IT SEC Research Group IT-Security | RWTH AACHEN UNIVERSITY

The present work was submitted to the Research Group IT-Security

Prof. Dr. Ulrike Meyer

Master Thesis

Secure Maximum Weight Matching Approximation for Privacy-Preserving Kidney Exchange

submitted by

Andreas Brüggemann

369345

Thesis Supervisor:	Prof. Dr. Ulrike Meyer
Second Examiner:	Dr. Walter Unger
Advisors:	Andreas Klinger, M.Sc. RWTH
	Malte Breuer, M.Sc. RWTH
Submitted:	December 2, 2021

Secure Maximum Weight Matching Approximation for Privacy-Preserving Kidney Exchange

Master Thesis

Abstract. For patients with end stage renal disease, kidney transplantation is an attractive alternative to treatments such as dialysis. A shortage of kidney transplants from deceased donors and therefrom resulting long waiting times can be bypassed by finding a willing living donor. In the case of medical incompatibility, it may still be possible to exchange donors with other patients also having incompatible donors. A special case of such a kidney exchange where only two patients swap their donors is called crossover exchange. While a matching of patients exchanging their donors can be computed by some central instance, such an approach is vulnerable to leakage of sensitive medical information.

In this thesis, we present the first privacy-preserving protocol for maximum weight matching approximation on general graphs. The protocol is statistically secure against a passive and static adversary, assuming an honest majority. For N parties, it runs in $\mathcal{O}(N \cdot \log N)$ rounds yielding a run time of less than 5 hours for up to 80 parties. Regarding crossover exchange, the protocol offers a privacy-preserving solution where weights allow for prioritization to be implemented. In this case, prioritization allows to increase the chances of getting matched for patients that for instance are difficult to match or already have a long waiting time.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 3
- 1.2 Outline . . . 4

2 Preliminaries **5**
- 2.1 Mathematical Notations . . . 5
- 2.2 Perfect and Statistical Indistinguishability . . . 5
- 2.3 Shamir's Secret Sharing Scheme . . . 6
- 2.4 Secure Multi-Party Computation . . . 7
  - 2.4.1 Behavior of the Adversary . . . 8
  - 2.4.2 Definition of Security . . . 9
  - 2.4.3 The BGW Protocol . . . 10
  - 2.4.4 Data Representation . . . 11
  - 2.4.5 Complexity Metrics . . . 11
  - 2.4.6 Obliviousness in Secure Multi-Party Computation . . . 12
- 2.5 Graph Theory . . . 13

3 Related Work **15**
- 3.1 Maximum Weight Matching Algorithms . . . 15
  - 3.1.1 Maximum Weight Matching Approximation . . . 16
- 3.2 Secure Protocols for Matching Problems . . . 17
- 3.3 Conventional Kidney Exchange . . . 18
- 3.4 Secure Protocols for Bartering and the KEP . . . 19

4 Ideal Functionality for Maximum Weight Matching Approximation **23**
- 4.1 MWM Approximation Algorithm . . . 23
- 4.2 Definition of the Ideal Functionality for Maximum Weight Matching 26
- 4.3 Application for Crossover Exchange . . . 27
  - 4.3.1 Graph Representation . . . 27
  - 4.3.2 Generation of the Compatibility Graph . . . 29

5 Building Blocks **31**
- 5.1 Elementary Gates . . . 31
  - 5.1.1 Secure Comparison . . . 31
  - 5.1.2 Secure Random Integer Generation . . . 32
  - 5.1.3 Auxiliary Operations . . . 33
  - 5.1.4 Mathematical Operations . . . 34
- 5.2 Edge Generation Functionality . . . 34
- 5.3 Edge Generation Gate for Crossover Exchange . . . 36

5.3.1 ABO-Compatibility . . . 36
5.3.2 Negative Virtual HLA-Crossmatch . . . 36
5.3.3 Definition of Edge Weights . . . 37
5.3.4 Gate Specification . . . 38
5.4 Secure Permutation . . . 40
5.4.1 Secure Shuffling . . . 40
5.4.2 Gate Specification . . . 42
5.5 Best Edge Search . . . 45
5.5.1 Gate Specification . . . 46

6 Privacy-Preserving Protocol for Maximum Weight Matching Approximation 49
6.1 Preliminary Algorithmic Considerations . . . 49
6.1.1 Specification of the Modified Algorithm . . . 51
6.2 Protocol Specification . . . 52
6.2.1 Input Phase . . . 53
6.2.2 Matching Phase . . . 53
6.2.3 Output Phase . . . 58
6.2.4 Complexity . . . 59
6.2.5 Correctness and Security . . . 59

7 Evaluation 63
7.1 Implementation in MP-SPDZ . . . 63
7.2 Evaluation Setup . . . 64
7.3 Results . . . 65
7.3.1 Performance of the Protocol . . . 65
7.3.2 Performance of the Protocol's Phases . . . 68
7.3.3 Impact of Parallelization of Comparisons . . . 69
7.3.4 Impact of Latency and Bandwidth . . . 70

8 Conclusion 73

A Random Edge Selection 75
A.1 Ideal Functionality Based on Random Edge Selection . . . 75
A.2 Protocol Based on Random Edge Selection . . . 77
A.2.1 Virtual Node Permutation . . . 78
A.3 Evaluation of the Different Approaches . . . 79

B Quality of the Computed Matchings for Crossover Exchange 81
B.1 Properties of the Compatibility Graphs . . . 81
B.2 Results for the Greedy MWM Algorithm . . . 83

Notation 87

Acronyms 89

Lists of Figures, Tables, Algorithms, Gates and Protocols 91

Bibliography 95

Chapter 1

Introduction

Chronic kidney disease cannot be cured to date and may progress to end stage renal disease [Bir+21]. The most common treatment for end stage renal disease is dialysis [Bir+21]. Kidney transplantation is an alternative treatment that is less costly and offers higher life expectancy and quality of life [Hal+11; Axe+18]. At the end of 2020, a total of 10827 patients were on waiting lists for kidney transplants from deceased donors in Eurotransplant member states[1] alone [Eur21]. To overcome resulting long waiting times, living donor kidney donations can be used where a patient tries to find a donor willing to donate one of her two redundant kidneys [Bir+21]. In addition, donations by living donors yield better results than those by deceased donors [Har+17].

Even if a patient finds a donor who is willing to donate a kidney, transplantation often is not possible due to medical incompatibility [ABS07]. To enable living donor kidney donations despite such incompatibility, kidney exchange can be used [ABS07]. In kidney exchange, multiple independent *parties*, i.e., pairs of a patient and her corresponding incompatible donor, form an exchange cycle of certain properties. A *crossover exchange* is such an exchange where only two parties are involved, and each party's donor is incompatible to his own but compatible to the other party's patient. Each of the involved patients then receives a kidney donation from the opposite party's donor. Such a construct yields a notion of fairness in the sense of each donor donating a kidney also having her corresponding patient receiving one. It is also possible to allow similar exchanges between more than two parties yielding some cycle of length three or higher. Figure 1.1 gives an example of a crossover exchange and a kidney exchange in a cycle of length 3.

While a kidney exchange in a cycle of 13 parties has been executed successfully [Wil09], shorter cycles usually are preferred with multiple kidney exchange programs limiting their length [Bir+21]. This is the case as long cycles imply additional logistical difficulties and are more likely fail in their execution [Bir+21]. The kidney exchange programs of France and Sweden that operate since 2014 and 2018 even limit kidney exchanges to cycles of length 2 and therefore only use crossover exchanges [Bir+19; Bir+21]. Furthermore, as an example the German transplantation law (Transplantationsgesetz, sect. 8, para. 1) only allows kidney donations

[1] The members are Austria, Belgium, Croatia, Germany, Hungary, Luxembourg, the Netherlands and Slovenia.

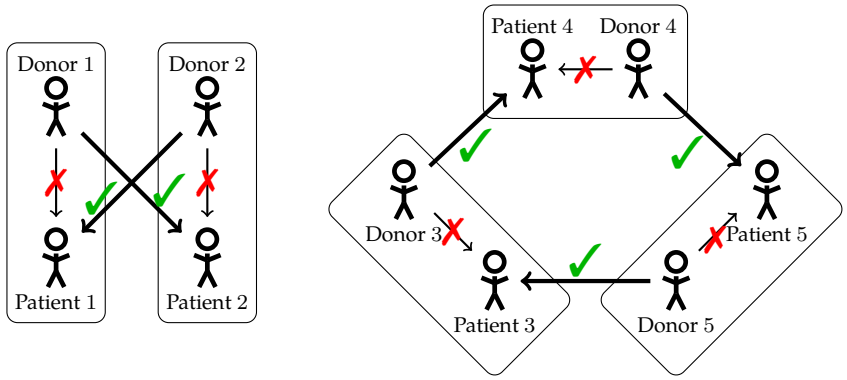


Figure 1.1.: Crossover exchange and kidney exchange in a cycle length 3. Medical incompatibility is denoted by "✗" and medical compatibility is denoted by "✓".

from donors that are especially close to the patient at the time of this writing. In recent years, motions to generally allow crossover exchanges have been filed and discussed in the German Federal Parliament after occasional crossover exchanges under special circumstances [Bro+; Rei20]. Therefore, it can be presumed that countries that currently do not maintain or take part in a kidney exchange program due to legal reasons may legalize crossover exchanges in the future instead of immediately adopting more liberal laws regarding kidney donations.

Finding pairs of two parties where a crossover exchange is possible is usually viewed as a graph theoretic problem [ABS07; Bir+21]. Parties are represented by nodes in some graph where two parties are connected if a crossover exchange between them is possible. Then, a selection of pairs of parties that can do a crossover exchange corresponds to a matching in such a graph [ABS07]. Furthermore, it is possible to add weights to the edges of the graph and to search for a matching that maximizes the cumulative weight of its edges. Such addition allows for the prioritization of some crossover exchanges over others. For instance, it is possible to prioritize patients who are difficult to match due their medical characteristics, especially so called *highly sensitized* patients, or those who already have a long waiting time [Bir+21].

Before searching for some matching, it is necessary to determine which parties are medically compatible. It is possible to estimate compatibility between two parties by taking separately gathered medical information of both parties' donors and patients into consideration [Bir+21]. In such a case, the final choice on whether a transplant is carried out or not lies with medical experts [Bre+20; Bir+21]. Furthermore, edge weights have to be determined given the parties' inputs. The required computation and the subsequent search for a matching is typically performed by a central instance maintaining a kidney exchange program [Bir+21].

A central instance receiving all sensitive information in a protected way and internally computing a matching displays a single point of failure even if considered trustworthy. This is critical as the required medical information of participating donors and patients should be protected. Besides medical information being sensitive information in general already, organ transplantation especially implies substantial risks in the case of insufficient protection. In particular, gaining knowledge about

possible donors could allow a single patient in need of a kidney transplant to identify and coerce some compatible donor into donation.

To protect medical information to an extent that is equivalent to the central instance being unconditionally trustworthy and secure in absence of such ideal party, *secure multi-party computation* (SMPC) which is a branch of cryptography can be used. SMPC allows a set of parties to compute a function on their own secret inputs together in a distributed fashion such that they learn nothing on the other parties' inputs and outputs that cannot be deduced from their own [Yao82]. Using SMPC to examine compatibility, determine weights and search for a *maximum weight matching* (MWM) on the resulting graph yields strictly more privacy than a centralized approach regarding medical information. The increase of privacy may even allow legalization of crossover exchange if insufficient protection of medical data was a critical reason not to allow it otherwise.

Approximating an MWM instead of searching for an optimal solution allows simpler and faster approaches to be used. This especially is beneficial regarding SMPC where overhead of necessary cryptographic operations can easily yield infeasible run times for complex problems or large inputs [Lin20]. Furthermore, searching for MWMs or corresponding approximations is a general problem that has a wide range of applications beyond crossover exchange [DP14]. By keeping an approach to privacy-preserving matchings general, it can also be used for such use cases besides crossover exchange.

This thesis' goal is to utilize SMPC to design, implement and evaluate a secure protocol for the privacy-preserving approximation of an MWM. We apply the resulting protocol to crossover exchange by securely using the medical inputs of multiple parties to generate a graph to compute a matching on. Thus, our approach enables privacy-preserving crossover exchange using prioritization expressed through weights and protects the necessary medical data of the involved parties.

1.1. Contributions

In this thesis, we devise a privacy-preserving protocol and secure building blocks for MWM approximation and its application in crossover exchange. In particular, our contributions are:

Secure MWM Approximation We present a secure protocol for privacy-preserving MWM approximation on general graphs. To the best of our knowledge, our protocol constitutes the first approach to privacy-preserving matchings of any kind on general graphs. The protocol generates a matching of at least half the optimal weight between all participating parties where the existence and weight of an edge between two parties is determined by their inputs. It prevents any bias in favor of some party that does not depend on the given inputs and, assuming an honest majority, is secure against a passive and static adversary. We implement and evaluate the protocol using the framework MP-SPDZ [Kel20] revealing that it runs in less than 5 hours for up to 80 parties.

Secure Crossover Exchange Using our novel secure protocol for privacy-preserving MWM approximation, we provide a building block for graph generation to be used for crossover exchange. The building block uses the medical inputs

of multiple parties to determine compatibility and weights for prioritization to build a corresponding graph modeling. To the best of our knowledge, the combination of our matching protocol and the building block for crossover exchange constitutes the first solution that explicitly handles prioritization for crossover exchange.

1.2. Outline

The remainder of this thesis is structured as follows: In Chapter 2, we establish the backgrounds and definitions regarding basic notation, SMPC and graph theory. Subsequently, related work regarding MWM algorithms, secure matching protocols, (secure) kidney exchange and secure bartering is discussed in Chapter 3. Chapter 4 defines the ideal functionality to be implemented by our protocol and describes how it can be used for crossover exchange. Several building blocks for our protocol are introduced in Chapter 5. In particular, the chapter defines an example of a gate that allows to use the protocol for crossover exchange. The protocol implementing the previously defined functionality is given in Chapter 6 where we also prove its security. In the evaluation in Chapter 7, we analyze the run times and generated traffic of our protocol. We close this thesis with a general conclusion and an outlook regarding future work in Chapter 8.

Chapter 2

Preliminaries

In the following chapter, we provide an overview of different notations, techniques and definitions used in this thesis. In particular, it contains the necessary foundations relating *secure multi-party computation* (SMPC) and matchings. After introducing basic notation in Section 2.1, we give definitions for perfect and statistical indistinguishability in Section 2.2 that are later required for SMPC. In Section 2.3, we present Shamir's secret sharing scheme [Sha79] that then is used in Section 2.4 concerning SMPC. The section on SMPC contains several parts on the general setting, definitions of security, relevant protocols, data representation, complexity metrics and further techniques commonly used in this field. We end the chapter by fixing relevant graph definitions and notations in Section 2.5.

2.1. Mathematical Notations

We define $\mathbb{N}$ and $\mathbb{N}_0$ as the sets of natural numbers excluding and including zero, $\mathbb{Z}$ as the set of all integers and $\mathbb{R}$ as the set of all real numbers. For the finite field obtained by taking the ring of integers modulo a prime p we write $\mathbb{F}_p$. We write $[a, b]$ for the interval of all integers $a \leq x \leq b$, i.e., $[a, b] := \{a, a + 1, a + 2, ..., b - 1, b\}$ where $a, b \in \mathbb{Z}$. The interval of all $x \in \mathbb{R}, a \leq x \leq b$ for fixed $a, b \in \mathbb{R}$ is denoted by $[a, b]_{\mathbb{R}}$. For sets A, B, we denote by $A \subseteq B$ that A is a subset of B allowing $A = B$. For $x \in \mathbb{R}$, by $\lfloor x \rfloor$ we denote the greatest $n \in \mathbb{Z}$ such that $n \leq x$ and by $\lceil x \rceil$ we denote the smallest $n \in \mathbb{Z}$ such that $n \geq x$. We denote the logarithm function with respect to base 2 by $\log$. The dot product of vectors A and B of equal dimension is denoted by $\langle A, B \rangle$. By $\{0, 1\}^*$, we denote the set of all binary strings and by S_X for a finite set X, we denote the symmetric group on X, i.e., the set of all permutations on X.

2.2. Perfect and Statistical Indistinguishability

Given two random variables, there is a difference measure called *statistical distance*.

Definition 2.1 (Statistical Distance [Sho09])**.** *Let A, B be two random variables taking values in a finite set S. The statistical distance between A and B is defined as*

$$\Delta(A, B) = \frac{1}{2} \sum_{s \in S} |\Pr(A = s) - \Pr(B = s)|.$$

The statistical distance gets lower with increasing similarity between the distributions of the given random variables. For a sufficiently small statistical distance, the distributions can be viewed as indistinguishable. To define whether a statistical distance is sufficiently small, *negligible functions* can be used as an upper bound. Such functions introduce a parameter that significantly decreases the resulting value if it is increased.

Definition 2.2 (Negligible Function [Can00])**.** *A function $\mu : \mathbb{N} \to [0, 1]_{\mathbb{R}}$ is* negligible *if for all $c > 0$ there exists some n_0 such that*

$$\mu(n) < n^{-c} \; \forall n \geq n_0.$$

A collection of infinitely many random variables can also be used to define a *probability ensemble*.

Definition 2.3 (Probability Ensemble [Lin17])**.** *A* probability ensemble *$X = \{X(a, \kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ is an infinite sequence of random variables taking values in a finite set and being indexed by $a \in \{0, 1\}^*$ and a security parameter $\kappa \in \mathbb{N}$.*

Such probability ensembles can be compared using the statistical distance between their contained random variables. Different definitions on what distances are sufficiently small lead to different notions of indistinguishability.

Definition 2.4 (Perfect Indistinguishability [Can00] (named as in [CH10; Wü18]))**.** *Let X, Y be two probability ensembles. X and Y are* perfectly indistinguishable *if for all $a \in \{0, 1\}^*, \kappa \in \mathbb{N}$ it holds that*

$$\Delta(X(a, \kappa), Y(a, \kappa)) = 0.$$

Definition 2.5 (Statistical Indistinguishability [Can00])**.** *Let X, Y be two probability ensembles. X and Y are* statistically indistinguishable *if there exists some negligible function μ and a $\kappa_0 \in \mathbb{N}$ such that for all $a \in \{0, 1\}^*, \kappa \geq \kappa_0$ it holds that*

$$\Delta(X(a, \kappa), Y(a, \kappa)) \leq \mu(\kappa).$$

Note that perfect indistinguishability implies statistical indistinguishability.

2.3. Shamir's Secret Sharing Scheme

Shamir's secret sharing scheme [Sha79] is a (t, N) *threshold secret sharing scheme*. Such a threshold secret sharing scheme allows to split a secret value s into N shares such that the secret can only be reconstructed from at least t shares. Another important property is that any set of less than t shares cannot be used to gain any knowledge about the secret in an information theoretic sense. The shares can be distributed among N different parties such that at least t parties need to cooperate to reconstruct the secret. Shamir's secret sharing scheme uses following steps to share a secret among N parties:

1. Select a prime p such that $p > N$ and the field $\mathbb{F}_p$ is sufficiently large to serve as domain for the secret value to be shared.

2. Let the secret value be some $s \in \mathbb{F}_p$. Then, draw coefficients $a_1, a_2, ..., a_{t-1}$ uniformly at random from $\mathbb{F}_p$ to build a polynomial $q(X) = s + a_1X + a_2X^2 + ... + a_{t-1}X^{t-1}$ of maximal degree $t - 1$.

3. Send share $q(i)$ to party i.

If at least t parties cooperate, they can reveal the secret by using polynomial interpolation, e.g. Lagrange interpolation, over their shares to reconstruct the polynomial $q(X)$ which is uniquely identified for the given amount of shares [Lin20]. Evaluating the polynomial then yields the secret value $s = q(0)$. For any number of cooperating parties that is less than t, it is not possible to gain any information about the secret value s. The proof idea for this claim is that even for $t - 1$ known shares there is a unique polynomial $q_{s'}(X)$ for each possible secret value $s' \in \mathbb{F}_p$ such that $q_{s'}(0) = s'$ which can be shown by interpolation. Thus, the known shares match a set of polynomials representing any possible secret value. Due to the random selection of coefficients, all these polynomials and therefore also all possible secret values s' are equally likely to match s.

Regarding notation, $[x]$ shall denote some integer $x \in \mathbb{F}_p$ being shared using Shamir's secret sharing scheme while x alone refers to the actual secret value. We also call such $[x]$ a secret shared value. Furthermore, sharing some value always refers to using Shamir's secret sharing scheme to share it throughout this thesis. Arithmetic operations on $[x]$ implicitly represent specific operations on the corresponding shares such that the result is a secret shared value representing the result of the operation if x was used as a public value. For instance, $a \cdot [x] + [y]$ represents operations on a and the shares of $[x]$ and $[y]$ such that the parties obtain shares for $[z]$ where $z = a \cdot x + y$, also written as $[a \cdot x + y]$.

2.4. Secure Multi-Party Computation

Secure multi-party computation (SMPC) was introduced by Yao [Yao82] and aims to allow evaluation of a function on multiple secret inputs without leaking more information about the inputs than what can be deduced from the output. In particular, parties $P_1, P_2, ..., P_N$, where party P_i has some secret value x_i unknown to other parties, want to compute $\mathcal{F}(x_1, ..., x_N)$ for some function $\mathcal{F}$. The parties carry out the computation such that they do not get more information about other parties' secret inputs than what can be deduced by their own secret inputs and the output.

Canetti [Can00] gives a more recent definition of security regarding SMPC that is currently used as the standard definition [Lin20]. His definition is known under the name *ideal/real simulation paradigm* [Lin20]. It describes an ideal process where all inputs are sent to a unconditionally trusted third party that evaluates the function and returns the outputs to the parties. An adversary corrupts some of the parties and only learns their inputs and outputs and, depending on the exact setting, can manipulate their inputs. Thus, it cannot interfere with the function evaluation or learn any information from it going beyond what can be deduced from inputs and outputs of the corrupted parties. In a real setting, we assume that there is no trusted

third party available. Instead, the parties need to run some protocol consisting of instructions regarding local computations and communication between the parties to evaluate the function. Intuitively speaking, such a protocol is secure if it emulates the corresponding ideal process. This means that what can be achieved by an adversary in the real setting is no more than what can be achieved by any adversary for the ideal process.

Canetti's definition implies several properties of SMPC. *Privacy* is one of these properties and means that no party gains any information which cannot be derived from its own input and output as previously described [Lin20]. *Correctness* as another of these properties denotes that each party receives the correct output corresponding to the given inputs by all parties and the function to evaluate [Lin20]. Some further properties include that the adversary cannot choose inputs in relation to the inputs of uncorrupted parties and that it cannot interrupt the protocol [Lin20].

2.4.1. Behavior of the Adversary

The security of a protocol has to be proven under assumptions about the adversary's behavior. Canetti [Can00] and Lindell [Lin20] give an overview of possible assumptions to make. An adversary can be *passive* (also called *semi-honest*) or *active* (also called *malicious*). Passive adversaries are able to use all information known by the corrupted parties to deduce additional knowledge while not being able to modify their behavior. Active adversaries can also change the corrupted parties' behavior in the protocol. Compared to active adversaries, passive adversaries are significantly weaker [Kol06]. Two common corruption strategies are the *static* strategy and the *adaptive* strategy. For a static strategy, the adversary corrupts a set of parties which is fixed before the start of the protocol. An adaptive adversary can choose which parties to corrupt during the protocol. Each corrupted party also stays corrupted for the remaining protocol. It can also be limited how many parties can be corrupted by the adversary. If only strictly less than half of all parties can be corrupted, we speak of an *honest majority*.

Developing a protocol only secure against a passive adversary can be justified by multiple points. Such a protocol can be used as basis for the development of protocols against an active adversary [Kol06; Lin17; EKR19]. Based on the setting where the protocol is used, security against a passive adversary may also be sufficient given some level of mutual trust [Kol06; Lin17; EKR19]. Such trust can be given if the parties trust each other not to attack the protocol while they cannot guarantee that traces of the execution stay secret [Kol06; EKR19]. Note that keeping the traces secret may be difficult due to behavior of networks, memory, and operating systems in practice as it can be difficult to reliably erase them [Gol04; Kol06]. Trust that parties do not participate in active attacks can also be justified if the risk of being caught outweighs the gain given by an attack [Kol06]. As a final remark, active attacks in the context of complex systems may be significantly more difficult than only recording and analyzing traces of the computation [Gol04; Kol06].

In this thesis, we consider an adversary that is passive, static and has unbounded computational power. Furthermore, we assume an honest majority and the existence of private channels between all pairs of parties. Those private channels can be used for communication between the parties where an adversary cannot read from a channel between parties that are not corrupted. Setting up such channels in a realistic

setting can be achieved efficiently [EKR19] while limiting the security to security against an adversary being restricted to probabilistic polynomial time [Can00].

2.4.2. Definition of Security

Next, we give an exact definition of security. We mainly use the definitions given by Asharov and Lindell [AL17] for security in the presence of a passive and static adversary. If not stated otherwise, the setup and definitions given in this subsection are taken from their work [AL17] with some adjustment regarding the specific notations. We extend their definitions by a security parameter κ according to Lindell [Lin17] which will be needed to work with statistical indistinguishability. For N parties let the function to be computed be $\mathcal{F} : (\{0,1\}^*)^N \to (\{0,1\}^*)^N, (x_1, ..., x_N) \mapsto (\mathcal{F}_1(x_1, ..., x_N), ..., \mathcal{F}_N(x_1, ..., x_N))$. The function $\mathcal{F}$ may be probabilistic. For more concise notation, we write $\hat{x} := (x_1, ..., x_N)$. Let $I = \{i_1, ..., i_\ell\} \subseteq [1, N]$ be the indexes of parties corrupted by an adversary. As we assume an honest majority, let $\ell < N/2$. We define $\hat{x}_I := (x_{i_1}, ..., x_{i_\ell})$ and $\mathcal{F}_I(\hat{x}) := (\mathcal{F}_{i_1}(\hat{x}), ..., \mathcal{F}_{i_\ell}(\hat{x}))$.

For some execution of a protocol π, let x_i be the private input and r_i be the random tape of party P_i. Furthermore, let $m_{i,j}$ be the j'th message received by P_i during the protocol's execution. Then, the view of P_i on the protocol's execution is defined as $\text{VIEW}_i^\pi(\hat{x}, \kappa) := (x_i, r_i; m_{i,1}, ..., m_{i,k})$. For some adversary with given I, its view is defined as $\text{VIEW}_I^\pi(\hat{x}, \kappa) := (\text{VIEW}_{i_1}^\pi(\hat{x}, \kappa), ..., \text{VIEW}_{i_\ell}^\pi(\hat{x}, \kappa))$. The output of the execution of π consists of what each party perceives to be its output given its own view on the execution and is denoted by $\text{OUTPUT}^\pi(\hat{x}, \kappa)$.

We use Asharov and Lindell's [AL17] general security definition for deterministic and probabilistic $\mathcal{F}$. It requires that for a functionality and a corresponding protocol, there is a simulator $\mathcal{S}$ that can efficiently simulate the part of the protocol's execution that is visible to a set of corrupted parties I, i.e., the view of I. The simulator gets the private inputs and outputs for the corrupted parties as input. Simulation then means that the generated view of the corrupted parties cannot be distinguished from these parties' view on a real execution of the protocol. Intuitively, this implies that the view on a real protocol execution does not leak any knowledge that cannot be efficiently derived from the corrupted parties' inputs and outputs only as a simulator without additional knowledge could generate similar views [Lin17]. In addition, the outputs of the functionality and the protocol must be indistinguishable to ensure correctness. Views and outputs are statistically compared in a joined fashion such that a meaningful definition of security also can be achieved for a probabilistic functionality [Lin17].

In this thesis, we also use a relaxed variant of Asharov and Lindell's [AL17] definition called statistical security as described by Canetti [Can00] and Catrina and de Hoogh [CH10]. While the original definition is based on perfect indistinguishability, the relaxed variant only requires statistical indistinguishability. Both notions of security are formally defined as follows:

Definition 2.6 (Perfect and Statistical Security [AL17] (extended by κ as in [Lin17] for statistical security [Can00; CH10])). *Let $\mathcal{F} : (\{0,1\}^*)^N \to (\{0,1\}^*)^N$ be a possibly probabilistic functionality and π be a protocol. Protocol π computes $\mathcal{F}$ with perfect (statistical) security if there is a probabilistic polynomial-time algorithm $\mathcal{S}$, also called a simulator, such*

that for each $I \subseteq [1, N]$ with $|I| < N/2$, the probability ensembles

$$\{(\mathcal{S}(1^\kappa, I, \hat{x}_I, \mathcal{F}_I(\hat{x})), \mathcal{F}(\hat{x}))\}_{\hat{x},\kappa}$$

and

$$\{(\text{VIEW}_I^\pi(\hat{x}, \kappa), \text{OUTPUT}^\pi(\hat{x}, \kappa))\}_{\hat{x},\kappa}$$

are perfectly (statistically) indistinguishable where $\kappa \in \mathbb{N}$ and $x_1, ..., x_N \in \{0,1\}^*$ such that $|x_1| = ... = |x_N|$.

2.4.3. The BGW Protocol

Based on Shamir's secret sharing scheme, Ben-Or, Goldwasser and Wigderson introduced a secure protocol [BOGW88] which is known today as the *BGW protocol* [EKR19]. Assuming secure channels, the BGW protocol allows N parties to evaluate any arithmetic circuit in a secure fashion with a passive adversary and an honest majority or with an active adversary that can only control less than $N/3$ parties. We focus on a passive adversary according to our own security model. Perfect security, as in our definition, is formally proven for the passive adversary case of the BGW protocol and a deterministic functionality by Asharov and Lindell [AL17]. Equal results for a probabilistic functionality can be inferred by viewing the randomness as explicit random tape being an input variable of a deterministic functionality which can be calculated from different parties' inputs [Gol04].

The protocol is divided into three stages:

Input Stage In the input stage, all secret inputs are shared among all parties using Shamir's secret sharing scheme. For N parties, the BGW protocol requires that the threshold t of Shamir's secret sharing scheme is chosen such that $2t-1 \leq N$. This requirement is satisfied by choosing $t := \lceil \frac{N}{2} \rceil$ that also ensures that a dishonest minority cannot reveal a secret shared value. Recall that for all secret shared inputs, it holds that the coefficients of the used polynomial except for the secret value itself are distributed uniformly at random. If this property holds for some arbitrary polynomial representing a secret shared value, in the following we say that the polynomial is random.

Computation Stage In the computation stage, all gates are evaluated where intermediate values are secret shared values with the same properties as the inputs. It is sufficient to allow the calculation of $c \cdot [a] := [c \cdot a]$, $[a] + [b] := [a + b]$ and $[a] \cdot [b] := [a \cdot b]$ given secret shared values $[a]$ and $[b]$ and a constant $c \neq 0$. All calculations are in the field $\mathbb{F}_p$ used by Shamir's secret sharing scheme.

Output Stage In the output stage, the result of the computation is revealed according to Shamir's secret sharing scheme. It is also possible to reveal the result to a single party by letting all other parties secretly send their shares to the single party.

We proceed by giving a brief overview of how the BGW protocol's arithmetic operations can be implemented. In the following, let $a, b, c \in \mathbb{F}_p$.

An addition of secret shared values, i.e., $[a] + [b]$ can be achieved without communication. Let $q_a(X), q_b(X)$ be the polynomials representing the given secret shared

values. Furthermore, let $q_{a+b}(X) := q_a(X) + q_b(X)$. Clearly, $q_{a+b}(0) = q_a(0) + q_b(0)$ represents the target value $a + b$. The parties can obtain their shares of $q_{a+b}(X)$ by adding their shares of $q_a(X), q_b(X)$. Concerning the coefficients' distribution, $q_{a+b}(X)$ is random if at least $q_a(X)$ or $q_b(X)$ is. While the BGW protocol does not explicitly specify how to add a constant and a secret value, a similar approach can be applied to achieve such functionality. To compute $[a] + c$, it trivially is possible to just let all parties add c to their shares of $[a]$ yielding a random polynomial again.

The multiplication of a secret shared value $[a]$ by a constant c also is possible without communication. Similar to the addition, it is sufficient that each party multiplies its share of $[a]$ by c.

For the multiplication of two secret shared values $[a]$ and $[b]$, communication is necessary. With corresponding polynomials $q_a(X)$ and $q_b(X)$, let each party multiply its corresponding shares yielding a polynomial $q'_{a \cdot b}(X) := q_a(X) \cdot q_b(X)$. The degree of $q'_{a \cdot b}(X)$ can be up to $2t - 2$ so that further multiplications would eventually cause the result not to be uniquely identified by N shares anymore. Furthermore, it is not a random polynomial. Using $q'_{a \cdot b}(X)$, it is possible to obtain a random polynomial $q_{a \cdot b}(X)$ of degree at most $t - 1$ that represents $a \cdot b$ by applying a *randomization step* and a *degree reduction step*.

A simplification of the original multiplication is given by Gennaro et al. [GRR98]. Their solution replaces the randomization step and degree reduction step by a single step offering the same security properties as the original multiplication for a passive adversary. The idea of their solution essentially is to secret share the shares of polynomial $q'_{a \cdot b}(X)$ and apply Lagrange interpolation [EKR19]. Regarding communication, it only requires each party to secret share a single value.

For simplification of protocols, a basic extension of the BGW protocol is to allow for constant values to be shared. Given a constant value $c \in \mathbb{F}_p$, all shares can be initialized to c without communication. Trivially, the resulting shares represent the value correctly.

2.4.4. Data Representation

Gates that are not implemented by the original BGW protocol, for instance for secure comparison, may require a specific encoding of values [CH10]. Catrina and de Hoogh [CH10] define integers by a constant number of bits that we denote by b. The represented interval for some $b \in \mathbb{N}$ is $[-2^{b-1}, 2^{b-1} - 1]$. Using Shamir's secret sharing scheme on some field $\mathbb{F}_p$ requires that $p > 2^{b+\kappa}$ for a security parameter κ. A value $x \in [-2^{b-1}, 2^{b-1} - 1]$ is represented by $x \mod p \in \mathbb{F}_p$. Boolean values can simply be encoded as $0, 1 \in \mathbb{F}_p$. Note that the arithmetic operations that are supported by the BGW protocol as introduced in Section 2.4.3 can directly be executed in $\mathbb{F}_p$ instead of $[-2^{b-1}, 2^{b-1} - 1]$. In particular, for $x, y \in [-2^{b-1}, 2^{b-1} - 1]$ and their representations $x', y' \in \mathbb{F}_p$, it holds that $x' \odot y'$ is the $\mathbb{F}_p$ representation of $x \odot y$ in $[-2^{b-1}, 2^{b-1} - 1]$ for each of the BGW protocol's arithmetic operations "$\odot$". Therefore, these operations can be applied as if there was no designated representation being used.

2.4.5. Complexity Metrics

To assess the complexity of secure protocols, we use two complexity metrics described by Catrina and de Hoogh [CH10] that are the number of *invocations* and *rounds*. An

invocation is some operation where each party may send a share to each other party. A share refers to a single element of $\mathbb{F}_p$ that is used to share a secret value according to Shamir's secret sharing scheme. Note that a single invocation requires each party to send and receive at most $N - 1$ shares. The data to be sent and received by each party therefore linearly increases with an increasing number of invocations or an increasing number of parties.

Multiple invocations may be executed in parallel if none of these invocations depend on the result of another of the given invocations. In this context, parallelization means that the data of multiple invocations is sent simultaneously instead of executing each invocation only after the previous one has finished. Thus, it can be avoided that the network delay increases the overall run time for each single invocation. One or multiple simultaneously executed invocations are called a round. Parallelization does not influence the number of invocations while it decreases the number of required rounds. Thereby, it also reduces the run time overhead inflicted by network latency.

In gates and protocols that are introduced in Chapters 5 and 6, the annotation "in parallel" inside a loop always refers to the parallelization of invocations. If each iteration of such a loop requires k rounds, all invocations of round i in each iteration are executed in a single global round for $i \in [1, k]$.

Note that the operations in the BGW protocols that require communication are sharing secret inputs, revealing secret shared values, and multiplying secret shared values. Each of these operations only require a single invocation and round. Every party sharing or revealing a secret shared value can also be grouped into a single invocation according to the definition of invocations as each party exactly sends one share to and receives one share from each other party.

2.4.6. Obliviousness in Secure Multi-Party Computation

When running SMPC in a *random-access machine* (RAM)-like environment, it may become inefficient to access registers under certain conditions as described by Evans et al. [EKR19]. If some secret register with index $[i]$ is to be accessed, trivially accessing register i in $\mathcal{O}(1)$ is not possible as $[i]$ would have to be revealed. Even if there was a technique not requiring to reveal $[i]$, the memory access pattern could leak its value. In general, the requirement that a RAM algorithm does not leak information about its specific inputs by its memory access pattern can be formalized as follows:

Definition 2.7 (Oblivious RAM Algorithm [GO96])**.** *An algorithm running on some RAM is called* oblivious *if the pattern of memory accesses is equal for all inputs of the same length.*[1] *The memory access pattern is given by the sequence of indexes used for register accesses.*

Evans et al. [EKR19] describe how some register $[i]$ holding a secret shared value can be accessed in an oblivious way. It is possible to realize such functionality without revealing any secret shared value by accessing all registers. For the access of a single

[1] The original definition of Goldreich and Ostrovsky [GO96] actually requires equal patterns for equal run times instead. The difference regarding definition is not relevant in this thesis as we only work with oblivious algorithms having the same run time for same input length and different run time for different input length.

register, a building block is required that provides the required functionality at register i while leaving all other secret shared values unchanged (while their shares of course may change). Note that this approach causes the number of operations required for a single oblivious register access to scale linearly in the number of used registers instead of being constant.

2.5. Graph Theory

In general, a graph is a tuple (V, E) where V is a set of *nodes* and E is a set of *edges* between these nodes. We restrict the definition of graphs to *finite undirected graphs*. This means that there is a finite number of nodes $N := |V|$ and each edge is a set of two nodes. Formally, $E \subseteq \{\{v, w\} \mid v, w \in V \wedge v \neq w\}$. Without loss of generality, we let $V := [1, N]$. It follows that the number of edges $M := |E|$ is also finite. For an edge $\{v, w\}$, we also use the notation (v, w) if $v < w$. This notation is unique for a given edge (v, w) which could be written as $\{v, w\}$ and $\{w, v\}$ in standard notation. Uniqueness of the notation becomes useful for more compact notation and indexing purposes for instance. We extend the definition of a graph by a weight function $\sigma : E \to \mathbb{N}$ yielding a *weighted graph* $G = (V, E, \sigma)$. Furthermore, we extend this weight function's definition by allowing it to be applied to arbitrary sets of edges $E' \subseteq E$: $\sigma(E') := \sum_{e \in E'} \sigma(e)$.

For a given graph G, a *matching* is a selection of edges of G such that each node is assigned to exactly one or no other node by these edges.

Definition 2.8 (Matching [Edm65b])**.** *A* matching *of a (weighted) graph with edges E is a set of edges $\mathfrak{M} \subseteq E$ such that $\forall v \in V$ it holds that $|\{e \in \mathfrak{M} \mid v \in e\}| \leq 1$.*

There exist several optimization problems concerning matchings. One of those problems appearing in this thesis is defined on unweighted and weighted graphs. Its goal is to maximize the cardinality of the resulting $\mathfrak{M}$.

Definition 2.9 (Maximum Matching [Edm65b])**.** *A matching $\mathfrak{M}$ of some graph G is a* maximum matching *if for all matchings $\mathfrak{M}'$ of G it holds that $|\mathfrak{M}'| \leq |\mathfrak{M}|$.*

For weighted graphs, another optimization problem is to maximize the sum of weights over all edges in the matching.

Definition 2.10 (Maximum Weight Matching [Edm65a])**.** *A* maximum weight matching *(MWM) of some weighted graph $G = (V, E, \sigma)$ is a matching $\mathfrak{M}$ of (V, E) such that $\sigma(\mathfrak{M}) = \sum_{e \in \mathfrak{M}} \sigma(e)$ is maximal among all possible matchings of (V, E).*

Besides searching for matchings of maximal cardinality or weight, approximation of those may for instance allow for application of simpler or faster algorithms. In this case, approximation means to find a matching offering at least some fraction of the quality given by the maximum or maximum weight matching.

Definition 2.11 (c-Approximation for Maximum (Weight) Matching [DH05])**.** *An algorithm A for maximum (weight) matching is a c-approximation if for all (weighted) graphs G with maximum (weight) matching $\mathfrak{M}^*$, it holds that*

$$g(A(G)) \geq c \cdot g(\mathfrak{M}^*)$$

where $A(G)$ is the matching computed by A on input G. For matching $\mathfrak{M}$, $g(\mathfrak{M}) := |\mathfrak{M}|$ for the maximum matching case and $g(\mathfrak{M}) := \sigma(\mathfrak{M})$ for the MWM case. The constant c is called the algorithm's approximation ratio.

If a weighted graph is used in SMPC where the number of nodes is public but E and the weights are secret, an alternative representation that we call *secret graph* later becomes useful concerning obliviousness. A secret graph $\hat{G}$ of a weighted graph $G = (V, E, \sigma)$ is a tuple $(V, \hat{E}, \hat{\sigma})$ such that $\hat{E} = \{\{v, w\} \mid v, w \in V \wedge v \neq w\}$ and

$$\hat{\sigma} : \hat{E} \to \mathbb{N}_0, e \mapsto \begin{cases} \sigma(e), & \text{if } e \in E, \\ 0, & \text{otherwise.} \end{cases}$$

This representation encodes the existence of an edge in the weight function while $\hat{E}$ encodes the set of all possible edges and depends on N only. We call all elements of $\hat{E}$ (*potential*) *edges* and an edge $e \in \hat{E}$ a *real edge* if $\hat{\sigma}(e) > 0$ and therefore $e \in E$. Let $\hat{M} := |\hat{E}|$ be the number of potential edges. It holds that $\hat{M} = \binom{N}{2} = \frac{N^2-N}{2} \in \Theta(N^2)$.

If each weight can be encoded with some fixed length ℓ, the whole graph can be encoded with length $\hat{M} \cdot \ell$. Thus, there is a one-to-one relation between values for N and possible input lengths and N can be used to indicate the input length instead. Inputs of the same length therefore only differ in their edge weights that also encode whether some edge is a real edge. It follows that for fixed ℓ, an algorithm on some RAM getting such a graph encoding as input is oblivious if it does not access memory locations depending on edge weights. Towards SMPC, it is easy to encode secret weights and secret edge existence by secret sharing all values $\hat{\sigma}(e)$ for $e \in \hat{E}$.

Note that the definitions for matchings and corresponding problems can also be applied to secret graphs $\hat{G} = (V, \hat{E}, \hat{\sigma})$ by letting $\mathfrak{M} \subseteq E := \{e \in \hat{E} \mid \hat{\sigma}(e) > 0\}$. By this definition we only allow real edges to be part of a matching while all zero weight edges are ignored.

Chapter 3

Related Work

In this chapter, we review existing works in the domains of MWM algorithms and approximations (Section 3.1), secure protocols for matching problems (Section 3.2), conventional kidney exchange (Section 3.3) and secure protocols for bartering and kidney exchange (Section 3.4). In particular, we identify approaches from the different fields that can act as the foundation of our work while discussing why existing approaches cannot immediately be utilized for our use case.

3.1. Maximum Weight Matching Algorithms

In 1965, Edmonds [Edm65a] published the first [Gab90] algorithm to compute MWMs on general graphs in polynomial time. The algorithm's time complexity is $\mathcal{O}(M \cdot N^2)$ [DP14]. It is heavily based on Edmonds' [Edm65b] algorithm for maximum matchings on general graphs. Both algorithms are also known under the name *blossom algorithm* [Kol09].

Edmonds' results have been improved multiple times [e.g., Gab90; GT91; HK12; Pet12]. Without requiring randomization or restrictions on the weights of edges[1], the time complexity for MWM has been improved to $\mathcal{O}(M \cdot N + N^2 \cdot \log N)$ by Gabow [Gab90]. Only allowing integer edge weights where I_{max} is the maximum integer value among all edge weights allows solutions with time complexities $\mathcal{O}(M \cdot \sqrt{N \cdot \log N} \cdot \log(I_{\text{max}} \cdot N))$ [GT91] and $\mathcal{O}(I_{\text{max}} \cdot M \cdot \sqrt{N} \cdot \log(N^2/M)/\log N)$ [HK12]. It is even possible to use randomization to achieve a time complexity of $\mathcal{O}(I_{\text{max}} \cdot N^{\omega})$ where ω is the exponent of square matrix multiplication [Pet12]. Note that it holds that $\omega < 2.38$ [CW90].

There are multiple publications solely dedicated to an efficient implementation of Edmonds' blossom algorithm using sophisticated improvements of its original definition [e.g., CR99; Kol09]. While being polynomially bounded, the resulting run times may still be too high for graphs with a high number of nodes [DH03a]. Especially in SMPC, complex problems or large inputs can easily yield infeasible run times [Lin20]. Thus, we focus on less complex and more efficient algorithms for MWM.

[1] Weights of graph edges are restricted to positive integer values in our work while this is not the general case in literature [DP14].

3.1.1. Maximum Weight Matching Approximation

To allow for simpler and faster algorithms to be used, it is possible not to require solutions to be optimal [DH03a]. In this case, approximation algorithms can be applied.

A trivial approximation algorithm for MWM is a greedy algorithm as documented by Avis [Avi83] and Preis [Pre99].[2] It runs in time $\mathcal{O}(M \cdot \log N)$ and is a $1/2$-approximation algorithm [Pre99]. In each iteration, the algorithm selects a heaviest edge as part of the matching and removes all incident edges. Due to this simple selection strategy, it appears to be well suited to act as the foundation of a secure protocol. In particular, obliviously searching for a heaviest edge by accessing all potential edges has a time complexity of $\mathcal{O}(N^2)$. Removing incident edges has the same complexity by accessing all edges. As only $\mathcal{O}(N)$ edges can be added to a matching, it appears that an oblivious variant of the greedy algorithm can be designed to have a time complexity of $\mathcal{O}(N^3)$.

Further $1/2$-approximation algorithms that have a time complexity of $\mathcal{O}(M)$ are given by Preis [Pre99] and Drake and Hougardy [DH03a]. The algorithm by Preis essentially tries to add edges such that each added edge has a higher weight than all edges incident to it. It uses multiple nested loops, recursive calls and conditional blocks that depend on the exact input. Therefore, the algorithm is not oblivious and makes it difficult to assess how it could be modified to be oblivious. The algorithm by Drake and Hougardy does not require recursive calls and its formulation is significantly shorter than the one that Preis gives for his algorithm. It iterates over a path where the next edge is always chosen to be the heaviest one among all edges that can extend the path. This procedure is possibly repeated multiple times and the edges of the paths are added to two possible matchings in an alternating fashion. Finally, the matching that achieves the higher weight is returned. The algorithm still uses two nested loops and requires to search for a heaviest edge in the inner one. While this heaviest edge only extends the current path, an oblivious search for it has a time complexity of $\mathcal{O}(N^2)$ as the path depends on the exact input. As both loops can be executed for $\mathcal{O}(N)$ times each, a naïve oblivious variant of the algorithm already has a time complexity of at least $\mathcal{O}(N^4)$. Thus, the algorithm by Drake and Hougardy appears to be less suitable for SMPC compared to the greedy algorithm due to having the same approximation ratio but presumably a higher time complexity if modified to be oblivious.

For arbitrary $\epsilon > 0$, there also exist $(2/3 - \epsilon)$-approximations by Drake and Hougardy [DH03b][3] and more recently also by Pettie and Sanders [PS04]. The time complexity of the approach by Drake and Hougardy is $\mathcal{O}(\frac{1}{\epsilon} \cdot M)$ while Pettie and Sanders offer $\mathcal{O}(\log \frac{1}{\epsilon} \cdot M)$. Both approaches execute multiple iterations where each one increases the weight of an intermediate matching using *augmenting paths and cycles* bounded to a fixed length. Over the iterations, the weight of the intermediate matching converges towards $2/3$ of the optimal weight. Finding good augmenting paths and cycles in each iteration requires more sophisticated approaches than the previously discussed greedy algorithm and the $1/2$-approximation algorithm by Drake and Hougardy [DH03a] do. In addition, the corresponding approaches

[2] The greedy algorithm's exact origin appears to be unclear, and it is labeled as folklore and trivial by Duan and Pettie [DP14].

[3] Parts of the paper are faulty but Drake and Hougardy published a corrected version [DH05].

include techniques like extensively marking specific nodes and traversing paths. Implementing such techniques in an oblivious fashion is to be expected to require a significant number of oblivious register accesses as discussed in Section 2.4.6. Thus, a secure protocol based on these approaches would likely have a considerably higher time complexity than a linear one in the number of edges.

The idea of using augmenting paths and cycles of a bounded length to gradually improve a matching can be generalized by allowing longer paths and cycles to be used [PS04]. This generalization yields further approximations like a $(3/4 - \epsilon)$-approximation with time complexity $\mathcal{O}(\log \frac{1}{\epsilon} \cdot M \cdot \log N)$ and a $(4/5 - \epsilon)$-approximation with complexity $\mathcal{O}(\log \frac{1}{\epsilon} \cdot M \cdot \log^2 N)$, both proposed by Hanke and Hougardy [HH10]. Searching for longer augmenting paths and cycles requires more sophisticated optimizations than the $(2/3 - \epsilon)$-approximations while also having a higher time complexity. Thus, a secure protocol based on this approach can be expected to be significantly less feasible than for the $(2/3 - \epsilon)$-approximations.

Duan and Pettie [DP14] have shown that for each $\epsilon > 0$, there even is a $(1 - \epsilon)$-approximation algorithm for MWM running in $\mathcal{O}(\frac{1}{\epsilon} \cdot \log \frac{1}{\epsilon} \cdot M)$ in general or $\mathcal{O}(\frac{1}{\epsilon} \cdot \log I_{\max} \cdot M)$ for integer weights with maximum weight $I_{\max}$. Their approach is far more complex than the previously mentioned approximations such that the feasibility of using it as the foundation for a secure protocol is strongly questionable.

3.2. Secure Protocols for Matching Problems

To the best of our knowledge, to date there are no secure protocols regarding matching problems on general graphs. However, multiple protocols for diverse matching problems on *bipartite graphs* exist.

A bipartite graph is a graph that allows to partition its nodes into two sets such that no edge connects two nodes in the same set [Die17]. Bipartite graphs allow for different algorithms than the blossom algorithm [Edm65b] to be used. For instance, a well-known MWM algorithm for bipartite graphs is the *Hungarian method* [Kuh55].

Some matching problems on bipartite graphs can be solved by utilizing flow or shortest path algorithms [e.g., EK72; AC17]. There exists a selection of secure protocols for *shortest path* and *maximum flow* problems given by Aly et al. [Aly+13] and Blanton et al. [BSA13]. For instance, secure protocols for maximum flow can be used to build secure protocols for maximum bipartite matching [AC17].

In 2015, Blanton and Saraph [BS15] published a secure protocol for maximum matching on a bipartite graph. Instead of explicitly computing it, the protocol only determines the size of a maximum matching. Another secure protocol by Doerner et al. [DEs16] can be used to explicitly compute a stable matching. Stable matchings are a special type of bipartite matchings where each node in one of the partition's sets has a list of all nodes of the other set ordered by preference. The goal is to match all nodes such that there exist no two nodes v, w in opposite sets of the partition that are not matched to each other but where both v and w would prefer to be matched to each other compared to their actual matched node.

In 2017, Anandan and Clifton [AC17] proposed a secure protocol for *minimum weight perfect matchings* on bipartite graphs. The underlying problem is to find a matching of minimal weight among all matchings where each node is covered by the matching. Their solution is based on the Hungarian method and utilizes the

protocol for shortest paths given by Blanton et al. [BSA13].

Wüller et al. [Wü+17] published a protocol for bartering that is based on finding a *maximum weight perfect matching* on a bipartite graph, i.e., a matching of maximal weight that covers all nodes. Their approach is also based on the Hungarian method. It utilizes the protocol for maximum flow by Blanton et al. [BSA13].

To conclude, none of the mentioned secure protocols can act as the foundation for our protocol where we use general graphs. This is due to the fact that all of the existing protocols for matching strongly rely on the graphs being bipartite.

3.3. Conventional Kidney Exchange

To enable living donor kidney donations for multiple parties, i.e., pairs of a patient and her willing but medically incompatible donor, kidney exchanges are conducted [ABS07]. Abraham et al. [ABS07] define the *kidney exchange problem* (KEP) which is an optimization problem for selecting a possible kidney exchange that maximizes the overall utility. To this end, kidney exchange is considered a special case of bartering and modeled as a graph problem. The so-called *compatibility graph* (CG) for a fixed set of parties is a *directed graph* that is defined as follows: Each party P_i is represented by a node i. For parties P_i and P_j, a *directed arc* (that is not to be confused with our definition of undirected edges) goes from node i to node j if and only if the donor of party P_i is compatible and can therefore donate to party P_j. The arc is assigned a weight expressing the utility of the represented donation. Then, a kidney exchange is a set of disjoint cycles in such graph. If the utility of each arc is set to 1, the goal of the KEP therefore is to maximize the number of patients receiving a kidney transplant.

Compatibility between a donor and a patient is usually determined by compatibility regarding *ABO blood type* and *Human Leukocyte Antigens* (HLAs) [Bir+21]. Kidney transplantation despite incompatibility can result in severe reactions including rejection of the transplant and death of the patient [Dea05; Cho07]. It is possible to assess compatibility using a *virtual crossmatch test* where separately obtained medical data about a donor and a patient is compared [Bir+21]. A virtual crossmatch test is easier to execute while only being able to estimate the result of a laboratory crossmatch test which is necessary to perform before executing a transplantation [Bir+21].

It is possible to use different objectives besides the maximization of the number of executed kidney transplants alone [Bir+21]. Such objectives that are used in modern kidney exchange programs include but are not limited to the following [Bir+21]:

- Directly or indirectly minimizing the size of selected cycles (e.g., Czech Republic, the Netherlands, Spain, United Kingdom).
- Minimizing the age differences between patients and the donors designated to donate to them (e.g., Belgium, Poland, Spain).
- Prioritizing *highly sensitized* patients (e.g., Poland, Spain, United Kingdom). Highly sensitized patients are patients with a high amount of HLA antibodies and therefore significantly lower chances of being compatible to some donor where the exact definition depends on the specific kidney exchange program.

- Prioritizing patients with a long waiting time in a kidney exchange program (e.g., Spain, United Kingdom).
- Prioritizing patients with a long waiting time on dialysis (e.g., Belgium, the Netherlands, Portugal, Spain).
- Prioritizing donations from a patient to a donor in the same region (e.g., Spain).

To solve the KEP, it is common to use integer programming [e.g., ABS07; RSn07]. Possible formulations of the KEP using integer programming are the *edge formulation* and the *cycle formulation* [ABS07]. The edge formulation introduces a binary variable for each arc indicating if it is selected as part of some cycle. Constraints are used to ensure that only cycles and no paths can be selected and that cycles do not overlap. Additional constraints need to be applied if the length of the chosen cycles is to be bounded. The cycle formulation introduces a binary variable for each possible cycle indicating if it is selected. Constraints ensure that no overlapping cycles are selected. The maximal cycle length can be restricted by only introducing variables for sufficiently small cycles.

Abraham et al. [ABS07] have analyzed how limitations of the lengths of cycles influences the KEP's complexity. If the length of cycles is not limited, the problem can be solved in polynomial time by applying an algorithm for maximum weight perfect matching on bipartite graphs. This approach is unsuitable for our approach as limiting the cycle length is a usual requirement of kidney exchange programs [Bir+21] and especially contradicts our goal of implementing a protocol for crossover exchange. Only allowing cycles of length two allows for solving the problem in polynomial time using an MWM algorithm as introduced in Section 3.1. Note that this case exactly corresponds to crossover exchange and therefore matches our use case. For a fixed limit $L > 2$ of the cycles' lengths, the KEP is NP-complete.

If the length of cycles is bounded to some $L > 2$, the cycle formulation is significantly more efficient than the edge formulation [ABS07; Con+13]. Another exact approach is to use enumeration methods [Bir+21]. Both approaches are not suitable for our use case due to requiring superpolynomial time while crossover exchanges can be represented by matchings and therefore can be found in polynomial time. As using the cycle formulation still solves an NP-complete problem, for instance the Spanish kidney exchange program utilizes a polynomial time heuristic [Bof+17; Bir+21]. This heuristic does also not match our use case due to being tailored to also support cycles of length three. Instead of using a modification of it, it is possible to immediately use an algorithm specifically designed to find matchings.

3.4. Secure Protocols for Bartering and the KEP

Kidney exchange is considered a special case of bartering between multiple parties [ABS07]. Bartering is a process where commodities are traded without involvement of a currency [FO08]. In kidney exchange, patients represent agents in a market based on bartering and instead of trading with arbitrary commodities, incompatible donors are exchanged [ABS07]. Thus, protocols for privacy-preserving bartering are also of interest for solving the KEP in a privacy-preserving manner.

Kannan et al. [Kan+18] describe a protocol for privacy-preserving multi-party bartering. The protocol approximates a solution where each different solution would

be worse for at least one party. While offer and demand of each party are kept private, the set of commodities is public. In the case of kidney exchange where donors are exchanged instead of commodities, their medical characteristics are to be kept secret. Thus, the protocol by Kannan et al. is not applicable to our use case.

In the line of work by Wüller et al. [WMW17a; WMW17b; Wü18], two secure protocols for multi-party bartering are presented. One protocol [WMW17b; Wü18] is secure against a passive adversary while the other one [WMW17a; Wü18] even provides security against an active adversary. The protocols use CGs where each party is represented by a node. For parties P_i and P_j, a directed arc goes from node i to node j if and only if party P_i's offer can satisfy the demand of party P_j. A set of disjoint exchange cycles on the graph is drawn uniformly at random from all such sets that maximize some welfare function. To search for sets of disjoint exchange cycles, for all such sets on a complete graph of the size of the CG it is tested if all arcs of the cycles are also present in the CG. Note that the arcs are also given in the CG exactly if the cycles represent trades where it is possible to trade along these cycles' arcs. It is possible to only use a subset of exchange cycle constellations, for instance to limit the length of exchange cycles.

Breuer et al. [Bre+20] modify the protocol by Wüller et al. [WMW17b] providing security against a passive adversary to obtain a protocol that can be used for the KEP. Note that the definition of CGs by Wüller et al. already matches the definition of CGs used for the KEP. In this case, offer and demand correspond to the relevant medical characteristics of each party's patient and donor. They use a welfare function that maximizes the number of selected edges and thus the number of patients that may receive a kidney transplant. Prioritization is not applied. The protocol is evaluated for exchange cycles with length limited to three. There is an exponential increase of run time for an increasing number of parties due to the exponentially increasing number of exchange cycle constellations. Breuer et al. measured a run time of two hours for eight parties and a run time of already 13 hours for nine parties.

The protocol by Breuer et al. [Bre+20] and therefore also the discussed protocols by Wüller et al. [WMW17a; WMW17b; Wü18] are not suitable for our use case as they are designed to support cycles of lengths larger than two. Thus, the problem of finding an optimal set of exchange cycles is NP-complete [ABS07]. The measured run times by Breuer et al. confirm that the resulting run time grows exponentially. Restricting all exchanges to only two participating parties may decrease the run time while implying that the protocols simply enumerate all possible matchings. In this case, we reason that the use of an algorithm designed specifically for a matching problem is more suitable. In addition, we state that the protocol by Breuer et al. does not allow for prioritization as required by our use case.

Another multi-party bartering approach by Wüller et al. [Wü+17; Wü18] is based on maximum weight perfect matchings for bipartite graphs. For each party, two distinct nodes are introduced. These nodes are connected by an (undirected) edge of weight 1. In addition, the first node of a party P_i is connected to the second node of a party P_j by an edge of weight 2 if party P_i's offer satisfies the demand of party P_j. Then, a maximum weight perfect matching is computed on the resulting graph as discussed in Section 3.2. The selection of an edge connecting the same party's nodes represents that the party does not take part in an exchange. Other edges in the matching represent the part of an exchange where a commodity is transferred along the edge. Note that this approach exactly corresponds to one by Abraham

et al. [ABS07] as discussed in Section 3.3. The protocol's complexity is polynomial in contrast to the previously discussed ones by Wüller et al. [WMW17a; WMW17b; Wü18]. However, it performs significantly worse for a number of parties where the resulting run times still are adequately small so that for six parties, the run time already exceeds two days [Wü18]. Due to its high run time and the resulting low number of parties that can take part in a computation, the approach appears not to be suited for our use case. In addition, it does not allow to limit the length of exchange cycles as also stated in Section 3.3 such that it is not applicable in the case of crossover exchange.

Chapter 4

Ideal Functionality for Maximum Weight Matching Approximation

In this chapter, we introduce the ideal functionality for MWM that is to be implemented by our protocol. To this end, we select and modify an approximation algorithm for MWM in Section 4.1. Based on the approximation algorithm, we introduce the ideal functionality for a generic graph generation in Section 4.2. In Section 4.3, we conclude the chapter by reviewing how our functionality can be used for crossover exchange.

4.1. MWM Approximation Algorithm

In Section 3.1.1, we have discussed multiple approximation algorithms for MWM that achieve different approximation ratios. Recall that a protocol based on one of the discussed algorithms achieving higher approximation ratios than $1/2$ is expected to have a significantly higher run time than a protocol based on a simpler approximation. Regarding $1/2$-approximation algorithms, we have identified the greedy MWM approximation algorithm [Avi83; Pre99] to be the most promising one to base a secure protocol on. We proceed by formally introducing the greedy algorithm that our ideal functionality and protocol are based on. After further considerations we can then propose a modification that finally yields our ideal functionality.

The greedy algorithm as formulated by Preis [Pre99] is given in Algorithm 4.1. It gets a weighted graph as input and executes multiple iterations to construct a matching that initially is an empty set. In each iteration, it selects an edge $\{a, b\}$ of maximal weight and adds it to the matching. After adding the edge, all incident edges and the selected edge itself are removed from the graph's set of edges. This step guarantees that no edge incident to an edge in the matching can be selected which would yield a result that is no matching. The addition of edges is repeated until no edges of the graph remain. Note that the algorithm computes a maximal matching, i.e., a matching where for each $e \in E \setminus \mathfrak{M}$, $\mathfrak{M} \cup \{e\}$ is no matching. Thus, it also is a $1/2$-approximation algorithm for maximum matching [Hou09].

Algorithm 4.1 only gives a high-level view and does not already fix the exact input-output behavior of the algorithm. The edge $\{a, b\}$ that is to be selected in

each iteration may not be uniquely identified due to multiple edges of maximal weight existing in the current state. For instance, on a graph with three nodes and edges $\{1,2\}, \{2,3\}$ of equal weight, matchings $\{\{1,2\}\}$ and $\{\{2,3\}\}$ both are possible outputs.

Algorithm 4.1 Greedy MWM Algorithm [Pre99]

$\mathfrak{M} \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
  Let $\{a,b\} \in E$ be an edge of maximal weight
  $\mathfrak{M} \leftarrow \mathfrak{M} \cup \{\{a,b\}\}$
  $E \leftarrow E \setminus \{\{v,w\} \in E \mid \{v,w\} \cap \{a,b\} \neq \emptyset\}$
end while
return $\mathfrak{M}$

It is easy to see that specifying which edge $\{a,b\}$ among possibly multiple edges of maximal weight is to be selected in each iteration yields a fixed input-output behavior. This can be achieved by introducing some fixed ordering of edges. Let such ordering for a graph with N nodes be defined as follows:

$$\{1,2\} \prec_d \{1,3\} \prec_d ... \prec_d \{1,N\} \prec_d \{2,3\} \prec_d \{2,4\} \prec_d ... \prec_d \{2,N\} \prec_d ...$$

Such ordering then allows to select the edge $\{a,b\}$ that is minimal with respect to "$\prec_d$" among all edges of maximal weight in each iteration. The resulting deterministic algorithm is given in Algorithm 4.2. As it is specified inside the framework provided by Algorithm 4.1, it trivially still has an approximation ratio of $1/2$.

Algorithm 4.2 Strictly Deterministic Greedy MWM Algorithm (based on [Pre99])

$\mathfrak{M} \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
  Let $E_{\text{max}} \subseteq E$ exactly contain all edges of maximal weight in E
  Let $\{a,b\} \in E_{\text{max}}$ be the minimal edge in E_{max} w.r.t. "$\prec_d$"
  $\mathfrak{M} \leftarrow \mathfrak{M} \cup \{\{a,b\}\}$
  $E \leftarrow E \setminus \{\{v,w\} \in E \mid \{v,w\} \cap \{a,b\} \neq \emptyset\}$
end while
return $\mathfrak{M}$

Depending on the specific graph instances, the outputs of Algorithm 4.2 may be biased in favor of nodes represented by low values. For instance, on a graph with three nodes and edges $\{1,2\}, \{2,3\}$ of equal weight, the edge $\{1,2\}$ is added to the matching in the first iteration. As edge $\{2,3\}$ is blocked by selection of edge $\{1,2\}$, the resulting matching is $\mathfrak{M} = \{\{1,2\}\}$. Thus, the algorithm matches node 1 instead of node 3 to node 2 despite both cases being symmetrical aside from the values representing the nodes.

Note that such behavior may be unwanted in a case where multiple parties are represented by nodes and compete in getting matched to some other party. This kind of competition may especially exist for crossover exchange due to each party's possibly vital significance of getting matched. In general, let there be multiple parties where party P_i is represented by node i and an MWM is to be approximated by

Algorithm 4.2. Then, a bias in favor of party P_1 and weaker bias for further parties with low indexes may exist while solely resulting from the indexing of parties instead of their specific inputs.

To define the absence of a bias resulting from the specific values of nodes later, we use permutations on the set of all nodes to formally describe such bias. Let there be a weighted graph $G = (V, E, \sigma)$ and a permutation $\tau \in S_V$. Note that τ induces a graph isomorphism between G and a weighted graph $G' = (V, E', \sigma')$ that is defined as follows:

$$E' := \{\{\tau(v), \tau(w)\} \mid \{v, w\} \in E\},$$
$$\sigma' : E' \to \mathbb{N}, \{v, w\} \mapsto \sigma(\{\tau^{-1}(v), \tau^{-1}(w)\}).$$

Thus, each edge $\{v, w\}$ of G is exactly represented by one edge $\{\tau(v), \tau(w)\}$ of equal weight in G'. We write $\tau(G) := G'$. In addition, for some matching $\mathfrak{M}$ on G we write $\tau(\mathfrak{M}) := \{\{\tau(v), \tau(w)\} \mid \{v, w\} \in \mathfrak{M}\}$. Clearly, these operations can be inverted using τ^{-1}, i.e., $\tau(\tau^{-1}(G)) = \tau^{-1}(\tau(G)) = G$ and $\tau(\tau^{-1}(\mathfrak{M})) = \tau^{-1}(\tau(\mathfrak{M})) = \mathfrak{M}$.

Now, we can define the discussed bias. For an algorithm, there is such bias if there is a weighted graph $G = (V, E, \sigma)$ and a permutation $\tau \in S_V$ such that directly running the algorithm on G yields differently distributed matchings than running it on $\tau(G)$ and applying τ^{-1} to its output. In other words, for isomorphic graphs G and $\tau(G)$, the algorithm's output on G is not equally likely to be $\mathfrak{M}$ as its output on $\tau(G)$ is to be $\tau(\mathfrak{M})$. The formal definition of the absence of such bias is as follows:

Definition 4.1 (Invariance Under Permutation)**.** *Let there be a probabilistic algorithm A computing matchings on weighted graphs and let $A(G)$ be the algorithm's output on a weighted graph G. The algorithm is invariant under permutation if for each input $G = (V, E, \sigma)$ and each permutation $\tau \in S_V$, it holds that the distributions $A(G)$ and $\tau^{-1}(A(\tau(G)))$ are equal.*

We stress that only requiring Algorithm 4.1 to be invariant under permutation is not sufficient to precisely define an ideal functionality. This is the case as there are variants of Algorithm 4.1 that are invariant under permutation but have different input-output behaviors. In the following, we define a variant of Algorithm 4.1 that has a fixed but probabilistic input-output behavior and is invariant under permutation. This algorithm then is used to define the ideal functionality. In Appendix A, we sketch another variant of the algorithm that also is invariant under permutation but has a different input-output behavior.

A straightforward solution to prevent any bias depending on the values representing the nodes is to use a random permutation on such values. For a weighted graph $G = (V, E, \sigma)$, permutation τ is picked uniformly at random from S_V. Using Algorithm 4.2 on $G' := \tau(G)$ then yields a matching $\mathfrak{M}'$. Finally, a matching $\mathfrak{M} := \tau^{-1}(\mathfrak{M}')$ on G is induced. Thus, Algorithm 4.2 is used on a random graph among all graphs that are isomorphic to G to obtain a matching on G. A detailed version of this approach is given in Algorithm 4.3. Again, it is trivial that it has an approximation ratio of $1/2$ regarding MWM.

Using a setting where parties $P_1, ..., P_N$ are represented by nodes $1, ..., N$ now does not yield a bias in favor of parties with low indexes. This is due to the presented approach being equivalent to running Algorithm 4.2 on a graph where party P_i is represented by a random node $\tau(i)$ and edges and weights are adapted as discussed

Algorithm 4.3 Greedy MWM Algorithm Using Node Permutation (based on [Pre99])

Draw permutation τ uniformly at random from S_V
$E' \leftarrow \{\{\tau(v), \tau(w) \mid \{v, w\} \in E\}\}$
$\sigma' : E' \to \mathbb{N}, \{v, w\} \mapsto \sigma(\{\tau^{-1}(v), \tau^{-1}(w)\})$
$\mathfrak{M}' \leftarrow \emptyset$
while $E' \neq \emptyset$ **do**
  Let $E'_{\text{max}} \subseteq E'$ exactly contain all edges of maximal weight w.r.t. σ' in E'
  Let $\{a', b'\} \in E'_{\text{max}}$ be the minimal edge in E'_{max} w.r.t. "$\prec_d$"
  $\mathfrak{M}' \leftarrow \mathfrak{M}' \cup \{\{a', b'\}\}$
  $E' \leftarrow E' \setminus \{\{v, w\} \in E' \mid \{v, w\} \cap \{a', b'\} \neq \emptyset\}$
end while
$\mathfrak{M} \leftarrow \{\{\tau^{-1}(v), \tau^{-1}(w) \mid \{v, w\} \in \mathfrak{M}'\}\}$
return $\mathfrak{M}$

previously. We proceed by formally proving that Algorithm 4.3 is invariant under permutation.

Proposition 4.1. *Algorithm 4.3 is invariant under permutation.*

Proof. Let A be Algorithm 4.3. For some weighted graph G with nodes V, A picks a permutation τ uniformly at random from S_V, deterministically computes a matching on $\tau(G)$, and then applies τ^{-1} to the matching. Formally, let g be the deterministic mapping from a permuted graph to its matching. Then, it holds that $A(G) = \tau^{-1}(g(\tau(G)))$ with τ drawn uniformly at random from S_V.

Now, let $\varphi \in S_V$ be arbitrary but fixed. It holds that

$$\begin{aligned}\varphi^{-1}(A(\varphi(G))) &= \varphi^{-1}(\tau^{-1}(g(\tau(\varphi(G))))) \\ &= (\tau \circ \varphi)^{-1}(g((\tau \circ \varphi)(G)))\end{aligned}$$

with τ drawn uniformly at random from S_V. Note that $\psi : S_V \to S_V, \tau \mapsto \tau \circ \varphi$ is a bijection. Thus, the distributions of τ and $\tau \circ \varphi$ are equal as τ is drawn uniformly at random from S_V. Then, we can derive that the distributions $A(G)$ and $\varphi^{-1}(A(\varphi(G)))$ are also equal. □

4.2. Definition of the Ideal Functionality for Maximum Weight Matching

As discussed in Section 4.1, we explicitly use an algorithm to define the ideal functionality so that the functionality's definition in precise. We begin by giving an informal description and then proceed by formally introducing the ideal functionality.

Parties $P_1, ..., P_N$ are represented by nodes in a weighted graph. For two parties P_i and P_j, their inputs are used to determine if an edge $\{i, j\}$ exists and, if it exists, what its weight is. The exact generation of edges is kept generic so that the functionality can be easily adapted to specific use cases. Algorithm 4.3 then determines a matching on the resulting graph. Finally, each party gets the index of the party it was matched to as output or receives output 0 if it was not matched. It directly follows that the

resulting matching has at least half of the maximally possible weight. Furthermore, by the algorithm's invariance under permutation it holds that no party P_i gets any advantage in getting matched based on its index i.

For a formal definition, we need to define the generic edge generation first:

Definition 4.2 (Edge Generation Function)**.** *Let L be a fixed number of inputs that are expected per party. Then, the edge generation function ϑ is a function mapping the input values $inputs_i := (inputs_{i,1}, ..., inputs_{i,L})$ and $inputs_j := (inputs_{j,1}, ..., inputs_{j,L})$ of any two parties P_i and P_j to a value in $\mathbb{N}_0$. If based on its inputs, parties P_i and P_j are not to be connected by an edge, then $\vartheta(inputs_i, inputs_j) = 0$. Otherwise, if P_i and P_j are to be connected by an edge of weight $x \in \mathbb{N}_0$, then $\vartheta(inputs_i, inputs_j) = x$.*

The ideal functionality is defined for a fixed number of parties N and a fixed edge generation function ϑ:

Definition 4.3 ($\mathcal{F}_{\text{MWM}}^{N,\vartheta}$: Secure Maximum Weight Matching Approximation)**.** *Let $N \in \mathbb{N}$ and an edge generation function ϑ be fixed. Then, the ideal functionality $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ is defined as follows:*

Let the inputs of all parties $P_1, ..., P_N$ be $inputs_1, ..., inputs_N$ with lengths as given by the definition of ϑ. Furthermore, let the weighted graph $G = ([1, N], E, \sigma)$ is defined by

$$E := \{\{i, j\} \mid i, j \in V \wedge i \neq j \wedge \vartheta(inputs_i, inputs_j) > 0\},$$
$$\sigma(\{i, j\}) := \vartheta(inputs_i, inputs_j)\ \forall \{i, j\} \in E.$$

Then,

$$\mathcal{F}_{\text{MWM}}^{N,\vartheta}(inputs_1, ..., inputs_N) = (y_1, ..., y_N)$$

with $y_1, ..., y_N$ as follows: Let the output of the probabilistic Algorithm 4.3 on G be a matching $\mathfrak{M}$. Then, for $i \in [1, N]$ let $y_i = j$ if $\{i, j\} \in \mathfrak{M}$ and $y_i = 0$ if $\{i, j\} \notin \mathfrak{M}\ \forall j \in [1, N]$.

Note that $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ is probabilistic as Algorithm 4.3 is.

4.3. Application for Crossover Exchange

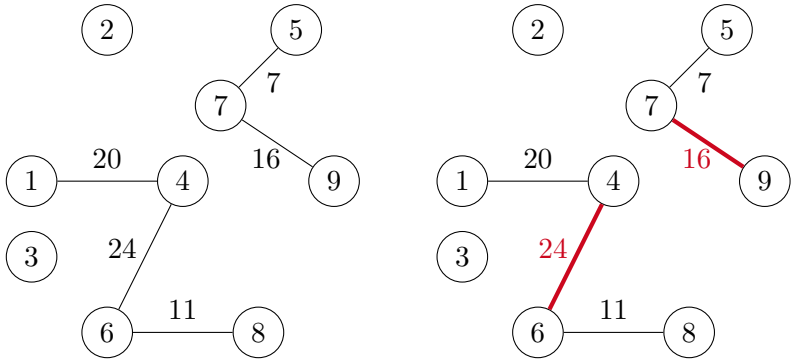
Having defined a generic ideal functionality for approximating MWMs between multiple parties, we review its possible application for crossover exchange. Note that for kidney exchange, by a party P_i we denote a pair consisting of a patient and her corresponding incompatible donor. For SMPC, a party P_i is a participant in the execution of a secure protocol. Here, we let both definitions coincide by letting each party in the execution of a secure protocol represent a pair of patient and donor. In the following, we introduce a graph representation of the KEP restricted to crossover exchanges and elaborate how the required graph can be generated.

4.3.1. Graph Representation

Recall that the general KEP can be modeled as a graph problem using a directed graph [ABS07]. As we solely consider crossover exchanges, only bidirectional compatibility between parties P_i and P_j, i.e., compatibility between the patient of P_i and

the donor of P_j and vice versa, are of relevance here. Thus, compatibility can be represented without any directionality. Consequently, our definition of weighted graphs suffices to define CGs for crossover exchange.

For crossover exchange, we let a CG be a weighted graph $G = (V, E, \sigma)$ where for each party P_i, there is one node i according to the definition of $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$. There is an edge $\{i, j\}$ for each pair of parties P_i and P_j where the patient of P_i is estimated to be medically compatible to the donor of P_j based on their inputs and vice versa. Each edge is assigned a weight depending on the inputs of the corresponding parties. This weight can be used to express prioritization or can be set to a constant value so that solely the number of patients that may receive a kidney transplant defines the objective [ABS07]. Figure 4.1a gives an example of a CG where non-uniform weights are used.



(a) CG (b) Resulting matching

Figure 4.1.: CG as used by $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ and matching of at least half of the optimal weight obtained by evaluating $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$. The edges in the matching are represented by thick edges. The given CG solely acts as an example and does not necessarily represent realistic instances for crossover exchange.

A matching on a CG corresponds to a set of crossover exchanges where each crossover exchange is estimated to be feasible and there are no conflicts, i.e., each party is assigned to take part in at most one crossover exchange. The number of patients that are allocated to receive a kidney transplant is maximized for a maximum matching on the corresponding CG. Note that a maximum matching exactly is an MWM if all weights are set to a uniform value. Computing an MWM on the CG maximizes the aggregated weight of all crossover exchanges planned to be executed. This allows for prioritization of specific crossover exchanges. For instance, crossover exchanges involving highly sensitized patients, i.e., patients with a high amount of HLA antibodies and resulting low chances of getting matched, could be prioritized [Bir+21]. Note that as a direct consequence, a single crossover exchange with a sufficiently high weight could be preferred over multiple other crossover exchanges.

The evaluation of $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ and therefore also the evaluation of our protocol yields a set of presumably feasible crossover exchanges such that the aggregated weight is at least half of the corresponding weight of an MWM. Recall that such set also represents at least half of the maximally feasible crossover exchanges as the greedy algorithm also is a $1/2$-approximation algorithm for maximum matching. An example of such

resulting matching of parties is given in Figure 4.1b. In the example, nine parties participate in the evaluation of $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$, for instance by a secure protocol. The parties' medial characteristics given in their inputs lead to the assumption that each pair of parties connected by an edge in Figure 4.1a is compatible. Furthermore, specific inputs are used for prioritization as discussed in Section 3.3 and yield to the given edge weights. Then, party P_4 receives the information that it was matched to party P_6 and vice versa. Similarly, parties P_7 and P_9 are informed of their match. Other parties receive output 0 encoding that they were not matched. The goal of using a secure protocol is that besides of what can be deduced from its inputs and outputs, a party gains no knowledge of the CG as shown in Figure 4.1b, other matching edges or similar potential intermediate steps in the calculation.

4.3.2. Generation of the Compatibility Graph

For each pair of parties P_i and P_j, the edge generation ϑ used by $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ has to define if there is an edge $\{i, j\}$. Furthermore, it has to give a weight for such edge if it exists. The edge should only exist if the patient of party P_i and the donor of party P_j are medically compatible and vice versa. Based on the inputs separately obtained for both parties, it is only possible to estimate medical compatibility by a virtual crossmatch test [Bir+21]. The parties are estimated to be compatible if both corresponding virtual crossmatches are negative [Bir+21].

For a negative virtual crossmatch, we require ABO-compatibility and a negative virtual HLA-crossmatch which is a compatibility test for HLA-compatibility [Bir+21]. Note that for instance, Breuer et al. [Bre+20] use the same definition for their protocol for kidney exchange. In the following, we briefly introduce each of these criteria. We stress that alternative compatibility checks according to a specific kidney exchange program's criteria can also be used by providing a different suitable definition of ϑ.

There are four ABO blood types, i.e., O, A, B and AB, which are determined by the absence or presence of A and B antigens on a human's red blood cells [Dea05]. Humans develop antigens against A or B antigens if they do not possess these antigens themselves [Dea05]. Thus, ABO-compatibility is only given if each of these types of antigens that is present for the donor also is present for the patient such that the patient has no antibodies against them [Dea05]. A summary of ABO-compatibility between all possible pairs of ABO blood types is given in Table 4.1.

Table 4.1.: ABO-Compatibility between donor and patient according to blood type [Dea05].

Donor \ Patient	O	A	B	AB
O	✓	✓	✓	✓
A	✗	✓	✗	✓
B	✗	✗	✓	✓
AB	✗	✗	✗	✓

A negative HLA-crossmatch between a donor and a patient is given if the patient has no antibodies against the donor's HLAs [Bir+21]. For a virtual HLA-crossmatch test, HLA data that is separately obtained for patient and donor is compared to esti-

mate the result of a laboratory crossmatch test [Bir+21]. Like Breuer et al. [Bre+20], we assume that there is a fixed list of HLAs that are considered for virtual HLA-crossmatch testing.

Thus, the input values $inputs_i$ of party P_i must define the ABO blood types of the corresponding patient and donor, the donor's HLAs and the patient's HLA antibodies. Concerning the case where medical compatibility is estimated to be given between two parties, we do not suggest specific weights at this point. This is due to great variation of prioritization between different kidney exchange programs and due to medical and ethical questions raised by prioritization [Bir+21] that are out of scope of this thesis.

Chapter 5

Building Blocks

Before introducing a secure protocol implementing the ideal functionality $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ as introduced in Definition 4.3, we need to introduce gates that can be used as building blocks for such protocol. In Section 5.1, we discuss elementary gates that are existing gates from literature and further gates that are trivial. A generic edge generation functionality based on a fixed edge generation function as in Definition 4.2 is introduced in Section 5.2. Then, an example for a specific edge generation gate for crossover exchange is discussed in Section 5.3. In Sections 5.4 and 5.5 we introduce functionalities and gates for randomly permuting a list of items and efficiently searching for an edge of highest weight in a list of potential edges.

5.1. Elementary Gates

The elementary gates consist of gates that implement comparisons, random integer generation, trivial auxiliary operations, and mathematical operations. We introduce several gates from literature that are also used by MP-SPDZ [Kel20] so that the gates used for our theoretical protocol formulation in Chapter 6 matches its implementation that is evaluated in Chapter 7. To this end, we review the source code of MP-SPDZ[1] regarding the use of Shamir's secret sharing scheme as introduced in Section 2.3 in presence of a passive adversary and an honest majority. This review reveals that MP-SPDZ uses multiple gates by Catrina and de Hoogh [CH10] that are of relevance for our protocol. In addition, we discuss trivial gates that implement further functionalities and directly follow from the gates from literature and arithmetic operations supported by the BGW protocol as introduced in Section 2.4.3. After introducing all elementary gates, we provide their complexities in Table 5.1.

5.1.1. Secure Comparison

For secure comparison, we require two gates that all necessary comparisons can be based on. We start by providing definitions of the implemented functionalities.

[1] https://github.com/data61/MP-SPDZ/

Definition 5.1 ($\mathcal{G}_{\text{EQZ}}$: Secure Equal Zero Comparison [CH10])**.** *Let $[x]$ be some secret shared integer. Then, gate functionality $\mathcal{G}_{\text{EQZ}}$ is defined as*

$$\mathcal{G}_{\text{EQZ}}([x]) := \begin{cases} [1], & \textit{if } x = 0, \\ [0], & \textit{otherwise.} \end{cases}$$

Definition 5.2 ($\mathcal{G}_{\text{LTZ}}$: Secure Less Than Zero Comparison [CH10])**.** *Let $[x]$ be some secret shared integer. Then, gate functionality $\mathcal{G}_{\text{LTZ}}$ is defined as*

$$\mathcal{G}_{\text{LTZ}}([x]) := \begin{cases} [1], & \textit{if } x < 0, \\ [0], & \textit{otherwise.} \end{cases}$$

Further comparison functionalities can trivially be expressed by a combination of the given ones and arithmetic operations supported by the BGW protocol [CH10]. For example, $x > y \Leftrightarrow y - x < 0$ yields that $\mathcal{G}_{\text{LTZ}}([y] - [x])$ checks if $x > y$ [CH10]. For better readability, in our code examples we annotate the comparison that is expressed by a call to one of the comparison gates.

Catrina and de Hoogh [CH10] propose gates ρ_{EQZ} and ρ_{LTZ} implementing $\mathcal{G}_{\text{EQZ}}$ and $\mathcal{G}_{\text{LTZ}}$ securely. Their gates provide statistical security in the presence of a passive adversary controlling a minority of corrupted parties. They justify that only providing statistical security yields more efficient gates. It is to be noted that there are also gates that implement the required comparison functionalities with perfect security [e.g., NO07; Rei09].

5.1.2. Secure Random Integer Generation

Recall that the ideal functionality $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ is probabilistic. For the randomization techniques that we use later, the secure generation of random integers is required. To allow generating an integer using $k \in [1, b-1]$ bits, we define the following functionality:

Definition 5.3 ($\mathcal{G}_{\text{RND}}$: Secure Random Integer Generation (based on [CH10]))**.** *Let $k \in [1, b-1]$ be some constant integer. Then, gate functionality $\mathcal{G}_{\text{RND}}$ is defined as $\mathcal{G}_{\text{RND}}(k) = [r]$ where r is drawn uniformly at random from $[0, 2^k - 1]$.*

Catrina and de Hoogh [CH10] sketch how a gate implementing $\mathcal{G}_{\text{RND}}$ can be formulated. As a basis, they use the secure generation of random bits:

Definition 5.4 ($\mathcal{G}_{\text{RNDB}}$: Secure Random Bit Generation [CH10])**.** *Gate functionality $\mathcal{G}_{\text{RNDB}}$ is defined as $\mathcal{G}_{\text{RNDB}}() = [r]$ where r is drawn uniformly at random from $\{0, 1\}$.*

A gate ρ_{RNDB} for random bit generation that implements $\mathcal{G}_{\text{RNDB}}$ with statistical security is given by Catrina and de Hoogh [CH10]. Their idea of using the random bit generation to obtain a random integer can be used to implement $\mathcal{G}_{\text{RND}}$ as in Gate 5.1.

It is easy to see that ρ_{RND} implements $\mathcal{G}_{\text{RND}}$ with statistical security in our setting as it can be simulated by calling the simulator for statistical secure ρ_{RNDB} k times while further operations are local. As correctness trivially holds, we conclude that ρ_{RND} securely implements $\mathcal{G}_{\text{RND}}$. A perfectly secure implementation is also possible using perfectly secure random bit generation [e.g., Dam+06].

Gate 5.1 $[r] \leftarrow \rho_{\text{RND}}(k)$ (based on [CH10])

for $i \in [0, k-1]$ in parallel **do**
  $[b_i] \leftarrow \rho_{\text{RNDB}}()$
end for
return $\sum_{i=0}^{k-1} 2^i \cdot [b_i]$

5.1.3. Auxiliary Operations

We proceed by briefly discussing simple auxiliary functionalities and corresponding gates that simplify the formulation of the protocol. The first functionality can be used to select one of two secret shared integers depending on a secret shared bit encoding a condition:

Definition 5.5 ($\mathcal{G}_{\text{IF_ELSE}}$: Secure If-Else)**.** *Let* $[cond]$ *be some secret shared bit, i.e.,* $cond \in \{0, 1\}$*, and let* $[x_{true}], [x_{false}]$ *be secret shared integers. Then, gate functionality* $\mathcal{G}_{\text{IF_ELSE}}$ *is defined as*

$$\mathcal{G}_{\text{IF_ELSE}}([cond], [x_{true}], [x_{false}]) := \begin{cases} [x_{true}], & \text{if } cond = 1, \\ [x_{false}], & \text{if } cond = 0. \end{cases}$$

A corresponding implementation is given in Gate 5.2.

Gate 5.2 $[x] \leftarrow \rho_{\text{IF_ELSE}}([cond], [x_{true}], [x_{false}])$

return $[cond] \cdot ([x_{true}] - [x_{false}]) + [x_{false}]$

The correctness of the implementation can be easily verified by setting *cond* to 0 and 1. Furthermore, the gate can be simulated by simulating the used operations that are perfectly secure operations of the BGW protocol. Thus, it is trivial that $\rho_{\text{IF_ELSE}}$ implements $\mathcal{G}_{\text{IF_ELSE}}$ with perfect security.

Gate $\rho_{\text{IF_ELSE}}$ yields the option to swap secret shared values based on a secret shared condition which also is required by our protocol. Formally, we implement the following gate functionality:

Definition 5.6 ($\mathcal{G}_{\text{SWAP}}$: Secure Conditional Swap)**.** *Let* $[cond]$ *be some secret shared bit, i.e.,* $cond \in \{0, 1\}$*, and let* $[x], [y]$ *be secret shared integers. Then, gate functionality* $\mathcal{G}_{\text{SWAP}}$ *is defined as*

$$\mathcal{G}_{\text{SWAP}}([cond], [x], [y]) := \begin{cases} ([y], [x]), & \text{if } cond = 1, \\ ([x], [y]), & \text{if } cond = 0. \end{cases}$$

A straightforward implementation of this functionality is given in Gate 5.3.

The correctness of the implementation trivially follows the correctness of $\rho_{\text{IF_ELSE}}$. Furthermore, ρ_{SWAP} can be simulated by calling the simulator of perfectly secure $\rho_{\text{IF_ELSE}}$ twice using the corresponding input values and returning their results according to the specification of ρ_{SWAP}. Thus, ρ_{SWAP} implements $\mathcal{G}_{\text{SWAP}}$ with perfect security.

Gate 5.3 $([x'], [y']) \leftarrow \rho_{\text{SWAP}}([cond], [x], [y])$

$[x'] \leftarrow \rho_{\text{IF_ELSE}}([cond], [y], [x])$
$[y'] \leftarrow \rho_{\text{IF_ELSE}}([cond], [x], [y])$
return $([x'], [y'])$

5.1.4. Mathematical Operations

To introduce a gate that generates the edges for crossover exchange in Section 5.3, we require gates for the computation of absolute values and dot products. Note that these gates are not required for the matching protocol itself. We start by discussing the computation of absolute values.

Definition 5.7 ($\mathcal{G}_{\text{ABS}}$: Secure Absolute Value)**.** *Let $[x]$ be some secret shared integer. Then, gate functionality $\mathcal{G}_{\text{ABS}}$ is defined as*

$$\mathcal{G}_{\text{ABS}}([x]) := [|x|].$$

The functionality can easily be implemented by utilizing the previously introduced gates ρ_{LTZ} and $\rho_{\text{IF_ELSE}}$. The resulting gate is given in Gate 5.4. The gate trivially is correct. It implements $\mathcal{G}_{\text{ABS}}$ with statistical security as it can be simulated by calling simulators for the gates ρ_{LTZ} and $\rho_{\text{IF_ELSE}}$ that provide statistical and perfect security.

Gate 5.4 $[y] \leftarrow \rho_{\text{ABS}}([x])$

return $\rho_{\text{IF_ELSE}}(\rho_{\text{LTZ}}([x]), -[x], [x])$ $\triangleright\ x < 0?$

The secure computation of the dot product of two vectors containing secret shared integers can be defines as follows:

Definition 5.8 ($\mathcal{G}^n_{\text{DOT_PROD}}$: Secure Dot Product [CH10])**.** *Let $n \in \mathbb{N}$ and $[A_i]_{i\in[1,n]}$ and $[B_i]_{i\in[1,n]}$ be vectors of n secret shared integers each. Furthermore, let*

$$(C_i)_{i\in[1,n]} := \langle (A_i)_{i\in[1,n]}, (B_i)_{i\in[1,n]} \rangle.$$

Then, gate functionality $\mathcal{G}^n_{\text{DOT_PROD}}$ is defined as

$$\mathcal{G}^n_{\text{DOT_PROD}}([A_i]_{i\in[1,n]}, [B_i]_{i\in[1,n]}) := [C_i]_{i\in[1,n]}.$$

Catrina and de Hoogh [CH10] propose a gate $\rho^n_{\text{DOT_PROD}}$ that implements $\mathcal{G}^n_{\text{DOT_PROD}}$ with perfect security. Their implementation is an optimization of the naïve computation of a dot product and requires only one invocation.

We conclude the section on elementary gates by providing an overview of the complexities of these gates in Table 5.1.

5.2. Edge Generation Functionality

Recall that the ideal functionality $\mathcal{F}^{N,\vartheta}_{\text{MWM}}$ is defined using a generic edge generation function ϑ as introduced in Definition 4.2. In this section, we give a generic gate functionality that can be used to generate single edges of a graph.

Table 5.1.: Complexities of the elementary gates regarding the required rounds and invocations. For the gates from literature, we also provide their names in the corresponding original publications.

Gate	Rounds	Invocations
$\rho_{\text{EQZ}}([x])$ (via KOpL[2])	$\mathcal{O}(\log b)$	$\mathcal{O}(b)$
$\rho_{\text{LTZ}}([x])$ (LTZL)	$\mathcal{O}(\log b)$	$\mathcal{O}(b)$
$\rho_{\text{RNDB}}()$ (PRandBit)	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\rho_{\text{RND}}(k)$	$\mathcal{O}(1)$	$\mathcal{O}(k)$
$\rho_{\text{IF_ELSE}}([\mathit{cond}], [x_{\mathit{true}}], [x_{\mathit{false}}])$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\rho_{\text{SWAP}}([\mathit{cond}], [x], [y])$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
$\rho_{\text{ABS}}([x])$	$\mathcal{O}(\log b)$	$\mathcal{O}(b)$
$\rho^{n}_{\text{DOT_PROD}}([A_i]_{i\in[1,n]}, [B_i]_{i\in[1,n]})$ (Inner)	$\mathcal{O}(1)$	$\mathcal{O}(1)$

The edge generation function ϑ defines which edges exist and the weight of existing edges given the parties' inputs. Edge existence and weight can be encoded by a single value using secret graphs as introduced in Section 2.5. Recall that a secret graph $\hat{G} = (V, \hat{E}, \hat{\sigma})$ corresponding to a weighted graph G encodes the existence of edges in its weight function. The set $\hat{E}$ depends on N only and potential edges $e \in \hat{E}$ that are not present for G have weight 0 while others keep their original weight.

Encoding existence and weight of an edge as a single integer yields the output format for the edge generation gate. For each potential edge $(v, w) \in \hat{E}$, a secret shared integer $[\mathit{weight}_{v,w}]$ is returned where $\mathit{weight}_{v,w} := \hat{\sigma}(\{v, w\})$. A protocol can call the edge generation gate for the inputs of the parties corresponding to each $(v, w) \in \hat{E}$. The resulting values $[\mathit{weight}_{v,w}]$ for $(v, w) \in \hat{E}$ can then be seen as a strict upper triangular adjacency matrix, i.e., an adjacency matrix that is restricted to entries above its main diagonal, as the represented graph is undirected. We proceed by giving a formal definition of the generation of a secret shared value $[\mathit{weight}_{v,w}]$ given some edge generation function ϑ.

Definition 5.9 ($\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$: Secure Edge Generation)**.** *Let ϑ be some edge generation function as introduced in Definition 4.2. Furthermore, let $\mathit{inputs}_i, \mathit{inputs}_j$ be the inputs of two parties P_i and P_j. Gate functionality $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ is defined as*

$$\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}([\mathit{inputs}_i], [\mathit{inputs}_j]) := [\vartheta(\mathit{inputs}_i, \mathit{inputs}_j)].$$

We denote a generic gate securely implementing $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ for a fixed edge generation function ϑ by $\rho^{\vartheta}_{\text{EDGE_GEN}}$. For a specific use case, a concrete implementation based on the particular inputs that are required by the considered use case has to be provided. Furthermore, its security and correctness must be proven. We require a specific implementation of the gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$ to be at least statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties.

[2] Catrina and de Hoogh [CH10] only give the complexity of the variant EQZC which is based on KOrCL while we use KOpL instead of KOrCL. The complexity given here can be obtained by comparing KOrCL and KOpL.

5.3. Edge Generation Gate for Crossover Exchange

In this section, we provide an example for a specific gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$ that can be used for crossover exchange. Our gate solely serves as an example of how compatibility and prioritization could be defined and computed. We can later use this gate to evaluate our protocol with a specific graph generation.

Recall that as discussed in Section 4.3.2, it is necessary to assess the compatibility between each pair of parties. In particular, for two parties P_i and P_j we require that for the patient of P_i and the donor of P_j there is a negative virtual crossmatch and vice versa. For a negative virtual crossmatch, we require ABO-compatibility and a negative virtual HLA-crossmatch. In the following, we discuss how these criteria can be tested for.

5.3.1. ABO-Compatibility

The ABO-compatibility test used by Breuer et al. [Bre+20] requires a secure less-than comparison. Note that we work in a different setting using Shamir's secret sharing scheme. In our case, the gate ρ_{LTZ} would be necessary requiring $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations. We present a different approach that is less expensive in our setting.

To test for ABO-compatibility, we introduce Boolean values A, B that encode the presence of A or B antigens by 1 and their absence by 0. These values have to be given for the patient and the donor of each party. Let the corresponding inputs of party P_i be $A_i^d, B_i^d, A_i^p, B_i^p$ where d refers to the donor's antigens and p refers to the patient's antigens. Recall that for ABO-compatibility between a patient and a donor, the A or B antigen must be present for the patient if the same antigen is present for the donor as discussed in Section 4.3.2. Thus, ABO-compatibility between the patient of party P_i and the donor of party P_j can be expressed as

$$(A_j^d \to A_i^p) \wedge (B_j^d \to B_i^p) \equiv \neg(A_j^d \wedge \neg A_i^p) \wedge \neg(B_j^d \wedge \neg B_i^p).$$

This expression can be rewritten as an arithmetic term:

$$(1 - A_j^d \cdot (1 - A_i^p)) \cdot (1 - B_j^d \cdot (1 - B_i^p)).$$

If the inputs are represented by secret shared Boolean values, the given expression can be evaluated using only three invocations and two rounds for the multiplications by executing the outer multiplications in parallel.

5.3.2. Negative Virtual HLA-Crossmatch

Recall that for a negative virtual HLA-crossmatch as discussed in Section 4.3.2, it must hold that there is no HLA known to be present for the donor where antibodies against this HLA are known to be present for the patient. We assume that a fixed number of H many HLAs is considered. Let there be some fixed ordering of the considered HLAs. We encode the HLA antibodies of the patients and the HLAs of the donors analogously to Breuer et al. [Bre+20]. For the patient of party P_i, let $\mathcal{K}_i \in \{0,1\}^H$ be chosen such that for $a \in [1, H]$, $\mathcal{K}_{i,a}$ equals 1 if the patient has antibodies against the a'th antigen with respect to the fixed ordering of the considered

HLAs, and 0 otherwise. Similarly, for the donor of party P_j let $\mathcal{J}_j \in \{0,1\}^H$ be a vector encoding which HLAs are present.

Note that using a flag for each possible antigen and antibody may lead to longer encodings than just encoding all present antigens and antibodies as a list or similar data structure. We choose the presented vector encoding as it has a fixed length. Therefore, the input length does not leak any information about the number of present HLAs and antibodies for a party and matches our formal requirement of equal input lengths for SMPC.

A positive virtual HLA-crossmatch for the patient of party P_i and the donor of party P_j is given if and only if there exists a considered HLA that is present for the donor and that the patient has antibodies against. This is equivalent to some HLA with index $1 \leq a \leq H$ existing such that $\mathcal{K}_{i,a} = \mathcal{J}_{j,a} = 1$ which then implies that $\langle \mathcal{K}_i, \mathcal{J}_j \rangle \geq 1$. Thus, for a patient with vector $\mathcal{K}_i$ and a donor with vector $\mathcal{J}_j$, there is a negative virtual crossmatch if and only if $\langle \mathcal{K}_i, \mathcal{J}_j \rangle = 0$.

5.3.3. Definition of Edge Weights

To define a graph generation gate for crossover exchange that can later be used to evaluate our protocol, it is necessary to define edge weights. In particular, an exemplary weight computation is required as a part of the protocol so that the run time of the actual implementation in MP-SPDZ can be evaluated. In addition, an evaluation of the resulting graphs and our protocol's performance on them compared to an optimal MWM algorithm is given in Appendix B. As discussed in Section 4.3.2, a meaningful definition of edge weights is out of scope here. Thus, we only give a brief overview of the used edge weights here that solely demonstrates how weights may be defined.

We prioritize crossover exchanges where one or both involved patients are highly sensitized or have a long waiting time. Furthermore, we prioritize cases where each transplant is executed between a donor and a patient who have similar ages and are geographically close. Each of these aspects is included in the objectives of some operating kidney exchange programs as discussed in Section 3.3.

In the following, we define the weight of the edge $\{i, j\}$ for parties P_i and P_j where each patient and the opposite party's donor have a negative virtual crossmatch. Let $\mathit{highly_sensitized}_i$ equal a fixed value greater than zero if the patient of party P_i is highly sensitized according to an arbitrary definition and 0 otherwise. Furthermore, let $\mathit{waiting_time}_i$ be the waiting time of party P_i's patient corresponding to a fixed time measure. For instance, the value could be increased by one for each day, week, or month of waiting time. Let inputs age_i^p and age_i^d be the ages of patient and donor of party P_i. As an example method for assessing the geographic distances, we let each person be in some region and use some closeness measure between their regions. In general, such closeness measures should decrease with increasing geographic distance. A region may for instance be defined by the area where a specific transplant center is in charge. Let there be R regions $1, ..., R$. The patient of party P_i being registered in region $a \in [1, R]$ is encoded by a vector $\mathcal{I}_i \in \{0,1\}^R$ such that only entry $\mathcal{I}_{i,a}$ equals 1. For the donor of party P_j, let $\mathcal{D}_j \in \mathbb{N}_0^R$ encode the closeness of the donor's region to each region, i.e., $\mathcal{D}_{j,a}$ describes the closeness between the donor's region and region a. Thus, $\langle \mathcal{I}_i, \mathcal{D}_j \rangle$ describes the closeness between the regions of the patient of P_i and the donor of P_j. All inputs are also defined for the symmetrical

case where i and j are exchanged. Each edge has some base weight to guarantee that the weight is positive. We use base weight of 10 as using a value higher than 1 lowers the impact of further prioritization. Directly scaling down the values used for prioritization instead is not possible as we only use weights in $\mathbb{N}$.

5.3.4. Gate Specification

We can now define an edge generation function ϑ_{KEP} inducing a gate functionality $\mathcal{G}^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$.

Definition 5.10 (Edge Generation Function for Crossover Exchange)**.** *Let there be fixed $H, R \in \mathbb{N}$. For parties P_i, P_j, let the collections of input values inputs_i contain $A_i^d, B_i^d, A_i^p, B_i^p \in \{0,1\}, \mathcal{K}_i, \mathcal{J}_i \in \{0,1\}^H$, $\mathit{highly_sensitized}_i, \mathit{waiting_time}_i, \mathit{age}_i^p, \mathit{age}_i^d \in \mathbb{N}_0$, $\mathcal{I}_i \in \{0,1\}^R$, and $\mathcal{D}_j \in \mathbb{N}_0^R$ and let inputs_j be defined symmetrically. We consider parties P_i and P_j to be compatible if*

$$\begin{aligned}(1 - A_j^d \cdot (1 - A_i^p)) \cdot (1 - B_j^d \cdot (1 - B_i^p)) = 1 \\ \wedge\ (1 - A_i^d \cdot (1 - A_j^p)) \cdot (1 - B_i^d \cdot (1 - B_j^p)) = 1 \\ \wedge\ \langle \mathcal{K}_i, \mathcal{J}_j \rangle = 0 \ \wedge\ \langle \mathcal{K}_j, \mathcal{J}_i \rangle = 0.\end{aligned}$$

If the parties a considered to be compatible, let the weight x of the corresponding edge be

$$\begin{aligned}x := 10 &+ \mathit{highly_sensitized}_i + \mathit{highly_sensitized}_j \\ &+ \mathit{waiting_time}_i + \mathit{waiting_time}_j \\ &+ \max\{20 - |\mathit{age}_i^p - \mathit{age}_j^d|, 0\} + \max\{20 - |\mathit{age}_j^p - \mathit{age}_i^d|, 0\} \\ &+ \langle \mathcal{I}_i, \mathcal{D}_j \rangle + \langle \mathcal{I}_j, \mathcal{D}_i \rangle.\end{aligned}$$

Then, let ϑ_{KEP} be defined as

$$\vartheta_{KEP}(\mathit{inputs}_i, \mathit{inputs}_j) := \begin{cases} x, & \text{if parties } P_i \text{ and } P_j \text{ are considered to be compatible,} \\ 0, & \text{otherwise.} \end{cases}$$

Gate functionality $\mathcal{G}^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ is defined according to Definition 5.9 by utilizing ϑ_{KEP} as edge generation function. Gate 5.5 then implements gate functionality $\mathcal{G}^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$.

First, the ABO-compatibility between the patient of party P_i and the donor of party P_j is computed and the Boolean result is written to $[\mathit{comp}_{ABO}]$. Symmetrically, $[\mathit{comp}'_{ABO}]$ is set to the ABO-compatibility between the patient of P_j and the donor of P_i. Note that both computations do not depend on each other's execution and can therefore be parallelized. As such parallelizations do not influence asymptotical complexity, we do not explicitly mention them in the remainder of this thesis. A negative virtual crossmatch can be tested for by utilizing the gate $\rho^H_{\text{DOT_PROD}}$ for computing the relevant dot products. As the result $[\mathit{hla_matches}]$ and $[\mathit{hla_matches}]$ are nonnegative, it holds that $\mathit{hla_matches} + \mathit{hla_matches}' = 0 \Leftrightarrow \mathit{hla_matches} = \mathit{hla_matches}' = 0$. Note that this restatement requires one comparison less than comparing each dot product to 0 separately.

The secret shared value $[\mathit{weight}_{age}]$ first is set to 20 minus the absolute age difference of the patient of P_i and the donor of P_j. In a second step, it is set to 0 if it would

be negative otherwise according to Definition 5.10. For $[weight'_{age}]$, the symmetrical steps with exchanged i and j are executed. Then, $[pre_weight]$ representing x as used in Definition 5.10 is set to the weight that the edge between i and j has if parties P_i and P_j are considered to be compatible.

The final output $[weight]$ representing a potential edge between parties P_i and P_j is set to $[pre_weight]$ multiplied by $[comp_{ABO}]$, $[comp'_{ABO}]$ and $[comp_{HLA}]$ that is one if both involved virtual HLA-crossmatches are negative. Thus, the output is $[0]$ if parties P_i and P_j are not considered to be compatible.

Gate 5.5 $[weight] \leftarrow \rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}([inputs_i], [inputs_j])$

$[comp_{ABO}] \leftarrow (1 - [A^d_j] \cdot (1 - [A^p_i])) \cdot (1 - [B^d_j] \cdot (1 - [B^p_i]))$
$[comp'_{ABO}] \leftarrow (1 - [A^d_i] \cdot (1 - [A^p_j])) \cdot (1 - [B^d_i] \cdot (1 - [B^p_j]))$

$[hla_matches] \leftarrow \rho^{H}_{\text{DOT_PROD}}([\mathcal{K}_{i,k}]_{k\in[1,H]}, [\mathcal{J}_{j,k}]_{k\in[1,H]})$
$[hla_matches'] \leftarrow \rho^{H}_{\text{DOT_PROD}}([\mathcal{K}_{j,k}]_{k\in[1,H]}, [\mathcal{J}_{i,k}]_{k\in[1,H]})$
$[comp_{HLA}] \leftarrow \rho_{\text{EQZ}}([hla_matches] + [hla_matches'])$ ▷ $\ldots + \ldots = 0$?

$[weight_{age}] \leftarrow 20 - \rho_{\text{ABS}}([age^p_i] - [age^d_j])$
$[weight_{age}] \leftarrow \rho_{\text{IF_ELSE}}(\rho_{\text{LTZ}}([weight_{age}]), [0], [weight_{age}])$ ▷ $weight_{age} < 0$?
$[weight'_{age}] \leftarrow 20 - \rho_{\text{ABS}}([age^p_j] - [age^d_i])$
$[weight'_{age}] \leftarrow \rho_{\text{IF_ELSE}}(\rho_{\text{LTZ}}([weight'_{age}]), [0], [weight'_{age}])$ ▷ $weight'_{age} < 0$?

$$\begin{aligned}[pre_weight] \leftarrow 10 &+ [highly_sensitized_i] + [highly_sensitized_j] \\ &+ [waiting_time_i] + [waiting_time_j] \\ &+ [weight_{age}] + [weight'_{age}] \\ &+ \rho^{R}_{\text{DOT_PROD}}([\mathcal{I}_{i,k}]_{k\in[1,R]}, [\mathcal{D}_{j,k}]_{k\in[1,R]}) \\ &+ \rho^{R}_{\text{DOT_PROD}}([\mathcal{I}_{j,k}]_{k\in[1,R]}, [\mathcal{D}_{i,k}]_{k\in[1,R]})\end{aligned}$$

$[weight] \leftarrow [comp_{ABO}] \cdot [comp'_{ABO}] \cdot [comp_{HLA}] \cdot [pre_weight]$
return $[weight]$

The number of inputs L provided by each party is dominated by the HLA and region indicators and therefore $L \in \mathcal{O}(H + R)$. A constant number of multiplications and calls to ρ_{EQZ}, ρ_{LTZ}, $\rho_{\text{IF_ELSE}}$, ρ_{ABS}, $\rho^{H}_{\text{DOT_PROD}}$ and $\rho^{R}_{\text{DOT_PROD}}$ is used. Each multiplication or call to $\rho_{\text{IF_ELSE}}$, $\rho^{H}_{\text{DOT_PROD}}$ or $\rho^{R}_{\text{DOT_PROD}}$ requires $\mathcal{O}(1)$ rounds and invocations. Furthermore, each call to ρ_{EQZ}, ρ_{LTZ} or ρ_{ABS} requires $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations. Thus, the gate's overall complexity consists of $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations.

Finally, we prove the gate's security:

Theorem 5.1. *Gate $\rho^{\vartheta_{KEP}}_{\text{EDGE_GEN}}$ implements functionality $\mathcal{G}^{\vartheta_{KEP}}_{\text{EDGE_GEN}}$ statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties.*

Proof. For the correctness of gate $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$, note that the gate exactly follows the definition of the edge generation function given in Definition 5.10. We have already pointed out why it is valid to compare the results of both virtual HLA-crossmatch tests in a joined fashion. Furthermore, the computation of $[weight_{age}]$ and $[weight'_{age}]$ exactly represents subtracting the absolute age difference from 20 and setting the

result to 0 if it would be negative otherwise. The final multiplication ensures that the result is set to 0 if and only if parties P_i and P_j are considered not to be compatible. Due to correctness of all used gates, $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ correctly implements $\mathcal{G}^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$.

We proceed by defining a simulator for $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$. On its given inputs, the simulator simulates all operations of $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ following its flow of execution. Arithmetic operations of the BGW protocol trivially can be simulated due to its perfect security. Furthermore, each gate call can be simulated by calling the corresponding simulator of the gate that is statistically or perfectly secure. As all operations are at least statistically secure, the resulting simulated view is statistically indistinguishable from a real view on the gate's execution. □

5.4. Secure Permutation

Our protocol that is defined in Chapter 6 requires to securely generate a random permutation and apply it to shuffle a list of items. In particular, n items that we denote by $\mathit{item}_1, ..., \mathit{item}_n$ have to be permuted according to a permutation τ that is drawn uniformly at random from $S_{[1,n]}$. Each item_i consists of ℓ integers $\mathit{item}_{i,1}, ..., \mathit{item}_{i,\ell}$. All integers are secret shared and we write $[\mathit{item}_i] := ([\mathit{item}_{i,1}], ..., [\mathit{item}_{i,\ell}])$. The required functionality is defined as follows:

Definition 5.11 ($\mathcal{G}^{n,\ell}_{\text{PERM}}$: Permutation of Items)**.** *Let $[\mathit{item}_1], ..., [\mathit{item}_n]$ be n items where each item consists of ℓ secret shared integers, i.e., $[\mathit{item}_i] = ([\mathit{item}_{i,1}], ..., [\mathit{item}_{i,\ell}])$ for all $i \in [1, n]$. Then, gate functionality $\mathcal{G}^{n,\ell}_{\text{PERM}}$ is defined as*

$$\mathcal{G}^{n,\ell}_{\text{PERM}}([\mathit{item}_1], ..., [\mathit{item}_n]) := ([\mathit{item}_{\tau(1)}], ..., [\mathit{item}_{\tau(n)}], [\tau(1)], ..., [\tau(n)])$$

where τ is drawn uniformly at random from $S_{[1,n]}$.

Note that not only the permuted items but also the permuted indexes are part of the output. This allows to identify the used permutation given the functionality's output.

To implement functionality $\mathcal{G}^{n,\ell}_{\text{PERM}}$, we begin by reviewing techniques that can be used for secure shuffling that exactly matches our requirement of permuting a list of items by a random permutation. Based on these techniques, we then propose a gate $\rho^{n,\ell}_{\text{PERM}}$ that implements $\mathcal{G}^{n,\ell}_{\text{PERM}}$ with statistical security.

5.4.1. Secure Shuffling

Shuffling a list of items in an oblivious context requires to access items independently of the randoms that are used to determine the corresponding permutation. Movahedi et al. [MSZ15] propose a protocol for secure shuffling which they claim to be efficient and easy to implement compared to other approaches. We use their algorithm's high-level approach which builds elements consisting of a secret shared random value and an item to shuffle and to then securely sort the elements according to their assigned randoms. Due to differences regarding the general setting between their and our protocol and for ease of implementation, we replace their implementation of assigning randoms and sorting. Assigning randoms can be achieved by using ρ_{RND} while secure sorting requires more sophisticated techniques.

Batcher [Bat68] proposes *sorting networks* that consist of comparison gates that order two elements each and are wired in some static way. In this case, wiring means that the elements to sort go into comparison gates and outputs of comparison gates go into other comparison gates or eventually an output which is ordered. Static wiring implies that running such a sorting network is possible in an oblivious way if the comparison gates can be implemented obliviously. Comparison gates can be grouped into different levels such all gates on the same level are independent of each other and can run in parallel. This implies the potential to parallelize multiple invocations as part of a single round when using a sorting network in a secure protocol.

One class of Batcher's [Bat68] proposed sorting networks are *bitonic sorters*. For any $k \in \mathbb{N}$, there exists a bitonic sorter to sort 2^k elements in ascending order. Such sorter requires $\mathcal{O}(k^2)$ levels. For ease of implementation, we use an iterative variant proposed by Christopher [Chr] instead of the original recursive formulation of bitonic sorters yielding the same complexity. An example for the resulting sorting networks is given in Figure 5.1.

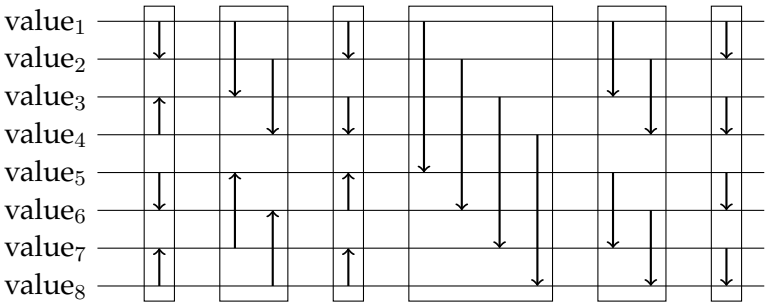


Figure 5.1.: Bitonic sorting network [Bat68] according to the implementation by Christopher [Chr] for $n = 8$ elements. Arrows indicate comparisons and point towards the larger elements according to some order after potential swapping. Rectangles include simultaneously running comparisons. Notation is based on [Lan].

As functionality $\mathcal{G}_{\text{PERM}}^{n,\ell}$ is not restricted to numbers of items that are powers of 2, we extend the sorter based on an idea by Lang [Lan]. For n items to shuffle, let $k := \lceil \log n \rceil$. We pad the n elements consisting of items and random values by $2^k - n$ more elements. Each padding element consists of an arbitrary item and a fixed value instead of a random one. Using fixed values that are larger than all possibly used random values yields that the padding elements are at the last $2^k - n$ positions after sorting all elements. The padding elements can then simply be removed from the output. The number of required levels is in $\mathcal{O}(k^2) = \mathcal{O}(\lceil \log n \rceil^2) \subseteq \mathcal{O}((\log n + 1)^2) = \mathcal{O}(\log^2 n)$.

To conclude, we can use ρ_{RND} to generate a secret shared random value using $b - 2$ bits for each item to shuffle and add padding elements that use 2^{b-2} instead of a random value. Note that for $b - 2$ bits, the output of ρ_{RND} is in $[0, 2^{b-2} - 1]$ and therefore strictly less than 2^{b-2}. We then use a sorting network as discussed previously to shuffle all elements and later ignore the padding elements that are sorted to the last positions. It is easy to see that this technique permutes the input elements according to some permutation drawn uniformly at random from $S_{[1,n]}$ if

all used random values are different.

5.4.2. Gate Specification

To implement functionality $\mathcal{G}_{\text{PERM}}^{n,\ell}$, we use a modification of Christopher's [Chr] sorting network algorithm. In the original algorithm, single numbers are sorted by their value. We need to modify the algorithm to sort elements where each element consists of an item to shuffle and its assigned secret shared random value. Furthermore, recall that the functionality's output contains $[\tau(1)], ..., [\tau(n)]$ where τ is the permutation that the items are shuffled according to. To obtain these additional outputs, we also append the index of each item to the corresponding element. Sorting a list of elements

$$([item_1], [rnd_1], [1]), ..., ([item_n], [rnd_n], [n])$$

where $[rnd_i]$ denotes the secret shared random value assigned to item i then yields a list

$$([item_{\tau(1)}], [rnd_{\tau(1)}], [\tau(1)]), ..., ([item_{\tau(n)}], [rnd_{\tau(n)}], [\tau(n)])$$

according to some permutation τ drawn uniformly at random from $S_{[1,n]}$.

The specification of $\rho_{\text{PERM}}^{n,\ell}$ using the previous considerations about secure random shuffling is given in Gate 5.6. Besides the preceding considerations about modifying the sorting network algorithm by Christopher [Chr], we also adapt it to work obliviously and use an indexing that starts at 1 according to our other gates' and protocol's indexing while the original one starts at 0. By BITWISE_XOR and BITWISE_AND, we denote bitwise "and" and "exclusive-or" operations that work on public values.

After calculating the number of elements to sort including padding elements, the gate assigns random numbers to each of the first n elements. Furthermore, the index of $item_i$ is assigned to $[id_i]$ as a secret shared value. This is necessary to reconstruct $[\tau(1)], ..., [\tau(n)]$ after shuffling. Then, all padding elements are initialized using fixed secret shared values instead of random ones. The remaining part of the gate consists of the previously explained modification of the sorting network algorithm by Christopher [Chr]. Note that we parallelize the loop in line 13 as its iterations access disjoint indexes while the loops in lines 11 and 12 cannot be parallelized due to corresponding to different levels of the sorting network. Each case where the original execution flow depends on the secret shared random values is replaced by constructs that use the gate ρ_{SWAP} to swap values obliviously. The remaining execution flow solely depends on publicly known values.

Complexity

Recall that the gate sorts n items consisting of ℓ secret shared integers each. Generating the required random values is possible in $\mathcal{O}(1)$ rounds using $\mathcal{O}(n \cdot b)$ invocations as $b - 2$ random bits are used to generate each secret shared random value.

Regarding the sorting phase, the body of the conditional block in line 17 requires $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations to compute $[swap]$. Obliviously swapping the necessary secret shared integers requires $\mathcal{O}(1)$ rounds and invocations for each value. As there are $\mathcal{O}(\ell)$ secret shared integers to be swapped which can be done in parallel,

Gate 5.6 $([item_{\tau(1)}], ..., [item_{\tau(n)}], ..., [\tau(1)], ..., [\tau(n)]) \leftarrow \rho^{n,\ell}_{\text{PERM}}([item_1], ..., [item_n])$

1: $n_{\text{padded}} \leftarrow 2^{\lceil \log n \rceil}$

2: **for** $i \in [1, n]$ in parallel **do**
3: $[id_i] \leftarrow [i]$
4: $[rnd_i] \leftarrow \rho_{\text{RND}}(b-2)$
5: **end for**
6: **for** $i \in [n+1, n_{\text{padded}}]$ **do**
7: $[id_i] \leftarrow [0]$
8: $[rnd_i] \leftarrow [2^{b-2}]$
9: $[inputs_i] \leftarrow ([0], ..., [0])$ ▷ $[0]$ repeated ℓ times
10: **end for**

11: **for** $k = 2, 4, 8, ... \leq n_{\text{padded}}$ **do**
12: **for** $j = \frac{k}{2}, \frac{k}{4}, \frac{k}{8}, ..., 1$ **do**
13: **for** $i \in [0, n_{\text{padded}} - 1]$ in parallel **do**
14: `ixj` $\leftarrow$BITWISE_XOR(i, j)
15: $a \leftarrow i + 1$
16: $b \leftarrow$ `ixj` $+ 1$
17: **if** `ixj` $> i$ **then**
18: **if** BITWISE_AND$(i, k) = 0$ **then**
19: $[swap] \leftarrow \rho_{\text{LTZ}}([rnd_b] - [rnd_a])$ ▷ $rnd_a > rnd_b$?
20: **else**
21: $[swap] \leftarrow \rho_{\text{LTZ}}([rnd_a] - [rnd_b])$ ▷ $rnd_a < rnd_b$?
22: **end if**
23: $([id_a], [id_b]) \leftarrow \rho_{\text{SWAP}}([swap], [id_a], [id_b])$
24: $([rnd_a], [rnd_b]) \leftarrow \rho_{\text{SWAP}}([swap], [rnd_a], [rnd_b])$
25: **for** $x \in [1, \ell]$ in parallel **do**
26: $([item_{a,x}], [item_{b,x}]) \leftarrow \rho_{\text{SWAP}}([swap], [item_{a,x}], [item_{b,x}])$
27: **end for**
28: **end if**
29: **end for**
30: **end for**
31: **end for**

32: **return** $([item_1], ..., [item_n], [id_1], ..., [id_n])$

the body of the conditional block in line 17 requires $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b + \ell)$ invocations in total.

The loop in line 13 executes n parallel iterations where by design of the algorithm by Christopher [Chr], the body of the conditional block in line 17 is executed for exactly one half of all iterations. This yields $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(n \cdot (b+\ell))$ invocations. The loops in lines 11 and 12 run for a combined amount of $\mathcal{O}(\log^2 n_{\text{padded}}) = \mathcal{O}(\log^2 n)$ iterations. This implies a complexity of $\mathcal{O}(\log^2 n \cdot \log b)$ rounds and $\mathcal{O}(n \cdot \log^2 n \cdot (b + \ell))$ invocations for the sorting phase. As this complexity dominates the overall complexity of the gate, the gate $\rho^{n,\ell}_{\text{PERM}}$ also requires $\mathcal{O}(\log^2 n \cdot \log b)$ rounds and

$\mathcal{O}(n \cdot \log^2 n \cdot (b + \ell))$ invocations.

Correctness and Security

We end this section by proving the gate's security:

Theorem 5.2. *Let the number of bits b be chosen such that it is not smaller than the security parameter κ, i.e., $b \geq \kappa$. Then, gate $\rho^{n,\ell}_{\text{PERM}}$ implements functionality $\mathcal{G}^{n,\ell}_{\text{PERM}}$ statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties.*

Proof. Let the inputs of gate $\rho^{n,\ell}_{\text{PERM}}$ be $[item_1], ..., [item_n]$. After the execution of lines 1-10 in Gate 5.6, there are $n_{\text{padded}} \geq n$ elements

$$([item_1], [rnd_1], [id_1]), ..., ([item_n], [rnd_n], [id_n]), ..., ([item_{n_{\text{padded}}}], [rnd_{n_{\text{padded}}}], [id_{n_{\text{padded}}}])$$

initialized to

$$([item_1], [rnd_1], [1]), ..., ([item_n], [rnd_n], [n]), ..., (([0], ..., [0]), [2^{b-2}], [0]).$$

Note that for all $i \in [1, n]$ it holds that $rnd_i \leq 2^{b-2} - 1 < 2^{b-2}$. Therefore, and due to the correctness of the sorting network, the elements' order after the sorting phase is

$$([item_{\tau(1)}], [rnd_{\tau(1)}], [\tau(1)]), ..., ([item_{\tau(n)}], [rnd_{\tau(n)}], [\tau(n)]), ..., (([0], ..., [0]), [2^{b-2}], [0])$$

where τ is some permutation in $S_{[1,n]}$ such that $rnd_{\tau(1)} \leq ... \leq rnd_{\tau(n)}$.

Assume that all random values rnd_i for $i \in [1, n]$ have different values, i.e., there exist no $i \neq j$ such that $rnd_i = rnd_j$. As the values $rnd_1, ..., rnd_n$ are drawn independently and uniformly at random from $[0, 2^{b-2} - 1]$, they then induce an ordering of elements that corresponds to a permutation τ that is drawn uniformly at random from $S_{[1,n]}$. Thus, $\rho^{n,\ell}_{\text{PERM}}$ implements $\mathcal{G}^{n,\ell}_{\text{PERM}}$ correctly if all used random values rnd_i for $i \in [1, n]$ have different values.

Now, assume that for rnd_i for $i \in [1, n]$ that are drawn independently and uniformly at random from $[0, 2^{b-2} - 1]$, there exist $i \neq j$ such that $rnd_i = rnd_j$. Note that the probability of such occurrence is essentially discussed by the *birthday problem* [KL14], and thus the upper bound is

$$\Pr(\exists i \neq j \in [1, n] : rnd_i = rnd_j) \leq \frac{n^2}{2 \cdot 2^{b-2}} = \frac{n^2}{2^{b-1}} \leq \frac{n^2}{2^{\kappa-1}}.$$

Hence, the probability for a non-uniform permutation is negligible in the security parameter κ. It directly follows that the gate implements functionality $\mathcal{G}^{n,\ell}_{\text{PERM}}$ correctly except for a negligible probability.

We proceed by defining a simulator for $\rho^{n,\ell}_{\text{PERM}}$. On its inputs, the simulator follows the flow of execution given by $\rho^{n,\ell}_{\text{PERM}}$ that only depends on publicly known values. As the BGW protocol is perfectly secure, each of its operations can trivially be simulated. Each call to a gate is simulated by calling its corresponding simulator. Note that all used gates are statistically or perfectly secure. Thus, the resulting view is statistically indistinguishable from a real view on the gate's execution. □

5.5. Best Edge Search

Our protocol definition in Chapter 6 requires to search real edges in a secret graph that are to be added to a matching according to the edge selection strategy of the greedy algorithm (cf. Algorithm 4.1 and its variants). To this end, it is necessary to find a real edge of maximal weight. It is easy to see that this can be achieved by utilizing a linear search over all $\hat{M}$ potential edges. This approach requires $\mathcal{O}(\hat{M}) = \mathcal{O}(N^2)$ iterations. Comparing the weight of each edge to the weight of the currently best known candidate in each iteration implies that the iterations are not independent. Thus, a linear search requires at least $\mathcal{O}(N^2)$ rounds.

Instead of utilizing a linear search, we use an approach that achieves a higher degree of parallelization. To this end, we carry out multiple comparisons between the weights of different pairs of potential edges in parallel. This can be achieved by carrying out comparisons in a structure as shown in Figure 5.2. We note that a similar idea is used by Catrina and de Hoogh [CH10] for k-ary operations.

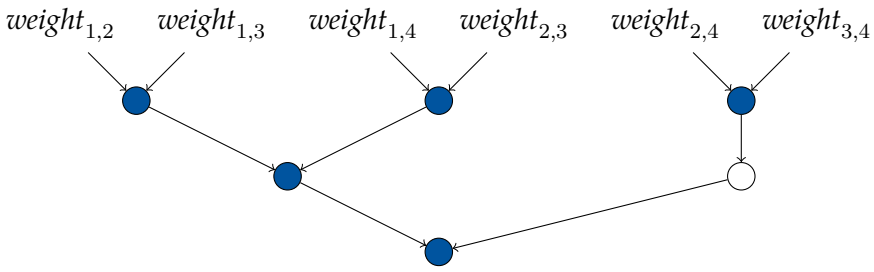


Figure 5.2.: Structuring of edge weight comparisons for $N = 4$ (based on [CH10]). Comparisons are performed in filled nodes only.

The structuring of comparisons groups the comparisons into different layers. For some layer containing n potential edges, the next layer contains $\lceil n/2 \rceil$ potential edges. To implement the search for an edge of highest weight, we use a recursive approach where each recursive step represents one of the layers.

The functionality that returns a real edge of highest weight among $n \in \mathbb{N}$ potential edges is defined as follows:

Definition 5.12 ($\mathcal{G}^n_{\text{SEARCH_MAX}}$: Maximal Weight Search)**.** *Let $[v_i], [w_i], [weight_i]$ for $i \in [1, n]$ be secret shared integers such that $weight_i \geq 0$ for all $i \in [1, n]$. Furthermore, let $k \in [1, n]$ be the minimal number such that $weight_k$ is maximal. Then, gate functionality $\mathcal{G}^n_{\text{SEARCH_MAX}}$ is defined as*

$$\mathcal{G}^n_{\text{SEARCH_MAX}}([v_i]_{i\in[1,n]}, [w_i]_{i\in[1,n]}, [weight_i]_{i\in[1,n]}) := \begin{cases} ([v_k], [w_k]), & \text{if } weight_k > 0 \\ ([0], [0]), & \text{otherwise.} \end{cases}$$

Note that potential edges are encoded by entries of three lists, i.e., the i'th potential edge is $\{v_i, w_i\}$ and has weight $weight_i$. This representation instead of an adjacency matrix is required to support the comparison of an arbitrary number of edges in each of the previously discussed comparison layers. Furthermore, note that in cases where multiple real edges of maximal weight exist, the real edge with the lowest index is selected so that the gate functionality is defined exactly. If the maximal weight among all potential edges is 0, no real edge exists and $\{[0], [0]\}$ as an encoding of this special case is returned.

5.5.1. Gate Specification

An implementation of the gate $\rho^n_{\text{SEARCH_MAX}}$ that implements $\mathcal{G}^n_{\text{SEARCH_MAX}}$ is given in Gate 5.7. If the input contains one single potential edge, this edge can immediately be returned if its weight is greater than zero, i.e., the edge is a real edge. If its weight equals zero, both nodes $[v_1]$ and $[w_1]$ are set to $[0]$ before returning them. Note that only checking if an edge is real for $n = 1$ is sufficient and requires only one comparison. Comparing the weights to zero in another layer of the structuring of comparisons as depicted in Figure 5.2 would require more comparisons and thus would be less efficient.

For an input that contains multiple potential edges, pairs of potential edges are compared and the potential edge of higher weight is transferred to new lists representing only $\lceil n/2 \rceil$ potential edges. If two compared weights are equal, the first potential edge is transferred so that for multiple potential edges of maximal weight, the potential edge with the lowest index is selected. If n is odd, the last potential edge is not a part of a pair and is also transferred to the new lists representing $\lceil n/2 \rceil$ potential edges. Finally, the gate calls itself on the newly generated lists. Note that the control flow including the recursion solely depends on publicly known values.

Gate 5.7 $([v], [w]) \leftarrow \rho^n_{\text{SEARCH_MAX}}([v_i]_{i\in[1,n]}, [w_i]_{i\in[1,n]}, [\mathit{weight}_i]_{i\in[1,n]})$

if $n = 1$ **then**
  $[\mathit{invalid}] \leftarrow \rho_{\text{EQZ}}([\mathit{weight}_1])$ ▷ $\mathit{weight}_1 = 0$?
  $[v_1] \leftarrow \rho_{\text{IF_ELSE}}([\mathit{invalid}], [0], [v_1])$
  $[w_1] \leftarrow \rho_{\text{IF_ELSE}}([\mathit{invalid}], [0], [w_1])$
  return $([v_1], [w_1])$
else
  $n' \leftarrow \lceil \frac{n}{2} \rceil$
  $\mathit{decisions} \leftarrow \lfloor \frac{n}{2} \rfloor$
  for $i \in [1, \mathit{decisions}]$ in parallel **do**
    $[\mathit{first}] \leftarrow 1 - \rho_{\text{LTZ}}([\mathit{weight}_{2i-1}] - [\mathit{weight}_{2i}])$ ▷ $\mathit{weight}_{2i-1} \geq \mathit{weight}_{2i}$?
    $[v'_i] \leftarrow \rho_{\text{IF_ELSE}}([\mathit{first}], [v_{2i-1}], [v_{2i}])$
    $[w'_i] \leftarrow \rho_{\text{IF_ELSE}}([\mathit{first}], [w_{2i-1}], [w_{2i}])$
    $[\mathit{weight}'_i] \leftarrow \rho_{\text{IF_ELSE}}([\mathit{first}], [\mathit{weight}_{2i-1}], [\mathit{weight}_{2i}])$
  end for
  if $n' > \mathit{decisions}$ **then**
    $[v'_{n'}] = [v_n]$
    $[w'_{n'}] = [w_n]$
    $[\mathit{weight}'_{n'}] = [\mathit{weight}_n]$
  end if
  return $\rho^{n'}_{\text{SEARCH_MAX}}([v'_i]_{i\in[1,n']}, [w'_i]_{i\in[1,n']}, [\mathit{weight}'_i]_{i\in[1,n']})$
end if

Complexity

We proceed by analyzing the complexity of a call to $\rho^n_{\text{SEARCH_MAX}}$. Assume that $n > 1$. For each pair of edges, the decision on which element to pick requires calling ρ_{LTZ}. The resulting complexity is $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations dominating the

complexity of the additionally required calls to $\rho_{\text{IF_ELSE}}$. Running the operations for each pair of edges in parallel yields a complexity of $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(n \cdot b)$ invocations. Further operations besides the recursive call run locally.

Now, assume that n is 1 or a power of 2, i.e., $n = 2^k$ for some $k \in \mathbb{N}_0$. For $k = 0$, $\rho^1_{\text{SEARCH_MAX}}$ has a complexity of $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations due to the calls to ρ_{EQZ} and $\rho_{\text{IF_ELSE}}$. For $k > 0$, the complexity of $\rho^{2^k}_{\text{SEARCH_MAX}}$ consists of $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(2^k \cdot b)$ invocations plus the complexity of $\rho^{2^{k-1}}_{\text{SEARCH_MAX}}$. It is easy to solve this recurrence relation yielding a complexity of

$$\mathcal{O}\left(\log b + \sum_{i=1}^{k} \log b\right) = \mathcal{O}(k \cdot \log b)$$

rounds and

$$\mathcal{O}\left(b + \sum_{i=1}^{k} 2^i \cdot b\right) = \mathcal{O}(2^k \cdot b)$$

invocations. Thus, the complexity of $\rho^n_{\text{SEARCH_MAX}}$ with n being a power of 2 consists of $\mathcal{O}(\log n \cdot \log b)$ rounds and $\mathcal{O}(n \cdot b)$ invocations.

Finally, let $n > 1$ be arbitrary. The number of rounds and invocations for $\rho^n_{\text{SEARCH_MAX}}$ is monotonically increasing with respect to n. Let $n_1 = 2^{\lfloor \log n \rfloor}$ and $n_2 = 2^{\lceil \log n \rceil}$. The round complexity of $\rho^{n_1}_{\text{SEARCH_MAX}}$ and $\rho^{n_2}_{\text{SEARCH_MAX}}$ is $\mathcal{O}(\lfloor \log n \rfloor \cdot \log b) = \mathcal{O}(\lceil \log n \rceil \cdot \log b) = \mathcal{O}(\log n \cdot \log b)$. The corresponding complexity regarding invocations is $\mathcal{O}(2^{\lfloor \log n \rfloor} \cdot b) = \mathcal{O}(2^{\lceil \log n \rceil} \cdot b) = \mathcal{O}(n \cdot b)$. As $n_1 \leq n \leq n_2$, it holds that $\rho^n_{\text{SEARCH_MAX}}$ requires $\mathcal{O}(\log n \cdot \log b)$ rounds and $\mathcal{O}(n \cdot b)$ invocations.

Correctness and Security

After having analyzed its complexity, we prove the security of the gate:

Theorem 5.3. *Gate $\rho^n_{\text{SEARCH_MAX}}$ implements functionality $\mathcal{G}^n_{\text{SEARCH_MAX}}$ statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties.*

Proof. Let k be the index of the optimal potential edge represented by the inputs of $\rho^n_{\text{SEARCH_MAX}}$, i.e., let k be the minimal number such that weight_k is maximal. We prove that $\rho^n_{\text{SEARCH_MAX}}$ implements $\mathcal{G}^n_{\text{SEARCH_MAX}}$ correctly, i.e., that it returns $([v_k], [w_k])$ if $\mathit{weight}_k > 0$ and $\{[0], [0]\}$, by induction.

For $n = 1$, $\rho^n_{\text{SEARCH_MAX}}$ implements $\mathcal{G}^n_{\text{SEARCH_MAX}}$ correctly as the output is set to $\{[0], [0]\}$ if and only if $\mathit{weight}_1 = 0$. Note that if $\mathit{weight}_1 \neq 0$ it holds that $\mathit{weight}_1 > 0$ as only nonnegative weights are allowed per definition.

Let $n > 1$ such that for all $n' < n$, $\rho^{n'}_{\text{SEARCH_MAX}}$ implements $\mathcal{G}^{n'}_{\text{SEARCH_MAX}}$ correctly. Then, the k'th potential edge is transferred to the input lists for $\rho^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$ if it is not compared to a potential edge of equal weight as it has maximal weight. If another potential edge at index ℓ has equal weight and is compared to the k'th potential edge, there exists an $i \in \mathbb{N}$ such that $k = 2i - 1$ and $\ell = 2i$. As $\mathit{weight}_{2i-1} \geq \mathit{weight}_{2i}$, the potential edge at index k is transferred to the input lists for $\rho^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$. Gate $\rho^n_{\text{SEARCH_MAX}}$ then returns $([v_k], [w_k])$ if $\mathit{weight}_k > 0$ and $([0], [0])$ otherwise as $\rho^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$ implements $\rho^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$ correctly.

We proceed by defining a simulator for $\rho^{n}_{\text{SEARCH_MAX}}$. Given its inputs, the simulator follows the flow of execution given by the gate. Note that this flow only depends on publicly known values. Each arithmetic operation of the BGW protocol can be simulated as the BGW protocol is perfectly secure. For each call to a gate, the corresponding simulator is called. In particular, for a call to $\rho^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$ the inductively defined simulator is called.

Gate $\rho^{1}_{\text{SEARCH_MAX}}$ implements $\mathcal{G}^{1}_{\text{SEARCH_MAX}}$ with statistical security as ρ_{EQZ} and $\rho_{\text{IF_ELSE}}$ are statistically and perfectly secure. Now, inductively assume that $\rho^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$ implements $\mathcal{G}^{\lceil n/2 \rceil}_{\text{SEARCH_MAX}}$ with statistical security. Then, all gates called by $\rho^{n}_{\text{SEARCH_MAX}}$ are statistically or perfectly secure. Thus, the resulting view is statistically indistinguishable from a real view on the gate's execution. □

Chapter 6

Privacy-Preserving Protocol for Maximum Weight Matching Approximation

In this chapter, we propose a secure protocol implementing the ideal functionality $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ for MWM approximation as introduced in Definition 4.3. We use Algorithm 4.3 that is also used to define $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ as the foundation of our protocol that we call $\pi_{\text{MWM}}^{N,\vartheta}$. In Section 6.1, we discuss how far this algorithm has to be modified before translating it to a secure protocol. Section 6.2 then contains our novel privacy-preserving protocol for MWM approximation, an analysis of its complexity and a proof of its security. In this section, we also review how the novel protocol can be used for secure crossover exchange.

6.1. Preliminary Algorithmic Considerations

Algorithm 4.3 which forms the basis of our privacy-preserving protocol must be modified slightly in order to translate it to a secure protocol. Recall that party P_i is represented by node i. In this section, we propose a modification of it including the generation of the graph and informing each party of its matched party or of it not being matched. Recall that the algorithm applies a permutation to its input graph and then executes multiple iterations where each iteration selects a heaviest edge, adds it to the matching, and removes it and all edges incident to it from the set of all edges. At most $\lfloor N/2 \rfloor$ edges can be in a matching so that only $\lfloor N/2 \rfloor$ iterations are required. We begin by briefly sketching how each of these steps can be executed in an oblivious way.

Generating and Permuting the Graph

Recall that to generate a graph, a fixed edge generation function ϑ of the properties given in Definition 4.2 is required. As discussed in Section 5.2, a secret graph can be used to represent the graph generated from the input values inputs_i from party P_i for $i \in [1, N]$. For each potential edge $(v, w) \in \hat{E}$, we can set $\mathit{weight}_{v,w} :=$

$\vartheta(\{inputs_v, inputs_w\})$. The resulting values $weight_{v,w}$ for $(v, w) \in \hat{E}$ can then be seen as a strict upper triangular adjacency matrix.

Modifying the edges and edge weights according to a permutation τ on the nodes requires to permute rows and columns of the adjacency matrix. As only the required entries above the main diagonal are computed, it is not immediately possible to swap two rows or columns. In addition, this approach needs to access all $\mathcal{O}(N^2)$ entries of the matrix. We use an alternative approach where we permute all inputs before generating the graph. This approach needs to access only $\mathcal{O}(N \cdot L)$ distinct variables where L is the constant number of inputs provided by each party.

Let G be the secret graph induced by ϑ without using a permutation, i.e., the graph with the adjacency matrix consisting of entries $weight_{v,w}$ as above. It holds that $\vartheta(inputs_{\tau(v)}, inputs_{\tau(w)})$ exactly corresponds to the potential edge $\{\tau(v), \tau(w)\}$ of G. Thus, by setting $weight_{v,w} = \vartheta(inputs_{\tau(v)}, inputs_{\tau(w)})$ for all $(v, w) \in \hat{E}$ we obtain a graph G' where each edge (v, w) of G' corresponds to an edge $\{\tau(v), \tau(w)\}$ of G. Thus, it holds that $G' = \tau^{-1}(G)$ using the notation from Section 4.1.

Recall that Algorithm 4.3 permutes a graph G to $G' := \tau(G)$. Thus, our modified approach deviates from the original one by replacing τ by τ^{-1}. Note that the behavior still is the same as τ is drawn uniformly at random from S_V. In particular, each τ in S_V being equally likely to be used implies that each τ^{-1} in S_V is equally likely to be used.

Selecting a Heaviest Edge

A heaviest edge $\{a', b'\}$ can be found by accessing all potential edges to determine the real edge, i.e., edge of weight greater than 0, of maximal weight. As the exact number of edges depends on the input, all $\mathcal{O}(N^2)$ potential edges need to be accessed so that no information about the number of edges is leaked.

Adding an edge to the Matching

The previously determined edge $\{a', b'\}$ has to be added to the matching. It remains to introduce a suitable data structure. Recall that for each matching edge $\{a', b'\}$, Algorithm 4.3 eventually reconstructs $\{a, b\} := \{\tau^{-1}(a), \tau^{-1}(b)\}$. Then, the ideal functionality $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ for MWM approximation as introduced in Definition 4.3 outputs b to party P_a and a to party P_b. Each party that is not matched to any other party receives output 0.

While Algorithm 4.3 generates a matching on the permuted graph and inverts the permutation on the matching in the end, we immediately invert it on a newly selected edge. This only requires inverting the permutation on a node twice in each iteration and thus $2 \cdot \lfloor N/2 \rfloor$ times. Inverting the permutation in the end instead would require to invert the permutation on $2N$ nodes, i.e., on v and $matching_partner_v$ for each $v \in V$.

Thus, two operations are necessary. First, we set $a := \tau(a')$ where τ is the permutation used on the inputs in the graph generation. Analogously, we set $b := \tau(b')$. Recall that we do not use τ^{-1} as in contrast to Algorithm 4.3, our graph generation computes $\tau^{-1}(G)$ instead of $\tau(G)$. Now, we can represent the matching by setting an output variable $matching_partner_a$ to b and an output variable $matching_partner_b$ to a. For each party P_i, $matching_partner_i$ is initialized to 0 before searching for the

first edge. Then, after execution of the greedy algorithm, *matching_partner*$_i$ can be revealed to the corresponding party P_i.

Removing Incident Edges

In the last step of each iteration, the weights of edges that are incident to $\{a', b'\}$ have to be set to 0 so that they cannot be picked as a part of the matching anymore. An alternative approach is to introduce Boolean variables *free*$_i$ initialized to True for each party P_i. The selection of a heaviest edge is then extended to only select edges $\{v, w\}$ where *free*$_v$ = *free*$_w$ = True. Thus, instead of removing all incident edges, it is sufficient to set *free*$_{a'}$ and *free*$_{b'}$ for a selected edge $\{a', b'\}$ to False. Note that this implies that *free*$_v$ = True if and only if node v is not covered by the current matching. The exact benefit of this modification is explained for the resulting protocol itself in Section 6.2 as it depends on the complexities of the used gates.

6.1.1. Specification of the Modified Algorithm

The resulting algorithm is given in Algorithm 6.1. It serves as an orientation for the corresponding protocol that we introduce in Section 6.2. The exact oblivious implementation of the algorithm's steps is left to be precisely defined by the protocol.

In lines 1 and 2, a permuted graph is built from the inputs. The newly introduced variables are initialized in lines 3 and 4. The search for the next edge $\{a', b'\}$ to add in each iteration is extended by filtering out all blocked edges in line 6. In lines 9-12, the output variables representing the matching are updated. Finally, line 13 blocks the nodes of the selected edge. Note that in this case, the permuted nodes are used again as the greedy construction of the matching itself works on the permuted graph.

Theorem 6.1. *Algorithm 6.1 exactly implements the ideal functionality* $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ *for MWM approximation as introduced in Definition 4.3.*

Proof. Let G be the graph generated by the ideal functionality that is used as input for Algorithm 4.3. Recall that as its first step, Algorithm 4.3 draws a permutation φ uniformly at random from S_V and generates $G' := \varphi(G)$. We compare an execution of Algorithm 6.1 using a permutation τ to an execution Algorithm 4.3 using a permutation $\varphi = \tau^{-1}$. This is sufficient to show that Algorithm 6.1 implements the ideal functionality as each τ being equally likely to be used implies that each $\varphi = \tau^{-1}$ in S_V is equally likely to be used.

Now, Algorithms 6.1 and 4.3 both generate $G' := \tau^{-1}(G)$. Observe that Algorithm 4.3 executes $x \leq \lfloor N/2 \rfloor$ iterations as each iteration adds an edge of G' to the matching and a matching cannot be larger than $\lfloor N/2 \rfloor$. Algorithm 6.1 executes exactly $\lfloor N/2 \rfloor$ iterations. We claim that in the first x iterations of both algorithms, the selected edges $\{a', b'\}$ are equal for both algorithms.

To prove our claim, it suffices to show that the sets E'_{max} of both algorithms are equal. Instead of deleting edges in each iteration, Algorithm 6.1 sets the flag *free*$_v$ of a matched node v to False. Algorithm 4.3 only deletes edges that are incident to a matched node. Thus, in each iteration it holds that the remaining edges $\{v, w\}$ of Algorithm 4.3 equal the edges $\{v, w\}$ of Algorithm 6.1 where flags *free*$_v$ and *free*$_w$ are set to True.

Algorithm 6.1 Modified Greedy MWM Algorithm Using Node Permutation (based on [Pre99]

Draw permutation τ uniformly at random from S_V
$\mathit{weight}_{v,w} \leftarrow \vartheta(\mathit{inputs}_{\tau(v)}, \mathit{inputs}_{\tau(w)})\ \forall (v,w) \in \hat{E}$

$\mathit{matching_partner}_v \leftarrow 0\ \forall v \in V$
$\mathit{free}_v \leftarrow$ True $\forall v \in V$
for $\lfloor N/2 \rfloor$ iterations **do**
    Let $E'_{\max} \subseteq \hat{E}$ exactly contain all edges $\{v,w\}$ in $\hat{E}$ where $\mathit{free}_v = \mathit{free}_w =$ True, $\mathit{weight}_{v,w} > 0$ and $\mathit{weight}_{v,w}$ is maximal
    Let $\{a',b'\} \in E'_{\max}$ be the minimal edge in $E'_{\max}$ w.r.t. "$\prec_d$"
    if some $\{a',b'\}$ was found **then**
        $a \leftarrow \tau(a')$
        $b \leftarrow \tau(b')$
        $\mathit{matching_partner}_a \leftarrow b$
        $\mathit{matching_partner}_b \leftarrow a$
        $\mathit{free}_{a'}, \mathit{free}_{b'} \leftarrow$ False
    end if
end for

for $v \in V$ **do**
    Output $\mathit{matching_partner}_v$ to party P_v
end for

For each edge $\{a',b'\}$ that Algorithm 4.3 adds to the matching, it outputs an edge $\{\varphi^{-1}(a'), \varphi^{-1}(b')\} = \{\tau(a'), \tau(b')\}$. Then, the ideal functionality defines the output of party $P_{\tau(a')}$ as $\tau(b')$ and vice versa. Observe that Algorithm 6.1 sets the value $\mathit{matching_partner}_{\tau(a')}$ to $\tau(b')$ and vice versa. As nodes a' and b' then are blocked, these values can never be overwritten again. Thus, Algorithm 6.1 also outputs $\mathit{matching_partner}_{\tau(a')} = \tau(b')$ to party $P_{\tau(a')}$ eventually and vice versa.

Finally, if $x < \lfloor N/2 \rfloor$ then Algorithm 6.1 executes more iterations than Algorithm 4.3. It is easy to see that in the last iterations $\lfloor N/2 \rfloor - x$ iterations of Algorithm 6.1, the output variables $\mathit{matching_partner}_v$ for $v \in V$ are not changed as no edge $\{a',b'\}$ is found. This is due to the fact that for all $\{v,w\} \in \hat{E}$ with $\mathit{weight}_{v,w} > 0$, flag free_v or flag free_w is set to False. If this was not the case, there would be an edge left that is not incident to any edge in the matching. This together with the equivalent behavior of Algorithms 6.1 and 4.3 in the first x iterations contradicts that Algorithm 4.3 only executes x iterations as there would be an edge left to be added to the matching. □

6.2. Protocol Specification

Our protocol for privacy-preserving MWM approximation is based on a fixed edge generation function ϑ of the properties given in Definition 4.2. The function ϑ induces a gate functionality $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ as in Definition 5.9. For a specific implementation of the protocol, a gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$ that implements $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ with statistical or perfect

security is required.

In the following, we describe the different phases of our protocol that is based on Algorithm 6.1. The complete protocol $\pi_{\text{MWM}}^{N,\vartheta}$ is given in Protocol 6.1 that we refer to in the following sections. By π_{KEP}^{N}, we denote the variant of $\pi_{\text{MWM}}^{N,\vartheta}$ where the edge generation for crossover exchange (cf. Section 5.3) is used. Section 6.2.1 describes the input phase where all parties provide their secret shared inputs and a permuted graph is built. In Section 6.2.2, we describe the matching phase itself where edges of the permuted graph are added to a matching. The output phase where each party receives the party it was matched to if it was matched is explained in Section 6.2.3. We analyze the protocol's overall complexity in Section 6.2.4 and prove its correctness and security in Section 6.2.5.

6.2.1. Input Phase

The input phase is given in lines 1-7 of Protocol 6.1. In the first step, all parties provide their inputs using Shamir's secret sharing scheme. Recall that each party has L inputs. Then, a permuted secret graph $\hat{G}$ is generated as described in lines 1 and 2 of Algorithm 6.1. Recall that in Algorithm 6.1, $\mathit{weight}_{v,w}$ is set to $\vartheta(\mathit{inputs}_{\tau(v)}, \mathit{inputs}_{\tau(w)})$ for all $(v, w) \in \hat{E}$.

We implement the graph generation by permuting the inputs of different parties using the secure permutation gate $\rho_{\text{PERM}}^{N,L}$ from Section 5.4.1. The gate permutes all inputs according to a permutation τ in $\mathcal{O}(\log^2 N \cdot \log b)$ rounds and $\mathcal{O}(N \cdot \log^2 N \cdot (b+L))$ invocations. Note that τ is drawn uniformly at random by $\rho_{\text{PERM}}^{N,L}$ except for a probability that is negligible in κ. Then, we set $[\mathit{weight}_{v,w}] = \rho_{\text{EDGE_GEN}}^{\vartheta}([\mathit{inputs}_{\tau(v)}], [\mathit{inputs}_{\tau(w)}])$ for all $(v, w) \in \hat{E}$ in parallel. Thus, our approach yields the adjacency matrix as given by the initial graph permutation of Algorithm 6.1 but using secret shared integers as its entries.

To conclude, $[\mathit{weight}_{v,w}]_{(v,w)\in\hat{E}}$ is the adjacency matrix of a graph permuted by a permutation drawn uniformly at random from S_V except for a negligible probability. Note that the output of $\rho_{\text{PERM}}^{N,L}$ does also contain $[\tau(1)], ..., [\tau(N)]$ for $\tau \in S_V$ such that $\hat{G}' = \tau^{-1}(\hat{G})$. These values are written to $[\mathit{id}_1], ..., [\mathit{id}_N]$ to allow for later reconstruction of matching edges as in lines 9 and 10 of Algorithm 6.1.

Complexity

Retrieving the parties' inputs is possible in $\mathcal{O}(1)$ rounds using $\mathcal{O}(L)$ invocations. The gate $\rho_{\text{EDGE_GEN}}^{\vartheta}$ is called $\mathcal{O}(N^2)$ times in parallel and $\rho_{\text{PERM}}^{N,L}$ is called once. Thus, the first phase of the protocol requires $\mathcal{O}(\log^2 N \cdot \log b)$ rounds and $\mathcal{O}(N \cdot \log^2 N \cdot (b + L))$ invocations plus one time the rounds and $\mathcal{O}(N^2)$ times the invocations required by $\rho_{\text{EDGE_GEN}}^{\vartheta}$.

6.2.2. Matching Phase

The matching phase is given in lines 8-36 of Protocol 6.1. We use secret shared integers $[\mathit{matching_partner}_v]$ and $[\mathit{free}_v]$ for all $v \in V$ representing the corresponding flags used in Algorithm 6.1. The matching phase executes $\lfloor N/2 \rfloor$ iterations of the

Protocol 6.1 $\pi^{N,\vartheta}_{\text{MWM}}$: Protocol for Maximum Weight Matching Approximation

for $v \in V$ in parallel **do**
  Get inputs $[inputs_v]$ from party P_v
end for
$([inputs'_1], ..., [inputs'_N], [id_1], ..., [id_N]) \leftarrow \rho^{N,L}_{\text{PERM}}(([inputs_1], ..., [inputs_N]))$
for $(v, w) \in \hat{E}$ in parallel **do**
  $[weight_{v,w}] \leftarrow \rho^{\vartheta}_{\text{EDGE_GEN}}([inputs'_i], [inputs'_j])$
end for

$[matching_partner_v] \leftarrow [0]\ \forall v \in V$
$[free_v] \leftarrow [1]\ \forall v \in V$
for $\lfloor N/2 \rfloor$ iterations **do**
  for $e \in [1, \hat{M}]$ in parallel **do**
    $(v, w) \leftarrow edge_to_v_w_e$
    $[v'_e] \leftarrow [v]$
    $[w'_e] \leftarrow [w]$
    $[weight'_e] \leftarrow [weight_{v,w}] \cdot [free_v] \cdot [free_w]$
  end for
  $([a'], [b']) \leftarrow \rho^{\hat{M}}_{\text{SEARCH_MAX}}([v'_e]_{e\in[1,\hat{M}]}, [w'_e]_{e\in[1,\hat{M}]}, [weight'_e]_{e\in[1,\hat{M}]})$

  for $v \in V$ in parallel **do**
    $[hit_a'] \leftarrow \rho_{\text{EQZ}}(v - [a'])$ ▷ $v = a'$?
    $[candidate_a_v] \leftarrow [hit_a'] \cdot [id_v]$
    $[hit_b'] \leftarrow \rho_{\text{EQZ}}(v - [b'])$ ▷ $v = b'$?
    $[candidate_b_v] \leftarrow [hit_b'] \cdot [id_v]$
  end for
  $[a] \leftarrow \sum_{v=1}^{N} [candidate_a_v]$
  $[b] \leftarrow \sum_{v=1}^{N} [candidate_b_v]$
  for $v \in V$ in parallel **do**
    $[hit_a] \leftarrow \rho_{\text{EQZ}}(v - [a])$ ▷ $v = a$?
    $[matching_partner_v] \leftarrow \rho_{\text{IF_ELSE}}([hit_a], [b], [matching_partner_v])$
    $[hit_b] \leftarrow \rho_{\text{EQZ}}(v - [b])$ ▷ $v = b$?
    $[matching_partner_v] \leftarrow \rho_{\text{IF_ELSE}}([hit_b], [a], [matching_partner_v])$
  end for

  for $v \in V$ in parallel **do**
    $[hit] \leftarrow \rho_{\text{EQZ}}(v - [a']) + \rho_{\text{EQZ}}(v - [b'])$ ▷ $v = a' \vee v = b'$?
    $[free_v] \leftarrow \rho_{\text{IF_ELSE}}([hit], [0], [free_v])$
  end for
end for

for $v \in V$ in parallel **do**
  reveal $[matching_partner_v]$ to party P_v
end for

loop starting in line 10 of the protocol. We divide each iteration into three parts that we discuss in the following.

Selection of the Next Edge

Lines 6 and 7 of Algorithm 6.1 that contain the search for the next edge $\{a', b'\}$ to add to the matching correspond to lines 11-17 of the protocol. Among all permuted real edges that are not already blocked and have maximal weight, the minimal one with respect to "$\prec_d$" has to be selected. To this end, the best edge search gate $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ from Section 5.5 is used. Recall that from a list of potential edges, it returns the earliest real edge in such list among all real edges of maximal weight. If the list contains no real edge, i.e., all weights are 0, then $([0], [0])$ is returned.

To use $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$, it is necessary to generate its inputs such that the gate cannot select an edge that is already blocked and such that it selects a minimal edge with respect to "$\prec_d$" if multiple candidates exist. To achieve that blocked edges cannot be selected anymore, we use a copy of the weight of each edge $\{v, w\}$ multiplied by the corresponding flags free_v and free_w. Thus, blocked edges have a modified weight of 0. Ordering the input lists according to "$\prec_d$" leads to $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ returning an edge that is minimal with respect to "$\prec_d$" among all edges of maximal modified weight.

We use a static mapping from the index of an edge in the ordering given by "$\prec_d$" to the corresponding nodes. Recall that "$\prec_d$" is defined as

$$(1,2) \prec_d (1,3) \prec_d ... \prec_d (1,N) \prec_d (2,3) \prec_d (2,4) \prec_d ... \prec_d (2,N) \prec_d ... .$$

For a node v, there are $N - v$ edges (v, w): $(v, v+1), (v, v+2), ..., (v, N)$. For nodes $v < w$, there are $w - v - 1$ edges $(v, w') \prec_d (v, w)$: $(v, v+1), (v, v+2), ..., (v, w-1)$. Thus, the index of a given edge $(v, w) \in \hat{E}$ is therefore given by

$$\begin{aligned}\sum_{v'=1}^{v-1}(N - v') + w - v - 1 + 1 &= (v-1)\cdot N - \sum_{v'=1}^{v-1} v' + w - v \\ &= vN - N + w - v - \frac{v(v-1)}{2} \\ &= -\frac{1}{2}v^2 + (N - \frac{1}{2})v + w - N.\end{aligned}$$

The required mapping can therefore be computed by each party locally and only once as given in Algorithm 6.2.

Algorithm 6.2 Edge Index to Node Mapping

for $(v, w) \in \hat{E}$ **do**
  $\mathit{edge_to_v_w}_{-\frac{1}{2}v^2+(N-\frac{1}{2})v+w-N} \leftarrow (v, w)$
end for

Then, inputs $[v'_e]_{e\in[1,\hat{M}]}$, $[w'_e]_{e\in[1,\hat{M}]}$ and $[\mathit{weight}'_e]_{e\in[1,\hat{M}]}$ for $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ can be generated by a parallelized loop. For each $e \in \hat{M}$, $[v'_e]$ is set to $[v]$ and $[w'_e]$ is set to $[w]$ where (v, w) is the e'th potential edge according to the ordering "$\prec_d$". Furthermore, $[\mathit{weight}'_e]$ is set to $[\mathit{weight}_{v,w}] \cdot [\mathit{free}_v] \cdot [\mathit{free}_w]$. Finally, $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ is called on the generated inputs and returns two secret shared integers $[a']$ and $[b']$.

To conclude, the selection of the next edge yields $[a']$ and $[b']$. If no edge was found, then it holds that $a' = b' = 0$. Otherwise, the next edge to add to the matching is $\{a', b'\}$.

Update of the Matching

In lines 18-31 of the protocol, the matching is updated according to lines 9-12 in Algorithm 6.1. Recall that from the previous step, $[a']$ and $[b']$ are given that represent an edge on the permuted graph if there are edges left which are not blocked yet. In the case that no edge was found, it holds that $a' = b' = 0$.

The original parties corresponding to the permuted edge $\{a', b'\}$ have to be reconstructed if an edge was found. Recall that to this end, the original edge $\{a, b\}$ can be reconstructed by setting $a := \tau(a') = id_{a'}$ and $b := \tau(b') = id_{b'}$. The protocol must not leak if an edge $\{a', b'\}$ was found as otherwise, it would leak the size of the computed matching. Recall that the edges and hence also the size of the resulting matching depend on the secret inputs of the parties. Thus, the protocol must in particular not reveal the values a' and b'.

For the following, assume that an edge $\{a', b'\}$ was found, i.e., it does not hold that $a' = b' = 0$. Lines 18-25 of the protocol are used to compute $[a] = [id_{a'}]$ and $[b] = [id_{b'}]$ representing the original edge $\{a, b\}$. As a' and b' must not be revealed, it is necessary to obliviously access $[id_v]$ for all $v \in V$ similarly to the pattern for oblivious register that is sketched in Section 2.4.6. For each $v \in V$, we compare v to $[a']$ and set a secret shared integer $[candidate_a_v]$ to the secret shared integer $[id_v]$ if $v = a'$. Otherwise, we set $[candidate_a_v]$ to $[0]$. This can be achieved obliviously by setting $[candidate_a_v]$ to $[id_v]$ multiplied by the output of $\rho_{\text{EQZ}}(v - [a'])$ that is 1 if $v = a'$ and 0 otherwise. Finally, we set

$$
\begin{aligned}
[a] &= \sum_{v=1}^{N} [candidate_a_v] \\
&= \sum_{v=1}^{N} \rho_{\text{EQZ}}(v - [a']) \cdot [id_v] \\
&= \sum_{v=1}^{N} [\tau(v)] \cdot \begin{cases} 1, & \text{if } v - a' = 0, \\ 0, & \text{otherwise} \end{cases} \\
&= \sum_{v=1}^{N} [\tau(v)] \cdot \begin{cases} 1, & \text{if } v = a', \\ 0, & \text{otherwise} \end{cases} \\
&= [\tau(a')].
\end{aligned}
$$

Note that writing intermediate results to secret shared integers $[candidate_a_v]$ first and locally computing the sum of all intermediate results in the end allows for parallelization of the iteration over all $v \in V$. Analogously, we compute secret shared integers $[candidate_b_v]$ to obtain $[b]$.

Now, it is necessary to set $[matching_partner_a]$ to $[b]$ and $[matching_partner_b]$ to $[a]$. Again, the values a and b must not be revealed as otherwise it would be leaked which parties are matched to all participating parties. Thus, we obliviously access $[matching_partner_v]$ for all $v \in V$. We discuss how it is possible to set $[matching_partner_a]$ to $[b]$ while the symmetrical case follows trivially.

The gate $\rho_{\text{IF_ELSE}}$ can be used to keep $[\textit{matching_partner}_v]$ unchanged by setting it to $[\textit{matching_partner}_v]$ again if $v \neq a$. If $v = a$, it can be used to set $[\textit{matching_partner}_v]$ to $[b]$. We set

$$\begin{aligned}[\textit{matching_partner}_v] &= \rho_{\text{IF_ELSE}}(\rho_{\text{EQZ}}(v - [a]), [b], [\textit{matching_partner}_v]) \\ &= \begin{cases} [b], & \text{if } v - a = 0, \\ [\textit{matching_partner}_v], & \text{if } v - a \neq 0 \end{cases} \\ &= \begin{cases} [b], & \text{if } v = a, \\ [\textit{matching_partner}_v], & \text{if } v \neq a. \end{cases}\end{aligned}$$

The iteration over all $v \in V$ can be parallelized.

Now, we discuss the case where $a' = b' = 0$, i.e., no edge is to be added to the matching. Note that in this case, $\rho_{\text{EQZ}}(v - [a'])$ and $\rho_{\text{EQZ}}(v - [b'])$ evaluate to $[0]$ for all $v \in V$ as $0 \notin V$. Thus, the secret shared integers $[\textit{matching_partner}_v]$ for all $v \in V$ remain unchanged. In particular, this approach does not require to use a condition as in line 8 of Algorithm 6.1 which would leak if an edge was found.

Blocking of Matched Nodes

In lines 32-35 of the protocol, the matched nodes are blocked according to line 13 in Algorithm 6.1. Recall that the secret shared integers $[a']$ and $[b']$ represent the matched nodes or $a' = b' = 0$ if no nodes were matched. We need to set $[\textit{free}_{a'}]$ and $[\textit{free}_{b'}]$ to $[0]$ if $a', b' > 0$. From the previous part on the update of the matching, recall that values a' and b' must not be revealed. Thus, we need to use an oblivious access pattern as sketched in Section 2.4.6 again.

We iterate over all nodes $v \in V$. For each v, we test if v equals $[a']$ or $[b']$. If no edge was found, it holds that $a' = b' = 0$ so that no $v \in V$ equals a' or b'. In particular, we set

$$\begin{aligned}[\textit{hit}] &= \rho_{\text{EQZ}}(v - [a']) + \rho_{\text{EQZ}}(v - [b']) \\ &= \left\{\begin{matrix} [1] & \text{if } v - a' = 0 \\ [0] & \text{if } v - a' \neq 0 \end{matrix}\right\} + \left\{\begin{matrix} [1] & \text{if } v - b' = 0 \\ [0] & \text{if } v - b' \neq 0 \end{matrix}\right\} \\ &= \left\{\begin{matrix} [1] & \text{if } v = a' \\ [0] & \text{if } v \neq a' \end{matrix}\right\} + \left\{\begin{matrix} [1] & \text{if } v = b' \\ [0] & \text{if } v \neq b' \end{matrix}\right\} \\ &= \begin{cases} [1] & \text{if } v = a' \vee v = b', \\ [0] & \text{otherwise.} \end{cases}\end{aligned}$$

Note that the last equation holds as $a' = b' = 0 \neq v$ or $a' \neq b'$ due to $\{a', b'\}$ being an edge so that it can never hold that $a' = b' = v$. The iterations over $v \in V$ can be executed in parallel. If $a' = b' = 0$, no nodes are blocked.

Complexity

We conclude our discussion of the matching phase by discussing its complexity. The number of iterations of the loop beginning in line 10 of the protocol is $\lfloor N/2 \rfloor$.

The selection of the next edge in lines 11-17 of the protocol executes $\hat{M} \in \mathcal{O}(N^2)$ parallel iterations. Each iteration requires two multiplications. Thus, the loop requires $\mathcal{O}(1)$ rounds and $\mathcal{O}(N^2)$ invocations. The complexity of $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ consists of $\mathcal{O}(\log \hat{M} \cdot \log b) = \mathcal{O}(\log N \cdot \log b)$ rounds and $\mathcal{O}(\hat{M} \cdot b) = \mathcal{O}(N^2 \cdot b)$ invocations. Thus, lines 11-17 require $\mathcal{O}(\log N \cdot \log b)$ rounds and $\mathcal{O}(N^2 \cdot b)$ invocations.

Updating the matching in lines 18-31 of the protocol uses two loops while the additions in lines 24 and 25 run locally. Each of the loops executes N iterations in parallel. The bodies of both loops contain calls to ρ_{EQZ} that require $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations. Further operations that require communication are multiplications and calls to $\rho_{\text{IF_ELSE}}$ that require only $\mathcal{O}(1)$ rounds and $\mathcal{O}(1)$ invocations. Thus, both loops require $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(N \cdot b)$ invocations.

The part blocking matched nodes in lines 32-35 of the protocol also executes N iterations in parallel. Again, the complexity of each iteration is dominated by the calls to ρ_{EQZ} that require $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations. Thus, the loop requires $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(N \cdot b)$ invocations.

To conclude, each iteration requires $\mathcal{O}(\log N \cdot \log b)$ rounds and $\mathcal{O}(N^2 \cdot b)$ invocations. Thus, the matching phase's complexity consists of $\mathcal{O}(N \cdot \log N \cdot \log b)$ rounds and $\mathcal{O}(N^3 \cdot b)$ invocations.

Advantage of Blocking Nodes Instead of Edges

Recall that our approach blocks nodes in lines 32-35 of the protocol using $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(N \cdot b)$ invocations. In line 15 of the protocol, the corresponding flags $[\textit{free}_v]$ for $v \in V$ are multiplied to the weight of each edge. Thus, all blocked edges have a weight of 0 when calling gate $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ so that they cannot be selected. This approach raises the question why blocking nodes is used as an intermediate step instead of setting the weights of blocked edges to 0 immediately.

We argue that our approach is more efficient. Note that obliviously updating values as in lines 32-35 of the protocol requires to call the gate ρ_{EQZ} for each value that may be accessed. Obliviously updating the flags $[\textit{free}_v]$ for $v \in V$ only requires $\mathcal{O}(N)$ calls to ρ_{EQZ} while obliviously updating the edge weights would require $\mathcal{O}(N^2)$ calls to ρ_{EQZ}. Thus, our approach only calls ρ_{EQZ} that requires $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations $\mathcal{O}(N)$ times instead of $\mathcal{O}(N^2)$ times. While it needs to execute two additional multiplications for each edge, it is more efficient as a multiplication requires only a single invocation which is less expensive than a call to ρ_{EQZ}.

6.2.3. Output Phase

The output phase corresponds to lines 37-39 of Protocol 6.1. According to Algorithm 6.1 where each party receives its output in lines 16-18, it is sufficient to reveal $[\textit{matching_partner}_v]$ to party P_v for all $v \in V$.

Complexity

As described in Section 2.4.5, revealing $[\textit{matching_partner}_v]$ to party P_v for all $v \in V$ is possible in a single invocation and thus a single round.

6.2.4. Complexity

For the complexity analysis of the protocol $\pi^{N,\vartheta}_{\text{MWM}}$ for MWM approximation, recall that by N we denote the number of parties and the number of nodes as each party is represented by one node. Furthermore, recall that by L we denote the number of inputs provided by each party and that by b, we denote the number of bits used for secret shared integers as described in Section 2.4.4. We let $\rho^{\vartheta}_{\text{EDGE_GEN}}$ be a gate implementing $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ for an arbitrary edge generation function ϑ of the properties given in Definition 4.2.

Recall that the input phase as discussed in Section 6.2.1 requires $\mathcal{O}(\log^2 N \cdot \log b)$ rounds and $\mathcal{O}(N \cdot \log^2 N \cdot (b + L))$ invocations plus one time the rounds and $\mathcal{O}(N^2)$ times the invocations required by $\rho^{\vartheta}_{\text{EDGE_GEN}}$. The matching phase that we have discussed in Section 6.2.2 requires $\mathcal{O}(N \cdot \log N \cdot \log b)$ rounds and $\mathcal{O}(N^3 \cdot b)$ invocations. For the output phase from Section 6.2.3, only a single invocation is required.

Thus, the total protocol requires $\mathcal{O}(N \cdot \log N \cdot \log b)$ rounds plus the rounds required for one call to $\rho^{\vartheta}_{\text{EDGE_GEN}}$. Furthermore, it requires $\mathcal{O}(N^3 \cdot b + N \cdot \log^2 N \cdot L)$ invocations plus $\mathcal{O}(N^2)$ times the invocations required by $\rho^{\vartheta}_{\text{EDGE_GEN}}$.

If we use the edge generation gate $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ from Section 5.3 to use the protocol for crossover exchange, we obtain a specific complexity for a given gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$. Recall that one call to $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ requires $\mathcal{O}(\log b)$ rounds and $\mathcal{O}(b)$ invocations. Thus, the complexity of the protocol π^{N}_{KEP} for crossover exchange consists of $\mathcal{O}(N \cdot \log N \cdot \log b)$ rounds and $\mathcal{O}(N^3 \cdot b + N \cdot \log^2 N \cdot L)$ invocations. Also recall that $L \in \mathcal{O}(H + R)$ where H denotes the number of considers HLAs and R denotes the number of used regions.

6.2.5. Correctness and Security

We conclude the chapter on our novel protocol for MWM approximation by proving its correctness and security. Recall that as our protocol is defined using a generic edge generation function ϑ and a corresponding gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$. To prove the security of a protocol for fixed ϑ, it is necessary that a corresponding gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$ exists and is secure.

Theorem 6.2. *Let the number of bits b be chosen such that it is not smaller than the security parameter κ, i.e., $b \geq \kappa$. Furthermore, let a specific edge generation function ϑ of the properties stated in Definition 4.2 be given that induces a functionality $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ as in Definition 5.9. Let there be an edge generation gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$ that implements functionality $\mathcal{G}^{\vartheta}_{\text{EDGE_GEN}}$ perfectly or statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties. Then, protocol $\pi^{N,\vartheta}_{\text{MWM}}$ implements functionality $\mathcal{F}^{N,\vartheta}_{\text{MWM}}$ statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties.*

Proof. Algorithm 6.1 implements the ideal functionality correctly as stated by Theorem 6.1. Furthermore, in the protocol specification in Section 6.2 we have already discussed how each operation executed by Algorithm 6.1 is correctly implemented by the protocol. Recall that gate $\rho^{N,L}_{\text{PERM}}$ is called once and is correct except for some probability that is negligible in κ as shown in Theorem 5.2. In Theorem 5.3, we have shown that gate $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ implements functionality $\mathcal{G}^{n}_{\text{SEARCH_MAX}}$ correctly. In addition, we have discussed the correctness of all elementary gates in Section 5.1. Thus,

we can conclude that except for some probability that is negligible in κ, our protocol $\pi^{N,\vartheta}_{\text{MWM}}$ implements the ideal functionality $\mathcal{F}^{N,\vartheta}_{\text{MWM}}$ from Definition 4.3 correctly.

We proceed by defining a probabilistic polynomial-time simulator $\mathcal{S}$ for the protocol as follows: The input stage is simulated as in the security proof of the BGW protocol by Asharov and Lindell [AL17]. For each corrupted party, a random tape is selected uniformly at random and used to share the inputs of the party that are known to $\mathcal{S}$ according to Shamir's secret sharing scheme. For other parties, $\mathcal{S}$ shares the value 0. In particular, sharing means that the shares that would be sent to the corrupted parties in a real execution of the protocol are appended to the simulated views of the corrupted parties.

For the computation stage, $\mathcal{S}$ uses the previously selected shares to follow the protocol's order of execution. All required operations of the BGW protocol that are discussed in Section 2.4.3 can be sufficiently simulated as the primitives are perfectly secure. The elementary gates from Section 5.1 are statistically or perfectly secure and thus can also be simulated by calling their corresponding simulators. The gates $\rho^{N,L}_{\text{PERM}}$ and $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ are statistically secure as stated by Theorems 5.2 and 5.3. In addition, gate $\rho^{\vartheta}_{\text{EDGE_GEN}}$ is perfectly or statistically secure by the premise of this theorem. Thus, calls to gates $\rho^{N,L}_{\text{PERM}}$, $\rho^{\hat{M}}_{\text{SEARCH_MAX}}$ and $\rho^{\vartheta}_{\text{EDGE_GEN}}$ can also be simulated by calling their corresponding simulators. Our protocol does not explicitly reveal any secret shared value in the computation stage except for operations inside of used gates that are already shown to be secure [CH10].

For the final stage, recall that the simulator $\mathcal{S}$ has knowledge of the ideal functionality's outputs to all corrupted parties. The stage is simulated as given by Asharov and Lindell [AL17] by incorrectly revealing the outputs for the corrupted party in such a way that the ideal functionality's outputs result. Each value that would be revealed to some corrupted party in a real protocol execution is defined by the corresponding shares held by all parties. For each output to a corrupted party, $\mathcal{S}$ instead appends shares that match the expected output of the corrupted parties to their views. This is possible by selecting a polynomial uniformly at random from all polynomials of maximal degree $t-1$ that represent the expected output and match the actual shares of the corrupted parties. Note that this is possible as for shares of at most $t-1$ corrupted parties and an expected output, at least one such polynomial exists.

The simulator trivially runs in polynomial time as the number of operations of $\pi^{N,\vartheta}_{\text{MWM}}$ is polynomial in the input size of $\mathcal{S}$ and all simulators called for the gates run in polynomial time. As already stated, all used gates are statistically or perfectly secure. Thus, we can infer that the resulting view is statistically indistinguishable from a real view on the protocol's execution. Finally, we can conclude that our protocol $\pi^{N,\vartheta}_{\text{MWM}}$ for MWM approximation implements the ideal functionality $\mathcal{F}^{N,\vartheta}_{\text{MWM}}$ for MWM approximation with statistical security in the presence of a static and passive adversary controlling a minority of corrupted parties. □

Concerning the protocol's variant π^{N}_{KEP} for crossover exchange, the edge generation function ϑ is given by ϑ_{KEP} as in Definition 5.10. The corresponding edge generation gate $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ is given in Gate 5.5 and is statistically secure in the presence of a passive adversary controlling a minority of corrupted parties as shown in Theorem 5.1. Thus, Theorems 6.2 in particular implies that protocol π^{N}_{KEP} implements functionality $\mathcal{F}^{N,\vartheta}_{\text{MWM}}$

using edge generation ϑ_{KEP} for crossover exchange statistically secure in the given setting.

Chapter 7

Evaluation

In this chapter, we present an evaluation of our privacy-preserving protocol for MWM approximation as introduced in Chapter 6. In Section 7.1, we discuss the implementation of the protocol. Section 7.2 given an overview of the setup that acts as the basis of our evaluation. The results of our evaluation are presented in Section 7.3.

7.1. Implementation in MP-SPDZ

We have implemented our privacy-preserving protocol $\pi_{\text{MWM}}^{N,\vartheta}$ for MWM approximation using the framework MP-SPDZ [Kel20]. MP-SPDZ allows for the implementation of SMPC protocols using high-level Python code. The protocols are compiled into bytecode that then can be executed on virtual machines that are also provided by the framework. Among other protocols, MP-SPDZ in particular allows to run BGW-like protocols based on Shamir's secret sharing scheme for a passive adversary and an honest majority. Thus, it matches the general setting of our protocol.

Recall that the protocol $\pi_{\text{MWM}}^{N,\vartheta}$ is defined for an edge generation function ϑ and a corresponding gate $\rho_{\text{EDGE_GEN}}^{\vartheta}$ as discussed in Section 5.2. We base our evaluation on the protocol's variant π_{KEP}^{N} for privacy-preserving crossover exchange as we have already introduced a corresponding edge generation gate $\rho_{\text{EDGE_GEN}}^{\vartheta_{\text{KEP}}}$ in Section 5.3. Note that this choice influences the protocol's performance in the input phase defined in Section 6.2.1 where an adjacency matrix of the graph to compute a matching on is calculated. The subsequent phases and in particular the matching phase defined in Section 6.2.2 do not use the edge generation gate and run oblivious of the actual edge weights computed in the input phase.

On top of the protocol's implementation in MP-SPDZ, we have designed and implemented an environment that allows for the generation of the parties' inputs to the protocol and for easy testing. The exact inputs that are expected to be provided by each party for crossover exchange are exactly defined in Definition 5.10. For verification purposes, we have developed unit tests that compare the protocol's outputs to its expected outputs. To this end, we have implemented the ideal functionality for MWM approximation that is given in Definition 4.3 as a conventional program. Recall that the ideal functionality is defined using our modified randomized greedy algorithm given in Algorithm 4.3. Thus, our conventional program extends Algo-

rithm 4.3 by the edge generation for crossover exchange as defined in Definition 5.10. Also recall that due to the randomization, it is not immediately possible to compare inputs and outputs. To tackle this problem, we have added an extension of the protocol that reveals the permutation that is selected by the permutation gate $\rho^{n,\ell}_{\text{PERM}}$ from Section 5.4. This extension is only used during unit tests as it violates the original protocol specification. Our environment then reads the permutation from the protocol's outputs and forwards it to the conventional algorithm. The conventional algorithm selects the same permutation and our environment compares its outputs to the protocol's outputs.

7.2. Evaluation Setup

We run MP-SPDZ on a single LXC container having access to 64 logical cores and 256 GB of RAM. The container runs on a machine equipped with an AMD EPYC 7702P with a base clock of 2.0 GHz. Each party runs a single process.

We connect all parties by virtual network interfaces to simulate a network locally as illustrated in Figure 7.1. To add bandwidth restrictions and network delay to our network simulation, we apply traffic shaping by using `tc` to configure the traffic control of the virtual interfaces in the Linux kernel. In particular, we add qdiscs that are schedulers for data packets to simulate bandwidth restrictions and network delay. For each party, the outgoing traffic is limited to some fixed bandwidth. A virtual network bridge forwards the packets coming from the parties to their corresponding destination parties. The outgoing traffic of the network bridge to each single party again is limited to some fixed bandwidth. Thus, bandwidth restrictions for the upload and download of each party are simulated. Furthermore, the network bridge delays all transmissions to simulate network delay.

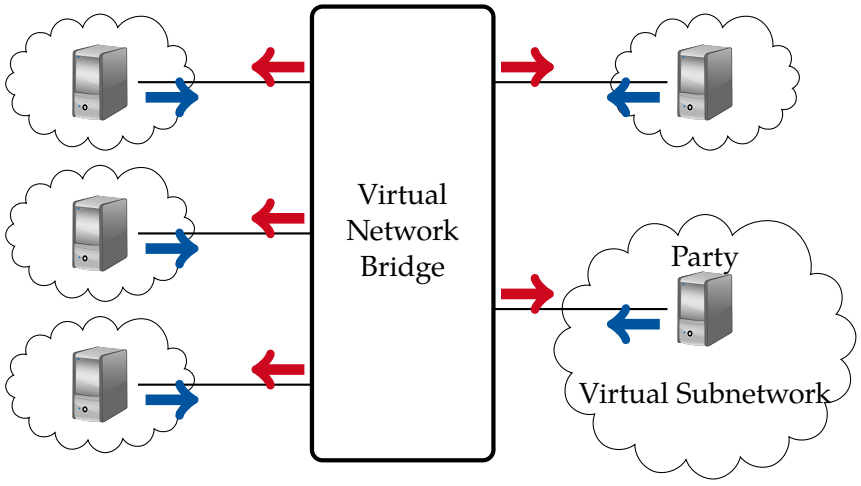


Figure 7.1.: Virtual network setup that is used for the evaluation. Red arrows denote simulated bandwidth restrictions and network delay. Blue arrows denote simulated bandwidth restriction only.

Note that our simulation of a network could be slower than a real network of the same properties regarding bandwidth and latency due to insufficient CPU speed or

similar restrictions. Thus, our results may not exactly represent the protocol's run time in a more realistic setting while offering an upper bound for such run time.

Choice of Parameters

For our evaluation, we simulate 1 Gbit/s ingoing and outgoing bandwidth for each party and a network delay of 5 ms if not stated otherwise. Furthermore, we must fix the number of bits b used for the representation of integers (cf. Section 2.4.4) and the security parameter κ that are to be used by our protocol. We follow the default configuration of MP-SPDZ that is $b = 64$ and $\kappa = 40$.

As inputs for our protocol, we have used real-world kidney exchange data that originates from the United Network for Organ Sharing (UNOS) and contains the relevant medical inputs used in KEP matching runs between October 2010 and December 2020.[1] The number of HLAs considered in the data is $H = 207$ while $R = 11$ regions are used. Thus, the number of inputs per parties amounts to $L = 444$ (cf. Definition 5.10). We give value here due to its impact on the permutation of inputs that requires $\mathcal{O}(N \cdot \log^2 N \cdot (b + L))$ invocations in the context of our protocol (cf. Sections 6.2.1).

7.3. Results

The remainder of this chapter is structures as follows: In Section 7.3.1, we evaluate the overall run time and generated network traffic for our protocol π^N_{KEP}. Then, we analyze how the run time is distributed among different parts of the protocol in Section 7.3.2. Finally, we assess the impact of parallelization, network delay and bandwidth on the protocol's performance in Sections 7.3.3 and 7.3.4.

7.3.1. Performance of the Protocol

We execute our protocol ten times for each number of parties $N \in \{5, 10, 15, ..., 60\}$. Note that higher N would yield less reliable results due to cores being shared by multiple parties as only 64 cores are available for our benchmarks.

The measured run times and global traffic, i.e., the sum of incoming traffic for all parties, are given in Table 7.1 and Figure 7.2. For up to 20 parties, the protocol runs in less than a minute. In less than ten minutes, an MWM between up to 35 parties can be approximated. For 60 parties, the protocol's run time is on average still below 61 minutes while for 50 parties or less, it is less than half an hour. The traffic generated by all parties together exceeds 100 GB for 35 parties. For 60 parties, it amounts to 1162 GB. The measured run times deviate up to 15.5% from their corresponding arithmetic mean. We note that for higher N, this deviation eventually decreases down to 8.1%. Among all protocol runs for fixed N, the traffic is constant.

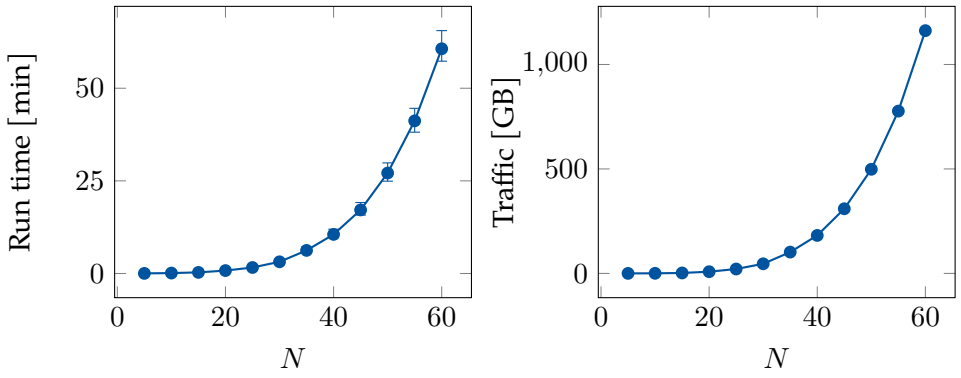
We observe that run time and traffic are polynomially bounded. This matches our previous theoretical analysis of the protocol's complexity in Section 6.2.4. Recall that the protocol requires $\mathcal{O}(N \cdot \log N)$ rounds and $\mathcal{O}(N^3)$ invocations for a fixed number

[1] The data reported here have been supplied by the United Network for Organ Sharing as the contractor for the Organ Procurement and Transplantation Network. The interpretation and reporting of these data are the responsibility of the author(s) and in no way should be seen as an official policy of or interpretation by the OPTN or the U.S. Government.

Table 7.1.: Arithmetic means of run times and global traffic for protocol π^N_{KEP} using 1 Gbit/s bandwidth and 5 ms delay. Data gathered from 10 iterations.

N	5	10	15	20	25	30
Run time	1.8 s	7.1 s	17.9 s	47.4 s	1.6 min	3.2 min
Traffic	26.7 MB	410.4 MB	2.1 GB	8.1 GB	21.0 GB	46.0 GB

N	35	40	45	50	55	60
Run time	6.2 min	10.5 min	17.2 min	27.1 min	41.2 min	60.6 min
Traffic	102.0 GB	181.8 GB	309.4 GB	498.3 GB	776.8 GB	1162.0 GB



(a) Run times (b) Traffic

Figure 7.2.: Arithmetic means, minima and maxima for run times and traffic of protocol π^N_{KEP} using 1 Gbit/s bandwidth and 5 ms delay. Data gathered from 10 iterations.

of inputs per party L and a fixed number of used bits b. As a single invocation implies that up to N parties send a share to up to $N-1$ parties each, the global traffic produced by a single invocation is quadratic in N. Thus, we can infer that the global traffic as depicted in Figure 7.2b is in $\mathcal{O}(N^5)$.

Recall that to the best of our knowledge there exist no protocols for privacy-preserving matching on general graphs. As the benchmarked protocol's variant π^N_{KEP} is specifically designed for crossover exchange, we briefly compare our results to those of the protocol for privacy-preserving kidney exchange by Breuer et al. [Bre+20]. Recall that their protocol that we have discussed in Section 3.4 has exponentially growing run time and traffic. In particular, it already requires a run time of two hours for eight parties and 13 hours for nine parties. Thus, our protocol requires significantly less run time for 60 parties than the protocol by Breuer et al. for only nine parties. Regarding global traffic, their protocol is more efficient for five parties (10 MB) while already generating more traffic for nine parties (5.5 GB) than ours does for ten parties. While our protocol has a significantly better performance, it is to be noted that the work of Breuer et al. addresses the general KEP which is NP-complete [ABS07] while even optimally solving the MWM problem is possible in polynomial time [Edm65a].

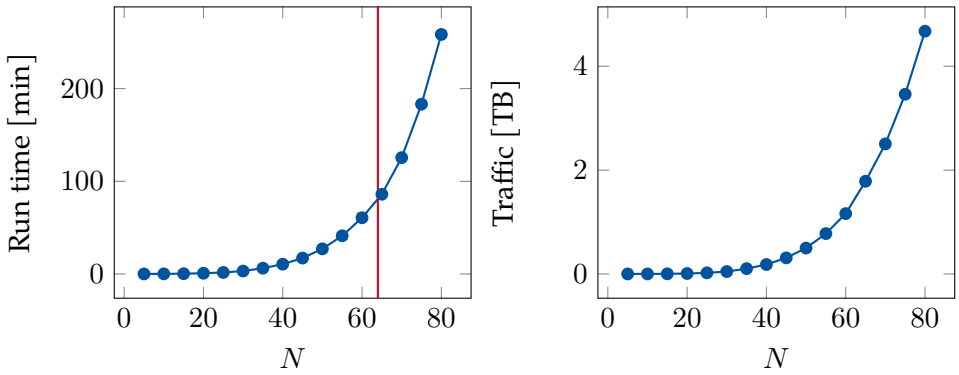
Performance for More than 60 Parties

To assess the protocol's performance for $N > 60$, we additionally execute it three times for $N \in \{65, 70, 75, 80\}$. Recall that only 64 virtual cores being available leads to parties sharing cores and therefore only yields an upper bound regarding run time. For more than 80 parties, the amount of RAM available to us is insufficient to run the protocol. Note that in a real network with different parties using different machines, a higher RAM usage would be feasible as for $N = 80$, our limit still corresponds to only 3.2 GB of RAM per party. Also note that the benchmarks yield the exact generated traffic as it is independent of the computation resources.

The results are given in Table 7.2 and Figure 7.3. For $N = 80$, the run time of protocol π^N_{KEP} is on average 4.3 hours. The generated traffic increases up to 4.7 TB. During the three iterations, the run time for fixed N deviated at most 1.4% from its arithmetic mean. The generated traffic is constant for fixed N.

Table 7.2.: Arithmetic means of run times and global traffic for protocol π^N_{KEP} using 1 Gbit/s bandwidth and 5 ms delay. Data gathered from 3 iterations.

N	65	70	75	80
Run time	1.4 h	2.1 h	3.1 h	4.3 h
Traffic	1.8 TB	2.5 TB	3.5 TB	4.7 TB



(a) Run times (b) Traffic

Figure 7.3.: Arithmetic means, minima and maxima for run times and traffic of π^N_{KEP} using 1 Gbit/s bandwidth, 5 ms delay and up to 80 parties. The vertical line for the run times indicates where the number of parties N exceeds the number of available virtual cores. Data gathered from 10 iterations for $N \leq 60$ and 3 iterations for $N > 60$. Note that minima and maxima are hardly visible as the measured deviation is at most 1.4%.

These results indicate that our protocol has a feasible run time for a significantly larger number of parties than for instance the protocol by Breuer et al. [Bre+20] does. Again, recall that their protocol addresses a problem that has a significantly higher complexity. Regarding crossover exchange, it is to be noted that many kidney exchange programs only execute their matching runs once a month or less frequently [Bir+21]. As our protocol runs in less than 4.3 hours for up to 80 parties and the run time increases only polynomially, it is to be expected that it still runs in

less than a month for a number of parties that is significantly higher than 80. Thus, it could be used for KEP in practice allowing for a significantly higher number of parties to participate in a matching run than the protocol by Breuer et al. [Bre+20] does.

7.3.2. Performance of the Protocol's Phases

We proceed by analyzing the run times of different steps of the protocol. As the most dominant phases, we identify the input phase and the matching phase. We split the input phase into the permutation of inputs by gate $\rho^{n,\ell}_{\text{PERM}}$ and the edge generation consisting of all calls to $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$.

The results are given in Figure 7.4. It is immediately visible that the matching phase's run time dominates the run time of the entire protocol. For $N = 5$, its share of the overall run time is 52.0%. This share increases up to 82.7% for $N = 60$. For $N > 10$, the permutation of inputs requires less time than the generation of edges using gate $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$. For $N = 60$, it only has a share of 4.1% of the overall run time. Recall that the matching phase does not depend on the used generation of edges. Thus, our measurements of the run time of the matching phase itself are universal for the generic protocol $\pi^{N,\vartheta}_{\text{MWM}}$ when used with an arbitrary edge generation. As the matching phase dominates the run time of protocol π^{N}_{KEP}, we infer that similar overall run times are to be expected for other arbitrary edge generations if they are not significantly more demanding than ours.

Note that the increase of run time regarding the call to $\rho^{n,\ell}_{\text{PERM}}$ is not smooth as it can be seen in Figure 7.4a. Recall that the sorting network used by $\rho^{n,\ell}_{\text{PERM}}$ (cf. Section 5.4.1) pads N to a power of 2. The observed behavior corresponds exactly to this padding. Note that each significant increase of run time corresponds to N exceeding some power of 2. Such increases can be seen for $N = 20$ and $N = 35$ where the padded number of nodes is increased from 16 to 32 and from 32 to 64.

Recall that for a fixed number of inputs per party L and a fixed number of used bits b, the permutation of inputs requires $\mathcal{O}(\log^2 N)$ rounds and $\mathcal{O}(N \cdot \log^2 N)$ invocations. For the edge generation, $\mathcal{O}(1)$ rounds and $\mathcal{O}(N^2)$ invocations are required while the matching phase requires $\mathcal{O}(N \cdot \log N)$ rounds and $\mathcal{O}(N^3)$ invocations. Thus, the increasing impact of the matching phase on the overall run time matches the theoretical analysis where the complexity of the matching phase is also higher than the complexities of input permutation and graph generation. Furthermore, the edge generation's share of the run time of input permutation and edge generation together increases with increasing N despite its round complexity being $\mathcal{O}(1)$ compared to $\mathcal{O}(\log^2 N)$. Regarding invocations, the complexity of the edge generation is higher than the one of the input permutation. This leads to the assumption that the impact of rounds on the run time is less than the impact of invocations on it. The analysis of the input permutation's run time attests an efficient implementation of secure permutation by gate $\rho^{n,\ell}_{\text{PERM}}$. Recall that the permutation is used to achieve invariance under permutation (cf. Definition 4.1) so that for a party P_i, no bias depending on its index i exists. Thus, we can conclude that removing such bias by achieving invariance under permutation only yields a small overhead regarding run time for our implementation.

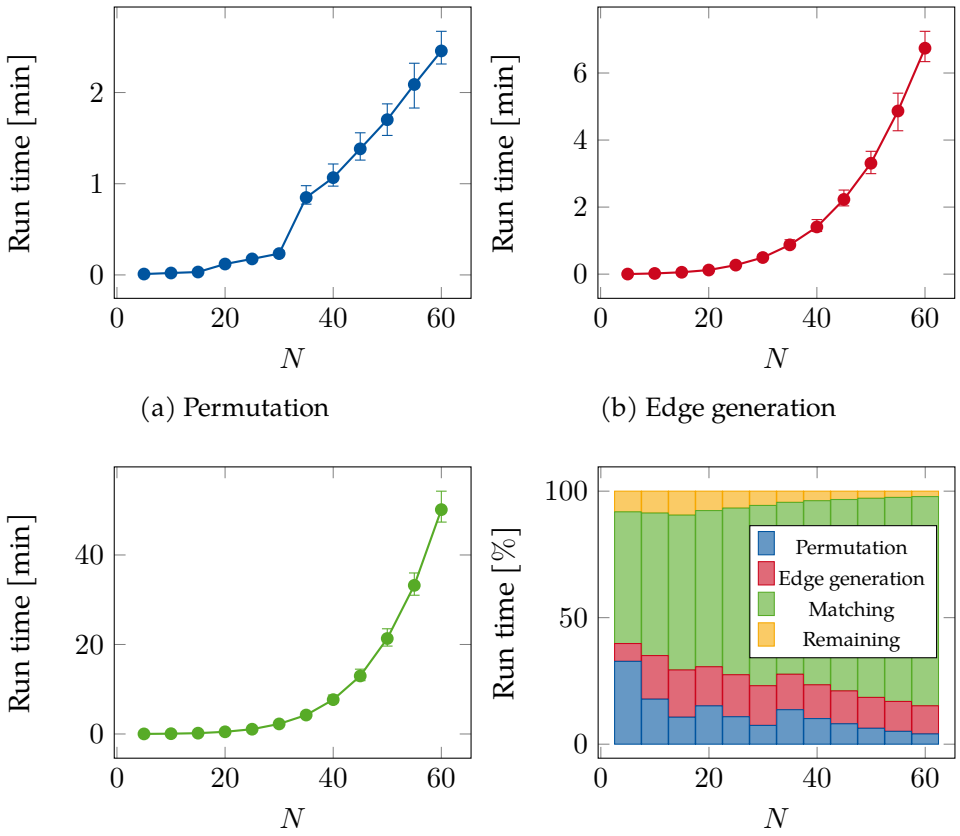


Figure 7.4.: Arithmetic means, minima and maxima for run times of the input permutation using gate $\rho^{n,\ell}_{\text{PERM}}$, the edge generation using gate $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ and the matching phase, and their shares of the overall run time using 1 Gbit/s bandwidth and 5 ms delay. Data gathered from 10 iterations.

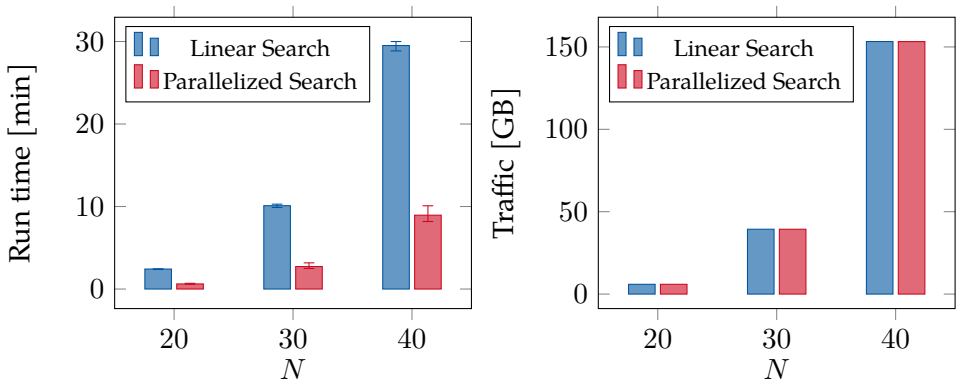
7.3.3. Impact of Parallelization of Comparisons

Recall that to search for an edge of maximal weight to add to the matching, we utilize gate $\rho^{n}_{\text{SEARCH_MAX}}$ that significantly parallelizes comparisons as discussed in Section 5.5. To assess the impact of this parallelization, we benchmark a modification of the protocol where the search for an edge to add to the matching (cf. lines 11-17 of Protocol 6.1) is replaced by a trivial linear search over all edges. In this protocol modification, we also remove the randomization, i.e., the call to $\rho^{n,\ell}_{\text{PERM}}$ to permute edges and the reconstruction of the selected edge $\{a, b\} \leftarrow \{id_{a'}, id_{b'}\}$ in lines 18-25 of Protocol 6.1. We compare the resulting protocol to a protocol where the parallelized edge search is utilized but randomization is also removed. The removal of randomization removes potential distortions that were observed in Section 7.3.2 due to the padding behavior of gate $\rho^{n,\ell}_{\text{PERM}}$. It does not obstruct an analysis of the parallelization's impact as the considered parallelization is independent of the randomization.

We execute the benchmarks for $N \in \{20, 30, 40\}$ to get an intuition on the impact of the parallelization. As for the variant without the additional parallelization is expected to have a significantly higher run time, we only execute 3 iterations of it for

each N. The results are given in Figure 7.5.

A comparison of the results confirms that the parallelization of searching for matching edges significantly improves the protocol's run time. The use of gate $\rho^n_{\text{SEARCH_MAX}}$ decreases the overall run time by 1.8 minutes for $N = 20$, by 7.4 minutes for $N = 30$, and by 20.6 minutes for $N = 40$. Relatively, the variant using a linear search has 3.92 times the run time of the variant where gate $\rho^n_{\text{SEARCH_MAX}}$ is used for $N = 20$. This factor decreases to 3.7 for $N = 30$ and to 3.3 for $N = 40$. Regarding traffic, parallelization yields nearly no difference as it requires a similar number of invocations that only is distributed among less rounds. This explains why the parallelization does not yield an asymptotical improvement regarding run time, i.e., why the multiplicative overhead of using a linear search that is 3.92 for $N = 20$ does not increase with greater N.



(a) Run times (b) Traffic

Figure 7.5.: Arithmetic means, minima and maxima for run times and traffic of variants of protocol π^N_{KEP} using 1 Gbit/s bandwidth and 5 ms where one variant executes a linear search to find edges and one variant utilizes the proposed parallelized search. Both variants do not permute inputs. Data gathered from 10 iterations for the variant using a parallelized search and 3 iterations for the variant using a linear search.

7.3.4. Impact of Latency and Bandwidth

Finally, we evaluate the impact of bandwidth restrictions and network delay on the run time of our protocol π^N_{KEP}. Recall that all previous benchmarks used a bandwidth of 1 Gbit/s and 5 ms of delay. We proceed by also benchmarking the protocol using a less performant network with only 100 Mbit/s bandwidth and a delay of 10 ms. To assess the influence of bandwidth and delay on the protocol's performance alone, we also run benchmarks for mixed settings with 1 Gbit/s bandwidth and 10 ms delay or 100 Mbit/s bandwidth and 5 ms delay.

The results are given in Figure 7.6. In the least performant network using a bandwidth of 100 Mbit/s and a delay of 10 ms, the maximal measured run time for $N = 60$ on average amounts of 94.1 minutes instead of 60.6 minutes as for the most performant network using 1 Gbit/s bandwidth and 5 ms of delay. Overall, the highest measured increase of run time is 105.6% that is given for $N = 10$. For over 40 parties, the increase of run time is below 70%. Recall that this overhead corresponds to using

only a tenth of the original bandwidth and double of the original network delay. This indicates that running our protocol still is feasible for a large number of parties in networks that are less performant than our default setup using a bandwidth of 1 Gbit/s and 5 ms of delay.

Doubling the network delay while using a bandwidth of 1 Gbit/s yields an overhead of 62.4% for $N = 5$ that monotonically decreases down to 5.9% for $N = 60$. Recall from Section 2.4.5 that the number of rounds directly influences the run time overhead resulting from network latency. As our protocol requires $\mathcal{O}(N \cdot \log N)$ rounds and $\mathcal{O}(N^3)$ invocations, the number of rounds is dominated by the number of invocations for increasing N. This exactly matches our observation where the significance of doubling the network delay gradually decreases.

Decreasing the bandwidth by a factor of ten while continuing to use 5 ms of delay yields an overhead of below 90.0% which decreases down to 52.1% for $N = 60$. Together with the previous observation of the run time only increasing insignificantly for sufficiently large N, this result attests that the network performance does not act as a bottleneck regarding the run time. Thus, we deduce that the individual processing speeds of the parties are the determining factor for the protocol's run time.

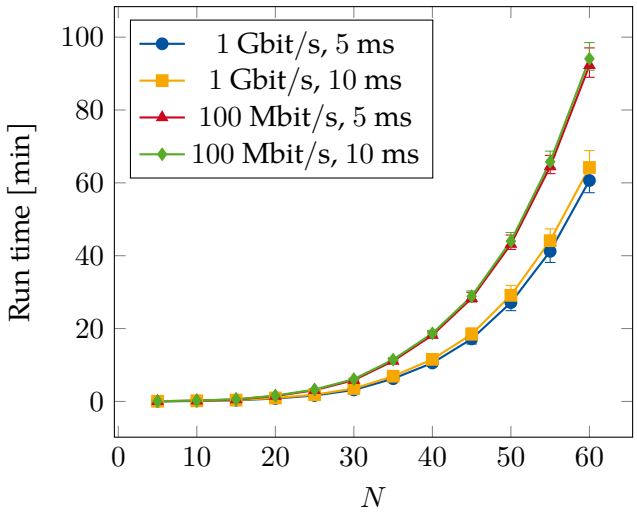


Figure 7.6.: Arithmetic means, minima and maxima for run times of π_{KEP}^{N} using different network settings. Data gathered from 10 iterations.

Chapter 8

Conclusion

In this thesis, we have designed, implemented, and evaluated a protocol for privacy-preserving maximum weight matching approximation on general graphs. It runs in less than 5 hours for up to 80 parties and, to the best of our knowledge, is the first contribution to privacy-preserving matchings on general graphs ever. We have demonstrated how our protocol can be used for crossover exchange in the domain of kidney exchange. In this domain it constitutes the first solution that explicitly handles prioritization for crossover exchange. Based on secure protocols like ours, existing kidney exchange programs may increase their privacy or even new programs may emerge.

Our protocol is based on a greedy MWM approximation algorithm that has an approximation ratio of $1/2$. We have identified a weakness of the original algorithm that leads to an undesired bias in favor of some parties. We have presented and used a modification of said algorithm that rules out such a bias by satisfying a requirement that we call invariance under permutation. Based on our modification of the greedy algorithm, we have defined the ideal functionality $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ for privacy-preserving MWM approximation.

Having defined said ideal functionality, we have implemented a general and secure protocol $\pi_{\text{MWM}}^{N,\vartheta}$ for privacy-preserving MWM approximation on general graphs. It allows for a generic secure graph generation to be used so that it can easily be adapted to different use cases where a MWM approximation needs to be computed to match different parties. We have proven our protocol to be statistically secure in the presence of a static and passive adversary controlling a minority of corrupted parties. Furthermore, we have analyzed its complexity proving that it runs in only $\mathcal{O}(N \cdot \log N)$ rounds and $\mathcal{O}(N^3)$ invocations for N parties. As an example of a secure graph generation, we have implemented a secure building block that specifically allows for our protocol to be used for crossover exchange.

We have implemented our protocol in MP-SPDZ [Kel20] and have evaluated its performance regarding run time and generated network traffic. For the evaluation, we have specifically used our graph generation for privacy-preserving crossover exchange. Our benchmarks have attested that the protocol runs in less than 5 hours for up to 80 parties, generating less than 5 TB of network traffic. For less than 60 parties, we have shown it to run in even less than an hour generating less than 1 TB of network traffic. Our results indicate that our protocol may be used for crossover

exchange using even larger numbers of parties.

Future Work

As a first approach to privacy-preserving matchings on general graphs, our protocol may act as the basis for future research into multiple directions.

While we have used several optimizations in our privacy-preserving protocol already, its performance could potentially be increased by additional techniques that we did not take into closer consideration. In particular, *oblivious RAMs* as introduced by Goldreich and Ostrovsky [GO96] have been adapted to SMPC [EKR19] and could potentially be used to allow for more efficient oblivious register access. Furthermore, each party could use multithreading as also supported by MP-SPDZ to decrease the impact of local computations on the overall run time. We did not pursue this approach as it requires multiple CPU cores per party to obtain meaningful benchmarks while we were limited to 64 cores shared between all parties.

Furthermore, different algorithmic approaches may be researched. For instance, $2/3$-approximations, as introduced by Drake and Hougardy [DH03b; DH05], Pettie and Sanders [PS04] and Hanke and Hougardy [HH10], achieve a higher approximation ratio than $1/2$ as for our approach. Exact MWM algorithms could also be used to provide optimal solutions while a significant decrease of performance is to be assumed.

For a specific use case, it can be evaluated in practice how the quality of matchings computed by our protocol can compete with optimal MWMs as our approximation ratio of $1/2$ only offers a lower bound. In particular, this may be researched for the specific prioritizations that are used by kidney exchange programs that only execute crossover exchanges. In this case, a dynamic model where new parties join a system, possibly take part in multiple matching runs, and leave when they are successfully matched could be simulated. Such dynamic models are already introduced for bartering in general by Wüller [Wü18].

We were only able to measure the performance of our protocol for up to 80 parties due to limited available resources. Thus, future benchmarks using a higher number of parties would be of interest to assess the protocol's scalability in detail.

Finally, our protocol could act as the foundation for protocols that are secure against a more powerful adversary. A major direction for such work would be to research similar protocols that are secure against an active adversary.

Appendix A

Random Edge Selection

Recall that as discussed in Section 4.1, Algorithm 4.1 does not specify which edge is to be selected if multiple options of maximal weight exist. Picking edge $\{a, b\}$ uniformly at random from the set of all edges of maximal weight in each iteration can be seen as the natural randomization inside the framework given by the greedy algorithm. This is since the only decision influencing the output that can be made inside the given framework is perfectly randomized. In Section A.1, we briefly discuss how this approach can be used to obtain an ideal functionality that is different from ours as in Definition 4.3. Section A.2 sketches how the ideal functionality can be implemented by a modification of our protocol. A short evaluation of the resulting different protocols is finally given in Section A.3.

A.1. Ideal Functionality Based on Random Edge Selection

We formally define the discussed random selection of edges as follows:

Definition A.1 (Random Edge Selection)**.** *A variant of Algorithm 4.1 performs random edge selection if in each iteration, it selects an edge $\{a, b\}$ uniformly at random from all possible edges of highest weight.*

It is easy to see that a variant of Algorithm 4.1 that performs random edge selection is invariant under permutation (cf. Definition 4.1). This is due to the fact that if edges $\{a, b\}$ are chosen uniformly at random, the algorithm does not depend on the values that represent the nodes at all. In contrast, for instance Algorithm 4.2 did specifically choose edges with respect to order "$\prec_d$" that depends on the nodes' values. We note that a variant of Algorithm 4.1 that performs random edge selection implements another functionality than Algorithm 4.3 that is also invariant under permutation:

Proposition A.1. *A variant of Algorithm 4.1 that performs random edge selection implements another functionality than Algorithm 4.3*

Proof. We prove our claim by providing a counterexample. Let there be a graph with four nodes and edges $\{1, 2\}, \{2, 3\}, \{3, 4\}$ of equal weight. We claim that in the first step, edge $\{2, 3\}$ is less likely to be chosen by Algorithm 4.3 than each of the other edges. This behavior yields that output $\mathfrak{M} = \{\{2, 3\}\}$ is less than half as likely to

result as $\mathfrak{M} = \{\{1,2\},\{3,4\}\}$. It is easy to see that for random edge selection, $\{2,3\}$ should be chosen with probability $1/3$ in the first step such that output $\mathfrak{M} = \{\{2,3\}\}$ is exactly half as likely to result as $\mathfrak{M} = \{\{1,2\},\{3,4\}\}$.

There are $4! = 24$ possible permutations of V that all are equally likely to be selected. $3! = 6$ of these permutations satisfy that $\tau(1) = 1$. For these permutations, only $\{1,2\}$ can be selected in the first iteration. Symmetrically, 6 permutations satisfy that $\tau(4) = 1$ implying that $\{3,4\}$ is selected in the first iteration. Edge $\{2,3\}$ can only be chosen if $\tau(2) = 1 \vee \tau(3) = 1$. For $\tau(2) = 1$, 3 out of $3! = 6$ permutations satisfy that $\tau(1) < \tau(3)$ leading to selection of edge $\{1,2\}$. For the other 3 permutations, $\{2,3\}$ is chosen. The case for $\tau(3) = 1$ is symmetrical where selection of edge $\{1,2\}$ is replaced by selection of edge $\{3,4\}$. In total, there are 9 permutations each such that $\{1,2\}$ or $\{3,4\}$ is chosen while only 6 permutations lead to selection of $\{2,3\}$. □

We present an algorithm that performs random edge selection by using a random permutation to shuffle all edges in Algorithm A.1. Algorithm A.1 can then serve as a basis for a secure protocol. It remains to prove that the approach performs random edge selection. To this end, we start with two basic observations concerning orderings given by random permutations that are necessary to prove that the algorithm performs random edge selection.

Algorithm A.1 Greedy MWM With Edge Permutation (based on [Pre99])

Draw permutation τ from $S_{[1,M]}$ uniformly at random
Let the edges be $e_1 \prec_d e_2 \prec_d ... \prec_d e_M$
$\prec_r \leftarrow \{(e_i, e_j) \mid i,j \in [1,M] \wedge \tau(i) < \tau(j)\}$
$\mathfrak{M} \leftarrow \emptyset$
while $E \neq \emptyset$ **do**
  Let $E_{\max} \subseteq E$ exactly contain all edges of maximal weight in E
  Let $\{a,b\} \in E_{\max}$ be the minimal edge in $E_{\max}$ w.r.t. "$\prec_r$"
  $\mathfrak{M} \leftarrow \mathfrak{M} \cup \{\{a,b\}\}$
  $E \leftarrow E \setminus \{\{v,w\} \in E \mid \{v,w\} \cap \{a,b\} \neq \emptyset\}$
end while
return $\mathfrak{M}$

Lemma A.1. *For some finite set X, let a strict total order $\prec$ on X be drawn such that each possible strict total order is equally likely to be selected. Then, for a subset $X' \subseteq X$ that is selected statistically independent of $\prec$, the strict total order $\prec'$ on X' implied by $\prec$ is equally likely to be selected as each other order from all possible strict total orders on X'.*

Proof. Due to statistical independence of $\prec$ and X', each strict total order on X still is equally likely to be selected as $\prec$ given X'. Each of the strict total orders on X' is implied by $(|X| - |X'|)! \cdot \binom{|X|}{|X'|}$ strict total orders on X as the remaining $|X| - |X'|$ elements can have an arbitrary order and $|X'|$ elements use arbitrary positions in the ordering among all $|X|$ elements. As each order on X is equally likely to be selected and each order on X' corresponds to the same number of orders on X, each order on X' also is equally likely to be selected. □

Lemma A.2. *For some finite set $X \neq \emptyset$, let a strict total order $\prec$ on X be drawn such that each possible strict total order is equally likely to be selected. Let $X' := X \setminus \{\min_{\prec} X\}$ where*

$\min_{\prec} X$ is the minimal element of X with respect to $\prec$. Then, the strict total order $\prec'$ on X' implied by $\prec$ is equally likely to be selected as each other order from all possible strict total orders on X'.

Proof. There are $|X|!$ possible strict total orders $\prec$ on X. For each possible but fixed $\min_{\prec} X$, there are $|X'|! = (|X| - 1)!$ possible strict total orders $\prec'$ on X'. Thus, each $\prec'$ on X' corresponds to exactly one $\prec$ on X with a fixed minimal element. As each $\prec$ is equally likely to be selected, each order $\prec'$ given a minimal element with respect to $\prec$ also is equally likely to be selected. □

Proposition A.2. *Algorithm A.1 performs random edge selection.*

Proof. Each strict total order on all edges given by $\prec_r$ is equally likely to be used due to random selection of τ. We partition E into a minimal number of sets of edges $F_1, ..., F_\ell \subseteq E$ where all edges in one set have equal weight. For all of these sets, each internal order given by $\prec_r$ still is equally likely to be used due to Lemma A.1 as the edge weights and the order $\prec_r$ are statistically independent. After each iteration of the algorithm, we remove all edges from $F_1, ..., F_\ell$ that have been removed from E due to being selected or blocked. In addition to the proposition, we claim that for all $F_i, i \in [1, \ell]$, each internal order on F_i implied by $\prec_r$ still is equally likely to be used after any iteration.

We prove the claim by showing that in each iteration where the additional claim has held before, it also holds after the iteration while an edge is selected as required by random edge selection. Suppose that the additional claim holds before some iteration. The iteration selects some edge from F_i where i is minimal such that $F_i \neq \emptyset$. As each order on F_i is equally likely to be used, each edge in F_i has an equal chance of $1/|F_i|$ to be selected due to being minimal in such order. Note that in this case, $F_i = E_{\max}$. Now, we update $F_1, ..., F_\ell$ as previously described. It remains to be shown that our additional claim holds after conducting the updates.

First, we remove the newly selected edge from F_i. As it was the minimal element in F_i with respect to the internal order, each internal order on the new F_i still is equally likely to be used due to Lemma A.2. Now, we remove all additionally blocked edges from $F_1, ..., F_\ell$. To this end, we replace each F_j by an $F'_j \subseteq F_j$. As F'_j does only depend on the previously selected edge that was already removed from $F_1, ..., F_\ell$, it is selected statistically independent of the internal order on F_j. Thus, Lemma A.1 yields that for F'_j, each internal order also is equally likely to be used. □

Replacing Algorithm 4.3 in Definition 4.3 by Algorithm A.1 yields a new ideal functionality. Recall that this is the case as both algorithms implement different functionalities as shown in Proposition A.1.

A.2. Protocol Based on Random Edge Selection

In this section, we briefly sketch how our protocol $\pi^{N,\vartheta}_{\text{MWM}}$ for MWM approximation (cf. Protocol 6.1) can be modified to perform random edge selection. Recall that protocol $\pi^{N,\vartheta}_{\text{MWM}}$ utilizes gate $\rho^{n,\ell}_{\text{PERM}}$ from Section 5.4 to permute its inputs. This gate can instead be utilized to permute all edges instead to order them randomly according to Algorithm A.1. Note that permuting all edges makes it difficult to utilize an adjacency matrix as in Protocol 6.1. Instead, lists can be used to represent all edges

as in lines 11-16 of Protocol 6.1. This requires substantial changes of the protocol's matching phase that we do not elaborate in detail here. Instead, we present an easier solution that we call *virtual edge permutation*:

Recall that the permutation gate $\rho^{n,\ell}_{\text{PERM}}$ from Section 5.4 generates a random number for each item to permute and then sorts all items by their random numbers. Instead of permuting all edges according to an order "$\prec_d$" as given in Algorithm A.1, we can simulate such ordering. For each edge e, a random number rnd_e is drawn uniformly at random from some interval as in gate $\rho^{n,\ell}_{\text{PERM}}$. Recall that it is possible to draw a fixed number of random numbers such that the probability of two random numbers being equal is negligible. Assuming that all random numbers are different, we let $e \prec_d e'$ if and only $rnd_e < rnd_{e'}$ for edges e and e'. It can easily be seen that this yields an order "$\prec_d$" of the same properties as in Algorithm A.1.

Now, it is possible to modify gate $\rho^{n}_{\text{SEARCH_MAX}}$ from Section 5.5 that is responsible for selecting an edge to add to the matching. Recall that when comparing the weights of a first edge e and a second edge e', gate $\rho^{n}_{\text{SEARCH_MAX}}$ chooses edge e if and only if

$$\sigma(e) \geq \sigma(e').$$

To ensure that eventually an edge that is minimal with respect to "$\prec_d$" among all edges of maximal weight is returned, this predicate can be modified. In the case of both weights being equal, it has to be ensured that the edge with the minimal random value is selected. To this end, the following predicate can be used:

$$\sigma(e) > \sigma(e') \vee \left(\sigma(e) = \sigma(e') \wedge rnd_e \leq rnd_{e'}\right).$$

Note that this approach replaces a single comparison executed by gate $\rho^{n}_{\text{SEARCH_MAX}}$ by three comparisons. It is also necessary to modify gate $\rho^{n}_{\text{SEARCH_MAX}}$ even more so that the required random values are carried along. However, as the nodes are not permuted, it is not necessary to invert a permutation on every edge that is added to the matching as in lines 18-25 of Protocol 6.1.

A.2.1. Virtual Node Permutation

Note that the approach of modifying gate $\rho^{n}_{\text{SEARCH_MAX}}$ instead of explicitly applying a permutation by using gate $\rho^{n,\ell}_{\text{PERM}}$ can also be used to implement our original functionality $\mathcal{F}^{N,\vartheta}_{\text{MWM}}$ in another way than used by our protocol $\pi^{N,\vartheta}_{\text{MWM}}$. In particular, it is possible to remove the permutation of inputs from the input phase of Protocol 6.1 and just to draw the necessary random values instead.

Let $\prec'_d$ be the order of edges after permutation if gate $\rho^{n,\ell}_{\text{PERM}}$ was applied. For two unshuffled edges e_1 and e_2, it is possible to check their order according to $\prec'_d$ by inspecting their corresponding nodes' randoms. As an example, let $e_1 = (1,2), e_2 = (3,4), rnd_1 = 3, rnd_2 = 8, rnd_3 = 1, rnd_4 = 5$. It follows that $e_2 \prec'_d e_1$ as e_1 would be permuted to $e'_1 = (2,4)$ and e_2 would be permuted to $e'_2 = (1,3)$ where $e'_2 \prec_d e'_1$. If two edges are compared and their weights are equal, the randoms can be used to decide which edge is chosen. The exact predicate to determine if unshuffled edge (v,w) is to be preferred over edge (v',w') is as follows:

$$\begin{aligned}\sigma((v,w)) > \sigma((v',w')) \vee \{\sigma((v,w)) = \sigma((v',w'))\\ \wedge [\min\{rnd_v, rnd_w\} < \min\{rnd_{v'}, rnd_{w'}\}\\ \vee (\min\{rnd_v, rnd_w\} = \min\{rnd_{v'}, rnd_{w'}\}\\ \wedge \max\{rnd_v, rnd_w\} \leq \max\{rnd_{v'}, rnd_{w'}\})]\}\,.\end{aligned}$$

Note that evaluating this predicate in a secure protocol is significantly more expensive than just testing if $\sigma(\{v,w\}) \geq \sigma(\{v',w'\})$ holds. However, as the nodes are not actually permuted, it again is not necessary to invert a permutation on every edge that is added to the matching as in lines 18-25 of Protocol 6.1.

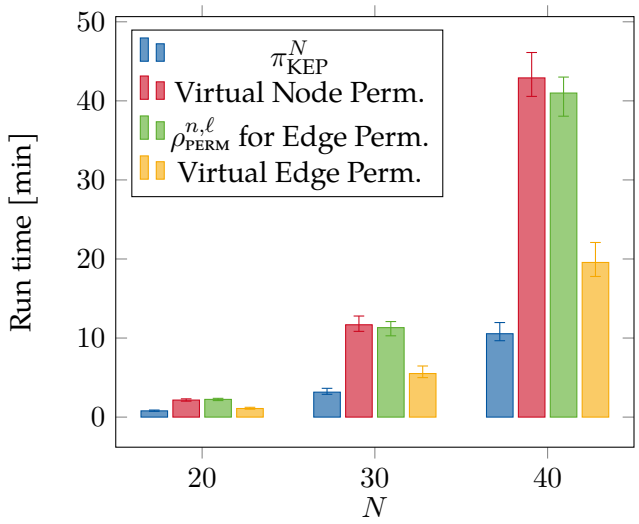
A.3. Evaluation of the Different Approaches

Using the same evaluation setup as in Chapter 7, we give a short comparison of the protocols that emerge from the previously discussed ideas. Again, we simulate 1 Gbit/s ingoing and outgoing bandwidth for each party and a network delay of 5 ms. Among the compared protocols is our original protocol π^N_{KEP} using the graph generation for crossover exchange. To this protocol, we compare a protocol that implements the same functionality but uses virtual node permutation as it was briefly discussed in Section A.2.1. For the new functionality introduced in Section A.1 that performs random edge selection, we benchmark a protocol using virtual edge permutation as described in Section A.2. Finally, we have implemented a protocol that utilizes gate $\rho^{n,\ell}_{\text{PERM}}$ to actually permute all edges to implement the functionality performing random edge selection. For our brief evaluation of the different approaches, we just use $N \in \{20, 30, 40\}$ and 3 iterations each to get an intuition on how the different approaches compare to each other.

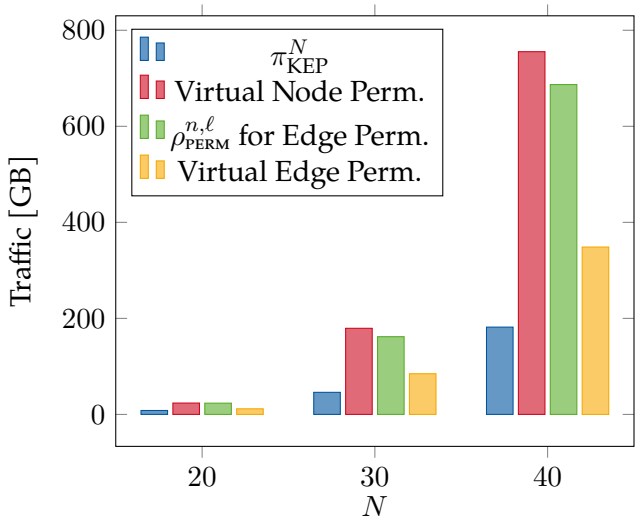
The results of our benchmarks are given in Figure A.1. Using virtual node permutation yields run times and traffic that are multiples of the corresponding values for π^N_{KEP}. This implies that the overhead given by calling $\rho^{n,\ell}_{\text{PERM}}$ and inverting the permutation of an edge in each iteration is significantly smaller than the overhead given by the evaluation of a predicate as in Section A.2.1 that requires a significantly higher number of comparisons. Thus, we argue that not using virtual node permutation for our protocol π^N_{KEP} is the more efficient solution. In contrast to π^N_{KEP} having better performance than its modification using virtual node permutation, virtual edge permutation outperforms our protocol that uses $\rho^{n,\ell}_{\text{PERM}}$ to permute all edges by a factor of approximately 2 for all tested N regarding both run time and traffic. This presumably is caused by the predicate for virtual edge permutation requiring significantly less comparisons than the predicate for virtual node permutation. In addition, the approach using $\rho^{n,\ell}_{\text{PERM}}$ requires substantial modifications of the original protocol that significantly increase its run time.

For 20 parties, the protocol using virtual edge permutation runs in 65.2 seconds compared to 47.4 seconds for π^N_{KEP}. For 30 parties, it requires 5.5 minutes compared to 3.2 minutes and for 40 parties it runs for 19.6 minutes instead of 10.5 minutes. Thus, virtual edge permutation requires 1.9 times the run time of π^N_{KEP} for 40 parties. In addition, also 1.9 times the network traffic is generated for $N = 40$. We have measured that for 60 parties, the run time overhead increases to a factor of 2.1

yielding a run time of 2.2 hours instead of 60.6 minutes. Finally, we conclude that a protocol that performs random edge selection according to Definition A.1 has a significant overhead compared to π^N_{KEP}. As both protocols offer invariance under permutation, π^N_{KEP} appears to generally be the better choice. This being said, specific use cases that require random edge selection are also covered by using virtual edge permutation where the resulting performance at least appears to be of the same order of magnitude as the performance of our protocol π^N_{KEP}.



(a) Run times



(b) Traffic

Figure A.1.: Arithmetic means, minima and maxima for run times and traffic of different MWM approximation protocols using 1 Gbit/s bandwidth and 5 ms delay. Data gathered from 10 iterations for $\pi^{\text{nshffl,par}}_N$ and 3 iterations for remaining protocols.

Appendix B

Quality of the Computed Matchings for Crossover Exchange

Recall that our protocol $\pi_{\text{MWM}}^{N,\vartheta}$ for privacy-preserving MWM approximation does not yield optimal solutions and its approximation ratio of $1/2$ only is a lower bound for the quality of its outputs. To assess the quality of its outputs in practice for a specific use case, we evaluate the resulting matchings for protocol π_{KEP}^{N} where the use case is crossover exchange. As the protocol yields the same outputs as Algorithm 4.3 that it is based on except for negligible probability, it suffices to use a conventional implementation of the algorithm. We use real-world kidney exchange data that originates from the United Network for Organ Sharing (UNOS) and contains the relevant medical inputs used in KEP matching runs between October 2010 and December 2020.[1] Recall that while real data is used, our results are not universal as they specifically correspond to our chosen generation of edge weights as in Definition 5.10. We use a static setting where a single test run consists of randomly sampling a set of N parties and their inputs from all pairs and running the algorithms on the resulting instance.

We split this chapter into two parts. As the quality of solutions may depend on the specific properties of the generated graphs, we briefly discuss the observed properties of these graphs in Section B.1. Then, we evaluate the quality of the matchings obtained by using Algorithm 4.3 on these graphs in Section B.2.

B.1. Properties of the Compatibility Graphs

Regarding the graphs, we measure how many edges they have and how the edge weights are distributed. Figure B.1 depicts how N and M relate. The generated CGs can be described as sparse graphs where the arithmetic means of graph densities for fixed N lie between 0.016 and 0.018. Recall that this result solely results from the selected compatibility criteria that we have discussed in Section 4.3.2.

[1] The data reported here have been supplied by the United Network for Organ Sharing as the contractor for the Organ Procurement and Transplantation Network. The interpretation and reporting of these data are the responsibility of the author(s) and in no way should be seen as an official policy of or interpretation by the OPTN or the U.S. Government.

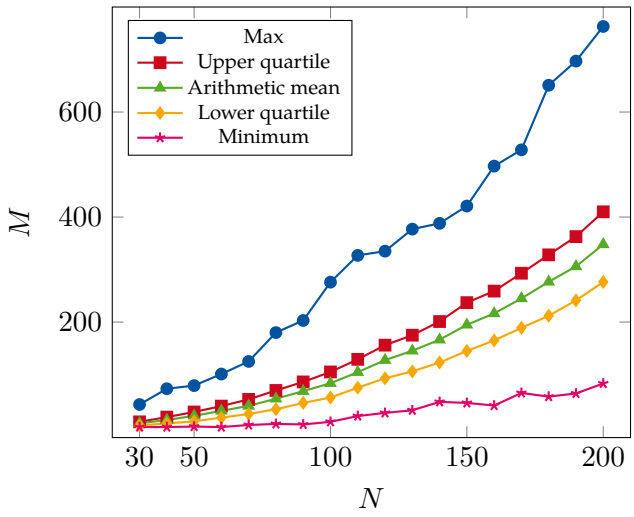


Figure B.1.: Number of generated edges for different N. Data gathered from 1000 CGs for randomly sampled parties for each N.

Furthermore, we give a short intuition on how the edge weights according to Definition 5.10 are distributed. Note that the distribution of weights is of importance as the algorithm's performance regarding its outputs' qualities may depend on it. Figure B.2 features the resulting distribution of edge weights. The distribution is bell curve like for low weights. For higher weights, the corresponding frequency gradually decreases due to a small number of patients that for instance have particularly long waiting times. Yet, it is to be noted that our edge weights just serve as an example and do not necessarily represent prioritization as used by today's kidney exchange programs.

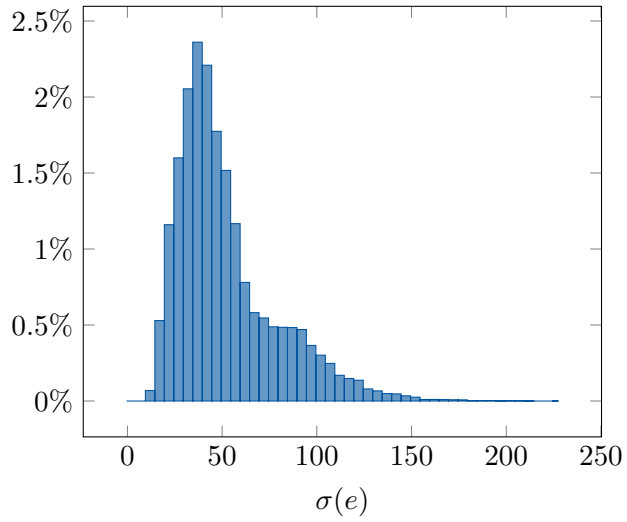


Figure B.2.: Distribution of edge weights. Data gathered from 1000 CGs for randomly sampled parties using $N = 100$.

B.2. Results for the Greedy MWM Algorithm

To assess the quality of the outputs of Algorithm 4.3, we deterministically build CGs from randomly sampled instances as discussed previously. For some CG G with $M > 0$, by the relative performance $r(G)$ of the greedy MWM algorithm on G we denote the weight of the matching computed by it divided by the weight of an optimal solution. Thus, the relative performance the fraction of the optimal solution's weight that is obtained by the greedy MWM algorithm. Recall that it holds that $1/2 \leq r(G) \leq 1$ due to the algorithm's approximation ratio. From our benchmarks, we exclude all generated instances that have no edges as the relative performance on such instances is undefined.

Table B.1 gives a statistical overview of the relative performances of Algorithm 4.3 on instances of different sizes. On average, the algorithm significantly outperforms its relative performance's lower bound of $1/2$. For each used N, 97% of all instances were solved with a relative performance of at least 84.0%, and 75% were solved with a relative performance over 93%. For $N = 100$, more than 30% of instances were solved to optimality. Even on instances with the worst results, the algorithm still achieved at least 75% of the optimal weight for $N \geq 70$.

Table B.1.: Arithmetic mean, standard deviation, minimum, third percentile, first quartile, median, third quartile, 97th percentile, maximum and interquartile range of relative performances and share of instances solved to optimality for Algorithm 4.3. Data gathered from 1000 CGs for randomly sampled parties for each N. Excluded $59, 12, 1$ instances with $M = 0$ for $N = 30, 40, 60$.

N	μ	σ	min	$P_{3\%}$	$P_{25\%}$	$P_{50\%}$	$P_{75\%}$	$P_{97\%}$	max	IQR	opt
30	0.988	0.050	0.526	0.862	1.000	1.000	1.000	1.000	1.000	0.000	0.906
40	0.985	0.044	0.553	0.840	1.000	1.000	1.000	1.000	1.000	0.000	0.829
50	0.979	0.045	0.694	0.845	0.986	1.000	1.000	1.000	1.000	0.014	0.717
60	0.976	0.045	0.695	0.842	0.965	1.000	1.000	1.000	1.000	0.035	0.644
70	0.974	0.040	0.766	0.873	0.956	1.000	1.000	1.000	1.000	0.044	0.549
80	0.974	0.037	0.757	0.879	0.955	0.994	1.000	1.000	1.000	0.045	0.456
90	0.967	0.039	0.779	0.876	0.944	0.981	1.000	1.000	1.000	0.056	0.339
100	0.967	0.037	0.765	0.878	0.946	0.978	1.000	1.000	1.000	0.054	0.309
110	0.966	0.033	0.835	0.893	0.946	0.974	0.999	1.000	1.000	0.053	0.239
120	0.962	0.034	0.820	0.881	0.942	0.969	0.990	1.000	1.000	0.048	0.163
130	0.959	0.034	0.810	0.887	0.939	0.964	0.988	1.000	1.000	0.049	0.128
140	0.959	0.033	0.812	0.888	0.938	0.964	0.987	1.000	1.000	0.048	0.103
150	0.958	0.030	0.805	0.893	0.940	0.961	0.980	1.000	1.000	0.041	0.063
160	0.957	0.029	0.815	0.897	0.939	0.959	0.980	1.000	1.000	0.041	0.036
170	0.955	0.029	0.816	0.893	0.938	0.958	0.976	0.999	1.000	0.039	0.024
180	0.955	0.027	0.857	0.897	0.939	0.958	0.975	0.999	1.000	0.037	0.026
190	0.952	0.028	0.795	0.892	0.936	0.955	0.972	0.998	1.000	0.036	0.017
200	0.952	0.026	0.859	0.900	0.934	0.953	0.970	0.998	1.000	0.037	0.019

Figure B.3 shows the arithmetic means, first quartiles, third percentiles and minima of the deterministic algorithm for different N. The arithmetic means and the first

quartiles slightly decrease for increasing N. In contrast to this behavior, the third percentiles and minima increase. This increase likely is to be explained by outliers produced by a small number of exceptionally poorly chosen edges. With higher N and therefore higher M, the algorithm is able to select more edges so that it becomes more unlikely that a bad selection as in the previous case is done for each added edge.

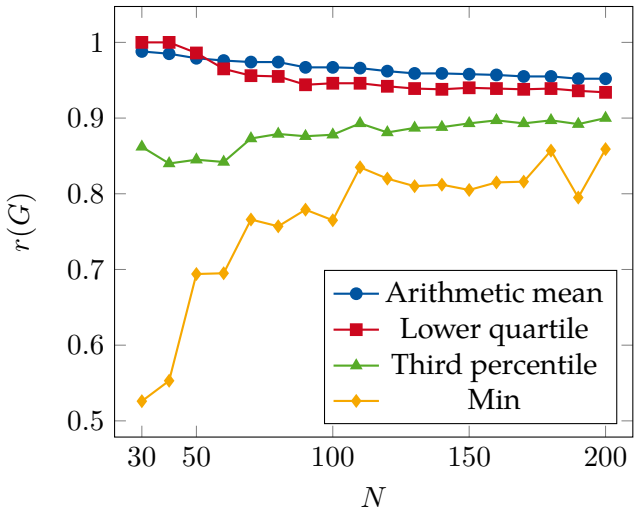


Figure B.3.: Arithmetic mean, first quartile, third percentile and minimum of relative performances for Algorithm 4.3. Data gathered from 1000 CGs for randomly sampled parties for each N. Excluded $59, 12, 1$ instances with $M = 0$ for $N = 30, 40, 60$.

We proceed by analyzing the number of possible different solutions for each CG that can be the output of Algorithm 4.3 as it is probabilistic. Independence under permutation as provided by the algorithm prevents bias in favor of parties P_i with low index i in theory while such advantage may not exist in practice. Recall that the discussed bias only affects iterations of the algorithm where multiple edges of equal weight may be chosen. For a sparse graph with broadly distributed weights, it may be the case that such iterations are rare. Then, the overhead of protocol π^N_{KEP} caused the application of a permutation would rarely yield solutions that are different from the ones given by a simpler deterministic approach. With $N = 50$ and 1000 CGs for randomly sampled parties, 11.7% of the instances had more than one possible result reachable by using different permutations for Algorithm 4.3. The arithmetic mean of possible solutions per instance was 1.13. For $N = 100$, this percentage increases to 24.6% with an arithmetic mean of 1.33 solutions per instance and for $N = 200$ even to 57.1% with an arithmetic mean of 2.50 solutions per instance. This shows that for crossover exchange, the randomization of the protocol indeed is justified as multiple solutions exist with a non-negligible probability.

Comparison to $1/2$-approximation by Drake and Hougardy [DH03a]

In the following, we compare the performance of Algorithm 4.3 on the generated graphs to the $1/2$-approximation by Drake and Hougardy [DH03a] that has time complexity $\mathcal{O}(M)$. The results are given in Figure B.4. While the $1/2$-approximation

by Drake and Hougardy [DH03a] can be implemented to run more efficient than the greedy MWM algorithm and has the same approximation ratio, it delivers significantly worse results for crossover exchange in practice. This demonstrates that even for approximation algorithms achieving the same approximation ratio, an evaluation of both algorithms should be executed for a given use case as their relative performances may significantly differ in practice.

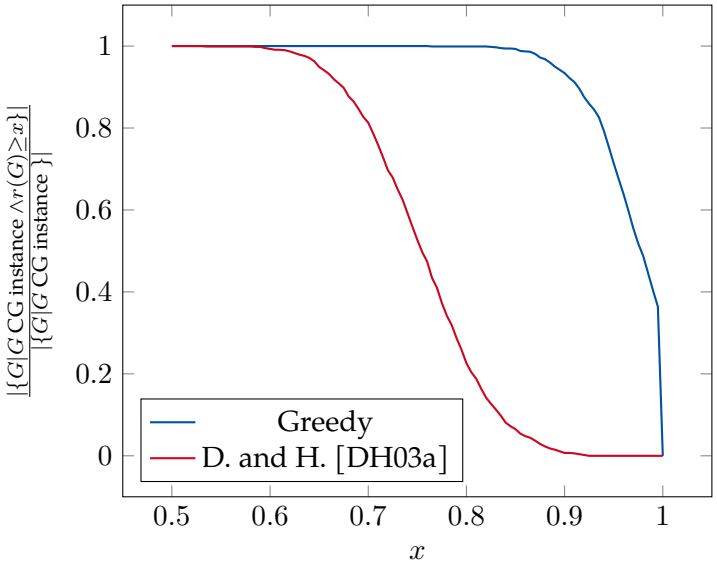


Figure B.4.: Inverse cumulative distribution functions for the given algorithms. Relative performance r matches the corresponding data row's algorithm. Data gathered from 1000 CGs for randomly sampled parties using $N = 100$.

Comparison to Uniform Edge Weights

We end this section by reviewing the relative performance of the greedy MWM algorithm for crossover exchange and $N = 100$ where uniform edge weights are used. Recall that in this case, the algorithm solely tries to maximize the number of crossover exchanges yielding an plain $1/2$-approximation for maximum matching. Note that this comparison is of interest to assess how the distribution of edge weights influences the quality of the results given the greedy MWM approximation.

Figure B.5 shows that in general, the edge weight generation given by ϑ_{KEP} leads to better results than using uniform edge weights. This may indicate that the algorithm profits from more broadly distributed edge weights. Such behavior can be justified by the main argument that Avis [Avi83] uses to prove that the algorithm has an approximation ratio of $1/2$: Each poorly selected edge can block at most two edges of an optimal matching. Due to the algorithm selecting an edge of maximal weight x, the blocked edges together may have a weight of at most $2x$. The broader edge weights are distributed, the more likely it is that the blocked optimal edges together have a weight that is significantly less than $2x$ but still more than x. Thus, less weight is expected to be lost by a suboptimal selection of an edge.

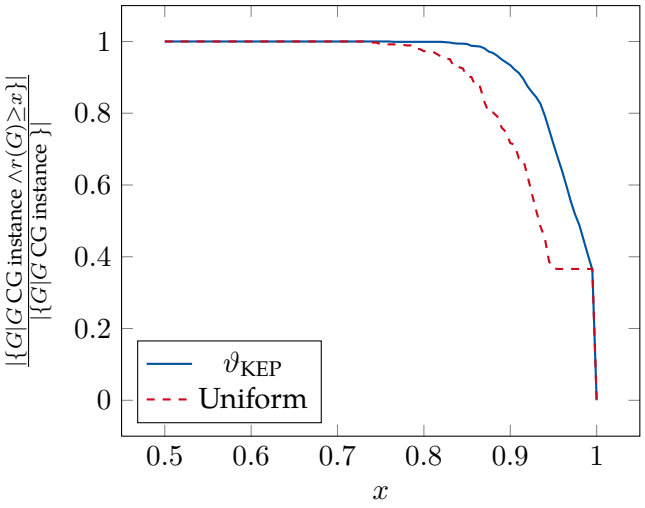


Figure B.5.: Inverse cumulative distribution functions for the greedy MWM algorithm on different edge weight generations. Relative performance r matches the corresponding data row's edge weight generation. Data gathered from 1000 CGs for randomly sampled parties using $N = 100$.

Notation

Symbol	Description
Sets	
$\mathbb{N}$	Natural numbers excluding zero: $\mathbb{N} := \{1, 2, 3, ...\}$
$\mathbb{N}_0$	Natural numbers including zero: $\mathbb{N} := \{0, 1, 2, ...\}$
$\mathbb{Z}$	Integers
$\mathbb{R}$	Real numbers
$\mathbb{F}_p$	Finite field, $\mathbb{Z}$ modulo prime p
$[a, b]$	Interval of integers $[a, b] := \{a, a + 2, a + 2, ..., b - 1, b\}$ where $a, b \in \mathbb{Z}$
$[a, b]_{\mathbb{R}}$	Interval of real numbers $x \in \mathbb{R}, a \leq x \leq b$ where $a, b \in \mathbb{Z}$
S_X	Symmetric group on finite set X, i.e., set of all permutations on X
$\subseteq$	Subset relation s.t. $A \subseteq B$ also if $A = B$
$\prec$	Strict total order relation
Mathematical Functions and Operations	
$\log$	Logarithm function w.r.t. base 2
$\langle A, B \rangle$	Dot product of vectors A, B
$\lfloor x \rfloor$	Greatest $n \in \mathbb{Z}$ such that $n \leq x$
$\lceil x \rceil$	Smallest $n \in \mathbb{Z}$ such that $n \geq x$
Probability Theory	
$\Pr(E)$	Probability of event E
$\Delta(A, B)$	Statistical distance between probability distributions A and B
Graphs	
G	Graph $G = (V, E)$ or $G = (V, E, \sigma)$
$\hat{G}$	Secret graph $\hat{G} = (V, \hat{E})$ or $\hat{G} = (V, \hat{E}, \hat{\sigma})$
V	Nodes of a graph: $V := [1, N]$
N	Number of nodes: $N := \|V\|$, coincides with number of parties
E	Undirected (real) edges of a graph
$\hat{E}$	Potential edges of a graph
M	Number of real edges: $M := \|E\|$
$\hat{M}$	Number of potential edges: $\hat{M} := \|\hat{E}\| = \binom{N}{2}$
σ	Edge weight function: $\sigma : E \to \mathbb{N}$
$\hat{\sigma}$	Secret edge weight function: $\hat{\sigma} : \hat{E} \to \mathbb{N}_0$
(v, w)	Edge $\{v, w\}$ such that $v < w$
$\mathfrak{M}$	Matching on a graph
$\prec_d$	Deterministic ordering of edges

SMPC	
N	Number of parties, coincides with number of nodes
P_i	i'th party
t	Threshold for secret sharing
$[x]$	Secret sharing of $x \in \mathbb{F}_p$
$\mathcal{F}$	Ideal functionality
π	Protocol
$\mathcal{G}$	Gate functionality
ρ	Gate
κ	Security parameter
I	Indexes of corrupted parties
$\text{VIEW}_i^{\pi}(\hat{x}, \kappa)$	View of party P_i on execution of π on input $\hat{x}$ using security parameter κ
$\text{OUTPUT}^{\pi}(\hat{x}, \kappa)$	Outputs of all parties after execution of π on input $\hat{x}$ using security parameter κ
$\hat{x}$	Inputs of all parties, $\hat{x} := (x_1, ..., x_N)$
$\mathcal{S}$	Simulator
ϑ	Mapping from inputs to a weighted graph
L	Number of inputs per party
b	Number of bits used to define secret shared integers
Miscellaneous	
$\{0,1\}^*$	Set of all binary strings
τ, φ	Permutations

Acronyms

CG *compatibility graph*

HLA *Human Leukocyte Antigen*

KEP *kidney exchange problem*

MWM *maximum weight matching*

RAM *random-access machine*

SMPC *secure multi-party computation*

Lists of Figures, Tables, Algorithms, Gates and Protocols

List of Figures

1.1 Crossover exchange and kidney exchange in a cycle length 3. 2

4.1 CG as used by $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$ and matching obtained by evaluating $\mathcal{F}_{\text{MWM}}^{N,\vartheta}$. . 28

5.1 Bitonic sorting network [Bat68] according to the implementation by Christopher [Chr] for $n = 8$ elements. 41
5.2 Structuring of edge weight comparisons for $N = 4$ (based on [CH10]). 45

7.1 Virtual network setup that is used for the evaluation. 64
7.2 Arithmetic means, minima and maxima for run times and traffic of protocol π_{KEP}^{N} using 1 Gbit/s bandwidth and 5 ms delay. 66
7.3 Arithmetic means, minima and maxima for run times and traffic of π_{KEP}^{N} using 1 Gbit/s bandwidth, 5 ms delay and up to 80 parties. . . . 67
7.4 Arithmetic means, minima and maxima for run times of the input permutation using gate $\rho_{\text{PERM}}^{n,\ell}$, the edge generation using gate $\rho_{\text{EDGE_GEN}}^{\vartheta_{\text{KEP}}}$ and the matching phase, and their shares of the overall run time using 1 Gbit/s bandwidth and 5 ms delay. 69
7.5 Arithmetic means, minima and maxima for run times and traffic of variants of protocol π_{KEP}^{N} using 1 Gbit/s bandwidth and 5 ms where one variant executes a linear search to find edges and one variant utilizes the proposed parallelized search. 70
7.6 Arithmetic means, minima and maxima for run times of π_{KEP}^{N} using different network settings. 71

A.1 Arithmetic means, minima and maxima for run times and traffic of different MWM approximation protocols using 1 Gbit/s bandwidth and 5 ms delay. 80

B.1 Number of generated edges for different N. 82

B.2 Distribution of edge weights. 82
B.3 Arithmetic mean, first quartile, third percentile and minimum of relative performances for Algorithm 4.3. 84
B.4 Inverse cumulative distribution functions for the given algorithms. . 85
B.5 Inverse cumulative distribution functions for the greedy MWM algorithm on different edge weight generations. 86

List of Tables

4.1 ABO-Compatibility between donor and patient according to blood type [Dea05]. 29

5.1 Complexities of the elementary gates regarding the required rounds and invocations. 35

7.1 Arithmetic means of run times and global traffic for protocol π^N_{KEP} using 1 Gbit/s bandwidth and 5 ms delay. 66
7.2 Arithmetic means of run times and global traffic for protocol π^N_{KEP} using 1 Gbit/s bandwidth and 5 ms delay. 67

B.1 Statistical analysis of the relative performance for Algorithm 4.3. . . . 83

List of Algorithms

4.1 Greedy MWM Algorithm [Pre99] 24
4.2 Strictly Deterministic Greedy MWM Algorithm (based on [Pre99]) . . 24
4.3 Greedy MWM Algorithm Using Node Permutation (based on [Pre99]) 26
6.1 Modified Greedy MWM Algorithm Using Node Permutation 52
6.2 Edge Index to Node Mapping 55
A.1 Greedy MWM With Edge Permutation (based on [Pre99]) 76

List of Gates

5.1 ρ_{RND} 33
5.2 $\rho_{\text{IF_ELSE}}$ 33
5.3 ρ_{SWAP} 34
5.4 ρ_{ABS} 34
5.5 $\rho^{\vartheta_{\text{KEP}}}_{\text{EDGE_GEN}}$ 39
5.6 $\rho^{n,\ell}_{\text{PERM}}$ 43
5.7 $\rho^{n}_{\text{SEARCH_MAX}}$ 46

List of Protocols

6.1 $\pi^{N,\vartheta}_{\text{MWM}}$: Protocol for Maximum Weight Matching Approximation 54

Bibliography

[ABS07] David J. Abraham, Avrim Blum, and Tuomas Sandholm. "Clearing algorithms for barter exchange markets: enabling nationwide kidney exchanges". In: *Proceedings of the 8th ACM conference on Electronic commerce*. EC '07. New York, NY, USA: Association for Computing Machinery, 2007, pp. 295–304. DOI: 10.1145/1250910.1250954.

[AC17] Balamurugan Anandan and Chris Clifton. "Secure minimum weighted bipartite matching". In: *2017 IEEE Conference on Dependable and Secure Computing*. 2017 IEEE Conference on Dependable and Secure Computing. 2017, pp. 60–67. DOI: 10.1109/DESEC.2017.8073798.

[AL17] Gilad Asharov and Yehuda Lindell. "A Full Proof of the BGW Protocol for Perfectly Secure Multiparty Computation". In: *Journal of Cryptology* 30.1 (2017), pp. 58–151. DOI: 10.1007/s00145-015-9214-4.

[Aly+13] Abdelrahaman Aly, Edouard Cuvelier, Sophie Mawet, Olivier Pereira, and Mathieu Van Vyve. "Securely Solving Simple Combinatorial Graph Problems". In: *Financial Cryptography and Data Security*. Ed. by Ahmad-Reza Sadeghi. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013, pp. 239–257. DOI: 10.1007/978-3-642-39884-1_21.

[Avi83] David Avis. "A survey of heuristics for the weighted matching problem". In: *Networks* 13.4 (1983), pp. 475–493. DOI: 10.1002/net.3230130404.

[Axe+18] David A. Axelrod, Mark A. Schnitzler, Huiling Xiao, William Irish, Elizabeth Tuttle-Newhall, Su-Hsin Chang, Bertram L. Kasiske, Tarek Alhamad, and Krista L. Lentine. "An economic assessment of contemporary kidney transplant practice". In: *American Journal of Transplantation* 18.5 (2018), pp. 1168–1176. DOI: 10.1111/ajt.14702.

[Bat68] K. E. Batcher. "Sorting networks and their applications". In: *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. AFIPS '68 (Spring). New York, NY, USA: Association for Computing Machinery, 1968, pp. 307–314. DOI: 10.1145/1468075.1468121.

[Bir+19] Péter Biró, Bernadette Haase-Kromwijk, Tommy Andersson, Eyjólfur Ingi Ásgeirsson, Tatiana Baltesová, Ioannis Boletis, Catarina Bolotinha, Gregor Bond, Georg Böhmig, Lisa Burnapp, Katarína Cechlárová, Paola Di Ciaccio, Jiri Fronek, Karine Hadaya, Aline Hemke, Christian Jacquelinet, Rachel Johnson, Rafal Kieszek, Dirk R. Kuypers, Ruthanne Leishman, Marie-Alice Macher, David Manlove, Georgia

Menoudakou, Mikko Salonen, Bart Smeulders, Vito Sparacino, Frits C. R. Spieksma, María Oliva Valentín, Nic Wilson, Joris van der Klundert, and on behalf of the ENCKEP COST Action. "Building Kidney Exchange Programmes in Europe—An Overview of Exchange Practice and Activities". In: *Transplantation* 103.7 (2019), pp. 1514–1522. DOI: 10.1097/TP.0000000000002432.

[Bir+21] Péter Biró, Joris van de Klundert, David Manlove, William Pettersson, Tommy Andersson, Lisa Burnapp, Pavel Chromy, Pablo Delgado, Piotr Dworczak, Bernadette Haase, Aline Hemke, Rachel Johnson, Xenia Klimentova, Dirk Kuypers, Alessandro Nanni Costa, Bart Smeulders, Frits Spieksma, María O. Valentín, and Ana Viana. "Modelling and optimisation in European Kidney Exchange Programmes". In: *European Journal of Operational Research* 291.2 (2021), pp. 447–456. DOI: 10.1016/j.ejor.2019.09.006.

[Bof+17] Miquel Bofill, Marcos Calderón, Francesc Castro, Esteve Del Acebo, Pablo Delgado, Marc Garcia, Marta García, Marc Roig, María O. Valentín, and Mateu Villaret. "The Spanish Kidney Exchange Model: Study of Computation-Based Alternatives to the Current Procedure". In: *Artificial Intelligence in Medicine*. Ed. by Annette ten Teije, Christian Popow, John H. Holmes, and Lucia Sacchi. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 272–277. DOI: 10.1007/978-3-319-59758-4_31.

[BOGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness theorems for non-cryptographic fault-tolerant distributed computation". In: *Proceedings of the twentieth annual ACM symposium on Theory of computing*. STOC '88. New York, NY, USA: Association for Computing Machinery, 1988, pp. 1–10. DOI: 10.1145/62212.62213.

[Bre+20] Malte Breuer, Ulrike Meyer, Susanne Wetzel, and Anja Mühlfeld. "A Privacy-Preserving Protocol for the Kidney Exchange Problem". In: *Proceedings of the 19th Workshop on Privacy in the Electronic Society*. WPES'20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 151–162. DOI: 10.1145/3411497.3420213.

[Bro+] Jens Brockmann, Rigmar Osterkamp, Stefan Reuter, and Hartmut Schmidt. *„Paired-kidney exchange" - Die Nieren-Überkreuzspende in Deutschland: die Hoffnung stirbt zuletzt*. Diatra-Verlag. URL: https://diatra-verlag.de/paired-kidney-exchange-die-nieren-ueberkreuzspende-in-deutschland-die-hoffnung-stirbt-zuletzt/ (accessed 10/30/2021).

[BS15] Marina Blanton and Siddharth Saraph. "Oblivious Maximum Bipartite Matching Size Algorithm with Applications to Secure Fingerprint Identification". In: *Computer Security – ESORICS 2015*. Ed. by Günther Pernul, Peter Y A Ryan, and Edgar Weippl. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 384–406. DOI: 10.1007/978-3-319-24174-6_20.

[BSA13] Marina Blanton, Aaron Steele, and Mehrdad Alisagari. "Data-oblivious graph algorithms for secure computation and outsourcing". In: *Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security*. ASIA CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, pp. 207–218. DOI: 10.1145/2484313.2484341.

[Can00] Ran Canetti. "Security and Composition of Multiparty Cryptographic Protocols". In: *Journal of Cryptology* 13.1 (2000), pp. 143–202. DOI: 10.1007/s001459910006.

[CH10] Octavian Catrina and Sebastiaan de Hoogh. "Improved Primitives for Secure Multiparty Integer Computation". In: *Security and Cryptography for Networks*. Ed. by Juan A. Garay and Roberto De Prisco. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2010, pp. 182–199. DOI: 10.1007/978-3-642-15317-4_13.

[Cho07] Sung Yoon Choo. "The HLA System: Genetics, Immunology, Clinical Testing, and Clinical Implications". In: *Yonsei Medical Journal* 48.1 (2007), pp. 11–23. DOI: 10.3349/ymj.2007.48.1.11.

[Chr] Thomas W. Christopher. *Bitonic Sort*. Tools of Computing. URL: https://www.tools-of-computing.com/tc/CS/Sorts/bitonic_sort.htm (accessed 08/16/2021).

[Con+13] Miguel Constantino, Xenia Klimentova, Ana Viana, and Abdur Rais. "New insights on integer-programming models for the kidney exchange problem". In: *European Journal of Operational Research* 231 (2013), pp. 57–68. DOI: 10.1016/j.ejor.2013.05.025.

[CR99] William Cook and André Rohe. "Computing Minimum-Weight Perfect Matchings". In: *INFORMS Journal on Computing* 11.2 (1999), pp. 138–148. ISSN: 1526-5528.

[CW90] Don Coppersmith and Shmuel Winograd. "Matrix multiplication via arithmetic progressions". In: *Journal of Symbolic Computation*. Computational algebraic complexity editorial 9.3 (1990), pp. 251–280. DOI: 10.1016/S0747-7171(08)80013-2.

[Dam+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. "Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation". In: *Theory of Cryptography*. Ed. by Shai Halevi and Tal Rabin. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2006, pp. 285–304. DOI: 10.1007/11681878_15.

[Dea05] Laura Dean. *Blood Groups and Red Cell Antigens*. National Center for Biotechnology Information (US), 2005.

[DEs16] Jack Doerner, David Evans, and abhi shelat. "Secure Stable Matching at Scale". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 1602–1613. DOI: 10.1145/2976749.2978373.

[DH03a] Doratha E Drake and Stefan Hougardy. "A simple approximation algorithm for the weighted matching problem". In: *Information Processing Letters* 85.4 (2003), pp. 211–213. ISSN: 0020-0190. DOI: 10.1016/S0020-0190(02)00393-9.

[DH03b] Doratha E. Drake and Stefan Hougardy. "Improved Linear Time Approximation Algorithms for Weighted Matchings". In: *Approximation, Randomization, and Combinatorial Optimization.. Algorithms and Techniques*. Ed. by Sanjeev Arora, Klaus Jansen, José D. P. Rolim, and Amit Sahai. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 14–23. DOI: 10.1007/978-3-540-45198-3_2.

[DH05] Doratha E. Drake and Stefan Hougardy. "A linear-time approximation algorithm for weighted matchings in graphs". In: *ACM Transactions on Algorithms* 1.1 (2005), pp. 107–122. DOI: 10.1145/1077464.1077472.

[Die17] Reinhard Diestel. *Graph Theory*. 5th ed. Graduate Texts in Mathematics. Berlin Heidelberg: Springer-Verlag, 2017. DOI: 10.1007/978-3-662-53622-3.

[DP14] Ran Duan and Seth Pettie. "Linear-Time Approximation for Maximum Weight Matching". In: *Journal of the ACM* 61.1 (2014), 1:1–1:23. DOI: 10.1145/2529989.

[Edm65a] J. Edmonds. "Maximum matching and a polyhedron with 0,1-vertices". In: *Journal of Research of the National Bureau of Standards Section B Mathematics and Mathematical Physics* (1965), pp. 125–130. DOI: 10.6028/jres.069B.013.

[Edm65b] Jack Edmonds. "Paths, Trees, and Flowers". In: *Canadian Journal of Mathematics* 17 (1965). Publisher: Cambridge University Press, pp. 449–467. DOI: 10.4153/CJM-1965-045-4.

[EK72] Jack Edmonds and Richard M. Karp. "Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems". In: *Journal of the ACM* 19.2 (1972), pp. 248–264. DOI: 10.1145/321694.321699.

[EKR19] David Evans, Vladimir Kolesnikov, and Mike Rosulek. *A Pragmatic Introduction to Secure Multi-Party Computation*. S.l.: Now Publishers Inc, 2019. 194 pp. ISBN: 978-1-68083-508-3.

[Eur21] *Annual Report 2020*. Eurotransplant, 2021. URL: https://www.eurotransplant.org/wp-content/uploads/2021/09/ETP_AR2020_opmaak_LR2.pdf (accessed 10/30/2021).

[FO08] Keith Frikken and Lukasz Opyrchal. "PBS: Private Bartering Systems". In: *Financial Cryptography and Data Security*. Ed. by Gene Tsudik. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 113–127. DOI: 10.1007/978-3-540-85230-8_9.

[Gab90] Harold N. Gabow. "Data structures for weighted matching and nearest common ancestors with linking". In: *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*. SODA '90. USA: Society for Industrial and Applied Mathematics, 1990, pp. 434–443. ISBN: 978-0-89871-251-3.

[GO96] Oded Goldreich and Rafail Ostrovsky. "Software protection and simulation on oblivious RAMs". In: *Journal of the ACM* 43.3 (1996), pp. 431–473. ISSN: 0004-5411. DOI: 10.1145/233551.233553.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. 2004. ISBN: 978-0-521-83084-3.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. "Simplified VSS and fast-track multiparty computations with applications to threshold cryptography". In: *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*. PODC '98. New York, NY, USA: Association for Computing Machinery, 1998, pp. 101–111. DOI: 10.1145/277697.277716.

[GT91] Harold N. Gabow and Robert E. Tarjan. "Faster scaling algorithms for general graph matching problems". In: *Journal of the ACM* 38.4 (1991), pp. 815–853. DOI: 10.1145/115234.115366.

[Hal+11] Maria Haller, Georg Gutjahr, Reinhard Kramar, Franz Harnoncourt, and Rainer Oberbauer. "Cost-effectiveness analysis of renal replacement therapy in Austria". In: *Nephrology Dialysis Transplantation* 26.9 (2011). Publisher: Oxford Academic, pp. 2988–2995. DOI: 10.1093/ndt/gfq780.

[Har+17] A. Hart, J. M. Smith, M. A. Skeans, S. K. Gustafson, D. E. Stewart, W. S. Cherikh, J. L. Wainright, A. Kucheryavaya, M. Woodbury, J. J. Snyder, B. L. Kasiske, and A. K. Israni. "OPTN/SRTR 2015 Annual Data Report: Kidney". In: *American Journal of Transplantation* 17 (S1 2017), pp. 21–116. DOI: 10.1111/ajt.14124.

[HH10] Sven Hanke and Stefan Hougardy. *New approximation algorithms for the weighted matching problem*. Report No : 101010. Research Institute for Discrete Mathematics, University Of Bonn, 2010. URL: http://www.or.uni-bonn.de/%7Ehougardy/paper.html (accessed 04/21/2021).

[HK12] Chien-Chung Huang and Telikepalli Kavitha. "Efficient algorithms for maximum weight matchings in general graphs with small edge weights". In: *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete algorithms*. SODA '12. USA: Society for Industrial and Applied Mathematics, 2012, pp. 1400–1412.

[Hou09] Stefan Hougardy. "Linear Time Approximation Algorithms for Degree Constrained Subgraph Problems". In: *Research Trends in Combinatorial Optimization: Bonn 2008*. Ed. by William Cook, László Lovász, and Jens Vygen. Berlin, Heidelberg: Springer, 2009, pp. 185–200. DOI: 10.1007/978-3-540-76796-1_9.

[Kan+18] Sampath Kannan, Jamie Morgenstern, Ryan Rogers, and Aaron Roth. "Private Pareto Optimal Exchange". In: *ACM Transactions on Economics and Computation* 6.3 (2018), 12:1–12:25. DOI: 10.1145/3105445.

[Kel20] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. CCS '20. New York, NY, USA: Association for Computing Machinery, 2020, pp. 1575–1590. DOI: 10.1145/3372297.3417872.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography*. 2nd edition. Boca Raton: Taylor & Francis Inc, 2014. 583 pp. ISBN: 978-1-4665-7026-9.

[Kol06] Vladimir Kolesnikov. "Secure two-party computation and communication". PhD thesis. University of Toronto, 2006.

[Kol09] Vladimir Kolmogorov. "Blossom V: a new implementation of a minimum cost perfect matching algorithm". In: *Mathematical Programming Computation* 1.1 (2009), pp. 43–67. DOI: 10.1007/s12532-009-0002-8.

[Kuh55] H. W. Kuhn. "The Hungarian method for the assignment problem". In: *Naval Research Logistics Quarterly* 2.1 (1955), pp. 83–97. DOI: https://doi.org/10.1002/nav.3800020109.

[Lan] Hans Werner Lang. *Bitonic sorting network for n not a power of 2*. URL: https://www.inf.hs-flensburg.de/lang/algorithmen/sortieren/bitonic/oddn.htm (accessed 08/16/2021).

[Lin17] Yehuda Lindell. "How to Simulate It – A Tutorial on the Simulation Proof Technique". In: *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich*. Ed. by Yehuda Lindell. Information Security and Cryptography. Cham: Springer International Publishing, 2017, pp. 277–346. DOI: 10.1007/978-3-319-57048-8_6.

[Lin20] Yehuda Lindell. "Secure multiparty computation". In: *Communications of the ACM* 64.1 (2020), pp. 86–96. ISSN: 0001-0782. DOI: 10.1145/3387108.

[MSZ15] Mahnush Movahedi, Jared Saia, and Mahdi Zamani. "Secure Multiparty Shuffling". In: *Structural Information and Communication Complexity*. Ed. by Christian Scheideler. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 459–473. DOI: 10.1007/978-3-319-25258-2_32.

[NO07] Takashi Nishide and Kazuo Ohta. "Multiparty Computation for Interval, Equality, and Comparison Without Bit-Decomposition Protocol". In: *Public Key Cryptography – PKC 2007*. Ed. by Tatsuaki Okamoto and Xiaoyun Wang. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2007, pp. 343–360. DOI: 10.1007/978-3-540-71677-8_23.

[Pet12] S. Pettie. "A simple reduction from maximum weight matching to maximum cardinality matching". In: *Information Processing Letters* 112.23 (2012), pp. 893–898. DOI: 10.1016/j.ipl.2012.08.010.

[Pre99] Robert Preis. "Linear Time 1/2-Approximation Algorithm for Maximum Weighted Matching in General Graphs". In: *STACS 99*. Ed. by Christoph Meinel and Sophie Tison. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 1999, pp. 259–269. DOI: 10.1007/3-540-49116-3_24.

[PS04] Seth Pettie and Peter Sanders. "A simpler linear time 2/3-ε approximation for maximum weight matching". In: *Information Processing Letters* 91.6 (2004), pp. 271–276. DOI: 10.1016/j.ipl.2004.05.007.

[Rei09] Tord Reistad. "Multiparty Comparison - An Improved Multiparty Protocol for Comparison of Secret-shared Values." In: SECRYPT 2009 - International Conference on Security and Cryptography, Proceedings. 2009, pp. 325–330.

[Rei20] Susanne Reitmaier. "Neue Initiative für Crossover-Nierenspenden in Deutschland". In: *DIATRA* 2 (2020). URL: https://diatra-verlag.de/neue-initiative-fuer-crossover-nierenspenden-in-deutschland/ (accessed 10/30/2021).

[RSn07] Alvin E. Roth, Tayfun Sönmez, and M. Utku Ünver. "Efficient Kidney Exchange: Coincidence of Wants in Markets with Compatibility-Based Preferences". In: *American Economic Review* 97.3 (2007), pp. 828–851. DOI: 10.1257/aer.97.3.828.

[Sha79] Adi Shamir. "How to share a secret". In: *Communications of the ACM* 22.11 (1979), pp. 612–613. DOI: 10.1145/359168.359176.

[Sho09] Victor Shoup. *A Computational Introduction to Number Theory and Algebra*. 2nd ed. Cambridge ; New York: Cambridge University Press, 2009. 580 pp. ISBN: 978-0-521-51644-0.

[Wü+17] Stefan Wüller, Michael Vu, Ulrike Meyer, and Susanne Wetzel. "Using Secure Graph Algorithms for the Privacy-Preserving Identification of Optimal Bartering Opportunities". In: *Proceedings of the 2017 on Workshop on Privacy in the Electronic Society*. WPES '17. New York, NY, USA: Association for Computing Machinery, 2017, pp. 123–132. DOI: 10.1145/3139550.3139557.

[Wü18] Stefan Wüller. "Privacy-preserving electronic bartering". Number: RWTH-2018-222969. PhD thesis. Dissertation, RWTH Aachen University, 2018, 2018. DOI: 10.18154/RWTH-2018-222969.

[Wil09] Val Willingham. *Massive transplant effort pairs 13 kidneys to 13 patients - CNN.com*. cnn.com. 2009. URL: http://www.cnn.com/2009/HEALTH/12/14/kidney.transplant/index.html (accessed 05/05/2021).

[WMW17a] Stefan Wüller, Ulrike Meyer, and Susanne Wetzel. "Privacy-Preserving Multi-Party Bartering Secure Against Active Adversaries". In: *2017 15th Annual Conference on Privacy, Security and Trust (PST)*. 2017 15th Annual Conference on Privacy, Security and Trust (PST). 2017, pp. 205–20509. DOI: 10.1109/PST.2017.00032.

[WMW17b] Stefan Wüller, Ulrike Meyer, and Susanne Wetzel. "Towards Privacy-Preserving Multi-party Bartering". In: *Financial Cryptography and Data Security*. Ed. by Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 19–34. DOI: 10.1007/978-3-319-70278-0_2.

[Yao82] Andrew C. Yao. "Protocols for secure computations". In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. 23rd Annual Symposium on Foundations of Computer Science (sfcs 1982). 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.