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

Supervisor:
Niv Gilboa

Vitali Lopushenko
317104347

Osher saragani
315348706

24.1.20

Contents

List of Figures

List of Tables

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 The team

The team include two members, Osher saragani and Vitali Lopushenko. We both from Arad, Israel, Educated in Arad high school and alumni of Prep school of Ben Guryon University. We started learning together in department of communication system engineering and took the same courses.

Vitali Lopushenko(28): 4th year student in Ben Guryon University in department of communication system engineering, fellow of Moshal scholarship program. lives in Beer Sheva, Israel with my Loving fiancee Avital. Served 3 years in Israel defense force as head of F-16 aircraft mechanical team, I was responsible for 20 soldiers and 8 F-16 aircraft's. My current position to day is at INT college as a Cyber and Python course lecturer. Love programming in C/C++ and DIY projects with microprocessors such as STM32 and Arduino, passion for cooking and baking.

Osher Saragani(24): 4th year student in Ben Guryon University in the department of communication system engineering.I'm a student-soldier at the Atuda program of the IDF. Additionally, i'm a teacher at Magshimim program in which i teach the course Advanced Programming Principles in

which i teach 11'th grade students. One of the main purposes of the program is preparing the students for the elite programing units of the IDF
relevant courses: introduction for computer science, advanced programming, Micro-Processors(Assembly), Data structures, Operation systems , Embedded systems Lab, Computer Communication Networks, information and data security.

1.5 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.

1.6 MPC Applications

MPC enables privacy-preserving applications where multiple mutually distrusting data owners cooperate to compute a function, simpler definition is, compute joint function without reviling the inputs. Here, we highlight a few examples of privacy-preserving applications that using MPC as mentions in [10]. This list is just few examples, and is meant to give an idea of the range of MPC applications.

Millionaires Problem: The toy problem that was used to introduce secure computation as mentioned above.

Secure auctions: The need for privacy in auctions is understandable. It is crucial for all participants, both bidders and sellers, to be able to rely on the privacy and non-malleability of bids. Bid privacy requires that no player may learn any other player's bid. For the seller, he need to know who is the highest bidder without learning nothing on the bids.

Voting: Secure electronic voting, in a simple form, is simply computation

of the addition function which tallies the vote. Privacy in voting is highly important.

Secure machine learning: MPC can be used to enable privacy in both the inference and training phases of machine learning systems. Oblivious model inference allows a client to submit a request to a server holding a pre-trained model, keeping the request private from the server S and the model private from the client C. In this setting, the inputs to the MPC are the private model from S, and the private test input from C, and the output (decoded only for C) is the model's prediction.

Boston wage equity study: An initiative of the City of Boston and the Boston Women's Workforce Council (BWWC) aims to identify salary inequities. across various employee gender and ethnic demographics at different levels of employment, from executive to entry-level positions.

Key management: One of the biggest problems faced by organizations today is safeguarding sensitive data as it is being used. This is best illustrated using the example of authentication keys. This use case lies at the core of the product o˙ering of Unbound Tech (Unbound Tech, 2018). Unlike other uses of MPC where the goal is to protect data owned by multiple parties from exposure, here the goal is to protect from compromise the data owned by a single entity.

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 $[[u]] = (u_i, u_{(i-1)})$.

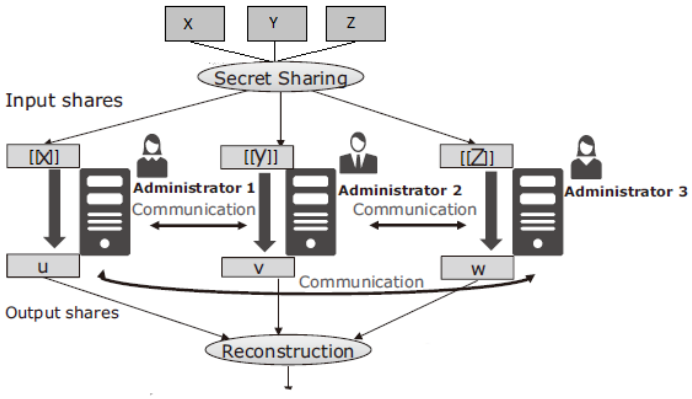

Figure 1: The Model

Figure 1 describe the process of the system, x, y, z are participants with secret input, in the first step all participant make connection to other participants, then they share their secret. the next step is to calculate the joint function with minimal communication, and the final step is to verify legit behavior and reconstruct the output.

2.1 definitions and concepts

KeyWords: MPC, Multi Party Computation, Security, secret sharing, secret input, verify, minimum communication,Replicated secret sharing, malicious

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}).

Loopback: Refers to the routing of digital data streams, or flows of items back to their source without intentional processing or modification. In our case loopback used for communicate through sockets from PC to the same PC using different ports. The advantage of using loopback is no need second(or more) PC for communicate through socket.

Reconstruct: In our case that means reconstruct data from shares. in the first phase in our project, after setting up connectivity is to take apart some secret input to shares and send them to other participants. the reconstruct is the process of getting those part back to valid data.

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 α_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 α_{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.

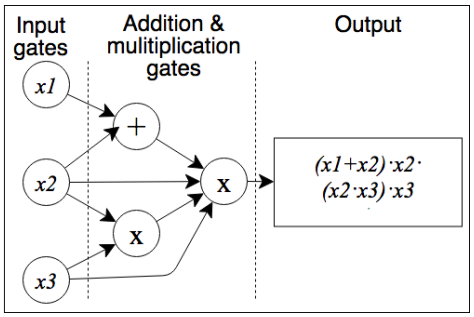

Figure 2: Function and its Boolean components

Figure 2 describe a simple function described with arithmetic gates.

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}.

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

Let c

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

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

Our hardware for developing and running simulation is:
Intel i5 5th generation CPU, 16 GB of RAM, 256GB SSD.

Visual Studio IDE: We choose to implement the code in C++, visual studio is a popular IDE for programming. we using 16.4.4 version of visual studio Community edition 2019.

Draw.io: Its a platform for creation of block diagram, we used an online edition. used to make the report.

TeXstudio: - LaTeX development environment, LaTeX is a document preparation platform use for academic and mathematics articles writing. version : TeXstudio 2.12.14 (git 2.12.14) Using Qt Version 5.8.0, compiled with Qt 5.8.0 R

Git: Is a distributed version-control system for tracking changes in source code during software development.

Exell: For the gant design and timeline.

Wireshark: Wireshark is the world's most popular network protocol analyzer. It is used for troubleshooting, analysis, development and education. version : 2.6.4.

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.

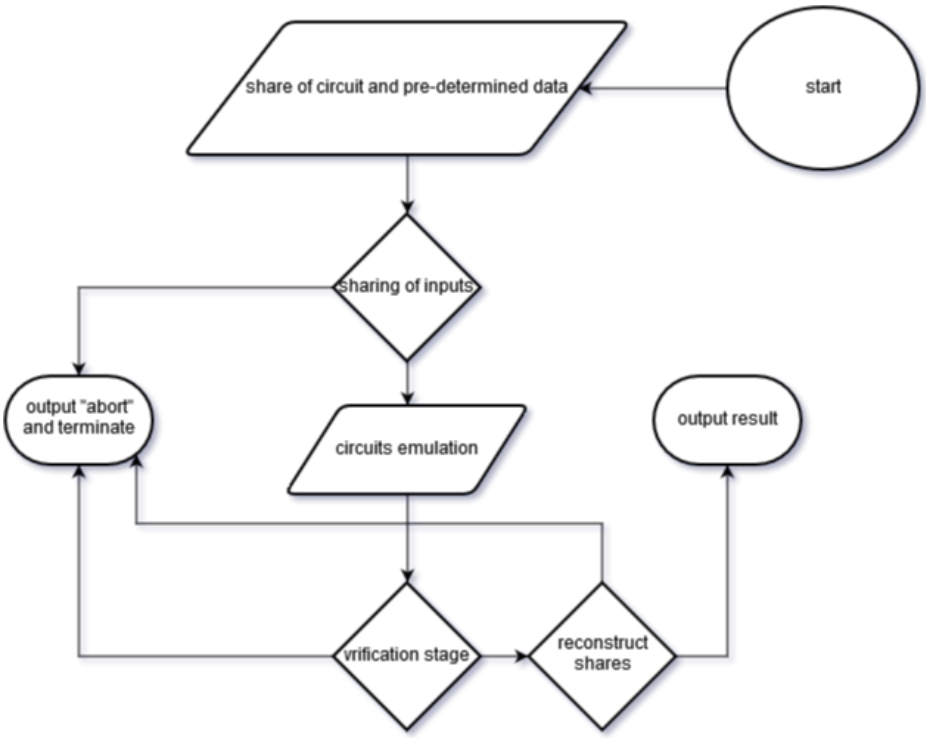

Figure 3: Flow diagram

The flow diagram describe on high level the states of the protocol. each

participant start the protocol with secret input to joint function and set the circuit, 2 more participants doing the same operation. after all participants ready to connect, the connection is made. Then the participants share their secret, and calculate the circuit, verify the behavior of all other participants and reconstruct the output. in each step, if one or more participants not behave by the protocol, abortion function called.

3 Progress

On this section we will describe our progress in time, deliberations and problems that accrued while working.

3.1 Connectivity

This is the first step on the project, the first Class created is Party. This class describes the participants, parameters such as: id, secret input, sockets fd and etc. Then we created the TcpSocket and TcpClient & TcpServer that inheritance from TcpSocket. Each party is a client and a server and hence each party has two sockets to connect other parties.

3.2 Secret shaering

After establishing the connectivity, the next step is to share the secret input. Each party generate a random number seq_i and broadcast to other parties. after receiving two random numbers seq_{i+1}, seq_{i-1} , party p_i calculate the pseudo random number $seq = seq_{i+1} + seq_{i-1} + seq_i$. Now each party generate another random number k_i and send it to party p_{i+1}. party p_i has its own value of k_i and received value k_{i-1}, those values are keys to AES_256 hash function. Eatch party calculate $\alpha_i = AES_{k_i}(seq)$ and $\alpha_{i-1} = AES_{k_{i-1}}(seq)$

3.3 Deliberations

While working on the project we have a few important deliberations:

1. **How to make the connectivity between 3 parties:**
   In our project there is 3 pc's that participate the calculations, its inconvenient to run the program each time on 3 different pc's. for the developing stage we used the loop back option to run the code on single pc. the connection is made with sockets on different ports.

2. **Threading:** We thought how to make our code more efficient, without any unnecessary busy waiting. We choose to implement thread for each socket.

3. **Random numbers** One way to generate random numbers is using rand() function that built in C++, However this function isn't cryptographically secure. We decided to use open ssl library because its Recommended by many users and articles.

3.4 Problems

While working on the project we assume that we will face with few problems. in heads-up section we will discuss few problems that we assume to deal with, and another section that describe actual problems that we solved.

3.4.1 Heads-up

1. Lack of understanding the protocols : our biggest fear is to think that we understand some function, implementing and then realize that something wrong. the solution is simple, in case of critical functionality in our project we will ask our project supervisor to verify our understanding.

2. Failure to meet deadlines : The presentation date of the project is set, and we need to be ready to display our presentation to this date. to avoid failure to meet times we set the timeline gant. the gant describes the progress in time and indicate in witch stage we suppose to be.

3. losing data: the scenario of losing data is harsh, hard work that go for nothing, that why we backup our project on Google Drive and GitHub to ensure that even if we lose our data on one platform there will be another backup.

3.4.2 Problems we faced with

1. At first we've had opened a socket but we didn't manage to send data.The first thing we checked was the return value of the read() function. The function returns the number of bytes that was successfully read from the socket or -1 if the operation failed. while setting up connectivity and preformming simulations the return value we got from the function was -1. The read() function is a blocking function by default,that is, the program stop the flow of the program and waits until information is read from the socket. In our case the function

didn't block, we've tried to figure out how come the function is in non-blocking mode for a while but no explanation was found.another important detail is that we were debugging the program using the loop back(ip = 127.0.0.1).At this point we decided to use the wireshark in order to determine whether the problem is at the sender or the receiver, however surprisingly, the wireshark didn't show any packects with ip=127.0.0.1. After a quick search at google we found out that packets which are transmitted via the loopback doen't pass through the network card of the computer, thus, can't be represented via the wireshark. We tried to find a solution and then Osher remembered that there is a tool which can sniff the packets being sent through the loopback with a program called RawCap which he'd used in Magshimim.we ran RawCap and found out that the packet was sent successfully but the receiver which was the server in the connection did not get it.So we figured that the problem was in the server an d not in the client as we originally thought. After a quick overview of the server code we found out the the server was trying to get the message via the welcome socket and not the new socket which was received from the accept function.After that the fix was very simple

4 Code Design

On this section we will describe the classes and their functions.

4.1 Party

this class describes each party.
Variables:

1. short id :unique identifier, lowest ip receive id = 1.
2. long input : secret input
3. vector <pair<long, long>> shares : index of vector is the id of input's party

Functions:

1. void connectToAllParties() : this function is responsible o setting connectivity between al parties. Each party p_i has two TCP connections to party $p_{(i+1)}$ and party $p_{(i-1)}$

4.2 TcpSocket:

the TcpSocket is base class for TcpServer & TcpClient.
Variables:

1. int socket : fd of the socket.
2. int port : port number
3. string hostAddress : IP address of the host.

Functions:

1. int id(void) : Return the socket descriptor.
2. int port(void) : Return the socket port number.
3. int writeBuffer(const void* buffer, long bufferSize, int flags = NULL) : Write a buffer over the socket connection. Returns the number of bytes written or -1 if an error occurs.

4. int readBuffer(void* buffer, long bufferSize, int flags = NULL) : Read an input buffer, up to length bufferSize. Returns the number of bytes read or -1 if an error occurs.

5. bool isValid() : Returns true if the socket descriptor is valid.

6. void close() : close all connections.

4.3 TcpClient:

This class is derived from TcpSocket.
Variables: No extra variables
Functions:

1. int connect(int port, std::string hostname = "localhost") : connect to some server

4.4 TcpServer:

This class is derived from TcpSocket.
Variables: No extra variables
Functions:

1. void serve() : bind + listen + accept. throws an exception in case of failure.

2. void accept(void) : accept a new connection

4.5 Library used

1. **WinSock2:** Winsock provides the programming interface for applications to communicate using popular network protocols such as TCP & UDP and provide a large set of network functions

2. **OpenSSL** its an open source library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. downloaded from *https* : *//www.openssl.org/*

5 Timeline

5.1 Milestones

Milestone planning is one of the most important aspects of project planning, because project milestones are the most visible indicators of project progress. Milestones typically mark critical decision points, the completion of major project tasks and the ends of various project phases.
As seen in Millstones table - table: 1 we changed the subject of our project. First research was on OT extension protocol, then on homomorphic secret sharing and finally fully secure 3MPC. we chose to change our project subject because OT extension and hommomorfic secret sharing are more research than coding. This decision caused delay in progress but increased our motivation.

Table 1: Millstones

	Task name	Meet time	Work days	Outcome
1	Research OT extension	20/10/19	14	knowlage
2	Research Hommomorphic secret sharing	05/11/19	14	knowlage
3	Research fully secure 3MPC	10/12/19	14	knowlage
4	Prework report	29/12/19	5	Prework for the progress report
5	Code design for Connectivity	10/1/20	3	Code design
6	Implementing Connectivity	25/1/20	5	Connectivity between 3 PC's
7	Progress report	26/1/20	7	Progress and research report
8	Code design for secret sharing	3/2/20	2	Code design
9	Implementing secret sharing	15/2/20	5	Code for sharing the secret input
10	Code design for the logic circuit	25/2/10	2	Code design
11	Implementing the logic circuit	5/3/20	7	working Boolean circuit function calculation
12	Code design for verify function	15/3/20	3	Design for the verify function
13	Implementing verify function	1/4/20	14	complete all part of the project
14	Testing and benchmark	22/4/20	7	Fixing bugs and getting results
15	Poster	06/20	14	Poster for presentation
16	Presentation	06/20	14	Presenting the project
17	Final submission	08/20	14	Submiting the project

5.2 Gant

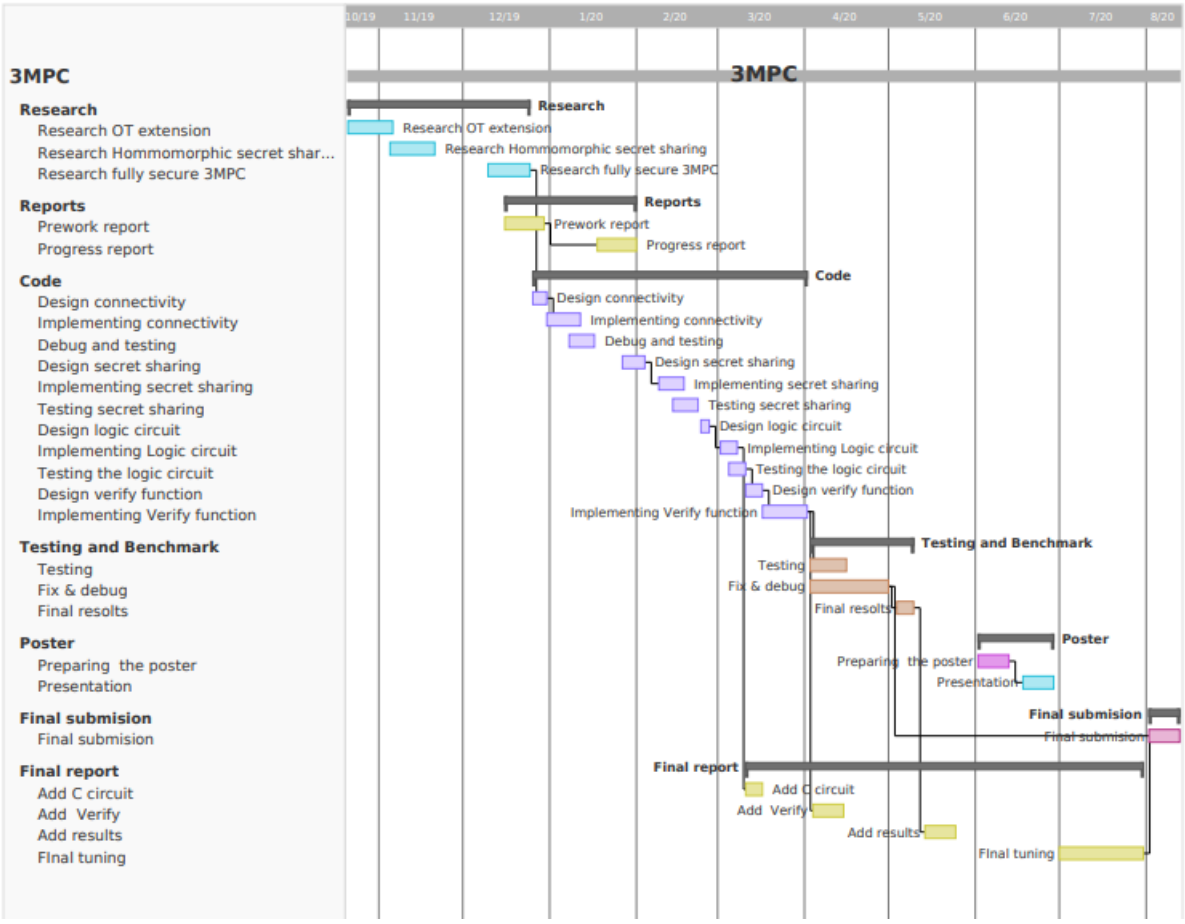

Figure 4: Gant

6 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.

[10] David Evans, Vladimir Kolesnikov and Mike Rosulek, A Pragmatic *"Introduction to Secure Multi-Party Computation.."* NOW Publishers, 2018