אוניברסיטת בן-גוריון בנגב
Ben-Gurion University of the Negev

Practical Fully Secure Three-Party Computation via Sub-linear Distributed Zero-Knowledge Proofs

Supervisor:
Niv Gilboa

Vitali Lopushenko
317104347

Osher saragani
315348706

30.12.2019

Communication Systems Engineering
Ben Gurion University of the Negev

Contents

List of Figures

1 Introduction

1.1 Abstract

Secure Multi-Party Computation (MPC) is a subfield in information security, the goal of the group is to calculate a certain function by collecting one input from each party while keeping their input private. The protocol which we are going to implement does not require a server or an access point and is fully distributed. Furthermore, the protocol minimizes the communication overhead by aspiring to do most of the computations locally with reasonable computation complexity. The protocol maintains fairness and security in a sense that a semi-honest party or malicious adversary, cannot manipulate the function output nor learn the output of the other parties as long as there is an honest majority.

1.2 Motivation

Traditional MPC is an asynchronous model, consider a simple setting with n parties that desire computes a function with their input without revealing it. The n parties send their inputs to some trusted party p and wait for an answer from p, the party p receives all inputs, calculates function f and sends the output to all parties. An Asynchronous MPC model achieves the same result without trusted party p by using the secret-sharing protocol to share the secret inputs and calculate distributedly the function f, for further reading [4]. 2PC formally introduced in the early 80's in the millionaire's problem where two millionaires wish to find out which one of them has more money but neither of them wanted to reveal their exact amount of cash each of them possesses. Furthermore, both of them agreed that the involvement of a third party was not desirable. Later the need for a general solution has risen, s.t. n parties wish to calculate some function f. Formally,The setting of secure multi-party computation consists of n parties with their private inputs $[x_1, \ldots, x_n]$ (where x_i is the private input of party i) that wish to jointly compute the functions $f_i(x_1, \ldots, x_n)$, where player i receives the output $f_i(x_1, \ldots, x_n)$. If there were a trusted party, then all parties could have sent their inputs to it, and the trusted party could have privately sent each $f_i(x_1, \ldots, x_n)$ to player i. In this setting it is clear that player i learns nothing but its designated output. its very efficient with two parties, however, as the number of parties increases the communication overhead

increases as well. Additionally, the communication cost greatly relies on channel performance and can expose the party to side-channel-attacks. 3PC topology has been growing in popularity in the last few years and simple enough to analyze.Moreover, it is the minimal size which is needed for an honest majority.

1.3 secure computation

We will focus on protocols that provide security against semi-honest parties controlled by an adversary. this type of adversary is assumed to follow the instructions that are prescribed for him by the protocol. However, he may try to learn additional information from the messages that his parties receive, In our case, the secret inputs of each party. A stronger type of corruption is performed by adversary who is malicious. That adversary can operate in any way he chooses and usually will not obey the protocol.

1.4 Our contribution

In this paper, we present an MPC protocol which minimizes the communication cost in expense of reasonable amount of local computations. The protocol is exactly the protocol described in [1]. We are going to present a full implementation of the protocol with a verification phase included, in which each party make sure that every member was honest throughout the execution of the protocol. Moreover, we wish to run different ways of implementation, analyze our results and achieve a better understanding of using the protocol in practice. We hope that our work will be used in the future and help provide practical results which push the work already done ahead.

2 The project and preliminaries

Our project is implementing the whole system described in the article by Niv Gilboa, Elette Boyle, Yuval Ishai and Ariel Nof in paper Practical Fully Secure Three-Party Computation [1].
In our setting of Three parties, $p = (p_1, p_2, p_3)$, connected by TCP connection. We assume an honest majority, in our settings e.g. one malicious party. The function f to be computed by the parties is a public circuit C that composed of addition and multiplication gates over F a finite field over the ring $\mathbb{Z}_{2^k}$ - the ring of integers modulo 2^k. $[n]$ is the set $[0 \dots n]$, $[[u]]$ is the part of u that party i holds $[[u]] = (u_i, u_{(i-1)})$.

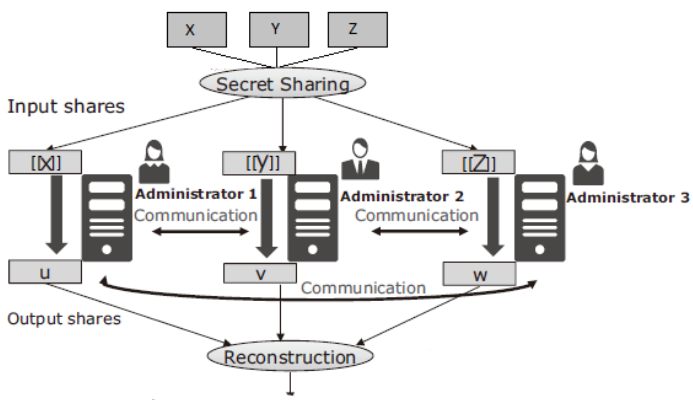

Figure 1: The Model

2.1 definitions and concepts

MPC- Multi-Party Communication.
Semi-honest MPC: A malicious party that tries to learn as much as he can about the protocol, even things he does not suppose to know in order to participate in the protocol. However, the party follows the protocol and function as you would expect from an honest party.

Malicious secure MPC: As long as there is an honest majority, the protocol guarantees that no malicious party can manipulate the system into false output.
Replicated secret sharing: In generally replicated secret sharing isn't efficient, however, in a small number of participants, it seems to perform very well. the replicated secret sharing scheme used in our protocol is optimized for 3 parties.
To share a secret v over the ring $\mathbb{R}$, The dealer needs three random values it's done by choosing two random values v_1, v_2 and compute by $v_3 = v - v_1 - v_2$. then the dealer construct shares for each party so that p_1s share is (v_1, v_3), p_2 shares is (v_2, v_1) and v_3 shares (v_3, v_2) in genarally p_i share is (v_i, v_{i-1}).

2.2 Parts of the project

2.2.1 secret sharing

As mentioned above, there is no need for a trusted third party which receives all the inputs, calculates the function and sends out the result to all parties, the computation is distributed between 3 parties that work synchronously in parallel with a reasonable amount of communication as described in [2]. The first step is to share the secret input of each party with other parties. $\mathcal{F}_{rand}$ - This functionality allows the generation of a random value(a member of the ring) and divide this value into shares, one for each party in the following manner. Firstly, Each party i generates a random key k_i send it to p_{i+1} and calculate $\mathcal{F}_{k_i}(id)$ to be its random element Alpha i. $\mathcal{F}_{k_i}()$ is a deterministic function which takes a counter that was agreed upon earlier. That way the random element α is divided into shares: α_1, α_2, α_3. Party p_i holds α_i and $\alpha_(i-1)$.
$\mathcal{F}_{input}$ - this functionality uses $\mathcal{F}_{rand}$ to generate a random value and divide it into shares as explained above. Then each party computes its secret input X_i minus α and broadcast it to all other parties. That way each party holds $[[X_i - \alpha]]$ shares. Each party has its shares of Alpha e.g $[[\alpha]]$ so he can compute his share of X_i by: $[[X_i - \alpha]] + [[\alpha]] = [[X_i]]$.
After each party receives its shares and rebuilt it to Three shares: (v_1, v_3), $(v_2, v_1), (v_3, v_2)$. Each party holds a different set of shares. That way, no party could learn about the output of the other parties and vice versa. this functionality fully described in [3].

2.2.2 The Calculation of C circuit

Every function can be broken down into its Boolean components. That is a Boolean function that uses XOR ($\bigoplus$) gates for addition and AND ($\cdot$) gate for multiplication.
In each round, either an addition gate or a multiplication gate is calculated by each party with its shares of the inputs. For example, in order to compute $u + v$ each party p_i sets its share to be $(u_i + v_i, u_{i-1} + v_{i-1})$ where the share of $[[u]] = (u_i, u_{i-1})$ and the share of $[[v]] = (v_i, v_{i-1})$. For the addition gates, there is no communication required whatsoever. However, for the multiplication gate, there is some communication required. After all the members finish to calculate the desired function, another phase is needed in the semi-honest model. That step is called the verification phase and is conducted distributedly as well. This extra step's goal is to make sure that every party did not lie on one or more of the communication steps that were carried out during the multiplication gates.In a case where everyone was honest every party output the function result as a success. In a case where the last phase discovered conflict, that is someone lied, every party outputs an abort signal.

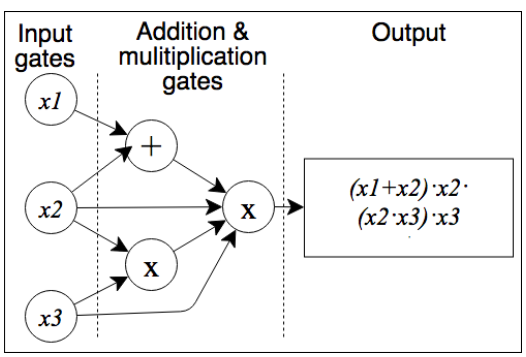

Figure 2: Function and its Boolean components

2.2.3 Verification

Verifying Correctness of Messages, the goal of this function is to verify that all messages that were sent by the parties during the execution are correct according to the protocol. this function extended the protocol mentioned in [2].
For each multiplication gate party p_i sends one message z_i to p_{i+1}.

$$z_i = u_i * v_i + u_i * v_{i-1} + u_{i-1} * v_i + \alpha_i \tag{1}$$

Let c

$$c(u_i, u_{i-1}, v_i, v_{i-1}, \alpha_i, z_i) = u_i * v_i + u_i * v_{i-1} + u_{i-1} * v_i + \alpha_i - z_i \tag{2}$$

we need to ensure that $c = 0$ on every communication stage of the function $\mathcal{F}_{vrfy}$ receives from each party p_j its inputs $(u_k, u_{k-1}, v_k, v_{k-1}, \alpha_k, z_k)$ for each k in m. For each stage, the input to c is distributed among p_{i+1} and p_{i-1}. specifically, u_i, v_i, α_i and z_i are known to $pi + 1$ and $u_{i-1}, v_{i-1}, \alpha_{i-1}$ known to p_{i-1}. value of α_{i-1} can be calculated since $\alpha_i = -\alpha_{i-1} - \alpha_{i+1}$

2.3 Prior knowledge

Knowledge needed for implementing the project, information security, C++ programming, STL library, principles in program design, work with git as a source control platform. basic knowledge in Boolean algebra, learn about possible security breaches so we can avoid them and write an implementation as secure as possible.

2.4 Tools and Programs we used

Visual Studio IDE: We choose to implement the code in C++, visual studio is a popular IDE for programming.
Draw.io: Its a platform for creation of block diagram.
TeXstudio: - LaTeX development environment, LaTeX is a document preparation platform use for academic and mathematic articles writing.
Git: Is a distributed version-control system for tracking changes in source code during software development.
Exell: For the gant design and time line.

2.5 Flow diagram

The first step in the process is sending the circuit which all of the parties are computing distributedly and may be some other preparation. Then, the parties divide and send their inputs' shares using $\mathcal{F}_{input}$. At the end of this step each party hold 2 shares of each input. In case the input sharing is not successful the protocol aborts. Otherwise, we move to the step of calculating the desired circuit.after calculating the function we move forward into the verification step where each party makes sure that the other parties were honest. In case the answer to this question is no then the protocol aborts or perform reconstruct to get the final output otherwise.

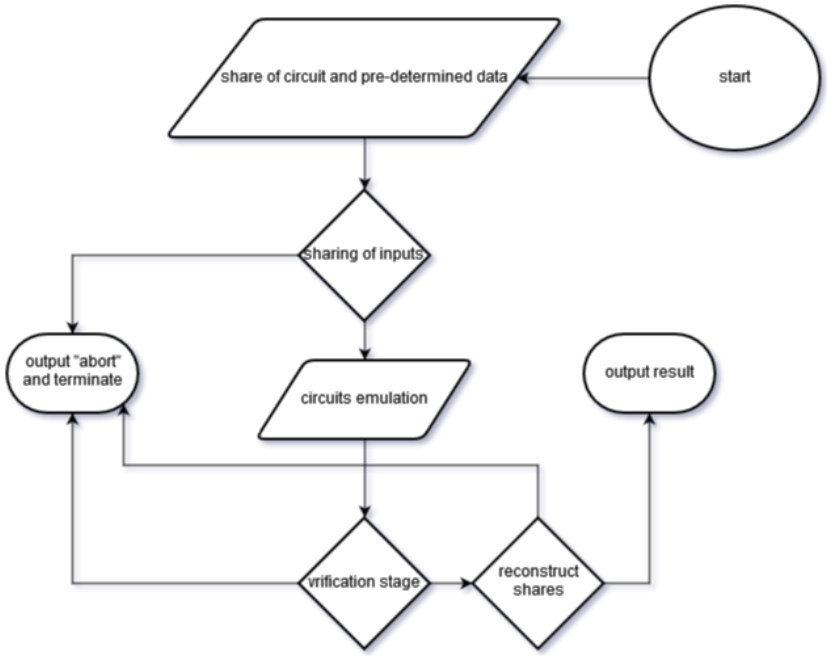

Figure 3: Flow diagram

3 Timeline

3.1 Milestones

We present our timeline and progress in tables on this section.
we set a clear goals for the project for improved further planning and punctuality.

Task Name	Start	End	Duration (days)
research OT extention	01-10-19	20-10-19	19
research homomorphic secret sharing	20-10-19	10-11-19	21
Reaserch fully secure 3MPC	20-11-19	10-12-19	20
Program design draft	10-12-19	16-12-19	6
Prework report	16-12-19	29-12-19	13
Research + Plannir	01-01-20	01-03-20	60
Program design	01-02-20	01-03-20	29
meeting with Yorar	16-12-19	16-12-19	0
meeting with Yorar	23-12-19	23-12-19	0
Programming	01-03-20	01-06-20	92
Fine tuning	01-06-20	30-06-20	29
Poster	01-06-20	15-06-20	14

Figure 4: Milestones

As seen in figure 5 we changed subject of the project, our first research was on extention OT protocol and homomorphic secret sharing.

3.2 Gant

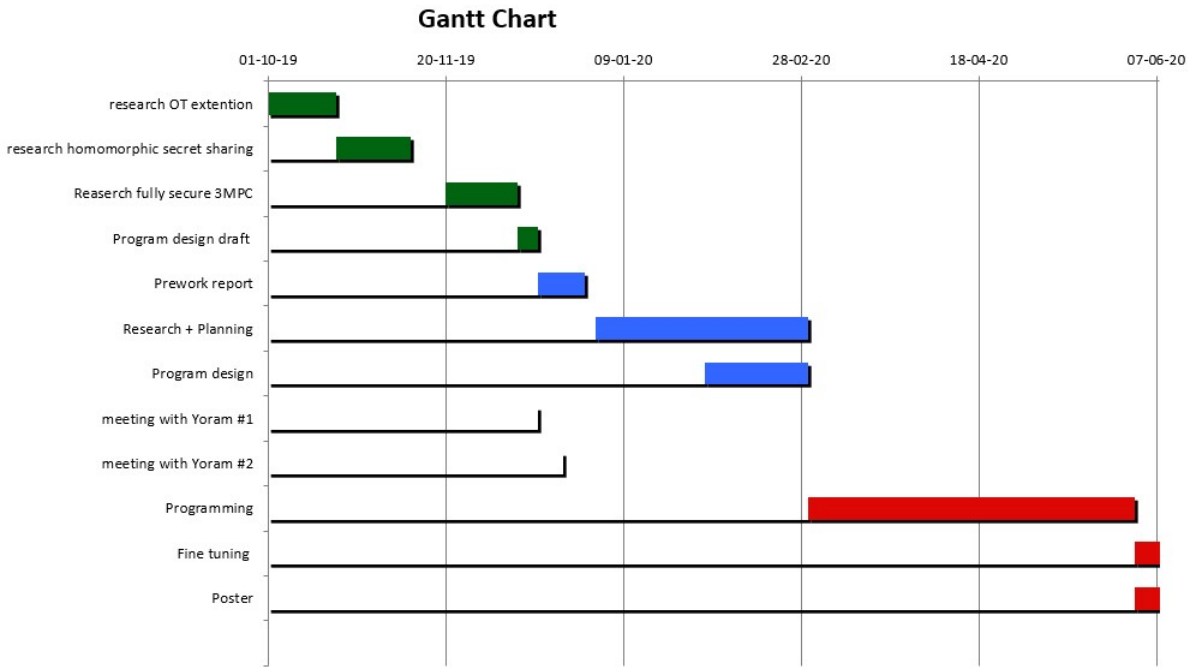

Figure 5: Gant

4 Bibliography

References

[1] Dan Boneh and Elette Boyle and Henry Corrigan-Gibbs and Niv Gilboa and Yuval Ishai *"Practical Fully Secure Three-Party Computation via Sublinear Distributed Zero-Knowledge Proofs."* In Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, pp. 869-886. ACM, 2019.

[2] Boyle, Elette, Niv Gilboa, Yuval Ishai, and Ariel Nof *"Zero-Knowledge Proofs on Secret-Shared Data via Fully Linear PCPs."* Cryptology ePrint Archive, Report 2019/188.

[3] Koji Chida and Daniel Genkin and Koki Hamada and Dai Ikarashi and Ryo Kikuchi and Yehuda Lindell and Ariel Nof *"Fast Large-Scale Honest-Majority MPC for Malicious Adversaries"* Cryptology ePrint Archive, Report 2018/570.

[4] Dani, Varsha, Valerie King, Mahnush Movahedi, Jared Saia, and Mahdi Zamani *"Secure multi-party computation in large networks"* Distributed Computing 30, no. 3 (2017): 193-229.

[5] Elette Boyle and Geoffroy Couteau and Niv Gilboa and Yuval Ishai and Lisa Kohl and Peter Rindal and Peter Scholl *"Efficient Two-Round OT Extension and Silent Non-Interactive Secure Computation"* Cryptology ePrint Archive, Report 2019/1159

[6] Elette Boyle and Geoffroy Couteau and Niv Gilboa and Yuval Ishai and Michele Orrù *"Homomorphic Secret Sharing: Optimizations and Applicationsks"* Cryptology ePrint Archive, Report 2018/419.

[7] Furukawa, Jun and Lindell, Yehuda *"Two-Thirds Honest-Majority MPC for Malicious Adversaries at Almost the Cost of Semi-Honest"* Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security.

[8] Eerikson, Hendrik, Claudio Orlandi, Pille Pullonen, Joonas Puura and Mark Simkin. *"Use your Brain! Arithmetic 3PC For Any Modulus with Active Security"* IACR Cryptology ePrint Archive 2019 (2019): 164.

[9] Furukawa, Jun and Lindell, Yehuda and Nof, Ariel and Weinstein, Or. *"High-Throughput Secure Three-Party Computation for Malicious Adversaries and an Honest Majority"* Advances in Cryptology – EUROCRYPT 2017.