CS523 Project 1 Report

Lucas Rollet, Lucie Hoffmann

***Abstract*—In this report we explain the design and implementation details of our Secure Multiparty Computation protocol. We also report the findings we made and a broad performance analysis of the impact of various parameters in our system.**

I. Introduction

We have implemented an SMC framework that allows several parties to compute a function on their respective sensitive data without revealing this data. We first describe the threat model of our framework, then we discuss implementation details and performance evaluations, and finally we introduce a concrete use-case of this implementation.

II. Threat model

In our SMC framework, a party can use the the computation's result to infer some other party's secret. This should be mitigated by making sure that the function to compute hides secrets (by adding a constant for instance). A party could also cooperate with other parties (or corrupt them), to gain more information on some secrets and infer the others.

This threat model considers only passive honest but curious adversaries, i.e. they comply to the protocol and share their honest results. What about active malicious adversaries? Our threat model does not consider a party that can broadcast wrong shares to harm others' computation result.

III. Implementation details

A. An abstract representation of function

Given the function to compute on shared secrets, every party needs to make their own computations on their respective shares before broadcasting their different results and combine them to get the final secret result.

For each global operation in the function (by global we mean that the function relates to the shared secret and not local shares themselves), a party needs to make adequate computations locally. In our implementation, we represent functions with an abstract syntax tree structure. Scalars and secrets constitute the leaves, nodes combine two scalars, secret or results of a previous operation into a new operation. Each party can then go through the operations of this abstract function tree and perform the corresponding concrete operations.

We represent elements of the abstract function as Expressions. These can be either `Scalar`, `Secret`, `AddOp`, `SubOp` or `MultOp`. For example, the function $(a+b+K)*c$, where a, b and c are shared secrets and K a scalar value, is represented as `MultOp(AddOp(AddOp(Secret,Secret), Scalar),Secret)`. Every `Expression` has an id that helps us determine which values it corresponds to. For example, we want to know which `Secret` to consider in a specific operation and use the share corresponding to that specific `Secret`.

Additionally, we represent a concrete share value from a secret by a `Share`, that we can add, subtract, multiply to/with another `Share`.

B. Beaver triplets and shares generation

For the Beaver triplets generation, we simply randomly generate two integers and multiply them to obtain the third one.

We produce shares from a secret by randomly picking positive integers in a set bounded by the secret value minus the number of shares we still need to assign.

IV. Performance evaluation

To realize the performance evaluation of our system we evaluated every parameter independently. We stored the computation time and cost for each party, then computed and plotted the mean values.

To evaluate the impact of the number of operations, the methodology is always the same: we take two parties `Alice` and `Bob` that both have a secret or a scalar number `a` and b. Given these expressions, we vary the number of operations (multiplications or additions) by performing the expression $a+b$ or $a*b$ $\boldsymbol{k}$ times for an increasing value of $\boldsymbol{k}$.

To evaluate the cost of increasing the number of parties, we created an expression summing 1000 different secrets that were shared between the parties. If we have 2 parties, they will have 500 secrets each, but if we have 1000 parties, they will have 1 secret each. This method makes use of the property that a party can own multiple secrets.

Our performance evaluation lead us the following conclusions:

- Addition and Subtraction do not add more communication cost to the initial communication of secret shares. The computational cost is linear with the number of such operations. Furthermore, the computation time stays nearly constant.

- Multiplication, on the other hand, adds a lot of communication overhead due to the Beaver triplets shares that have to be retrieved for every new operation. The computation time also increases a lot.

- The number of parties has an important impact on the computation time but the communication cost stays constant after a threshold of 10 parties.

The plots showing our results can be found at the end of the document.

V. Application

We want to compute the average cost of hospitalization for a patient with colon cancer over 3 hospitals. These hospitals have a secret number of such patients, we call them a_1, a_2 and a_3, and a secret average time they spent in the hospital, we call them b_1, b_2 and b_3. The cost of a patient staying in a hospital and the average insurance's reimbursement are known respectively as the constants K_0 and K_1.

We first use our SMC framework to compute the total number of patients over all hospitals with the following expression: $a_1 + a_2 + a_3$. Then we compute the total cost for every patient in every hospital as: $(a_1 * b_1 + a_2 * b_2 + a_3 * b_3)K_0 - K_1$. The average cost is then the division of the total cost by the total number of patients.

The number of patients with colon cancer and their time spent in hospital is sensitive and should stay confidential. This is ensured by the SMC framework as long as no two parties cooperate to get the third party's number of patients. Indeed if two parties put in common their respective shares and secrets, they can recover the third hospital's secret with the output of the function. Note that for this to happen, all hospitals but one need to cooperate and share their sensitive information, which might not be worth it.

Another privacy limitation appears in the case where only one hospital has one patient with colon cancer, one of the other hospital will learn this information as well as the time this patient stayed in the hospital with probability $\frac{1}{3}$. That is because the shares of the number of patients with colon cancer for the victim hospital, i.e. (1,0,0), will be randomly distributed. This could be mitigated by adding some noise to the secrets before using them in the SMC framework. We could then have to deal with a utility-privacy trade-off.

Figures

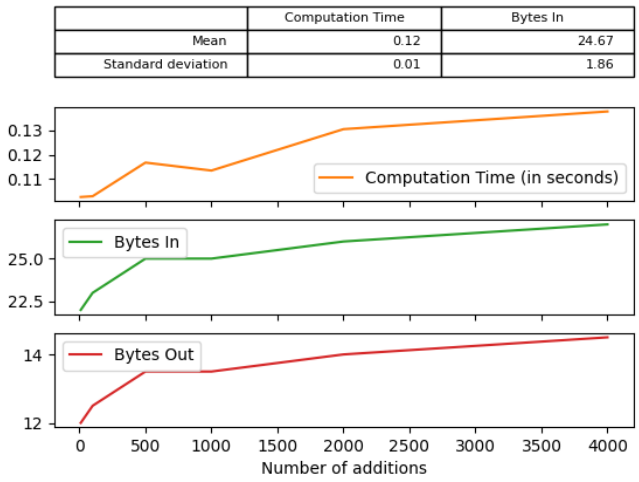

Fig. 1: Number of addition operations

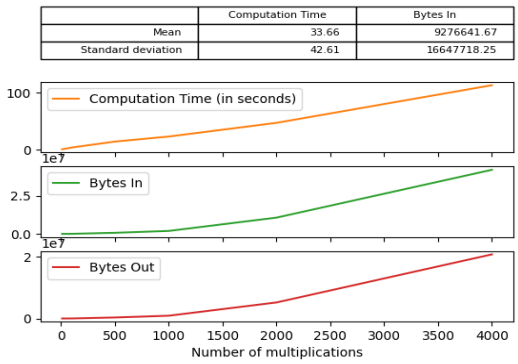

Fig. 2: Number of multiplication operations

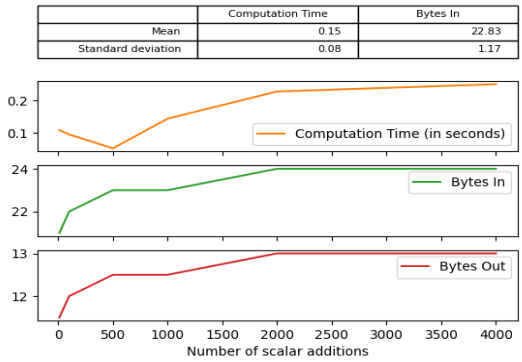

Fig. 3: Number of scalar addition operations

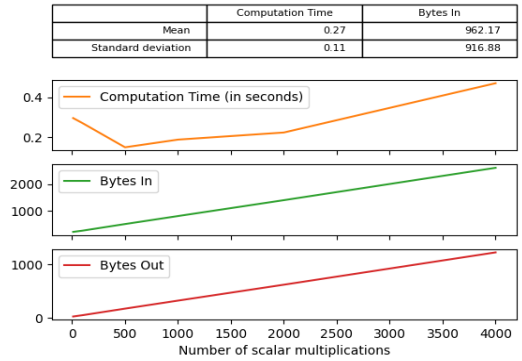

Fig. 4: Number of scalar multiplication operations

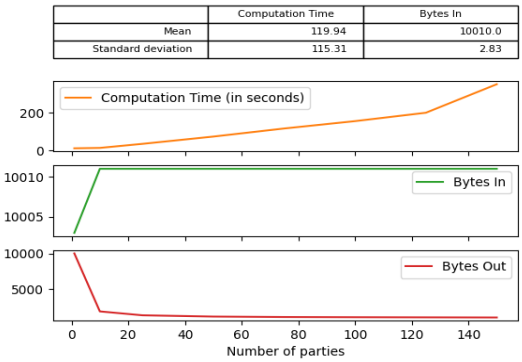

Fig. 5: Number of parties