CS-523 SMCompiler Report

Stuart Gueissaz, Kilian Lauener, Xavier Ogay

***Abstract*—This report presents a secure multi-party computation (SMC) engine developed in Python 3. The engine uses additive secret sharing and a trusted third party to perform secure arithmetic operations within a semi-honest adversarial model. Key operations include addition, subtraction, and multiplication using the Beaver triplet protocol. The SMC framework was applied to securely aggregate refugee data and financial contributions among EU member states, ensuring privacy and accurate decision-making.**

I. Introduction

This report details the development and application of a secure multi-party computation (SMC) engine, implemented in Python 3, as part of the CS-523 course. The project aims to enable secure computations on arithmetic circuits within a semi-honest adversarial setting, leveraging the presence of a trusted third party to facilitate secure operations. The overarching aim of the implemented SMC framework is to maintain the privacy of inputs while securely performing computations on arithmetic circuits. This is accomplished by employing additive secret sharing and utilizing a trusted third party for secure operations, including addition, subtraction, scalar multiplication, and multiplication using the Beaver triplet protocol.

We worked mainly together to grasp a full understanding of the project, also such that each member worked equally.

II. Threat model

The framework operates within a semi-honest (passive) adversarial model. In this context, adversaries are assumed to adhere to the protocol correctly but may attempt to extract additional information from the messages they receive. The design of the framework ensures that even in the presence of collusion among participants, no private input information is disclosed beyond what is inferable from their own inputs and the final output.

The framework relies on a trusted third party to facilitate secure computations, particularly for operations that require shared randomness or precomputed values, such as those in the Beaver triplet protocol.

III. Implementation details

We started by implementing expressions, we then went onto implementing the basic arithmetic operations on the `Share` class. The choice of prime for the field is rather arbitrary, we chose a prime of the appropriate size, here we went with a 64 bits hardcoded prime number. Given that the field is a public parameter, a hardcoded prime works well. For addition of scalars only participant 0 does it, same goes for the beaver case, only the participant 0 adds the $-(x-a)(y-b)$ final terms.

IV. Performance evaluation

Our performance evaluations measured the communication costs and runtime of the SMC framework for scalar additions, scalar multiplications, and general arithmetic operations. The results are summarized below.

A. Communications Costs

Table I and II represent the situation where only constants scalar are added or multiplied together. Table III and IV represent one secret per participants that are added or multiplied all together. Table V and VI represent 5 secret per participant.

Notes that the *Mean total bytes* is the count of bytes exchanged (sent and received) during the whole protocol by every parties. The *Mean bytes received/sent* are the mean numbers of bytes exchanged, computed on each individual parties during a protocol run.

TABLE I: Communications on scalar additions

Nb of add.	Mean total bytes	Mean bytes received	Mean bytes sent
10	136.53 ± 1.28	5.00 ± 0.00	40.51 ± 1.19
100	140.37 ± 1.26	6.00 ± 0.00	40.79 ± 1.61
500	144.72 ± 1.29	7.00 ± 0.00	41.24 ± 2.01
1000	144.23 ± 1.59	7.00 ± 0.00	41.08 ± 2.09

TABLE II: Communications on scalar multiplications

Nb of mult.	Mean total bytes	Mean bytes received	Mean bytes sent
10	136.41 ± 1.56	5.00 ± 0.00	40.47 ± 1.26
50	160.26 ± 1.38	11.00 ± 0.00	42.42 ± 3.85
100	180.58 ± 1.53	16.00 ± 0.00	44.19 ± 6.16

TABLE III: Communications on general additions

Nb of add.	Mean total bytes	Mean bytes received	Mean bytes sent
10	5632.47 ± 25.36	369.07 ± 2.59	194.17 ± 1.93
20	22886.14 ± 44.08	756.11 ± 3.35	388.19 ± 3.02
30	51884.00 ± 128.14	1147.28 ± 4.90	582.19 ± 3.28
40	92287.00 ± 114.27	1531.12 ± 4.57	776.05 ± 3.76
60	208294.29 ± 196.14	2307.66 ± 5.47	1163.91 ± 5.06

TABLE IV: Communications on general multiplications

Nb of mult.	Mean total bytes	Mean bytes received	Mean bytes sent
10	51161.14 ± 43.42	4573.06 ± 5.01	543.06 ± 2.99
20	362443.86 ± 217.77	16997.13 ± 10.55	1125.06 ± 4.58
30	1167500.71 ± 1148.11	37209.23 ± 34.79	1707.46 ± 5.99

TABLE V: Communication costs for general addition with different numbers of participants with 5 secrets each

Participants	Mean total bytes	Mean bytes received	Mean bytes sent
5	2716.92 ± 10.79	368.65 ± 2.73	310.58 ± 2.23
10	19596.14 ± 37.61	1067.41 ± 4.14	892.2 ± 4.10
30	186762.57 ± 87.09	3393.73 ± 7.73	2831.69 ± 6.94

TABLE VI: Communication costs for general multiplication with different numbers of participants with 5 secrets each

Participants	Mean total bytes	Mean bytes received	Mean bytes sent
5	23477.14 ± 33.67	4822.29 ± 7.99	1047.00 ± 4.73
10	267421.71 ± 220.70	23949.41 ± 20.37	2792.76 ± 7.99
30	5916371.00 ± 2419.19	188599.60 ± 74.39	8612.76 ± 11.97

B. Runtime

TABLE VII: Runtime for operations

Operation	Nb of operations	Mean runtime (ms)	Std Dev (ms)
Scalar addition	10	14.10	0.70
Scalar addition	100	14.20	0.58
Scalar addition	500	14.88	0.62
Scalar addition	1000	15.13	1.32
Scalar multiplication	10	14.62	1.56
Scalar multiplication	50	14.68	1.22
Scalar multiplication	100	23.60	4.69
General addition	5	29.86	2.78
General addition	10	83.35	6.31
General addition	20	322.69	15.86
General addition	30	694.97	24.21
General addition	40	1262.81	51.28
General addition	60	2846.07	67.61
General multiplication	5	99.21	12.45
General multiplication	10	613.42	28.22
General multiplication	20	4471.92	44.99
General multiplication	30	14827.38	155.94

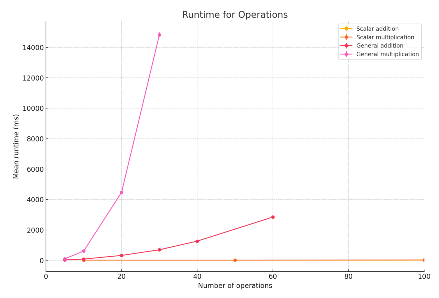

Fig. 1: Runtime for general operations

Based on the graphs provided, we can draw several conclusions about the runtime performance of scalar and general operations:

- **Scalar Operations**: Given the little reliance on the network these operations naturally scale very well.
- **General Operations**: The addition of secrets is as expected linear in the number of secrets. Multiplication require new beaver triplets for each of them, the growth is much larger and limits the scalability of the process.

TABLE VIII: Runtime for X participants with 5 secrets

Operation	Nb of participants	Mean runtime (ms)	Std Dev (ms)
General addition	5	51.59	4.72
General addition	10	286.77	26.44
General addition	30	2494.89	36.73
General multiplication	5	276.82	9.55
General multiplication	10	3118.82	30.72
General multiplication	30	76496.09	565.27

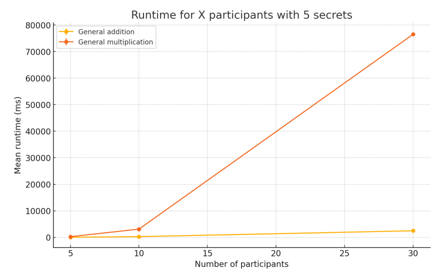

Fig. 2: Runtime for X participants with 5 secrets

We note the difference when participants hold a single secret or when the secret are spread among the participants the growth is more controlled.

V. Application

The European Union (EU) often needs to coordinate the reception of refugees and the allocation of related financial support among its member states. To address privacy concerns and ensure fair participation, a secure multi-party computation (SMC) framework is proposed. This framework securely aggregates the number of refugees and the funds provided by each country without revealing individual data during the computation process.

A. Aim

The aim is to securely compute the total number of refugees the EU can welcome and the total funding required, using inputs from multiple countries while preserving the privacy of each country's contributions during the computation process.

B. Adversarial Model

We assume a semi-honest adversarial model where parties follow the protocol correctly but may try to learn additional information from received messages. The SMC protocol ensures that no individual party can infer the contributions of others during the computation.

C. SMC Protocol

The protocol involves each country submitting two secrets: the number of refugees they can welcome and the amount of money they can provide per refugee. The following steps are performed:

1. Compute the total number of refugees:

$$\text{Total refugees} = \sum_{i=1}^{n} \text{refugees}_i$$

2. Compute the total cost:

$$\text{Total Cost} = \left(\sum_{i=1}^{n} \text{refugees}_i\right) \times \left(\sum_{i=1}^{n} \text{money\_per\_refugee}_i\right)$$

D. Use Case Description

The application securely computes the total number of refugees the EU can welcome and the total cost of receiving these refugees. This can be applied to secure planning and budgeting for humanitarian efforts. The results of the computation are then used by the EU to vote on the approval of the proposed budget. Each member state can subsequently agree or disagree with the total number of refugees and the total needed budget, knowing exactly how much they will need to contribute.

To ensure dynamic and continuous assessment, member states are required to use this system continuously until the vote passes. They can change their input (number of refugees they can welcome and the amount of money they can provide) at any time during this period. If the vote passes, all information regarding the individual contributions will be disclosed to ensure transparency and accountability.