CS-523 SMCompiler Report

Baccari Mariem, Borges Trivelli Caetano, Gasmi Donia

***Abstract*—We implement a Secure Multiparty Computation (SMC) protocol supporting arithmetic on secret shares, including addition, subtraction, multiplication by constants, and secure multiplication using Beaver triplets. A trusted third party generates these triplets and sends them to a secure server that handles all communication. The protocol operates under an honest-but-curious adversary model. We analyze computation and communication costs under varying numbers of parties and operations. While increasing parties leads to linear growth in both costs, the impact of operation count depends on the operation type. We also explore a medical risk assessment use case, where specialists contribute health scores to compute a risk value without revealing individual inputs through an arithmetic circuit.**

I. Introduction

Secure Multi-Party Computation (SMC) enables multiple parties to compute a function over private inputs without revealing them. It is widely applicable in privacy-critical domains such as healthcare analytics, finance, and secure voting.

In this project, we implemented a distributed SMC framework supporting arithmetic operations via secret sharing and Beaver triplets. A secure server manages communication—publishing and retrieving both private and public messages, including the distribution of shares and triplets. The triplets are generated by an additional trusted third party and forwarded to the server.

We aimed to build a functional prototype and verify its correctness and privacy. We also evaluated how varying parameters—such as party count or number of operations—impact performance.

II. Threat Model

The security of our Secure Multi-Party Computation (SMC) framework is based on a clearly defined threat model. We adopt a **semi-honest (honest-but-curious)** adversary model, where participants follow the protocol correctly but may attempt to infer private inputs from received messages. Privacy is preserved as long as no parties collude.

Assumptions:

1) **Trusted Server:** Fully trusted. It handles all communication, including secret share and Beaver triplet distribution via private and public messages. If compromised, it could reconstruct all secrets.
2) **Trusted Triplet Generator:** An external trusted entity generates Beaver triplets and sends them to the server.
3) **Secure Communication:** Communication is assumed secure against interception (e.g., no man-in-the-middle attacks), as it occurs via the trusted server.
4) **No Collusion:** Clients only access their own shares and do not collaborate to compromise privacy.

Potential Threats:

1) **Passive Eavesdropping:** While out of scope, a network-level observer might infer information. This is mitigated by centralizing all traffic through the trusted server.
2) **Malicious Server (Out-of-Scope):** A compromised server could collect shares, alter triplets, or tamper with computation. Our model does not address this.

Future work could explore stronger models, such as **malicious adversaries** who actively deviate from the protocol.

III. Implementation Details

Our Secure Multi-Party Computation (SMC) framework is designed to perform secure arithmetic computations among multiple participants without revealing individual inputs. The implementation consists of several key components, namely secret sharing mechanisms, expression representations, secure multiplication using Beaver triplets, performance tracking, and testing and validation.

The finite field is implemented as a singleton, as it is common to all parties. All operations are performed modulo 100000000003, our chosen prime number.

To communicate secret shares, we use the hash of each secret's ID as a label and send it as a private message to the intended client. The results of the computations are published as public messages under the label "result." All communication is routed through the trusted server, which ensures message delivery and central coordination.

IV. Performance Evaluation

We evaluate our SMC framework by measuring **computation time** and **communication cost** under varying circuit parameters.

A. Evaluation Setup

Experiments were conducted on the provided VM running on a student laptop with an Intel Core i7 CPU (4 cores, 4GB RAM). All parties were executed in parallel using Python's `multiprocessing` library. While this setup simulates concurrent execution, communication occurs locally and does not reflect real-world latency or bandwidth limitations.

B. Measured Parameters

We varied the following:

- **Number of Parties:** 3, 5, 7, 10, 20, 40.
- **Number of Operations:** 10, 30, 70, 100.
  Operation Type: Additions, Multiplications, Scalar Additions, Scalar Multiplications.

Each measurement includes both:

- **Computation Time:** Wall-clock time from the beginning to the end of the protocol.
- **Communication Cost:** Total bytes sent and received per party (sum of all incoming and outgoing traffic).

C. Environment Sensitivity

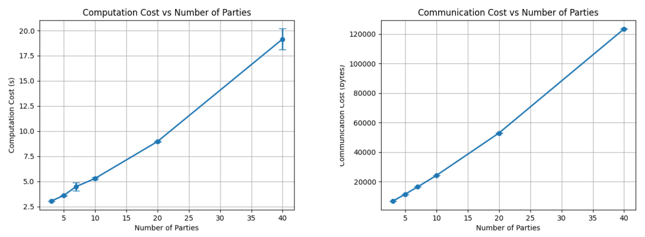
Since all parties run on the same physical machine, results are influenced by system load, CPU scheduling, and memory usage. As the number of operations increases, the system may experience bottlenecks leading to irregular results.
In a real distributed setup, network characteristics would play a larger role in communication cost.

D. Measurement Repeatability

To ensure statistical reliability, **each experiment was repeated five times**. We report both the mean and the standard deviation. These repetitions help reduce the impact of transient system noise or scheduling effects.

E. Results

Varying the Number of Parties:



(a) Computation Time (b) Communication Cost

Fig. 1: Effect of Number of Parties on (a) Computation Time and (b) Communication Cost

Varying Number of Operations:

Num of Operations	10	30	70	100
Addition	2.42±0.06	3.08±0.02	4.19±0.04	5.07±0.01
Multiplication	3.77±0.08	7.91±0.27	20.72±5.66	32.42±10.34
Scalar Addition	2.45±0.05	3.04±0.09	4.32±0.06	5.45±0.13
Scalar Multiplication	6.09±1.95	13.32±0.67	28.78±13.19	22.27±0.59

TABLE I: Mean **computation time** with standard deviation (s) for each operation type and number of operations.

Num of Operations	10	30	70	100
Addition	2841±8	7221±10	15974±13	22544±17
Multiplication	16878±15	52408±23	123417±26	176762±62
Scalar Addition	2846±5	7220±6	15980±12	22540±16
Scalar Multiplication	16856±23	52391±23	123460±58	176756±63

TABLE II: Mean **communication cost** with standard deviation (bytes) for each operation type and number of operations.

V. Application

Our custom application, implemented in `test_custom_app.py`, simulates a medical scenario where different healthcare professionals securely contribute scores related to a patient's health. The goal is to collaboratively compute a disease risk score without revealing individual inputs. This can enhance privacy in cases involving multiple institutions or specialists.

We arbitrarily define the following risk formula:

$$c = 5A + 2BC - D + K \tag{1}$$

where:

- $K = 50$ is the patient's known age (public constant),
- Doctor A (cardiologist) contributes A = Blood Pressure score,
- Doctor B (pulmonologist) contributes B = Smoking History score,
- Doctor C (endocrinologist) contributes C = Diabetes score,
- Doctor D (family doctor) contributes D = Healthy Lifestyle score.

The protocol assumes a semi-honest adversarial model, which is realistic given the professional setting. Professionals are assumed to follow the protocol, but might be curious to infer private data.

Privacy Risks and Mitigations

If an adversary repeatedly runs the protocol with varied inputs, some leakage is possible. For instance, if Doctor B sets $B = 0$ and then $B = 1$ while keeping other inputs unchanged, they may infer Doctor C's value based on the change in output. Similar inference strategies could apply to other multiplicative terms.

To mitigate this, we can:

- Limit the number of allowed queries per participant,
- Introduce differential privacy mechanisms to obfuscate results slightly,
- Restrict output visibility to a trusted aggregator (e.g., a central authority or patient).

However, assuming each party only knows their own input and observes only the final output once, no privacy leakage occurs. The risk score is a single aggregated result that cannot be reverse-engineered to reveal individual values.

VI. Individual Contributions

Caetano and Mariem implemented the arithmetic operations and custom application. Donia handled performance evaluation.