

CS-523 SMCompiler Report

Xingxing Chen, Yihan Huang

Abstract—The report presents a secure multi-party computation engine (SMC) and discusses its implementation and performance evaluation. The threat model assumes an Honest but Curious setting. The implementation details cover the general procedure of implementing the protocol, encryption and reconstruction of secrets, and expression processing. The performance evaluation analyzes the effect of different parameters on communication and computation costs, revealing that the number of parties and multiplication operations significantly affect costs.

I. INTRODUCTION

This project aims to develop a secure multi-party computation engine (SMC) using Python 3 programming language. The SMC engine will allow multiple parties to perform computations on their private inputs without revealing them to other parties. The SMC framework will be designed to work with generic arithmetic circuits, assuming the presence of a trusted third party.

II. THREAT MODEL

The threat model of the project is Honest but Curious setting, which assumes that the parties will follow the SMC protocol, but try to learn as much as possible from the messages they receive.

III. IMPLEMENTATION DETAILS

A. Implementation of SMC Protocol

We have implemented the SMC protocol framework in the `smc_party.py` file. The first step for a party is to create shares of its secret based on the number of parties and send them to the other parties. Once each party has completed this step, they publish a message to inform others. When a party has received the public messages from all other parties, it proceeds to the next step, which is to calculate the expression. In this step, the party obtains its own encrypted calculation result based on the shares. Next, the party broadcasts its result and retrieves the calculation results from other parties, summing them to reconstruct the secret.

B. Encryption and Reconstruction the secret

1) *Encryption of secret*: Assuming that a party owns a secret value, denoted by x , we set a field F as the integer modulo a prime p . We then uniformly sample $(n-1)$ random integers from the range $(0, F)$, denoted by x_2, \dots, x_N , where n is the number of parties. These integers are the shares generated for the other parties. The share x_1 for the party itself is generated using the following mathematical expression.

$$x_1 = x - \sum_{i=2}^N x_i \quad (1)$$

Therefore, we get the shares $[x]$ for the secret x , where $[x] = x_1, x_2, \dots, x_N$

2) *Reconstruction of secret*: Assume we have a set of shares $[x] = x_1, x_2, \dots, x_N$, then we reconstruct the secret x by $x = \sum_{i=1}^N x_i$.

C. Process Expression

We utilized recursion to process the expression and aimed to unify all elements to the share type, which facilitates the calculation of the final result. We divided the expression processing into four cases, based on the operator, and separated the expression into left and right expressions.

1) *Addition without scalar or Subtraction*: If the expression is an addition or subtraction operation and both the left and right expressions are not Scalar type, we can simply add or subtract them together.

2) *Addition with scalar*: If the expression is an addition operation and involves a scalar, we assign the first party in the `participants_list` as the captain to calculate this expression. Non-captain parties have three possible cases to process the expression. Firstly, if the left expression contains a scalar, the party only returns the right expression. Secondly, if the right expression contains a scalar, the party returns the left expression. Finally, if both the left and right expressions contain scalars, the party returns `Share(0)`.

3) *Multiplication with scalar*: If the `expr` is multiplication operation and one of left and right `expr` is the scalar. Then we directly do the multiplication. If the expression is a multiplication operation and either the left or right expression is a scalar, then we can directly perform the multiplication operation.

4) *Multiplication using the Beaver triplet protocol*: We implemented a trusted parameter generator to generate (a, b, c) for each multiplication operation. We uniformly sample a and b randomly from the field F and calculate c using $c = a * b$. Then, we generate shares of (a, b, c) for each party and save the Beaver triplet in a dictionary called `triplet_dict` with the key being `expr_id`. Each party retrieves its Beaver triplet share $[a], [b], [c]$ from the `triplet_dict`.

Each party locally computes $[x - a]$ and $[y - b]$, where $[x]$ and $[y]$ correspond to the calculation result of the left and right expressions, respectively. Each party broadcasts its result and calculates $(x - a)$ and $(y - b)$ by summing shares $[x - a]$ and $[y - b]$ retrieved from the other parties, respectively. The party then calculates the expression $[z] = [x - a] \cdot [y - b] + [x] \cdot [b] + [y] \cdot [a] + [c]$ to get its local result.

$$[z] = [c] + [x] * (y - b) + [y] * (x - a) - (x - a)(y - b) \quad (2)$$

Notice that we indicate the first party in the `participants_list` as the captain, and only the captain can add the term $-(x - a)(y - b)$ locally, while other parties should not add this term.

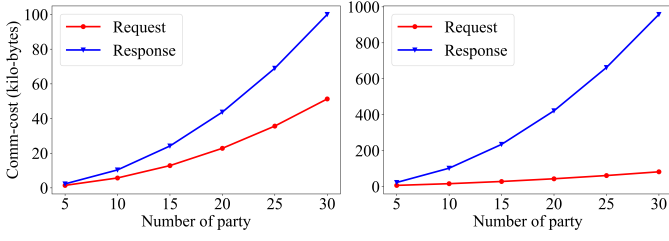


Fig. 1: The effect of the number of parties on the costs of addition(left) and multiplication(right)

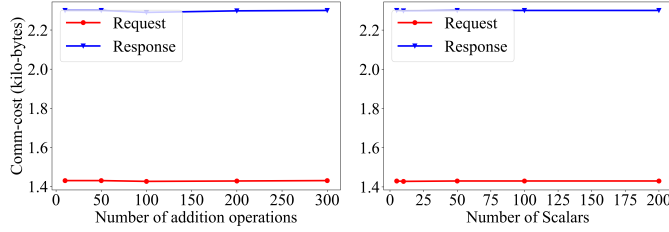


Fig. 2: The effect of the number of addition operations(left) and additions of scalars(right) on the costs

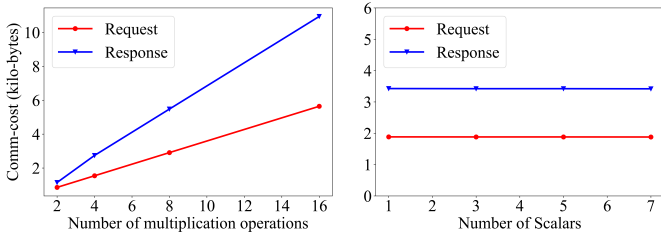


Fig. 3: The effect of the number of multiplication operations(left) and multiplications of scalars(right) on the costs

IV. PERFORMANCE EVALUATION

A. Communication Costs

We designed six experiments to test the effect of changing different parameters on the communication costs. The y-axis for all figures represents the communication cost in kilobytes. The blue line represents the bytes sent (response), and the red line represents the bytes received (request).

As shown in Figure 1, the left plot shows the relationship between the communication costs and the number of parties when implementing 50 addition operations. In contrast, the right plot shows the relationship between the communication costs and the number of parties when implementing 10 multiplication operations.

In Figure 2, we set 5 parties to test the effect of the number of addition operations and additions of scalars on the costs. We observed that, regardless of how we changed the number of addition operations and scalar additions, the communication costs for request and response remained stable.

In Figure 3, we used 3 parties to test the effect of the number of multiplication operations and scalar multiplications on the costs. As shown in the left plot, the communication

costs increased nearly linearly as the number of multiplication operations grew. However, since we processed the multiplication of multiple scalars before passing the expression to the `process_expression` function, increasing the number of scalars did not affect the costs. For example, $(leftexpr) * 5 * 5 * 5$ would be processed to $(leftexpr) * 125$ before being passed to the function.

B. Computation Costs

TABLE I: 5 parties implement 100 additions

Min	MAX	Mean	Std-Deviation
5.5300	5.7827	5.7054	0.0880

TABLE II: 5 parties implement 10 multiplications

Min	MAX	Mean	Std-Deviation
5.7284	7.7316	6.8391	0.6409

TABLE III: 5 parties implement 50 additions and 100 additions of scalars

Min	MAX	Mean	Std-Deviation
5.3375	5.7685	5.7037	0.1293

TABLE IV: 3 parties implement 5 multiplications and 3 multiplications of scalars

Min	MAX	Mean	Std-Deviation
5.3063	6.7042	5.8689	0.5547

The tables above show the mean and standard deviation of computation costs when parties implement the additive protocol, the addition of a constant protocol, the scalar multiplication protocol, and the multiplication using the Beaver triplet protocol, respectively.

V. APPLICATION

Assuming that there are three restaurants, A, B, and C, in a shopping mall, and all of them want to know the total revenue of all the restaurants in the mall to evaluate whether their own restaurant can continue to achieve good economic benefits in the next year. However, they do not want to disclose their individual revenue information to each other. Therefore, they decide to use an additive SMC protocol to calculate the total revenue of the three restaurants. We assume the threat model in this scenario is the Honest but Curious setting.

We believe that there is no privacy leakage as the SMC protocol uses additive operations to calculate the total revenue without revealing the individual revenues of each restaurant. This ensures that the privacy of each restaurant's revenue information is preserved.