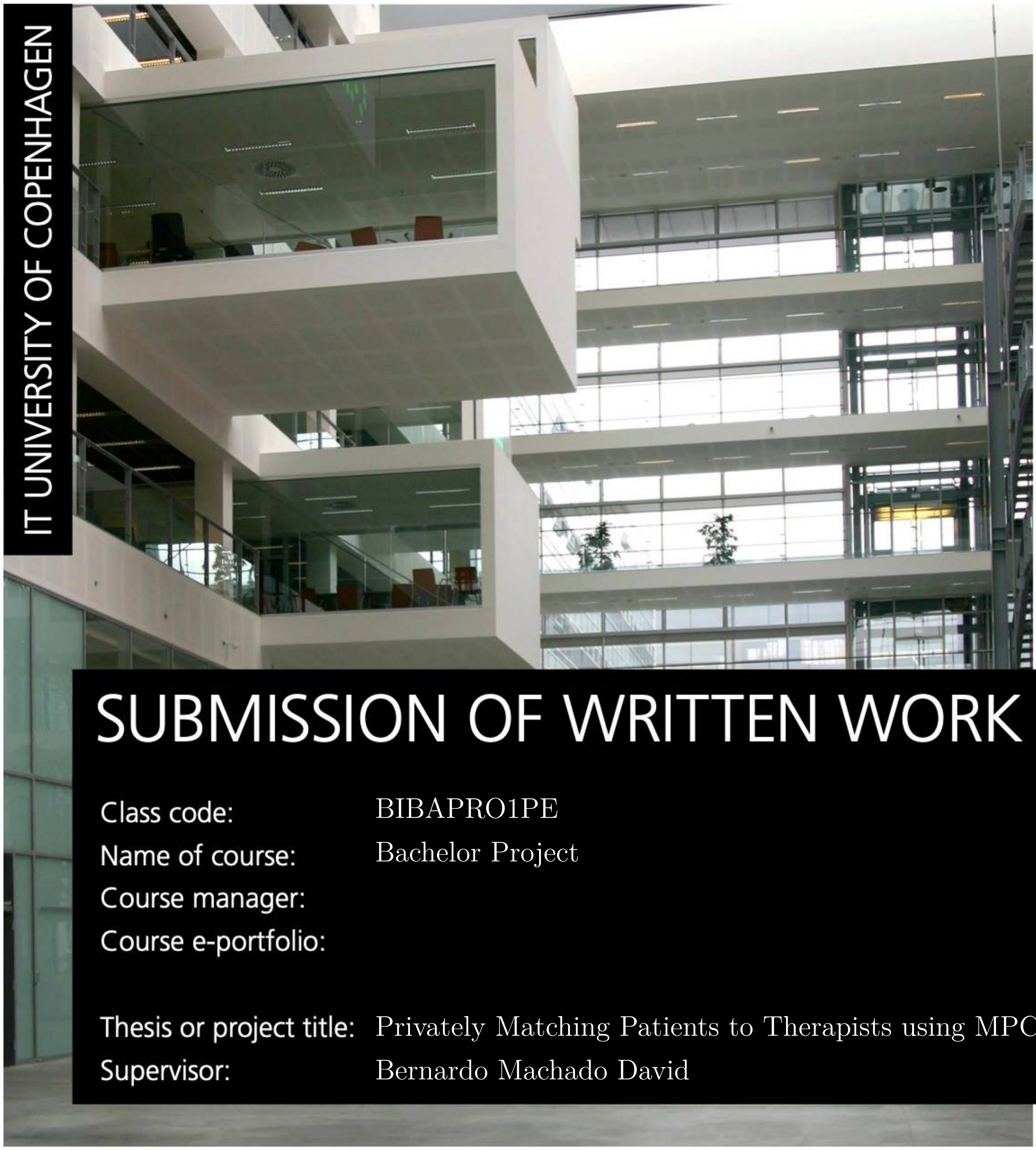

SUBMISSION OF WRITTEN WORK

Class code:	BIBAPRO1PE
Name of course:	Bachelor Project
Course manager:	
Course e-portfolio:	
Thesis or project title:	Privately Matching Patients to Therapists using MPC
Supervisor:	Bernardo Machado David

	Full Name:	Birthdate (dd/mm-yyyy):	E-mail:
1.	Sam Al-Sapti	09/01-2001	sals@itu.dk
2.			@itu.dk
3.			@itu.dk
4.			@itu.dk
5.			@itu.dk
6.			@itu.dk
7.			@itu.dk

Privately Matching Patients to Therapists using MPC

Sam Al-Sapti

May 15, 2023

Abstract

When patients need psychological therapy, they typically use a company to find a suitable therapist. This approach exposes sensitive medical information to a potentially untrusted party, the company. This project extends upon previous work done in the field of secure multi-party computation to adapt the Gale-Shapley algorithm to work in a secure setting. The project proposes an implementation based on a 3-party computation that ensures no single entity has access to sensitive information about patients. The implementation is benchmarked using the built-in benchmarking capabilities of MP-SPDZ, on which the implementation is built. The project shows that the implementation is secure in a semi-honest security model with dishonest majority, and that the implementation is fast enough that it can potentially scale to larger datasets.

Contents

1 Introduction **2**

1.1 The stable matching problem . . . 2

1.2 Problem statement . . . 2

2 Background **3**

2.1 Secure multi-party computation . . . 3

2.1.1 Secret sharing . . . 4

2.1.2 Oblivious transfer . . . 5

2.1.3 Garbled circuits . . . 5

2.2 The Gale-Shapley algorithm . . . 6

3 Implementation **7**

3.1 MP-SPDZ . . . 7

3.2 Design choices . . . 7

3.3 Security . . . 8

3.4 Protocol choices . . . 8

3.5 Input/output handling . . . 9

3.6 Algorithm . . . 9

3.7 Code . . . 9

4 Benchmarks **11**

4.1 System specifications . . . 11

4.2 Results . . . 11

5 Conclusion **12**

Chapter 1

Introduction

When patients need psychological therapy, they need to either find a therapist themselves, or delegate this activity to a company, such as Prescriba [9] or BetterHelp [1]. Both cases imply exposing private medical information to potentially untrusted parties. This might not be desirable, as malicious actors can share patient data with other third parties against the patients' interests.

1.1 The stable matching problem

This matching between patients and therapists is essentially what is known as *stable matching*. In the paper *Secure Stable Matching at Scale* [2], the authors not only adapt the Gale-Shapley [4] algorithm to work in a secure manner, using secure multi-party computation (MPC), but also to run efficiently, making it feasible to execute on large datasets. However, this is done as a two-party computation only, with one party holding the list of proposers and the other holding the list of acceptors. In the context of patients and therapists, this does not fully protect the privacy of patients, as it still requires a trusted party to execute the protocol for them, and such party will still be able to see the patients' private data in its entirety.

1.2 Problem statement

This project will attempt to improve upon the work done in the above mentioned paper, by extending the proposed implementation to work in an MPC with more than two parties, in such a way that no single actor has direct access to patient data, except for the therapist that the patient is matched to and the patient themself. Further, this project will benchmark the implementation with different sizes of datasets, as well as reflect upon the security properties guaranteed by the implementation.

Chapter 2

Background

2.1 Secure multi-party computation

In the field of cryptography, secure multi-party computation (MPC) is a tecnique that allows n mutually distrustful parties to compute a function $f(x_0, x_1, x_2 \cdots x_{n-1})$ that depends on private inputs from each party, but whithout revealing those private inputs to anyone. A single party p learns nothing other than the output of f and its own private input x_p [7].

When working with MPC, we consider a set of different adversarial models. Each MPC protocol is designed to be secure against an adversarial model chosen specifically for that protocol. An adversary is able to take control of a subset of the MPC parties, meaning t out of n parties. As such, an MPC protocol is usually designed to be secure against either honest majority or dishonest majority. Honest majority is defined as $t < \frac{n}{2}$, and dishonest majority is defined as $t \geq \frac{n}{2}$ [7].

A set of formal security properties are defined that are essential for an MPC protocol to be proven secure. The security properties are as follows [7]:

- **Privacy:** Any party should learn nothing more than its prescribed output and what can be derived thereof.
- **Correctness:** The output that a party receives is guaranteed to be correct.
- **Fairness:** Corrupted parties should receive their outputs if and only if the honest parties receive their outputs.
- **Guaranteed output delivery:** A corrupted party should not be able to prevent an honest party from receiving its output.
- **Independence of inputs:** A corrupted party must choose its input independently of the honest parties' inputs.

The properties guaranteed by a given protocol is oftentimes relaxed for specific scenarios, where for example privacy is not a concern for some of the data.

We also consider what capabilities corrupted parties have. The three types of adversaries are *semi-honest adversaries*, *malicious adversaries* and *covert adversaries*.

A semi-honest adversary can be viewed as a curious adversary. The parties corrupted by the adversary will behave according to the protocol, but they will gather as much information as possible from the data that gets sent to them during the protocol execution. This means that if a protocol is only secure against honest majority, an adversary controlling more than half of the parties would be able to jeopardize the privacy property of the protocol [7].

A malicious adversary will deviate arbitrarily from the protocol, for example by making the corrupted parties send arbitrary data instead of computing the correct data during protocol execution. Like the semi-honest adversary, they are able to break the privacy property of the protocol, but since they can also deviate from the protocol, they can also break correctness [7].

A covert adversary *may* behave like the malicious adversary, but it will only do so in specific circumstances. This security model implies, that there is a certain probability that the adversarial behavior will be detected. However, there is still a chance for the adversary to not be detected and can thus still break the protocol [7].

When talking about security in the context of MPC, we consider the real-ideal paradigm [3]. The real-ideal paradigm is the idea, that if we were to execute an MPC protocol in an ideal world, an ideal resource would be available. An ideal resource is an incorruptable, fully trusted function or third party. The ideal resource could for example be used to securely compute the sum of three parties' private inputs by sending them in plain text to this ideal resource, who would then execute the protocol for them and send back the result. In the real world however, there is no such ideal resource. Instead, we use tecniques such as MPC to do a secure computation of private inputs. When assessing the security of an MPC protocol, we compare what harm an adversary is able to do in the real world and in the ideal world. If they can do the same amount of harm (for example, in the context of a semi-honest adversary, learn the same amount of sensitive information) in both worlds, we say that a protocol is secure.

2.1.1 Secret sharing

Secret sharing is used to share a secret S among n parties, such that no party can see S in its entirety. One such variant is additive secret sharing, where S is split in n shares, such that $S = \sum_{i=0}^{n-1} s_i$, and s_p is then sent to p for any party p. This has the property, that all shares of S are required to recreate S. Thus, additive secret sharing provides perfect security if one or more shares are missing. One way to create such a sharing is by choosing s_0 through s_{n-2} randomly and then calculating $s_{n-1} = S - \sum_{i=0}^{n-2} s_i$.

This is one of the underlying technologies of many MPC protocols. In order for parties to collectively compute on private data, they can compute on shares of data. Both addition and multiplication is possible to compute on secret-shared data.

MPC can be done with arithmetic or binary circuits. Arithmetic circuits offer faster computation for arithemtic operations, namely addition and multiplication, as computation is made directly on integers. Binary circuits on the other hand offers faster computation times for non-arithmetic operations, such as comparisons. Secret sharing is mostly preferred for computation with arithmetic circuits. For binary circuits, we might be more interested in garbled circuits.

2.1.2 Oblivious transfer

An oblivious transfer (OT) is a secure way for a sender S to send n secrets to a receiver R, but R only receives one of them without S knowing which one [3]. For an OT with n secrets, we denote it as a 1-out-of-n OT. In the case of $n = 2$, a 1-out-of-2 OT works by S giving input secrets x_0, x_1 to the OT function F^{OT}, and R supplying a choice-bit $b \in \{0, 1\}$. Then R will receive x_b from F^{OT}. Thus, R has not learned x_{1-b} and S has not learned b, so S does not know which x_b was chosen by R. There are a variety of ways for a secure OT function to be constructed, but we will not look at those in this report.

2.1.3 Garbled circuits

A garbled circuit (GC) is a type of binary circuit used in MPC, that ensures that no party can see other parties' inputs in the clear. The MPC protocol best known for this is Yao's Garbled Circuit Protocol [3], which is regarded as one of the most performant MPC protocols. Yao's GC however only works with two parties. There are other protocols that work with more parties, such as the Beaver-Micali-Rogaway (BMR) protocol [3], which in fact builds on top of Yao's GC.

We will now take a high-level look at how a GC works, in the context of Yao's GC. p_0 takes the role of GC generator, and p_1 takes the role of GC evaluator. p_0 constructs a binary circuit with 2-input boolean gates. For each gate, p_0 evaluates all possible combinations of inputs $x, y \in \{0, 1\}^*$, and stores them in a lookup-table. For each cell in the lookup table, two symmetric encryption keys with length K, $k_x, k_y \in_R \{0, 1\}^K$, are randomly chosen. Then, the cells are encrypted with *both* keys, meaning the cell for inputs x and y is encrypted with k_x and k_y. The table is randomly permutated and sent to p_1. For the evaluation, p_0 knows their own input x, so they send k_x to p_1 in plain text. p_0 does not know p_1's input y, so they send k_y to p_1 using a 1-out-of-$|Y|$ OT (Y being the domain of possible values of y), and p_1 can now use the keys to decrypt the cell corresponding to the inputs. Due to the nature of binary gates, the table only has four cells, so p_1 can just brute-force their way to the correct cell. However, this is not always the case, or it can be expensive for deep circuits. Some tecniques exist to make p_1 able to determine which cell to decrypt. One of those is known as *Point-and-Permute* [3].

2.2 The Gale-Shapley algorithm

The Gale-Shapley algorithm was proposed in 1962 by David Gale and Lloyd Shapley [4] as a solution to the stable matching problem. Their algorithm proves that there always exists a match where all pairs are stable given an equal number of participants of each type, and that the match has the following properties:

- **Everyone is matched:** A proposer who is left unmatched at the end must have proposed to all acceptors, but when an acceptor receives a proposal, they will remain matched for the rest of the execution of the algorithm. Thus, no acceptor can be unmatched. Since the number of proposers and acceptors are equal, there can be no unmatched proposer either.
- **The pairs are stable:** If a proposer p is part of an unstable pair, they would have proposed to their preferred acceptor a prior to their actual match. If a receives a proposal from a proposer q that they prefer over p, they will dump p for q, but if they prefer p over q they will stick with p and reject q. Hence, no unstable pair can exist.

A modified revision of this algorithm, known as the Roth-Peranson algorithm, is currently in use by the National Resident Matching Program (NRMP) in the United States [10].

The Gale-Shapley algorithm has previously been implemented as a 2-party computation by Doerner et al. [2], but where one party holds an entire set of participants, meaning that party p_0 holds the data of all proposers and party p_1 holds all the data of acceptors. Their approach ensures privacy in that proposers never know what the acceptors' preferences are and vice versa, should either p_0 or p_1 be corrupted by an adversary. They learn their final matches, but they also learn the preferences of proposers (p_0) and acceptors (p_1).

Chapter 3

Implementation

As stated, an implementation of the Gale-Shapley algorithm exists for use in MPC [2]. That implementation is built with the MPC framework Obliv-C [11], which is rather outdated and difficult to compile on modern systems. It also only works with Yao's GC protocol.

An adoption of the code to the MP-SPDZ framework [6] has already been made, which is found in the source code for MP-SPDZ. As such, my implementation extends upon that to modify it to the setting of patients and therapists.

3.1 MP-SPDZ

MP-SPDZ is a multi-protocol MPC framework developed to provide easy implementation and benchmarking of secure computations [6]. It provides a high-level language based on Python, to write an MPC program in. It provides a variety of protocols implemented as virtual machines and a compiler to compile a high-level program to machine code that the virtual machines interpret. It supports both arithmetic and binary circuits, and has different protocols for various security settings.

3.2 Design choices

The implementation assumes that patients and therapists do not know each other. In fact, the setup is very similar to when patients use services like BetterHelp [1]. A patient registers at BetterHelp, and they match them to a suitable therapist. This also means, that BetterHelp is the only entity that knows both patient and therapist.

The implementation also assumes that a patient suffers from one of five cases, indexed as 1 to 5, and that a therapist is experienced with one case. As such, both patients and therapists really only have one preference, but there can be more than one patient with a given case or therapist with a given experience. Hence, patients and therapists actually have more than one preferred match. This means that there can be multiple stable matchings in this setting.

3.3 Security

The implementation is made as an outsourced 3-party computation. To ensure that no single party has access to the real data, the inputs from the patients and therapists are additively secret-shared into three shares (which is also the number of MPC parties), and one share is given to each MPC party. When the protocol execution is done, the parties end up with shares of the outputs that are then sent back to the patients and therapists.

As such, the parties compute the Gale-Shapley algorithm on shares of the real data. This ensures privacy of inputs even with dishonest majority, since additive secret sharing requires all shares to be present in order to reveal the secret. This means that up to two parties can be corrupted by an adversary without compromising the protocol. It is important to note however, that the protocol is only secure in the presence of a semi-honest adversary. While this is a rather weak security model, correctness is not a concerns since the results can easily be verified by the patients and therapists themselves when they meet.

In an ideal world, one entity (the ideal resource) will receive all inputs in plain text, do the computation and deliver the outputs to the participants (patients and therapists). Notice that this is exactly what BetterHelp does, thus we can say that BetterHelp is the ideal resource if we were to disregard privacy. In the real world however, we do care about privacy, and BetterHelp is not an ideal resource. Hence we need a secure protocol to execute the matching algorithm.

3.4 Protocol choices

Since the implementation does a lot of arithmetic operations, it seems obvious to use an arithmetic circuit based protocol. As such, the implementation uses the OT-based Semi2k protocol, which is a version of the SPDZ2k protocol stripped from all steps required for malicious security [5]. Semi2k runs in an arithmetic circuit with a mod 2^k domain.

As with the implementation by Doerner et al. [2], my implementation uses an oblivious RAM (ORAM) to store the data in while the algorithm is executed. An ORAM is a datastructure that makes accesses oblivious, meaning that an observer neither knows whether an access is a read or a write operation, nor which index is accessed. This is important to ensure that an adversary cannot infer which preferences patients and therapists have simply by observing how the data is accessed [2].

3.5 Input/output handling

The computation takes as input two lists, one containing the cases of each patient sorted by patient index, and one containing experiences of therapists sorted by therapist index. The list is secret-shared as stated in 3.3. Each party reads their shares of the list, and the shares are added together securely inside the protocol and stored in an ORAM.

The output is two new lists (both stored in ORAMs), one with indices of therapists ordered by patient index, and one with indices of patients ordered by therapist index. The lists do not exist as entire lists in reality. Each party end up with secret shares of the list elements. The list shares can now be sent to patients and therapists. The shares of the therapist list are then sent to the respective patients, and the shares of the patient list are sent to the respective therapists. Patients and therapists compute the real values from the shares themselves by adding the shares together. They now know who they have been paired with.

Sending of the lists to patients and therapists is not currently implemented. The values are printed to the screen so that the user of the implementation can view them. However, this could be an easy addition to the program. In fact, MP-SPDZ supports communicating with non-computing parties (also called clients) [5].

3.6 Algorithm

The Gale-Shapley implementation has been modified to accomodate for the patient and therapist setting. Due to the fact that patients and therapists do not know each other, they also do not know what index they prefer. They only know what case they suffer from (patients) or are experienced with (therapists). Thus, while the normal Gale-Shapley works on participants specifying what indices they prefer, we need a different approach. On each step in the main loop of the algorithm, another loop runs that iterates over the list of therapists. It performs a first-fit search for a therapist that is experienced with the current patient's case, and saves the index of said therapist to use in the rest of the main loop iteration. This of course increases time complexity by a factor of n, where n is the number of patients or therapists, making the overall time complexity $O(n^3)$ instead of $O(n^2)$ for normal Gale-Shapley [4]. On the other hand, as mentioned in 3.2, each participant only has one preference. This means that the matrices used to store patient and therapist preferences are n by 1 instead of n by n, essentially making them lists rather than matrices. This in turn reduces time complexity by a factor of n, such that we are back at a time complexity of $O(n^2)$.

3.7 Code

The implementation takes the form of a program written in MP-SPDZ's Python-based high-level language. The source code file is named `gale_shapley.mpc`. All source code files are located in the `src/` directory.

Further, there is a Bash script `gen_data.sh` that randomly generates data for patients and therapists, secret-shares the data and outputs the shares to three files for the MPC parties to use. The data is ordered as a list of patient cases, followed by a list of therapist experiences. All the integers are seperated by spaces and prepended by an identifier, namely `-100` for patients and `-200` for therapists. The script ensures that any patient case c is present the same number of times in both lists. This is because the list of therapist data is a random permutation of the patient list. Note that this way of generating data is only for the purposes of this project. A real world use case should use real data.

Lastly, a `Makefile` is supplied for building MP-SPDZ and compiling `gale_shapley.mpc`, and a Bash script `run.sh` is supplied to run the computation. Instructions for building and running can be found in `README.md`.

The code is expected to work on an Ubuntu 22.04 LTS environment on an x86_64 based machine. Other environments are not guaranteed to work. The code for the implementation can be found in the Git repository samsapti/bachelor-project on GitHub, as well as in the appendix.

Chapter 4

Benchmarks

To benchmark the implementation, MP-SPDZ's built-in benchmarking capabilities are used [5]. The benchmarks have been done locally with all computing parties running on one machine. This of course introduces a bottleneck. Due to a lack of resources, it was not possible to benchmark the implementation on cloud servers. This would have avoided the bottleneck of multiple parties running on the same CPU.

4.1 System specifications

The benchmarking has been done on a desktop computer featuring an Intel i7-13700K processor with multithreading enabled. The operating system is Ubuntu 22.04 LTS. MP-SPDZ was compiled with Clang version 14.0.0, and the Python version used is 3.10.6.

The computation is intended to be run on three powerful machines, one for each computing party.

4.2 Results

Using the Bash script `run.sh` to spawn three computing parties, and with multiple lengths of input lists, the results have been obtained and can be seen below. The results have been rounded to one decimal place.

Input length	10	20	50	100	200
Time in seconds	0.6	1.4	7.8	37.5	222.4

Table 4.1: A table showing the benchmarking results for different input lengths.

Overall it seems very much feasible to use this implementation for large scale computations. A simple modification that would speed up the computation even more is to switch to a next-fit search instead of a first-fit search. This would make it even more scalable.

Chapter 5

Conclusion

When we want to match patients to therapists, it is important for the privacy of patients that sensitive medical information does not fall into the wrong hands. It is therefore in many ways arguably not a good idea to delegate the task of matching patients to therapists to a central third party. Secure multi-party computation allows us to construct a protocol for computing on private data without revealing anything other than the outputs.

Doerner et al. [2] used this to construct a scalable 2-party computation for computing the Gale-Shapley algorithm on large datasets. This project has shown how their idea can be adapted to an MPC with three parties, with data secret-shared among them, such that no single party can view the raw data and that the protocol is secure against semi-honest adversaries with dishonest majority. It is also shown that the execution time cost is low enough to be used for moderate sizes of datasets.

This implementation has yet to show its potential on very large datasets, such as in the scale of the NRMP matching pool [8]. It is believed however, that it will perform equally well as the implementation by Doerner et al. due to the similar time complexity.

Bibliography

[1] BetterHelp. *Professional therapy with a licensed therapist.* (Visited on 05/14/2023). 2023. URL: `https://www.betterhelp.com`.

[2] Jack Doerner, David Evans, and abhi shelat. "Secure Stable Matching at Scale". In: *23rd ACM Conference on Computer and Communications Security.* 2016. DOI: `10.1145/2976749.2978373`. URL: `https://eprint.iacr.org/2016/861.pdf`.

[3] David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation.* (Version: June 11, 2022). NOW Publishers, Dec. 2018. ISBN: 978-1-68083-509-0. DOI: `10.1561/3300000019`.

[4] David Gale and Lloyd Shapley. "College Admissions and the Stability of Marriage". In: *American Mathematical Monthly* 69.1 (Jan. 1962). (Visited on 02/13/2023). DOI: `10.2307/2312726`. URL: `https://web.archive.org/web/20170925172517/http://www.dtic.mil/get-tr-doc/pdf?AD=AD0251958`.

[5] Marcel Keller. *MP-SPDZ documentation.* (Visited on 03/28/2023). 2022. URL: `https://mp-spdz.readthedocs.io/en/stable/`.

[6] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security.* 2020. DOI: `10.1145/3372297.3417872`. URL: `https://eprint.iacr.org/2020/521.pdf`.

[7] Yehuda Lindell. "Secure Multiparty Computation". In: *Communications of the ACM* 64.1 (Jan. 2021). (Visited on 02/02/2023). DOI: `10.1145/3387108`. URL: `https://web.archive.org/web/20220629061828/https://cacm.acm.org/magazines/2021/1/249459-secure-multiparty-computation/fulltext`.

[8] NRMP. *National Resident Matching Program.* (Visited on 05/15/2023). 2002. URL: `https://www.nrmp.org`.

[9] Prescriba. *Psykologisk rådgivning.* (Visited on 05/14/2023). 2023. URL: `https://prescriba.com`.

[10] Alvin E. Roth and Elliot Peranson. "The Redesign of the Matching Market for American Physicians: Some Engineering Aspects of Economic Design". In: *American Economic Review* 89.4 (Sept. 1999). (Visited on 05/03/2023). DOI: `10.1257/aer.89.4.748`. URL: `https://web.stanford.edu/~alroth/papers/rothperansonaer.PDF`.

[11] Samee Zahur and David Evans. "Obliv-C: A Language for Extensible Data-Oblivious Computation". In: *Cryptology ePrint Archive, Paper 2015/1153.* Nov. 2015. URL: `https://eprint.iacr.org/2015/1153.pdf`.