CS523 Project 1 Report

Malo Perez, Maxime Sierro

I. Introduction

The project's goal is to implement and apply a secure multi-party computation engine in a passive adversarial setting with a trusted third party. The framework should support generic arithmetic circuits and use additive secret sharing.

II. Threat model

We assume adversaries are semi-honest (passive), meaning they can cooperate to try to gather information but otherwise "follow the rules" : every client will behave as expected when participating in the computation of the output. In this threat model, our only concern is to prevent the leakage of any information on a fully-honest participant's secret no matter how many other semi-honest participants may be cooperating. Note that the case where all participants are passive adversaries and cooperate is trivial and thus not considered.

III. Implementation details

Here are some notable implementation choices we have made:

- We always use the same prime number in our ring, and we fixed it to 6827, which we considered big enough. One possible extension of this implementation would be to draw a different prime number for every circuit.
- For operations where only one of the parties has to do an action (for example scalar addition or secret multiplication), we chose the party that would do the additional action in a deterministic way. Since all parties have access to the list of participants, the party with the lowest id is selected. Making this more balanced, with one different party being randomly selected for each operation requiring a single party action in the circuit, could be an interesting extension, and could possibly decrease the variance in evaluation time.
- For simplicity, a party shares and retrieves from itself the shares of its own secrets.
- The beaver triplets are generated for each secret multiplication operation, at the moment when the first party tries to get its shares. They are stored in a double dictionary keeping track of the triplet's shares for each operation and each client.

IV. Performance evaluation

We constructed several experimental set-ups in order to evaluate some of the parameter's influence on the performance. Across all set-ups, we have measured the time taken to evaluate successfully one circuit for all participating and third-parties clients, which we repeated a total of 10 times for value of the parameter we are studying. For each value, we have computed the mean and standard deviation of the time taken across all 10 runs and across all participants.

The first performance evaluation is done to observe the effects of the number of participants on the performance. To evaluate this, we have created a simple fixed circuit: the addition of 100 secrets, that would all be distributed equally among the different participants (for example 2 secrets for each party in the case of 50 participants). This was done for 2, 10, 25, 50 and 100 participants. For each one of these values, the circuit is run 10 times, resulting in the mean across all run and participants.

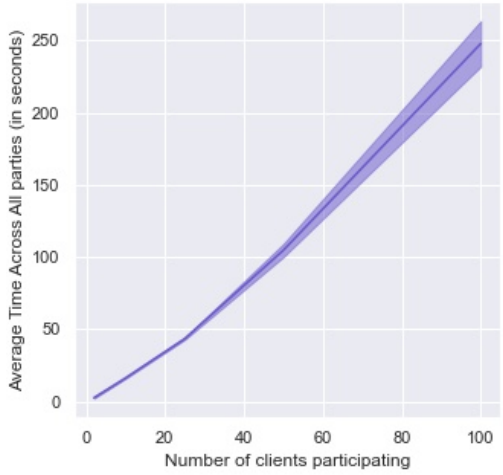

Fig. 1: Average time taken to evaluate the circuit (in seconds) depending on the number of participants (for a fixed circuit)

.

Fig. 1 shows us that the the evaluation time grows linearly with the number of participants. However, the standard deviation (depicted as the margin around the curve) does get bigger with the number of participants, when it's very small for a low number of participants.

The next set of experiments was to measure the effect of the number of operations on the performance for four types of operation: secret addition and multiplication and scalar addition and multiplication. For each of these 4 experiments, we have fixed 4 participants. For the two cases where we study the effect of the number of secret operations, we generate 4, 20, 100, 500 and 1000 secrets which we distribute equally between each participant, and the circuit being the addition/multiplication of all of those secrets. For the two

cases where we study the effect of the number of scalar operations, each participants has one secret and we generate 4, 20, 100, 500 and 1000 random scalars, with the circuit being the addition/multiplication of all of the scalars (and the 4 secrets from each party). Once again for each value, we run the circuit 10 times, and measure the time for each party before computing the mean and standard value.

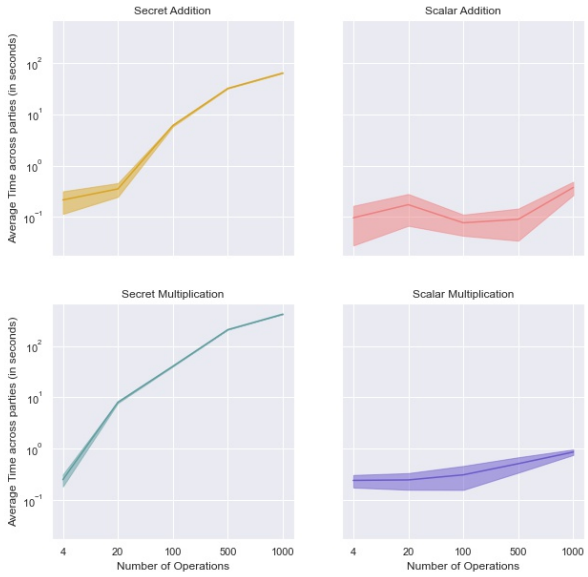

Fig. 2: Average time taken to evaluate the circuit (in seconds on log scale) depending on the number of operations(for a fixed circuit), for 4 operations types

.

The results show us that evaluation time (in seconds and on a log scale) has an almost exponential growth with the number of secret addition and multiplication operations, with a rather small standard deviation. However the effect of the number of scalar operations, whether addition or multiplication, on the evaluation is minimal. It's interesting to note that the standard deviation is larger than for secret operation. In this case of scalar addition this could be explained by the fact that only one party has to do an additional addition, and since our implementation chooses this party in a deterministic way, the same participant does all of these additional computations.

V. Application

We present a novel betting game played between 3 players who do not wish to rely on a casino as an external trusted party, and whose special rules justify a complex circuit containing every operation. Players A and B both have a random secret integer, which serves as an input to the computation of their own future final score. Both bet on their own victory, determined by the highest final score out of the 2. The twist is that player C influences both final score computations and wins instead if they are close enough. Thus, each player has a different goal : A and B both want to convincingly beat each other while C wants their "duel" to be as close to a draw as possible. The players engage in Poker-style betting (they can fold) and the SMC circuit is used mid-match to provide to everyone the noisy difference in the 2 final scores without revealing the secrets, which makes the next round of bets more strategic.

We assume that every player has access to a random source of integers whose result cannot be modified or denied. Players A and B will use theirs to generate their initial secret integers (whose range should be defined in the rules) while player C will use his to produce the needed integer noise of 0-mean. The final score of both A and B is computed as a simple multiplication of their secret integer with a multiplier (whose range should be defined) chosen by C for each of them. The players engage in a first round of bets. Players A and B have their own secret as their sole information while player C knows nothing. Once the bets are settled, player C tries to guess who has the better integer from the recent betting style of his opponents and chooses the 2 multipliers accordingly, aiming for a draw. The circuit output, defined as follows : $output = (secretA * multiplier1C) - (secretB * multiplier2C) + noiseC$, is published and the 3 players engage in the next round of bets with this new information. Finally, the game can end and the secrets are revealed thanks to the required assumption on the random sources. A unique winner is selected from those who have not folded, depending on the 2 final scores. Note that "public" scalar additions/multiplications could be added to the final score computation (and thus to the circuit as well) to introduce handicaps to better performing players. The integer ring must be large enough so that large outputs may safely be considered as negative (indicating that the player who is more likely to win is player B instead of player A).

There is "privacy leakage" : players A and B know that some values of their opponent's secret are likelier than others thanks to the circuit output. Player C, knowing the noise value, downright knows if he lost or won for the second round of bets (and this should be taken into account when balancing the game). However, in this particular application, it is by design and the participants willingly accept this as it adds another layer of strategy to their game. The real issue is preventing 2 semi-honest players from secretly teaming up. In this context, this should only be an issue if the game is poorly balanced. For example, if player C's role is disadvantageous due to the rules (the "draw" score difference range might be too small), he might give up on winning personally, use his multipliers to help Player A win, which then split the winnings with him afterwards. The best mitigation is thus to balance the game as much as possible, so that the best way for each participant to earn money is to try to win by themselves with no cooperation.