithmetic circuits made of additions, addition of a constant, substractions, multiplication by constants, and multiplication. For the Multiplication circuit we used the Beaver triplet protocol that uses ”blinding” values a, b, and c sampled randomly such that ab = c. Give a brief introduction about the project and its aims.

II. Threat model

The adversary would be any party trying to recover a secret from another party during the computation of the arithmetic circuit. As every party only receives a share of the secret, and since these parts are numbers in Zp, it is impossible for the adversary to recover the secret from only the shares he receives. However, an eavesdropper trying to reconstruct the secret from the parts sent on the insecure channel could also be part of the threat model due to the fact that there is no encryption on the network layer, that is why everything should be encrypted in the future.

III. Implementation details

We implemented some additional tests class : `test_expression` and `test_secret_sharing` . We also created a class to evaluate the performance : `performance_evaluation`. The computation for the artithmetic circuits are made in the group Z / pZ of prime order p , where p is a large prime number between 2^{29} and 2^{32}. We chose a prime group for the multiplicative context of the circuits computation to be sure we can manipulate invertible numbers.

IV. Performance evaluation

We evaluated the performance in term of communication (bytes sent and received) and computations costs (computation time) . To compute the length of bytes exchanged we defined a class variable `num_communication_bytes` in SMCParty and everytime some bytes are received or sent a client we increment the length of the total bytes exchanged. The way we computed our performance was by first calculating the mean across all participants of the time taken by each client to finish the circuit and the length the bytes sent and received by each participant for a circuit computation . We then repeated this mean calculation n = 10 times to obtain a representative statistics in order to compute the mean and the standard deviation . When trying with circuit of over 100 Additions we obtained a ” recursion error : maximum recursion depth exceeded” , thus we restrained ourselves to 50 operations for the Addition. We also observed that the standard deviation for the number of bytes exchanged was always 0 (except for the Multiplication) which is logical as every-time when running the protocol for each party the quantity of information exchanged is the same. For the Multiplication the number of bytes exchanged is huge and the deviation as well. This is because due to the Beaver scheme the quantity of bytes will hugely depend for every computation. We show our result here in form of tables containing the measurements of the costs for each value of the parameter with the mean and standard deviation .

	Time (s) : Mean +-SD	Bytes Length : Mean +-SD
3 Participants	0.044 +-0.03	296+-0.00
10 Participants	0.74+-0.099	296+-0.00
15 Participants	2.65+-0.16	296+-0.00

TABLE I: Effect of the number of parties on the cost.

	Time (s) : Mean +-SD	Bytes Length : Mean +-SD
3 Additions	0.057+-0.02	296.00+-0.00
10 Additions	0.14+-0.004	758+-0.00
50 Additions	0.31+-0.058	3398+-0.00

TABLE II: Effect of the number of Additions on the cost.

	Time (s) : Mean +-SD	Bytes Length : Mean +-SD
10 Scalar Additions	0.047+-0.02	120+-0.00
100 Scalar Additions	0.07+-0.003	1020+-0.00

TABLE III: Effect of the number of Scalar Additions on the cost.

	Time (s) : Mean +-SD	Bytes Length : Mean +-SD
10 Multiplications	0.53+-0.09	2201245873.6+-1070110034.089642
100 Multiplications	3.51+-0.57	2328374422.8+-1003267299.057814

TABLE IV: Effect of the number of Multiplications on the cost.

V. Application

For the custom application, we decided to implement an system where each party is a student. Each student stores their grades as their secrets, and can share them when the class needs to calculate the mean and the variance for a given exam. The idea is that each student wants to keep their grades secret

	Time (s) : Mean +-SD	Bytes Length : Mean +-SD
10 Scalar Multiplications	0.033+-0.007	29296875.0+-0.00
100 Scalar Multiplications	0.0561+-0.0042	29683875.0+-0.00
500 Scalar Multiplications	0.1+-0.012	232837442+-0.00

TABLE V: Effect of the number of Scalar Multiplications on the cost.

from the rest of the class, but at the same time wish to know what the mean and variance were for a given exam. In our code, we implemented the python file ”application.py” which basically applies the SMC protocol for a class of n students, where each student has its grade as a secret, and processes the mean and the variance in the runApplication method. The SMCompiler is perfect for classes of maximum 30 students, as a big n will greatly impact the performance. As we saw in the performance evaluation, SMCompiler hardly scales, hence is better for applications with just a few parties.

Let’s take for instance a class of 3 students. In our application, we suppose that the teacher adds 0.5 points to every student as a way of balancing the final mean. Firstly, each student calculates the sum of all grades, by sharing parts of their secrets, adding all secrets together and 0.5 for the balancing. Then, they can divide the result by 3 to find the mean. The expression that is used for the sum circuit is
((s1Secret + Scalar(0.5)) + (s2Secret + Scalar(0.5)) + (s3Secret + Scalar(0.5))).
Then, the students can use the mean value computed locally to calculate the variance, with the following expression:
(((Scalar(mean) - (s1Secret + Scalar(0.5))) * (Scalar(mean) - (s1Secret + Scalar(0.5))) + ((s2Secret + Scalar(0.5)) - Scalar(mean)) * ((s2Secret + Scalar(0.5)) - Scalar(mean))) + (Scalar(mean) - (s3Secret + Scalar(0.5))) * (Scalar(mean) - (s3Secret + Scalar(0.5))))
Once the result is computed for all parties, they can divide it by 3 (number of students) to obtain the variance.

There can still be some privacy leakage in this system as, in some cases, a student could infer the grades of the other students from the computed sum. For instance, in a class of 3 students, if the sum is 16, and the student’s grade is 4, he knows for sure that the other students both had 6. Same goes with the minimal grade (sum of 6). Also, an eavesdropper spying on a specific student could also gather all parts of its secret in the first part of the protocol and reconstruct the grade easily, as nothing is encrypted. The eavesdropper only has to know p and sniff the network when the target is sending its part to the other parties.