CS-523 SMCompiler Report

Michael Gysel, Valentim Romão

***Abstract*—There are numerous occasions when one needs to compute expressions that require sensitive information. Secure Multiparty Computation (SMC) can be used to accomplish this, allowing the computation of common expressions using secrets, without needing to trust these secrets with a third party.**

I. Introduction

This project implements a SMC framework in Python3, allowing participants to compute expressions with secrets without exposing these secrets to the other participiants. This framework handles 6 operations: secrets addition, subtraction, and multiplication and scalar addition, subtraction, and multiplication. The secrets multiplication requires a trusted third party to generate auxiliary information used for the multiplication of shares.

II. Threat model

The threat model focused on the following two privacy leakages: participants trying to discover other participants' secrets and participants publishing computation results. The first privacy leakage is more likely to be harmful to the participant with the secret exposed.

The threat model should protect user privacy against other participants in the protocol. Even a global passive adversary could not reconstruct the secret without the help of a participant, since this would require at least one secret share created from the participant's own secret. In this case, the participant could just provide the final output. Even if this happened, numerous secret value combinations could lead to the same result. Thus, this would not expose participant secrets unless the output value is an edge case.

The threat model also considers active adversaries. Because each participant has access to the final result of the computation, participants can collude, ultimately inferring another participants' secret. More precisely, in a scenario where there are N participants (N secrets), N-1 colluding participants can discover the remaining participant's secret. In this case, the colluding participants know N-1 secrets and the final value, leaving the remaining participant's secret as the only other variable to solve for.

III. Implementation details

For our implementation, we interpreted the expressions as abstract syntax trees (ASTs), where the operations are nodes and the secrets or scalars are leaves. Next, we use the Additive Secret Sharing scheme to share each of the participants' secrets, sending each share to the other respective participant. We use Beaver Triples for the multiplication operation. This requires a trusted third party to generate and distribute them to each participant so that each participant can obtain a correct multiplication share.

After this initial step, we process expressions according to each operation, using secret shares in place of the original secrets in the expression. When the participants finish this computation, they publish the result obtained for all participants. Finally, each participant retrieves all final shares and uses them to reconstruct the value, as if they replaced the expression secrets by the secret values.

We compute these expressions using the finite field of integers modulo $2^{17} - 1$. We chose $2^{17} - 1$ because it is sufficiently large to prevent multiplication wraparounds for the purposes of our application.

IV. Performance evaluation

To conduct the performance evaluation, a Macbook Pro was used with a 2.2GHz Intel Core i7 processor, 16GB 1600 MHz memory, and macOS High Sierra version 10.13.16.

Secret sharing, reconstructing secrets, addition of secrets and scalars, and multiplication of secrets and scalars were evaluated in terms of computation costs. To evaluate computation costs, 100 trials each were conducted for testing the secret sharing, secret reconstruction, and different addition and multiplication circuits.

For secret sharing and reconstruction, 10, 100, 500, and 1,000 shares were tested. As can be seen in Table 1, the mean and standard deviation of the computation cost of sharing and reconstructing secrets increased linearly as the number of shares increased. Furthermore, sharing the secret was significantly more computationally intensive than reconstructing the secret.

Computation	Shares	Mean (μs)	Standard Deviation (μs)
share secret	10	18.17	6.82
	100	187.39	57.60
	500	1,195.00	1,424.05
	1,000	1,919.21	1,401.48
reconstruct secret	10	1.42	0.99
	100	8.97	2.38
	500	49.77	9.47
	1,000	94.52	19.87

TABLE I: Computation Cost of Secret Sharing

The addition and multiplication circuits were tested using 10, 100, 500, and 1,000 additions and multiplications for both secrets and scalars. Per Table 2, the computation costs of both secrets and scalars for both addition and multiplication were relatively similar. These costs increased linearly with the number of operations and linearly with the number of users. The standard deviation however, increased at a much higher

rate, with particularly large standard deviations when using 1,000 operations.

To evaluate the communication costs, 100 trials each were conducted for addition and multiplication circuits. More specifically, 10, 25, and 50 addition and multiplication operations were tested for both secrets and scalars. The total number of Bytes sent and received by users was then recorded.

As can be seen in Table 3, the communication cost of the addition and multiplication operations using secrets increased exponentially as the number of operations or users increased. The standard deviation on the other hand, increased relatively linearly. The communication costs of the addition and multiplication circuits increased linearly with the number of operations and all had standard deviations of 0, as expected. Thus, as the number of secrets operations increases, the system burdens the user.

Computation	Operations	Mean (μs)	SD (μs)
Addition (Secret)	10	8.29	2.50
	100	80.24	23.01
	500	409.67	155.57
	1,000	959.60	1,131.13
Addition (Scalar)	10	6.91	2.36
	100	65.41	13.61
	500	460.90	1,162.02
	1,000	821.41	973.35
Multiplication (Secret)	10	8.39	3.67
	100	67.19	7.79
	500	561.61	1,071.89
	1,000	919.95	1,132.03
Multiplication (Scalar)	10	6.72	2.10
	100	57.73	7.29
	500	579.23	221.94
	1,000	800.99	992.40

TABLE II: Computation Cost of Arithmetic Circuits

Computation	Type	Operations	Mean (kB)	SD (B)
Addition (Secret)	Sent	10	12.28	5.94
		25	70.81	14.51
		50	275.50	22.66
Addition (Secret)	Received	10	12.36	15.40
		25	73.41	86.58
		50	288.33	206.23
Addition (Scalar)	Sent	10	0.55	0.00
		25	1.30	0.00
		50	2.55	0.00
Addition (Scalar)	Received	10	0.375	0.00
		25	22.15	0.00
		50	86.85	0.00
Addition (Secret)	Sent	10	2,553	2.50
		25	80.24	23.01
		50	409.67	155.57
Addition (Secret)	Received	10	8.29	2.50
		25	80.24	23.01
		50	409.67	155.57
Multiplication (Scalar)	Sent	10	8.29	2.50
		25	80.24	23.01
		50	409.67	155.57
Multiplication (Scalar)	Received	10	8.29	2.50
		25	80.24	23.01
		50	409.67	155.57

TABLE III: Communication Cost of Arithmetic Circuits

V. Application

Academic courses often have 4 member group projects, where students are given individual and group grades. Also to note, TAs are responsible for monitoring the contribution of each member, giving them an additional supervision grade. All group members must know the group grade but want to keep their grade secret. The grade is scaled from 0-100 to 0-200 and a 10 point bonus is added to compensate for some individual members under-performing. Therefore, the adversary model is a student or TA who wants to learn the secrets of other students in order to discover their grade. SMC is needed because neither the students nor the TA want others to know the grade they received. Using SMC, the group project grade can be computed without each member exposing their individual grade. This computation can also be useful for grading an exam and other scenarios with similar grading schemes, where only minor adjustments in the parameters would be required.

For this use case, we developed the following circuit:

$$f(g_1, g_2, g_3, g_4, s_1, s_2, s_3, s_4) = \\ (((g_1 * s_1 + g_2 * s_2 + g_3 * s_3 + g_4 * s_4) * Scalar(2)) + Scalar(10))$$

where g_i would represent the grade of the student i and s_i would represent the supervision grade given by the TA to student i. Both grades are scaled from 0 to 5 inclusively, and only having integer values.

Although SMC allows participants to compute expressions without knowing all values, each participant has access to the final computation. From this, they can infer some information about other participants' secrets.

For example, if the output was 10 (minimum grade), the students and TA can infer the group project grade was 0. In this case, a student who knows their grade was not 0 now knows their supervision grade was 0. Secondly, this student could observe that other group members either received 0 on their individual grade or their supervision grade. If the TA knows that no supervision grade was 0, they can infer that all students received 0 as their individual grade.

A second example is if the final grade is the maximum value of 210. In this case, the participants know the group grade was 100 and infer all the secrets are 5, a requirement for this maximum grade. In the case where the final grade is 200, the possibilities grow considerably. For example, if the final grade is 200, each student could have a supervision grade of 5 with one having a supervision grade of 4, or vice versa. In this case, there is still some privacy leakage as the participant with the grade of 4, would know that the grades of all the other students are 5.

To conclude, more privacy is achieved from lower grades than from higher grades, due to the secrets multiplication, resulting in a higher larger number of possibilities. Also, the application contains the previously mentioned privacy leakages as a consequence of SMC allowing all participants to view the final output. One possible mitigation is to add noise to the final output, as opposed to the bonus to increase the uncertainty of the secrets, although this would lower the utility of the system.