TECHNISCHE
UNIVERSITÄT
DARMSTADT

Bachelor Thesis

Efficient and Practical Privacy-Preserving Kidney Exchange Protocol

Timm Birka

March 31, 2022

ENCRYPTO
CRYPTOGRAPHY AND
PRIVACY ENGINEERING

Cryptography and Privacy Engineering Group
Department of Computer Science
Technische Universität Darmstadt

Supervisors: M.Sc. Tobias Kussel
M.Sc. Helen Möllering
Prof. Dr. Kay Hamacher
Prof. Dr.-Ing. Thomas Schneider

Erklärung zur Abschlussarbeit gemäß §23 Abs. 7 APB der TU Darmstadt

Hiermit versichere ich, Timm Birka, die vorliegende Bachelor Thesis ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§38 Abs.2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei der abgegebenen Thesis stimmen die schriftliche und die zur Archivierung eingereichte elektronische Fassung überein.

Thesis Statement pursuant to §23 paragraph 7 of APB TU Darmstadt

I herewith formally declare that I, Timm Birka, have written the submitted Bachelor Thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper. I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content. This thesis has not been handed in or published before in the same or similar form.

I am aware, that in case of an attempt at deception based on plagiarism (§38 Abs. 2 APB), the thesis would be graded with 5,0 and counted as one failed examination attempt. The thesis may only be repeated once.

In the submitted thesis the written copies and the electronic version for archiving are identical in content.

Darmstadt, March 31, 2022

Timm Birka

Abstract

In Kidney Exchange Programmes donor-recipient pairs exchange their incompatible donors such that each pair that has donated a kidney receives a compatible kidney. In general, these exchanges are performed in a cyclic fashion. The resulting cycles are called exchange cycles. The kidney exchange problem is to find the most robust set of exchange cycles with respect to transplantation success, so that as many recipients as possible receive a compatible kidney.

The evaluation of medical compatibility between donors and recipients requires highly sensitive health information. Therefore, privacy-preserving solutions are required to protect the sensitive health information of the patients and avoid accidentally leaking information to unauthorised personnel.

In our work, we design, implement, and benchmark an efficient and privacy-preserving solution to the kidney exchange problem. We outperform the run-time of current state-of-the-art works of Breuer et al. [BMWM20; BMW22] by a factor of approximately 30 000 for 9 pairs with cycle length $L = 3$ and by a factor of approximately 400 for 40 pairs with cycle length $L = 2$. We include additional biomedical factors to increase the likelihood for transplantation success.

Contents

1 Introduction **1**
- 1.1 Contributions . . . 2
- 1.2 Outline . . . 3

2 Preliminaries **4**
- 2.1 Medical Background . . . 4
  - 2.1.1 Human Leukocyte Antigens . . . 4
  - 2.1.2 ABO System . . . 7
  - 2.1.3 Age . . . 7
  - 2.1.4 Sex . . . 8
  - 2.1.5 Weight . . . 8
- 2.2 Cryptographic Background . . . 9
  - 2.2.1 Secure Multi-Party Computation . . . 9
  - 2.2.2 ABY Framework . . . 10
  - 2.2.3 Secure Outsourcing . . . 12
- 2.3 Graph Background . . . 14
  - 2.3.1 Graph Theory . . . 14
  - 2.3.2 Kidney Exchange Problem . . . 16

3 Related Work **17**
- 3.1 Non Privacy-Preserving Solutions . . . 17
  - 3.1.1 Robust Models for the Kidney Exchange Problem . . . 17
  - 3.1.2 Medical Analysis for Evaluating Compatibility in Kidney Exchange . . . 18
- 3.2 Privacy-Preserving Solutions . . . 19
  - 3.2.1 A Privacy-Preserving Protocol for the Kidney Exchange Problem . . . 19
  - 3.2.2 Privacy-Preserving Maximum Matching on General Graphs and its Application to Enable Privacy-Preserving Kidney Exchange . . . 20
  - 3.2.3 Secure Graph Analysis at Scale . . . 21
  - 3.2.4 Privacy-Preserving Linear Programming . . . 22

4 Design and Implementation **23**
- 4.1 Problem Statement . . . 24
- 4.2 Notation . . . 25
- 4.3 Compatibility Matching . . . 25
  - 4.3.1 Overview . . . 26
  - 4.3.2 HLA Cross-Match . . . 28

4.3.3 HLA Matching . . . 29
4.3.4 Blood Type Matching . . . 31
4.3.5 Age Matching . . . 32
4.3.6 Sex Matching . . . 33
4.3.7 Weight Matching . . . 34
4.3.8 Complexity Assessment . . . 35
4.4 Cycle Computation . . . 36
4.4.1 Overview . . . 36
4.4.2 Protocols . . . 37
4.4.3 Complexity Assessment . . . 38
4.5 Cycle Evaluation . . . 39
4.5.1 Overview . . . 39
4.5.2 Protocols . . . 40
4.5.3 Complexity Assessment . . . 45
4.6 Solution Evaluation . . . 46
4.6.1 Overview . . . 46
4.6.2 Protocols . . . 48
4.6.3 Complexity Assessment . . . 50
4.7 Overall Complexity Assessment . . . 51

5 Evaluation **52**
5.1 Security Discussion . . . 52
5.2 Performance Benchmarks . . . 53
5.3 Comparison to State-of-the-Art . . . 59

6 Conclusion **64**
6.1 Future Work . . . 64

List of Figures **66**

List of Tables **67**

List of Abbreviations **69**

Bibliography **70**

A Appendix **76**
A.1 Full Benchmarks . . . 76
A.2 Fitting Models . . . 88

1 Introduction

Kidneys remove wastes and toxins from the body, so if none of a person's kidneys are functioning, they require treatment. There are two ways to treat malfunctioning kidneys. The first is dialysis in which blood is cleansed from toxins, i.e., dialysis takes over the role of the kidney. However, dialysis is very time consuming and has several risks, e.g., low blood pressure, weight gain, sudden cardiac death [Bet22]. Second, it is possible to receive kidneys from deceased or living humans. In 2021, there were in total 9874 recipients on the active waiting list for kidneys in Eurotransplant countries [Eur22a]. In contrast, there were only 1573 deceased kidney donors [Eur22b]. In Europe, Eurotransplant [Eur21a] is an organisation which is specialised in organ transplantation such as kidneys and serves as mediator between donors and recipients. For kidney transplantation, Eurotransplant focuses on deceased donor donations [Eur21c]. This approach is not optimal as it comes with long waiting times for the recipients due to the scarcity of deceased donor organs and possibly degraded organ quality. These two problems significantly increase the risk of the recipients to pass away before receiving a compatible kidney.

Humans possess two kidneys and are able to live with only one functioning kidney, thus, living humans with functioning kidneys are able to donate one of their kidneys. This allows recipients who did not receive a kidney through a deceased donor waiting list to receive a kidney from a living donor. The first living donor transplantation was conducted in 1954 in the USA [Eur21b]. Since then living kidney transplants have become a very important part of today's kidney transplantation to significantly reduce waiting times for the recipients on the active waiting list. However, even when a recipient manages to find a living donor, it is not guaranteed that the donor and recipient are medically compatible. For these cases, the so called living donor exchange system was implemented in 1991 [SCM+14]. In living donor exchanges, recipients who have found a willing but incompatible donor, i.e., pairs, are able to exchange donors with other pairs in need of a kidney. These pairs exchange their incompatible donors such that ideally each recipient receives a compatible kidney.

In our work, we consider a scenario in which several donor-recipient pairs exchange their donors in a cyclic fashion, so that each donating pair receives a compatible kidney. These cycles are called *exchange cycles* [BKM+21]. Figure 1.1 shows an example exchange between donor-recipient pairs. Pairs and compatibility between pairs are encoded in a graph structure. The resulting graph is called *compatibility graph*. The problem of finding a set of exchange cycles on a graph which maximises the number of transplants being carried out is known

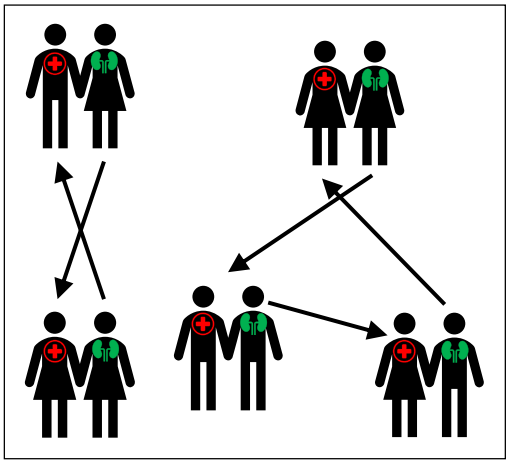

Figure 1.1: Cyclic exchange of donors in kidney exchange programmes.

as the kidney exchange problem. We use tools of graph theory and discrete optimisation to solve the kidney exchange problem.

Determining medical compatibility between pairs requires the analysis of highly sensitive medical health data of the donors and recipients, which makes it pivotal that no information is leaked accidentally to unauthorised personnel. Thus, our protocol requires the implementation of privacy-preserving techniques such that the plain text health information remains locally at each medical institution and the analysis is run on distributed data which is leaking nothing beyond the output: a set of exchange cycles with high transplantation success likelihood. In the end, these found cycles are checked by medical experts.

1.1 Contributions

We design and implement an efficient privacy-preserving kidney exchange protocol in the semi-honest security model. In comparison with current state-of-the-art [BMWM20; BMW22], we improve the medical quality of the compatibility matching by considering additional biomedical factors.

We benchmark our protocol and show that it achieves practical run-times and communication cost for real-world applications. Our protocol shows an about 30000× speedup over [BMWM20] and about 400× over [BMW22]. In addition, we provide benchmarks to further demonstrate the scalability and practicality of our protocol.

The software implementation is available in the following repository: https://github.com/encryptogroup/ppke.

The results of this work are presented in [BHK+22] which is still in submission and an abstract of this work was published at the 33rd Crypto Day [BKMS21].

1.2 Outline

In Chapter 2, we introduce the basic concepts for our privacy-preserving kidney exchange protocol. In Section 2.1, we provide the necessary medical background information that is needed for the reasoning behind choosing the biomedical compatibility factors. In Section 2.2, we explain the cryptographic building blocks and protocols used in our *Efficient and Privacy-Preserving Kidney Exchange Protocol (EPPKEP)*, and, lastly, in Section 2.3, we introduce fundamental graph theory and give an overview of the kidney exchange problem.

In Chapter 3, we discuss related work, separated into non privacy-preserving related work (Section 3.1) and privacy-preserving approaches to the kidney exchange problem (Section 3.2).

Chapter 4 contains the design and implementation details of our *EPPKEP*. In Section 4.3, we cover the compatibility graph construction given a set of pairs by evaluating their medical compatibility. Sections 4.4 and 4.5 cover the cycle evaluation and in Section 4.6, we compute the most robust set of exchange cycles. At last, we show the asymptotic complexity of our *EPPKEP* and compare it with current state-of-the-art works by Breuer et al. [BMWM20; BMW22] (cf. Section 4.7).

Chapter 5 contains the security discussion of our *EPPKEP* (cf. Section 5.1), and the run-time and communication performance of our *EPPKEP* (Section 5.2). Lastly, we compare run-time performance and communication with current state-of-the-art in Section 5.3.

Finally, in Chapter 6, we conclude this thesis 5 and give directions for future work.

2 Preliminaries

As the research problem of this thesis is located at the intersection of multiple fields - medicine, cryptography, and mathematics - we give a brief introduction into the relevant subjects. First, we explain the biomedical, then the cryptographic, and graph theory concepts.

2.1 Medical Background

First, we focus on understanding the cause of allograft rejection in kidney transplantation which is the main reason for failing kidney replacement interventions. In order to understand this, we explain the basic function of the immune systems and antigens that play a significant role in allograft rejection. In addition, we explain supplementing biomedical factors we consider in our Efficient and Privacy-Preserving Kidney Exchange Protocol (*EPPKEP*) to increase the medical quality of our solution.

2.1.1 Human Leukocyte Antigens

Every human possess an *immune system* which is responsible for protecting the body against potentially harmful invaders, called *pathogens* [AJL+02]. The immune system is a complex network of cells and proteins that defends the body against infections [Bet21]. It is divided into the *innate immune system*, which is fully developed after birth, and the *adaptive immune system* which evolves during the lifetime by building antibodies against encountered pathogens [AJL+02]. The adaptive immune reaction consists of an *antibody-mediated response* and a *cell-mediated response* [AJL+02]. The antibody-mediated response is caused by *B-cells* which are a type of white blood cells (*leukocytes*). B-cells produce antibodies which can dock on antigens and prevent them from docking on organs [AJL+02], thus, protecting the organs from possible harmful *pathogens*. The cell-mediated response is caused by *T-cells* which are another type of leukocytes. T-cells recognise unknown cell structures on antigens and trigger an antibody-mediated response by alarming B-cells, thus producing antibodies against the unknown antigen [AJL+02].

An antigen is the molecular structure on the outside of a pathogen, which can be bound by an antigen specific antibody. Antigens can be divided in two groups. First, there are *broad antigens*, which has a cell structure that can be further divided in two or more split antigens.

Second, *Split antigens* have a more refined cell surface than broad antigens and are, therefore, more specific than broad antigens. In general split antigens are assessed for determining compatibility [Eur21d].

Each human innately possesses a number of certain antigens called *Human Leukocyte Antigens (HLA)*. For every human the genetic information for the most HLA can be found on the so called *Major Histocompatibility Complex (MHC)*, which is a large area on chromosome 6 [NHK01] that is always at the same location, called *locus*. The *MHC* is a large area on the human DNA, which contains antigens encoding the surface of cells. *HLA* can be divided in up to three classes [CCA13], where only class I and II are relevant for our cause. The *HLA* classes are divided according to their position within the *MHC* and their function. *HLA* class I restricted T-cells recognise endogenous antigens synthesised within the target cell and are therefore responsible for a cell-mediated response of the adaptive immune system. *HLA* class II restricted T-cells recognise endogenously derived antigens and is restricted to antigen-presenting cells such as B-cells [Cho07]. Thus, *HLA* class II antigens are responsible for a antibody-mediated response of the adaptive immune system since they regulate how *T-cells* respond to an infection [OPS13].

A way to check whether a recipient, a person who is in need of a healthy kidney, and a donor, a person willing to donate their kidney, are compatible in general, is to perform a *HLA* cross-match [Eur21d]. For *HLA* cross-matches the *HLA* of a donor are tested against the *HLA* antibodies of a recipient. If a recipient has antibodies against one or more of the donor's *HLA*, a transplant carries a higher risk of an *Antibody Mediated Rejection (ABMR)* or allograft loss due to the already existing antibodies [LLH+10; NIK+11], i.e., they are deemed incompatible. But, in some cases, even a positive cross-match does not deem a transplant impossible because of possible immunosupressants [SCM+14], but those cases need a more in-depth assessment, which is out of scope for an algorithmic evaluation.

According to Eurotransplant, the typical *HLA* that are most frequently screened in kidney transplantation are HLA-A, -B, and -DR [Eur21e]. The split antigens are considered as they are more specific than broad antigens and are, therefore, better when it comes to determining the compatibility between a donor and a recipient. In addition, the HLA-DQ loci plays an important role, when it comes to *ABMR* [LPT+18].

The *HLA* classes we consider in our *EPPKEP* are listed in Table 2.1. They can be divided into *HLA* class I and *HLA* class II antigens. With the assessment of those *HLA*, we follow Eurotransplant's [Eur21d] guidelines for determining compatibility in kidney transplantation.

Table 2.1: *HLA* split antigens assessed for determining compatibility.

Class I			Class II	
HLA-A	HLA-B		HLA-DR	HLA-DQ
A23	B51	B38	DR11	DQ5
A24	B52	B39	DR12	DQ6
A25	B44	B57	DR13	DQ7
A26	B45	B58	DR14	DQ8
A34	B64	B49	DR15	DQ9
A66	B65	B50	DR16	
A29	B62	B54	DR17	
A31	B63	B55	DR18	
A32	B75	B56		
A33	B76	B60		
A74	B77	B61		
A68	B71			
A69	B72			

To further evaluate the quality of compatibility of a recipient and a donor, we compare the *HLA* of the recipients and donors in contrast to the cross-matching procedure, which compares antigens to antibodies. Different *HLA* in recipients and donors increase the chances of allograft loss after transplantation due to *ABMR*, which arise after the transplantation [OD12]. A *HLA* mismatch is described as a donor having a *HLA* that a recipient does not have, and vice versa. A higher amount of mismatches increases the risk of allograft failure. *HLA* mismatches are not an exclusion criteria for a transplantation because of modern immunosupressants, which reduce the risk of allograft loss. However, they decrease the allograft survival chances and are, therefore, an important factor to consider [Ope97; OD12; LCC+12]. Each person has up to two types of antigen of each *HLA* group which are inherited from their parents. Therefore, this results in a maximum of two mismatches per gene locus [NHK01].
One of the reasons why fewer mismatches lead to higher allograft survival chances is that it is less likely for a recipient to develop donor specific antibodies after the transplantation, which can be the cause for allograft rejection [OD12; LCC+12]. Especially HLA-DQ antigens play an important role in allograft survival because HLA-DQ mismatches result in a lower allograft survival rate, which comes from *ABMR* caused by the development of HLA-DQ antibodies post transplant [LPT+18].
According to Opelz et al. [OD12], we can group the number of mismatches in one of four different risk groups. Depending on the number of mismatches, the chances of allograft survival decrease. Fewer mismatches correlate with a higher allograft survival chance, but having no mismatches is very rare and mostly only happens in cases where donor and recipient are twins. The next best results achieve transplantation with only 1 to 2 *HLA* mismatches. Transplantation with 3 to 4 *HLA* mismatches also have a decent outcome, while transplantation with 5 or more *HLA* mismatches face serious *ABMR* risks.

2.1.2 ABO System

When it comes to compatibility in kidney donations, the blood group of the recipient and donor plays a significant role. One way to differentiate blood characteristics is the ABO blood system [Blu21]. In the *ABO* system, there are four different blood groups, O, which does not contain any antigens on the surface of the red blood cells (erythrocytes), A, which contains only antigen A on the surface of the erythrocytes, B, which contains only antigen B on the surface of the erythrocytes, and AB which contains antigen A and B on the surface of the erythrocytes [Blu21]. It is important that the blood groups of recipient and donor are compatible, in this case we talk about *ABO compatibility*.
While *ABO* incompatibility does not make a transplant impossible, thanks to modern immunosuppression, it increases the chance of success in kidney transplantation [WB18]. Additionally organs need to be processed such that there are no traces of an incompatible blood type left. It leads to a higher risk of allograft loss during the first year post transplantation and it increases the risk for recipients to get severe infections, e.g., viral infections, *ABMR*, and postoperative bleeding [WB18]. Still, in the long-term outcome, *ABO*-incompatible transplants have similar outcomes to *ABO*-compatible transplants. For this reason, we include *ABO* compatibility as an additional factor rather than an exclusion criteria contrary to the current state-of-the-art [BMWM20; BMW22].
A table of which blood group can donate to which blood groups is listed in Tab. 2.2.Note that we show the percentage of the European population that possess the respective blood groups.

Table 2.2: ABO compatibility according to [Blu21]

Blood Group	Population in % [FGHW14]	Can Receive From	Can Donate To
O	45	O	O, A, B, AB
A	40	O, A	A, AB
B	11	O, B	B, AB
AB	4	O, A, B, AB	AB

2.1.3 Age

According to Waiser et al. [WSB+00], age disparity plays an influential role when it comes to allograft survival post transplant. The authors have put donors and recipients in two groups according to their age. Old donors and recipients that are 55 years or older and young donors and recipients who are younger than 55 years old. They found out that old recipients with older and younger donors have better long-term allograft survival compared to young recipients. But kidneys from old donors in young recipients significantly reduce the actual allograft survival. We follow the categorisation w.r.t. age of this work [WSB+00]:

Age Groups:

Junior: This group contains all donors and recipients below the age of 55.
Senior: This group contains all donors and recipients at the age of 55 and above.

Waiser et al. [WSB+00] conclude that kidney donations within the same age group have the best outcomes and are more efficient when it comes to the usage of the donated kidney because, for older recipients, it is more common that they die from a cause unrelated to the transplantation when receiving the kidney from a younger donor, which means that the kidney could have survived longer in a younger recipient. In addition, the allograft survival of kidneys from senior donors in junior recipients is the worst.

2.1.4 Sex

The sex of donors and recipients also has an impact on allograft survival [ZCH+12]. Depending on the combination of sexes, i.e., female donor/female recipient (FDFR), female donor/male recipient (FDMR), male donor/female recipient (MDFR), male donor/male recipient (MDMR), the outcome of the transplant varies a little.
According to Zhou et al. [ZCH+12], the worst allograft survival have male recipients for female donor organs, while recipients who received their kidney from a same sex donor perform slightly better than female recipients for male donor organs.

2.1.5 Weight

In addition to the previously mentioned factors, the weight of donors and recipients has an impact on the outcome of allograft survival. It is known, that a weight difference between a donor and a recipient has a negative impact on allograft survival. According to Miller et al. [MKA+17], recipients who received a kidney from a lighter donor have higher chances of allograft loss than other recipients. El-Agroudy et al. [EHB+03] come to a similar conclusion. They reason that the allograft loss for recipients with kidneys from lighter donors can be caused by kidney not being able to support the body functions of a heavier recipient.

2.2 Cryptographic Background

In this section, we focus on understanding the cryptographic primitives and tools we use to implement our *EPPKEP*.

Our goal is to compute the results of our *EPPKEP* without leaking any sensitive medical information of the participants. To achieve that, we rely on secure computation techniques which enable us to securely compute arbitrary functions on distributed private inputs of the participants. Secure computation protocols are required to achieve that without relying on a trusted third party, i.e., a third instance that is trusted by the participants to not leak any information on the medical data of others other than what can be derived from one's own private input and the output. Proving the security of a protocol often requires the comparison of the protocol with a so-called *ideal functionality*, which can be described as the protocol being executed with a trusted third party, thus, no data is leaked.

So far there are two main paradigms that help realise that goal. The first one is *Homomorphic Encryption (HE)* which is based on public-key encryption, thus, *HE* is computationally expensive. The second main paradigm is *Secure Multi-Party Computation (SMPC)* which is based mainly on symmetric-key encryption, which is less computationally expensive than *HE* but requires more interaction. In addition, SMPC is more flexible as it can arbitrary functions. For these reasons, SMPC is more suited for complex real-world applications such as our *EPPKEP*.

2.2.1 Secure Multi-Party Computation

The main technique that is used for our *EPPKEP* is Secure Multi-Party Computation (*SMPC*). In *SMPC*, two or more parties jointly compute a functionality $\mathcal{F}$ while keeping the distributed inputs private.

The functionality of our protocol is that there are several parties where each party represents a hospital/kidney exchange centre which has one or more incompatible donor-recipient pairs with the medical data of the donor d_i and the medical data of the recipient r_i. Now, these parties jointly compute our *EPPKEP* which represents the functionality $\mathcal{F}$ without leaking any information. To achieve that, they compute compatibility between all donor-recipient pairs and evaluate the resulting graph with respect to existing cycles (cf. Subsection 2.3.2).

In *SMPC*, there are different security models for proving the protocols security:

- *semi-honest security*: The involved parties honestly follow the protocol specification, i.e., they do not try to manipulate inputs or local computations, but they try to learn additional information from the transcript which contains the communication between

the participating parties. Protocols designed for semi-honest security are not suited for all real-world applications. However, as we argue later in this section, this is an appropriate security model for this application. Furthermore, semi-honest protocols are more efficient with respect to communication.

- *covert security*: The involved parties can arbitrarily deviate from the protocol, but cheating is detected with a constant probability, e.g., $\frac{1}{2}$.
- *malicious security*: The involved parties can arbitrarily deviate from the protocol as well. However, in malicious security, cheating is detected with an overwhelming probability, e.g., very close 1. Protocols designed in a malicious security setting are considered to be very secure in the real world, but are less efficient.

We design our *EPPKEP* in a *semi-honest* security setting as the involved parties are collaborating medical institutes that can be assumed to honestly follow the protocols specification while trying to learn as much information as they possibly can since they are generally trusted and legally not allowed to share data among each other. Semi-honest security protects our *EPPKEP* against curious personnel or accidental data leakage. Therefore, semi-honest security is sufficient for our case. In addition, in contrast to malicious security, semi-honest security is more efficient which enhances the practicality of our *EPPKEP* and allows us to use *ABY* [DSZ15] as our SMPC framework.

2.2.2 ABY Framework

For implementing our *EPPKEP*, we utilise the *ABY* Framework [DSZ15]. *ABY* [DSZ15] is an efficient mixed-protocol secure Two-Party Computation framework that implements three well established *SMPC* protocols: Yao's Garbled Circuits (*Y*) [Yao86], Arithmetic Secret Sharing (*A*) [GMW87], and Boolean Secret Sharing (*B*) [GMW87]. Additionally, *ABY* implements state-of-the-art optimisations, efficient conversions between the three sharing types, and supports *Single Instruction Multiple Data (SIMD)* operations that can be used to parallelise identical operations on different data to reduce memory usage and increase the run-time performance. ABY [DSZ15] has been improved by [PSSY21], inter alia, they improve the online communication for multiplications and conversions. Unfortunately, we could not use these improved protocols for our *EPPKEP* as they have been implemented only recently in [BCS21],

For the construction of our *EPPKEP*, we consider all three sharing types as our protocol can be split into smaller portions for which we use different operations and, therefore, are possibly more efficient in different sharing types, e.g., if there are Boolean and arithmetic operations. We explain which sharing type we use for each protocol in chapter 4.

The evaluation of protocols implemented in *ABY* can be divided into an offline (setup) phase and an online phase. During the setup phase, we can precompute every operation that is independent of the inputs which allows for significant run-time improvements. During the online phase, we compute every operation interactively that depends on the inputs.

In the following, we give an overview on the three sharing types in *ABY*.

Notation

We denote the secret share of a variable x as $\langle x \rangle_i^S$ where S denotes the sharing type, i.e., $S \in \{A, B, Y\}$, and i indicates which party $P_i, i \in \{0, 1\}$ holds the share. We denote XOR operations between two secret shared values as $\oplus$, AND gates are denoted as $\wedge$, OR gates are denoted as $\vee$, Not gates are denoted as $\neg$, and MUX gates are denoted as the ternary operator condition **?** true statement **:** false statement.

Arithmetic Sharing

We use the protocols for Arithmetic Sharing used in ABY [DSZ15] that were introduced by Atallah et al. [ABL+04], Kerschbaum et al. [KSS14], and Pullonen et al. [PBS12] which are based on [GMW87]. Arithmetic sharing describes additively sharing an ℓ-bit value x as the sum of two integer values in an algebraic ring $\mathbb{Z}_{2^l}$. It is used to work on Arithmetic Circuits which are directed acyclic graphs where vertices represent the elementary operations of the represented function. For a ℓ-bit Arithmetic sharing $\langle x \rangle^A$ of x we have $\langle x \rangle_0^A + \langle x \rangle_1^A \equiv x \mod 2^\ell$, with $\langle x \rangle_0^A, \langle x \rangle_1^A \in \mathbb{Z}_{2^l}$. To share x, $P_i, i \in \{0, 1\}$ randomly chooses $r \in_R \mathbb{Z}_{2^l}$, sets $\langle x \rangle_i^A = x - r$ and sends r to P_{1-i}, who sets $\langle x \rangle_{1-i}^A = r$, thus leaking no information on x. Now, for the reconstruction of the shared value, P_{1-i} sends its share $\langle x \rangle_{1-i}^A$ to P_i who computes $x = \langle x \rangle_0^A + \langle x \rangle_1^A \mod 2^\ell$. In Arithmetic sharing, additions can be computed locally and multiplications are computed interactively in the online phase.

Boolean Sharing

In Boolean Sharing functions are represented by Boolean circuits that are evaluated using the protocol introduced in [GMW87]. Boolean Circuits are similar to Arithmetic Circuits directed acyclic graphs where the vertices consist of primitive gates which realise the represented function. Variables are secret shared using XOR operations. The Boolean Share $\langle x \rangle^B$ of a variable x is shared between two parties. It holds that $\langle x \rangle^B = \langle x \rangle_0^B \oplus \langle x \rangle_1^B$ for $\langle x \rangle_0^B, \langle x \rangle_1^B \in \mathbb{Z}_2$. Sharing a variable is similar to the secret sharing in Subsection 2.2.2: P_i randomly chooses $r \in_R \{0, 1\}$ and computes $\langle x \rangle_i^B = x \oplus r$, sets $\langle x \rangle_{1-i}^B = r$ and sends it to P_{1-i}. To reconstruct the variable x, P_{1-i} sends its share $\langle x \rangle_{1-i}^B$ to P_i who reconstructs $x = \langle x \rangle_0^B \oplus \langle x \rangle_1^B$. XOR gates can be computed locally $\langle z \rangle_i^B = \langle x \rangle_i^B \oplus \langle y \rangle_i^B$ and AND gates are evaluated interactively in the online phase.

Yao's Garbled Circuits

Yao's Garbled Circuits (Y) protocol [Yao82] is a secure two-party party computation protocol that consists of two parties, a garbler who "encrypts" the Boolean circuit to a garbled circuit and the evaluator who evaluates the garbled circuit. A garbled circuit is a Boolean circuit, i.e., a directed acyclic graph where the vertices are logic gates and the edges are wires. Each wire w is assigned two wire keys (k_0^w, k_1^w) with $k_0^w, k_1^w \in \{0,1\}^\kappa$ and κ being the symmetric security parameter. Now, the output wire keys of all gates are encrypted on all combinations of the two input wire keys by the garbler (cf. Table 2.3). The garbled truth tables are then permuted and sent to the evaluator, together with the corresponding input keys of the garbler's input. The corresponding keys of the evaluator's inputs are obliviously sent to the evaluator. After that, the evaluator iteratively decrypts the garbled gates using the input wire keys to obtain the output wire key and jointly reconstructs the clear-text output of the circuit together with the garbler. Yao Sharing requires no communication for the evaluation of XOR gate using the technique of [KS08]. For AND gates, Yao Sharing requires 2κ bits of communication [ZRE15]. The number of communication rounds is independent of the depth of the circuit. Therefore, Yao Sharing is more efficient for deep circuits and Boolean sharing is more efficient for circuits with a lower circuit depth as the number of communication rounds depends on it.

Table 2.3: Garbled AND Gate

Input w_0	Input w_1	Output w_2	Garbled Value
$k_0^{w_0}$	$k_0^{w_1}$	$k_0^{w_2}$	$Enc_{k_0^{w_0},k_0^{w_1}}(k_0^{w_2})$
$k_0^{w_0}$	$k_1^{w_1}$	$k_0^{w_2}$	$Enc_{k_0^{w_0},k_1^{w_1}}(k_0^{w_2})$
$k_1^{w_0}$	$k_0^{w_1}$	$k_0^{w_2}$	$Enc_{k_1^{w_0},k_0^{w_1}}(k_0^{w_2})$
$k_1^{w_0}$	$k_1^{w_1}$	$k_1^{w_2}$	$Enc_{k_1^{w_0},k_1^{w_1}}(k_1^{w_2})$

2.2.3 Secure Outsourcing

The *ABY* framework supports only two computation parties, but in our setting it is very likely that the input data is held by more than two parties, i. e., medical organisations. Outsourcing the computation parties from the input parties allows us to retrieve data from more than two parties [KR11]. That means that all input parties secret share their data and send one share to each of the two computation servers that are not allowed to collude. Choosing the outsourcing model provides several advantages:

- The computation servers can be located in the same network, thus allowing to compute our protocol in a setting with higher band-width and low latency. However, the computation still has to happen on two different servers operated by two different institutes.

- The communication of SMPC protocols scales linear or even quadratic in the number of computing parties. Outsourcing allows us to reduce the communication as we only have two computing parties.
- By outsourcing, the input owners do not participate in the protocol itself which protects our protocol from malicious data owners [KR11], as they can at most corrupt the correctness of our calculations but not breach confidentiality.

2.3 Graph Background

In this section, we focus on understanding the necessary graph theory and explain the kidney exchange problem.

2.3.1 Graph Theory

In our protocol, we are representing compatibility information as a graph. A graph $\mathcal{G}$ consists of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. Edges are represented as pairs of vertices. There exists an edge between two vertices $u, v \in \mathcal{V}$ if it holds that $(u, v) \in \mathcal{E}$. The edges within a graph $\mathcal{G}$ can be either directed or undirected. In an undirected graph, the pair $(u, v) \in \mathcal{E}$ for $u, v \in \mathcal{V}$ is equivalent to $(v, u) \in \mathcal{E}$, i.e., the edge pairs are unordered. In a directed graph, $(u, v) \in \mathcal{E}$ is not equivalent to $(v, u) \in \mathcal{E}$, thus the edge pairs are ordered. In addition, the edges within a graph can be either unweighted or weighted. Some edges can have more importance than other edges. To show the different importance of different edges, weighted edges are assigned different values. In our *EPPKEP*, we are evaluating a weighted directed graph to find cycles within our graph as a cycle represents two or more vertices that are connected with each other. A cycle c is a list of vertices $\{v_1, v_2, \ldots, v_{m-1}\}$ where an edge exists from vertex v_i to v_{i+1} for $i \in \{1, ..., m-1\}$. Additionally, it has to hold that there exists an edge from vertex v_{m-1} to vertex v_1 to close the cycle. A vertex disjoint cycle is a cycle c, where each vertex appears at most once in the cycle, hence it holds that for all $v, u \in c$ that $v \neq u$. We define the length of cycles as the number of edges that are used to create a cycle, i.e., the cycle $\{v_1, \ldots, v_m\}$ is of length m for $m > 1$. An example of a directed and weighted graph is shown in Figure 2.1.

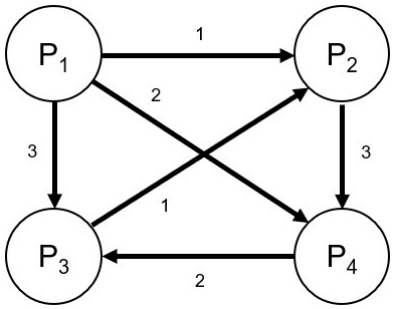

Figure 2.1: Example of a directed and weighted graph

We represent our graph as a weighted adjacency matrix which is a square matrix A where each row/column represents the edges from/to a vertex, i.e., the directed edge connecting vertex i with vertex j is encoded in the i-th row and the j-th column. Hence, if there exists an edge from vertex i to vertex j, the entry a_{ij} of A is set to the respective edge weight. If

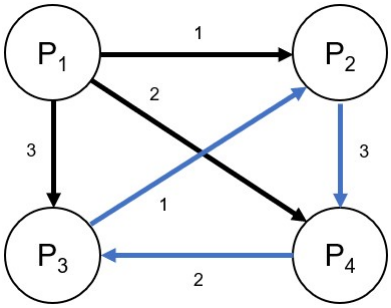

Figure 2.2: The cycle in the example graph in Figure 2.1.

there is no edge, the entry a_{ij} is set to 0. The adjacency matrix representing the graph in Figure 2.1 looks as follows:

$$\begin{pmatrix} 0 & 1 & 3 & 2 \\ 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 \end{pmatrix}$$

Unweighted adjacency matrices allow to easily compute how many paths of a given length exist in the topology by computing the respective power of the unweighted adjacency matrix, i.e., if we want to find out how may paths of length n exist, we compute the n-th power of the unweighted adjacency matrix A. In the resulting matrix, entries greater than 0 show that there exists at least one path of a given length, e.g., if the entry a_{ij} of A^n is greater than zero, then there exists a path of length n from vertex v_i to vertex v_j. The entries on the diagonal start and end with the same vertex, thus if an entry on the diagonal is greater than 0, it shows that there exists at least one cycle of length n within our graph. As an example, we show this computation with the graph shown in Figure 2.1. The graph consists of four vertices, thus the corresponding unweighted adjacency matrix A is in $\mathbb{N}^{4\times4}$:

$$A = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

To find out the number of cycles of a given length that exist in A, we compute the respective power of A. In this example, we are looking for cycles of length 3, the result looks as follows:

$$A^3 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}^3 = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

In the resulting matrix, the entries represent the number of paths of length 3 that exist in our graph. To find the cycles we have to look at the entries on the diagonal where the entries of

the second to fourth row are 1, which means that a cycle exists for vertices $v \in \{P_2, P_3, P_4\}$. As the three vertices are part of the same cycle, we know that there exists one cycle in our graph. The cycle is shown in Figure 2.2.

2.3.2 Kidney Exchange Problem

Kidney Exchange Programmes are a way to find medically compatible donors for recipients who have found a medically incompatible donor. A pair consists of a donor who is willing to donate their kidney and a recipient who is in need of a compatible kidney. Usually, the donor is a family member or close friend of the recipient who is willing to donate their kidney to the recipient but is medically incompatible. Multiple pairs take part in Kidney Exchange Programmes to find a mutually benefiting donor-recipient assignment. This way, more recipients are able to receive a compatible kidney. Hence, more lives can be saved. The goal of these Kidney Exchange Programmes is to find as many compatible pairs that can exchange their kidneys. The problem that stems of this goal is the so called *Kidney Exchange Problem (KEP)*. It is an optimisation problem on a compatibility graph [BKM+21; PCCS18].

A compatibility graph is a directed, potentially weighted graph where vertices represent donor-recipient pairs. To solve the kidney exchange problem, pairs try to exchange kidneys in a cyclic fashion.

Directed edges indicate compatibility between two pairs, i.e., if there exists the directed edge $(u, v) \in \mathcal{E}$ for $u, v \in \mathcal{V}$, then the donor from pair u is compatible with the recipient from pair v, thus pair u could donate a kidney to pair v. The goal of the *KEP* is to find either the largest exchange cycle or as many fixed-length exchange cycles as possible within the compatibility graph. Keeping the length of cycles flexible is also possible, however, it is less efficient than deciding on a fixed length of cycles beforehand. Most approaches to solve the *KEP* are based on Integer Linear Programming (*ILP*) without considering the privacy of the donor-recipient pairs' medical information. ILP computations are infeasible in a privacy-preserving fashion due to their high computational complexity. However, in kidney exchange data privacy is very important as we are working with sensitive medical data to evaluate compatibility between pairs. To include data privacy, we design and implement a Privacy-Preserving Kidney Exchange Protocol using secure computation techniques.

The first Kidney Exchange Programme was established in 2004 in the Netherlands [KKC+05] where the waiting time for kidneys from deceased donors was around 4 to 5 years which is too long for patients with severe kidney failure. After initial successes, more Kidney Exchange Programmes were introduced in Europe [BHA+18] and even international Kidney Exchange Programmes were introduced [BFS+17]. In most existing European Kidney Exchange Programmes, there are about 50 to 300 participating pairs and the Programmes are run about every three months [BGM+20].

3 Related Work

3.1 Non Privacy-Preserving Solutions

In Kidney Exchange Programmes it is possible that exchange cycles (cf. Subsection 2.3.2) fail after calculating an initial solution. One reason might be that a donor withdraws from the exchange as their recipient already has received a compatible kidney through other means or the recipient dies and the donor does not want to donate their kidney anymore. Another reason can be that a previously compatible deemed match is considered incompatible after further medical examination. This can happen due to the fact that the compatibility between pairs is first assessed virtually on an algorithmic base which cannot consider all criteria in full depth. Only after a possible exchange cycle has been found, the pairs are assessed by specialists to make sure that they are possible.
Those localised problems concerning only one pair propagate through the cycle and lead to its complete disruption [CKG+20].

3.1.1 Robust Models for the Kidney Exchange Problem

Carvalho et al. [CKG+20] present approaches to make Kidney Exchange Programmes more robust by considering possible failure modes. They propose three policies that allow to contain failing edges within the compatibility graph (cf. Subsection 2.3.2).
The first policy is called *Simple Recourse*. This policy allows to take cost (or missed gains) of planned transplants that do not proceed into account. This approach does not enable the recovery failing of exchange cycles to be recovered, but it allows better decisions regarding the global solution as failure is explicitly taken into account.
The second approach is called *Back-Arcs Recourse*. This policy allows part of a failing exchange cycle to be recovered, if the remaining pairs in the cycle can exchange their kidneys among themselves. This policy only considers the pairs that were part of the original, failing exchange cycle.
The third policy they introduce is called *Full Recourse*. This policy allows for a complete recovery of failing exchange cycles by using alternative pairs, thus replacing the failing parts.

The presented approaches are all implemented using integer linear programming which is already very computationally costly in itself. Thus, translating these implementations of the policies directly into an efficient SMPC implementation is not trivial. However, their

work shows the importance of robustness in solutions for the Kidney Exchange Problem, which is why we decided to consider robustness in a different, more computationally feasible way, by including more biomedical factors (cf. Subsection 2.1) to decrease the possibility of failing medical specialist assessments. In addition, we suggest to use smaller cycle sizes, e.g., cycles of size two or three, to reduce the damage caused by withdrawing pairs [BKM+21].

3.1.2 Medical Analysis for Evaluating Compatibility in Kidney Exchange

A 2017 study by Ashby et al. [ALR+17] examines the importance of various medical factors leading to a calculator for determining medical compatibility in kidney exchange.

The study takes 232 705 kidney-alone transplantation from 1998 to 2012 into account, where the kidney transplants stem from the Scientific Registry of Transplant recipients, which includes data on all donors, wait-listed candidates, and transplant recipients in the United States. The transplantation can be divided into three donor types: living related donors, where the donor and recipient have a biological relationship with one another, living unrelated, and deceased donors. Those three types were analysed separately. For their analysis they considered age, sex, obesity status (body mass index greater than 30), weight ratio, height ration, HLA mismatches (cf. Subsection 2.1.1) and ABO compatibility (cf. Subsection 2.1.2).

The risk calculator developed by the studies authors helped determining the leading high-impact factors used in this work.

3.2 Privacy-Preserving Solutions

Most approaches solving the Kidney Exchange Problem do not consider privacy of the recipient's and donor's medical data. However, due to the sensitive nature of medical data, it is essential to consider privacy. Breuer et al. [BMWM20] and [BMW22] are the first approaches to consider data privacy in Kidney Exchange Problems.

3.2.1 A Privacy-Preserving Protocol for the Kidney Exchange Problem

The protocol presented by Breuer et al. [BMWM20] is the first privacy-preserving solution for the kidney exchange problem. Their approach is based on Homomorphic Encryption (*HE*), to be more precise, on a threshold variant of the Paillier Cryptosystem and SMPC (cf. Subsection 2.2.1). For the determination of compatibility, they consider HLA cross matching (cf. Subsection 2.1.1) and ABO compatibility (cf. Subsection 2.1.2).

They publicly precompute the set of all exchange constellation graphs that are graphs which contain only disjoint exchange cycles (cf. Subsection 2.3.2) up to a given cycle length. They limit the length of cycles up to 3 due to the fact that the transplants within an exchange cycle are carried out simultaneously [FPS00]. This mitigates the risk that donors withdraw from the exchange after their incompatible recipient received another kidney. In addition, a lot of medical staff and other resources are required to carry out a transplantation which limits the number of transplants that can be carried out simultaneously.

In their protocol, the participating parties jointly compute the adjacency matrix which represents the compatibility graph (cf. Subsection 2.3.2) for the participating pairs. The graph is evaluated by determining compatibility between every donor and recipient of each pair. A donor and recipient of different pairs are deemed compatible if their HLA cross match is negative and they are ABO compatible. After that they evaluate the adjacency matrix by comparing it to the known sets of exchange constellation graphs to determine which of the exchange constellation graphs are potential solutions. Out of all the potential exchange constellation graphs the graph with the most edges is chosen. If there are several potential exchange constellation graphs, then one is chosen randomly. The selected graph will be returned as the solution.

The run-time of their protocol scales exponentially with the number of parties, i.e., pairs, that participate in the protocol. For two parties, the run-time is about 14 seconds, for four parties the run-time is about 44 seconds, for eight parties the run-time is about two hours, and for nine parties the run-time is already 13 hours. A higher number of parties was not evaluated in their work. This shows that the presented protocol by Breuer et al. [BMWM20]

is not feasible for a larger number of pairs, thus making it not feasible for real-world Kidney Exchange Programmes (cf. Subsection 2.3.2).

Our solution improves the run-time performance 30 000-fold for 9 pairs (cf. Section 5.3 by avoiding performance impending *HE* techniques and is implemented using highly optimised hybrid SMPC circuits (cf. Subsection 2.2.2). In addition, we have a better medical evaluation of the medical compatibility of donors and recipients.

3.2.2 Privacy-Preserving Maximum Matching on General Graphs and its Application to Enable Privacy-Preserving Kidney Exchange

Breuer et al. [BMW22] present an improved privacy-preserving protocol for crossover kidney exchange, which claims to be the first to have polynomial computational complexity. They (re-)implement their new protocol and the protocol presented in [BMWM20] in the SMPC framework MP-SPDZ [Kel20] and establish their new protocol in a dynamic setting where pairs are added and removed over time. Similar to the protocol in [BMWM20], their new protocol is secure against semi-honest adversaries but avoids the usage of *HE*. For the data aggregation, they use an outsourcing data model (cf. Subsection 2.2.3) with three computing parties and passive security.

In a crossover kidney exchange two pairs, each consisting of a recipient and a donor, exchange their kidney. Thus, crossover exchanges are essentially exchange cycles of size two. Therefore, their new work only allows the computation of exchange cycles of length two, where their previous work [BMWM20] allowed for exchange cycles of arbitrary length. By limiting the size of cycles, they achieve a significant performance boost over their previous work [BMWM20].

Their protocol consists of six phases. In the first phase, they initialise the compatibility graph. The computation of the compatibility between pairs is based on the same biological criteria as in their previous work, i.e., HLA cross matches (cf. Subsection 2.1.2) and ABO compatibility. For their graph generation, they use the same protocol as in [BMWM20], however, they implemented it in MP-SPDZ for enhanced performance. The second to fifth phase work on the computation of exchange cycles which is based on the matching algorithm by Pape and Conradt [PC80] which has an asymptotic complexity of $\mathcal{O}(|V|^3)$ where V denotes the number of vertices.

For the performance evaluation of their protocols, they used an AMD EPYC 7702P 64-core processor as a host for virtual input and computing peers. For each peer, they used a container running on Ubuntu 20.04 LTS with 4GB RAM and one core of the aforementioned processor. They tested their protocols on a 1GB per second bandwidth with varying latencies ranging from one to ten milliseconds. In comparison with their previous work [BMWM20], they are able to compute up to 64 pairs depending on the network setting. While the run-time of the

previous approach [BMW22] scaled exponentially with the number of pairs, this one scales polynomial with the number of pairs. Thus, their new protocol outperforms their old version by several orders of magnitude. However, their new approach is specialised on the computation of cycles of length but two, while their previous work allowed for arbitrary cycle lengths.

In addition to the design and implementation of their protocol [BMWM20], they establish their protocol in a dynamic setting where pairs come and go over time, e.g., due to receiving kidneys. In their dynamic setting, they have a pool of pairs where new pairs are added or old pairs removed from. They benchmark different arrival rates, i.e., time intervals after which new pairs are added to the pool. In addition, they test their approach for different match run intervals, i.e., time intervals after which the protocol is executed again. Before they execute their protocol, they randomly select a fixed subset of pairs to compute their compatibility graph. All pairs that have received a match are removed from the pool so that they won't be considered in future runs of the protocol. The removal and addition of new pairs results in an update of the compatibility graph. Their results have shown that their approach is very well suited for a dynamic setting.

3.2.3 Secure Graph Analysis at Scale

A recent work by Araki et al. [TAF+21] presents a highly scalable secure computation methodology for graph analysis. It hides all information on the topology of the graph, and the values associated with vertices and edges. It runs in a 3 server setting with either semi-honest or malicious security. One of their major contributions to efficiency is that they replaced secure sort operations with secure shuffles.

Their work is based on the *GraphSC* framework [TAF+21], which is a framework for message-passing algorithms in a secret-shared setting. Given a secret-shared directed graph $G(V, E)$, *GraphSC* enables the usage of message-passing algorithms on this secret shared graph. In message-passing algorithms, in each round all vertices send messages over their outgoing edges, and receive messages on their incoming edges. They update their state according to the received information. A round in a message-passing algorithm consists of each vertex sending and receiving messages, and updating accordingly. A current use case for these algorithms are contact tracing such as a COVID-19 contact graph [TAF+21]. Araki et al. [TAF+21] focus on the breadth-first search (BFS) and maximal independent set (MIS) algorithms. All of their protocols can be used for semi-honest and malicious adversaries.

Their results show that BFS and MIS scale linearly in the number of vertices, which is very good. Especially, BFS might be an interesting approach to analyse compatibility graphs (cf. Subsection 2.3.2) in regards to existing cycles. However, we have not considered

this approach in our work as it is not trivial to combine BFS with the rest of our protocols. This is a topic to further look into in another work.

3.2.4 Privacy-Preserving Linear Programming

As the approach of Carvalho et al. [CKG+20] showed, it is possible to use Integer Linear Programming (ILP) to solve the kidney exchange problem in a more robust manner. However, ILP is very expensive in itself and, therefore, cannot be trivially translated into a privacy-preserving setting. However, Linear Programming (LP) is less complex compared to ILP as it does not have the additional constraint that values have to be integers. One common algorithm in LP is the simplex algorithm which was implemented in a privacy-preserving way in [Tof09]. While the implementation provides information-theoretic security, it is too slow to be used in real-world applications because the values in the table representation of the coefficients of the constraint equations (tableau) grow too large during the initial iterations of the algorithm. An improved privacy-preserving variant of the simplex algorithm was implemented in [CH10] using fixed-point arithmetic. They circumvent the drawback of [Tof09] by using smaller tableaux which decreases the run-time. However, this approach is still only suitable for relatively small inputs due to its worst exponential worst case complexity. Dreier et al. [DK11] introduce an approach to improve the run-time performance of privacy-preserving LP solutions by using privacy-preserving problem transformations to reduce the size of the problem. Their approach is significantly faster than the works of [Tof09; CH10]. The use of privacy-preserving LP and ILP to solve the kidney exchange problem in a more robust manner is an interesting topic to look further into in another work.

4 Design and Implementation

In this chapter, we describe the privacy-preserving kidney exchange problem and introduce our Efficient Privacy-Preserving Kidney Exchange Protocol (*EPPKEP*). Our protocol consists of four parts each requiring the outputs of the previous parts while being separately executable, as interim results can be reused. Additionally, designing the parts to be able to be executed separately saves memory. Figure 4.1 gives a high-level overview of our *EPPKEP*. In the first part of our protocol, we compute the weighted compatibility graph which encodes the pair-wise compatibility of each donor-recipient pair-combination. Using this compatibility graph, the second part computes the number of exchange cycles that exist in the compatibility graph, i.e., the donor-recipient pairs who can potentially engage in kidney exchange. The third part computes all cycles and removes the found duplicates. The fourth and last part of our *EPPKEP* constructs sets of disjoint exchange cycles, i.e., each donor-recipient pair is part of at most one exchange cycle, and finally outputs the most robust set. The solution set of exchange cycles is the set with the highest likelihood that most transplants are successful. Note that we compute a locally optimal solution and not the global optimum. As each part of protocol can be executed independently, it is also possible to use it in a dynamic setting in which pairs are dynamically added and removed similar to the protocol of [BMW22] (cf. Subsection 3.2.2).

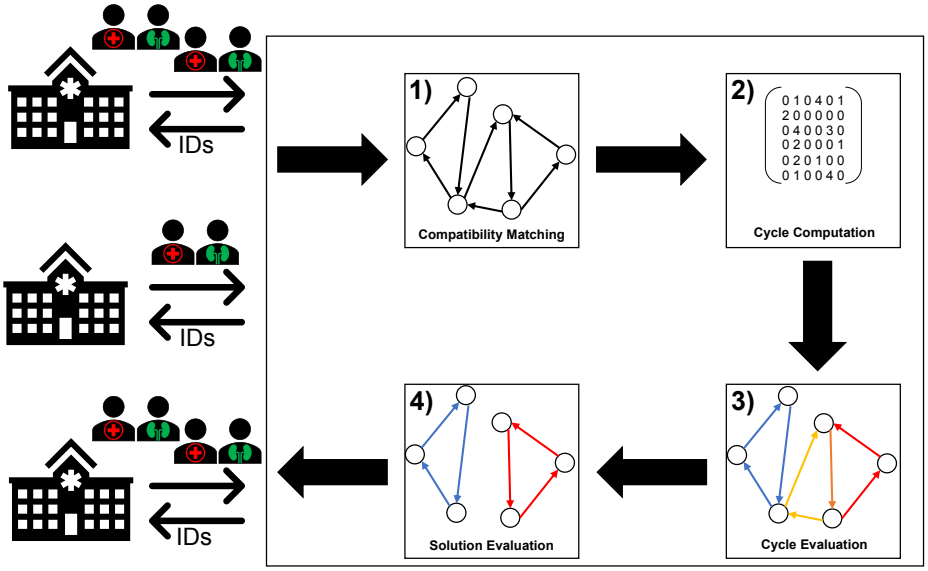

Figure 4.1: High-level overview of our *EPPKEP*.

4.1 Problem Statement

In Figure 4.2, we show the *ideal functionality* for solving the kidney exchange problem in a privacy-preserving way. Although it is unrealistic, we assume that a Trusted Third Party (TTP) is available. After receiving the medical health information of the donor-recipient pairs from medical institutes, the TTP calculates exchange cycles (cf. Section 2.3.2). The TTP outputs a set of disjoint exchange cycles that represents a local maximum with respect to the weight of the exchange cycles in the set, i.e., the set has a high probability of transplantation success. Each donor-recipient pair that is part of an exchange cycle receives information on the other donor-recipient pairs that are part of the same exchange cycle, i.e., the donor-recipient pairs they are potentially compatible with.

Our *EPPKEP* must realise the same functionality as the ideal functionality. However, as the existence of a TTP is unrealistic, it has to achieve the same functionality without relying on a TTP. Further, our protocol must achieve efficient communication cost and run-time performance such that it provides feasible run-time on standard server hardware. In order to protect the highly sensitive medical data of the donor-recipient pairs, we require our *EPPKEP* to be decentralised such that the plain-text medical health information of each pair is only stored locally at the respective medical institution. Additionally, it must offer high flexibility and adaptability for medical experts with respect to the selection of the biomedical factors and weighing the individual importance of the factors for evaluating the medical compatibility between donors and recipients. Lastly, our *EPPKEP* must be easily extendable to new biomedical factors and customise the considered HLA to adapt to new changes in research.

Note that our *EPPKEP* does not replace the evaluation of medical experts, so even when a donor and recipient are deemed compatible and exhibit a good likelihood for their exchange being carried out, medical experts must manually evaluate the match before proceeding. In general, the final decision is always in the hands of medical experts. The goal of our *EPPKEP* is to automate and, thus, accelerate the process of finding possible exchange cycles while preserving privacy.

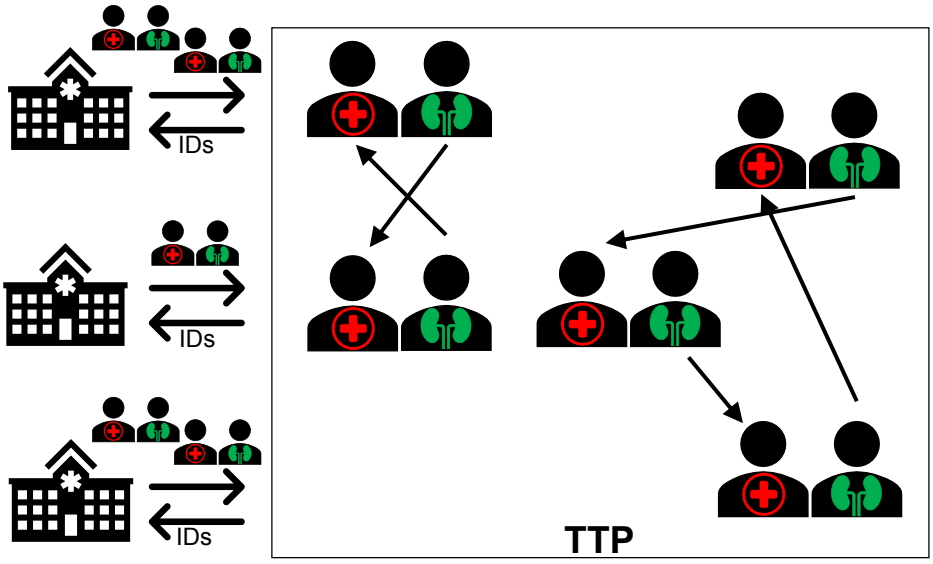

Figure 4.2: The ideal functionality for a privacy-preserving *KEP* [BHK+22].

4.2 Notation

In the following, we use the notation for secret shared values introduced in Subsection 2.2.2. In addition, we use the logical operators $\wedge$, $\vee$, and $\neg$ to denote AND gates, OR gates, and Not gates respectively. In the corresponding SMPC protocols, we use the ternary operator condition **?** true statement **:** false statement to denote Multiplexer (MUX) gates. The conversion into different sharing types is denoted as `a2b` for a conversion from Arithmetic Sharing ($\mathcal{A}$) to Boolean Sharing ($\mathcal{B}$) and `b2a` for a conversion from Boolean sharing to Arithmetic Sharing.

4.3 Compatibility Matching

First, we introduce the full protocol for the first part of our *EPPKEP* and show how each of the protocols for the medical factors are connected (cf. Subsection 4.3.1). After that, we introduce the sub-protocols that are used in the first part. We determine whether a donor and a recipient are compatible by performing a Human Leukocyte Antigen (HLA) cross-match (cf. Subsection 4.3.2). After determining a general compatibility, we further evaluate the probability of a compatible transplant being successfully carried out by including additional medical factors, i.e., HLA mismatches (cf. Subsection 4.3.3), ABO compatibility (cf. Subsection 4.3.4), age (cf. Subsection 4.3.5), sex (cf. Subsection 4.3.6), and the participants' weight (cf. Subsection 4.3.7). The result of our first part is a compatibility graph, which is used in the following two parts. Lastly, in Subsection 4.3.8, we show the asymptotic

complexities for the introduced protocols.

By parameterising the weight of the individual factors' contribution, we allow medical experts to adapt the matching process to changes or new advances in the medical field or to situational constraints. This allows to highlight or even exclude some or even all of the additional factors for each run of the protocol, so that situational irrelevant information is excluded.

4.3.1 Overview

In this subsection, we give an overview of all the protocols presented in the first part, *Compatibility Matching*, of our *EPPKEP* and how we construct our compatibility graph (cf. Subsection 2.3.2).
The most important factor to determine a general compatibility is the HLA cross-match (cf. Subsection 4.3.2). The further sub-protocols are necessary for pairs that have been deemed compatible in Subsection 4.3.2 and evaluate the compatibility quality between compatible pairs such that pairs with a high compatibility receive a higher weight. However, to avoid leaking any information on compatibility, we evaluate the sub-protocols for every combination of pairs.

Additionally, we introduce a vector $\mathsf{weight} \in \mathbb{N}^5$ as an input for our main protocol of part 1 which can be used to weight the individual biomedical factors to better adapt the current medical guidelines. We leave the choice of appropriate weight values to medical experts.

Tables 4.1 and 4.2 list the attributes of each donor and recipient. Each pair consists of a donor and a recipient. The medical data of the donor and recipient of a pair can be accessed with `pair.d` and `pair.r`, respectively.

Table 4.1: Overview of the attributes of a donor that are required to determine compatibility.

Attribute	Short Definition
hla	Encodes the HLA of a donor.
bg	Encodes the blood group of a donor.
age	Encodes the age of a donor.
sex	Encodes the sex of a donor.
weight	Encodes the body mass of a donor.

Table 4.2: Overview of the attributes of a recipient that are required to determine compatibility.

Attribute	Short Definition
hla	Encodes the HLA of a recipient.
ahla	Encodes the HLA antibodies of a recipient.
bg	Encodes the blood group of a recipient.
age	Encodes the age of a recipient.
sex	Encodes the sex of a recipient.
weight	Encodes the body mass of a recipient.

Protocol 4.1 `computeCompatibilityGraph`($\langle \text{pairs} \rangle^B$: vector of pairs, $\langle \text{weights} \rangle^A$: vector of integers) $\rightarrow \langle \text{weightedadjacencymatrix} \rangle^A$

1: $\langle \text{compG} \rangle^A \leftarrow matrix\{\langle 0 \rangle^A\}^{|\text{pairs}| \times |\text{pairs}|}$
2: **for** $i = 0, \ldots, |\text{pairs}| - 1$ **do**
3: **for** $j = 0, \ldots, |\text{pairs}| - 1$ **do**
4: **if** $i == j$ **then**
5: $\langle \text{compG} \rangle^A[i][j] \leftarrow \langle 0 \rangle^A$
6: **else**
7: $d \leftarrow \langle \text{pairs} \rangle^B[i].d$
8: $r \leftarrow \langle pairs \rangle^B[j].r$
9: $\langle \text{edge\_w} \rangle^A \leftarrow \langle 1 \rangle^A +$
$\langle \text{weights} \rangle^A[0] \cdot \texttt{b2a}(\texttt{evalHLA}(\langle \text{d.hla} \rangle^B, \langle \text{r.hla} \rangle^B)) +$ ▷ Sub-protocol 4.3
$\langle \text{weights} \rangle^A[1] \cdot \texttt{b2a}(\texttt{evalABO}(\langle \text{d.bg} \rangle^B, \langle \text{r.bg} \rangle^B)) +$ ▷ Sub-protocol 4.4
$\langle \text{weights} \rangle^A[2] \cdot \texttt{b2a}(\texttt{evalAge}(\langle \text{d.age} \rangle^B, \langle \text{r.age} \rangle^B)) +$ ▷ Sub-protocol 4.5
$\langle \text{weights} \rangle^A[3] \cdot \texttt{b2a}(\texttt{evalSex}(\langle \text{d.sex} \rangle^B, \langle \text{r.sex} \rangle^B)) +$ ▷ Sub-protocol 4.6
$\langle \text{weights} \rangle^A[4] \cdot \texttt{b2a}(\texttt{evalWeight}(\langle \text{d.weight} \rangle^B, \langle \text{r.weight} \rangle^B))$ ▷ Sub-protocol 4.7
10: $\langle \text{sel} \rangle^B \leftarrow \texttt{match}(\langle \text{d.hla} \rangle^B, \langle \text{r.ahla} \rangle^B)$ ▷ Sub-protocol 4.2
11: $\langle \text{compG} \rangle^A[i][j] \leftarrow \texttt{b2a}(\langle \text{sel} \rangle^B \texttt{ ? a2b}(\langle \text{edge\_w} \rangle^A) \texttt{ : } \langle 0 \rangle^B)$
12: **end if**
13: **end for**
14: **end for**
15: **return** $\langle \text{compG} \rangle^A$

In Protocol 4.1, we compute the compatibility graph. Given the participating pairs and the weights of each medical factor, we evaluate the sub-protocols for each medical factor and multiply the result with the respective weight.
It takes a secret shared vector pairs containing the medical information of all participating donor-recipient pairs and a secret shared vector weights which contains weights for each of the biological criteria except the HLA cross-match, i.e., how much it influences the overall likelihood for good compatibility compared to the other factors as input. The vector

weights allows medical experts to highlight certain factors or even exclude factors entirely if necessary. In Line 9, we compute the sum over all weighted biological factors, except the HLA cross-match. Afterwards, we check whether the current donor and recipient have a general medical compatibility by performing a HLA cross-match (cf. Protocol 4.2). If they have a general compatibility, we set the weight of the respective edge to the weight, otherwise we set it to 0 (cf. Line 11).
SMPC Cost. To run the complete matching process, all criteria have to be evaluated for each donor-recipient pair, i.e., Sub-protocols 4.2, 4.3, 4.4, 4.5, 4.6, and 4.7 are run $|\mathsf{pairs}|^2$ times. Then, in Protocol 4.1, we additionally evaluate five multiplications, five additions, one comparison, one AND gate, and one MUX gate. As discussed in the respective subsections, the compatibility evaluation sub-protocols are done in Boolean Sharing ($\mathcal{B}$). At the end, we convert the weight to Arithmetic Sharing ($\mathcal{A}$) to efficiently realise additions and multiplications. Using the protocols of ABY2.0 [PSSY21], the online communication could be reduced from $3 \times \ell^2 + 24 \times \ell + 2 \times \ell \times \kappa$ to $23 \times \ell + \ell \times \kappa$ in every iteration (without considering the sub-protocols) where ℓ denotes the length of the values and κ denotes the security parameter.

4.3.2 HLA Cross-Match

Given several pairs of incompatible donors and recipients, we first compute whether there exists a general immunological compatibility between some pairs based on the HLA. One of the most important approaches to do so is a HLA cross-match (cf. Subsection 2.1.1). The HLA we consider are listed in Table 2.1. The number of HLA is determined by the public parameter |HLA| which is per default set to 50 according to the observed HLA by Eurotransplant [Eur21d]. However, medical experts are free to add or remove HLA by altering the number of considered HLA.

We now encode the HLA and the corresponding antibodies for each donor and recipient as Boolean vector $\mathsf{hla} \in \{0,1\}^{|HLA|}$, for which each index encodes a different HLA. The number of HLA we account for is denoted by |HLA|, which is a publicly known parameter. If an entry is 1, the recipient possess the HLA which is encoded by the respective index. For recipients, we encode the HLA antibodies as a Boolean vector $\mathsf{ahla} \in \{0,1\}^{|\mathsf{HLA}|}$, using the same encoding as in hla. If an entry of ahla is 1, the recipient possesses an antibody against the respective HLA. The HLA antibodies of the donors are not stored since they are not relevant for kidney transplantation.

We show the general compatibility algorithm of a donor and a recipient:

Sub-protocol 4.2 $\texttt{match}(\langle \mathsf{hla_d} \rangle^B$: vector, $\langle \mathsf{ahla_r} \rangle^B$: vector$) \rightarrow \langle \mathsf{int} \rangle^B$

1: $\langle \mathsf{comp} \rangle^B \leftarrow \{\langle 0 \rangle^B\}^{|\mathsf{HLA}|}$ ▷ vector with result of each HLA comparison.
2: **for** $i = 0, \ldots, |\mathsf{HLA}| - 1$ **do** ▷ *SIMD*
3: $\quad \langle \mathsf{comp} \rangle^B[i] \leftarrow \langle \mathsf{hla_d} \rangle^B[i] \wedge \langle \mathsf{ahla_r} \rangle^B[i]$
4: **end for**
5: $\langle \mathsf{combined} \rangle^B \leftarrow \langle 0 \rangle^B$
6: **for** $i = 0, \ldots, |\mathsf{HLA}| - 1$ **do**
7: $\quad \langle \mathsf{combined} \rangle^B \leftarrow \langle \mathsf{combined} \rangle^B \vee \langle \mathsf{comp} \rangle^B[i]$
8: **end for**
9: **return** $\neg \langle \mathsf{combined} \rangle^B$

The HLA cross-match is shown in Sub-protocol 4.2 in which we determine general compatibility between donors and recipients. The number of observed HLA $|\mathsf{HLA}|$ is publicly known and the same for each donor and recipient. A vector comp stores whether the recipient possesses an antibody against any of the donor's HLA (cf. Line 3). For enhanced efficiency, we parallelise the cross-match comparison as a *Single instruction, multiple data* (*SIMD*) operation such that all HLA matches of one recipient are computed in just one step. If there was any positive match, the pair is marked as not compatible in comp (cf. Line 7), by combining the cross-match status of each HLA with bit-wise conjunctions. To prepare for further processing, we invert the result of the HLA cross-match in Line 9.
SMPC Cost. In Line 3, we evaluate $|\mathsf{HLA}| \times \mathsf{AND}$ gates (as *SIMD*). In Line 9, we evaluate $|\mathsf{HLA}| \times \mathsf{OR}$ gates, which can be created using three XOR gates and one AND gate for each OR gate[1]. Finally, we invert $combined$ once, which can be realised using an XOR gate. The circuit depth is, therefore, $|\mathsf{HLA}| + 1$ and the total number of AND gates is $2 \times |\mathsf{HLA}|$. To be most efficient, we use Boolean sharing ($\mathcal{B}$) for this protocol considering the type of operations (i.e., Boolean) and the increased efficiency thanks to our *SIMD* optimisation [DSZ15].
Note that Lines 6 to 8 can be computed as a tree to significantly reduce the depth of the circuit to $\log_2(|\mathsf{HLA}|) + 1$[2].

4.3.3 HLA Matching

After testing the general compatibility, more medical factors are evaluated to estimate the compatibility quality, i.e., evaluating positive and negative influences on allograft survival chances. After the determination of general compatibility between a donor and a recipient (cf. Section 4.3.2), we further evaluate all pairs for kidney transplantation by including more factors that influence allograft survival, i.e., increase the chance of a successful transplantation. Beyond HLA antibodies, we now compare the HLA of each donor and recipient (cf. Section 2.1.1). In contrast to the HLA cross-match, where antibodies and

[1] $A \vee B \iff 1 \oplus ((1 \oplus A) \wedge (1 \oplus B))$
[2] The benchmarks were conducted without this optimisation.

antigens presence are compared, we now compare the sets of considered antigens on both donor and recipient, as Opel et al. [OD12] show that allograft survival chances decrease with an increasing number of HLA mismatches (cf. Subsection 2.1.1). The following four groups can be derived from the results of [OD12]:

Perfect: no HLA mismatches
Good: 1-2 HLA mismatches
OK: 3-4 HLA mismatches
Bad: 5 or more HLA mismatches

This analysis uses the same Boolean vector encoding of the participants' HLA as described in Subsection 4.3.2.

We determine the result of the HLA match and place them into one of the previously defined groups with respect to the number of mismatches:

...Perfect, if sum is equal to 0, we return the value encode by the variable *A*.
...Good, if sum is greater than 0 but less than 3, we return *B*.
...OK, if sum is greater than 2 but less than 5, we return *C*.
...Bad, if sum is greater than 4, we return 0.

Sub-protocol 4.3 $\texttt{evalHLA}(\langle \mathsf{hla_d} \rangle^B$: vector of Boolean, $\langle \mathsf{hla_r} \rangle^B$: vector of Boolean$) \rightarrow \langle \mathsf{int} \rangle^B$

1: $\langle \mathsf{mm} \rangle^B \leftarrow \{\langle 0 \rangle^B\}^{|\mathsf{HLA}|}$ ▷ This list represents the number of mismatches
2: **for** $i = 0, \ldots, |HLA| - 1$ **do** ▷ *SIMD*
3: $\quad \langle \mathsf{mm} \rangle^B[i] \leftarrow \langle \mathsf{hla_d} \rangle^B[i] \oplus \langle \mathsf{hla_r} \rangle^B[i]$
4: **end for**
5: $\langle \mathsf{sum} \rangle^B \leftarrow \texttt{Hamming}(\langle \mathsf{mm} \rangle^B)$ ▷ Compute Hamming Weight.
6: $\langle \mathsf{ok} \rangle^B \leftarrow \langle \mathsf{sum} \rangle^B < \langle 5 \rangle^B$
7: $\langle \mathsf{good} \rangle^B \leftarrow \langle \mathsf{sum} \rangle^B < \langle 3 \rangle^B$
8: $\langle \mathsf{perfect} \rangle^B \leftarrow \langle \mathsf{sum} \rangle^B == \langle 0 \rangle^B$
9: **return** $\langle \mathsf{perfect} \rangle^B$ **?** $\langle \mathsf{A} \rangle^B$ **:** $\big(\langle \mathsf{good} \rangle^B$ **?** $\langle \mathsf{B} \rangle^B$ **:** $\big(\langle \mathsf{ok} \rangle^B$ **?** $\langle \mathsf{C} \rangle^B$ **:** $\langle 0 \rangle^B\big)\big)$

In Sub-protocol 4.3, we further evaluate the compatibility quality by comparing the HLA of donors and recipients.
It takes two vectors $\mathsf{hla_d}$ and $\mathsf{hla_r}$ with the HLA antigens of the donor and recipient respectively as input. The number of $|\mathsf{HLA}|$ is public as it is configurable. The vector mm indicates the HLA mismatches of the donor and the recipient. A mismatch occurs if either donor or recipient has a HLA antigen that the other does not have (cf. Line 3). For enhanced efficiency, we parallelise the comparison as *SIMD* operation such that the vector mm is computed in a single step. Afterwards, the number of HLA mismatches is determined with a Hamming Weight

Circuit (cf. Line 5). Based on the number of mismatches, the protocol outputs an indicator for the quality of the pairing w.r.t. the HLA antigens: Class *A* is an optimal fit with no mismatches, class *B* is a good fit, and class *C* is an acceptable fit with 3-4 mismatches (cf. Line 9).
SMPC Cost. To compute the HLA match, we compare the HLA vectors of a donor and recipient entry-wise and aggregate all $\langle \mathsf{hla_d} \rangle^B \neq \langle \mathsf{hla_r} \rangle^B$ into a sum by computing the Hamming Distance between the bit-field containing the differences and a $zero$ bit-field. Thus, $|\mathsf{HLA}|$ XOR gates as *SIMD* and one computation of the Hamming Weight $\mathtt{Hamming}(\mathsf{mm})$ are performed. We use the protocol from Bringer et al. [BCF+14] to compute the Hamming Weight. It has $\frac{2\times|\mathsf{HLA}|\times\log_2(\ell)}{o} + 3 \times \ell$ computation complexity and $\ell \times (|\mathsf{HLA}| \times \log_2(\ell) + \kappa)$ communication complexity where ℓ denotes the bit length, κ denotes the symmetric security parameter, and o denotes the size of the output of a pseudo-random function.
Line 3 in Sub-protocol 4.3 evaluates $|\mathsf{HLA}| \times$ XOR gates (as *SIMD*). Line 5 evaluates one Hamming Weight Protocol. Lines 6 to 9 contains three comparison and three MUX gates. Thus, the circuit depth is 7, which is equal to the number of *AND* gates. At first glance using Yao's Garbled Circuits ($\mathcal{Y}$) seems to be most efficient. However, considering that Sub-protocol 4.2 is done in $\mathcal{B}$ sharing, the conversion cost outweigh the benefits of using $\mathcal{Y}$ instead of $\mathcal{B}$, thus, $\mathcal{B}$ is used here as well.

4.3.4 Blood Type Matching

Another factor we include to further evaluate the compatibility between pairs is ABO compatibility (cf. Subsection 2.1.2).

In order to determine ABO compatibility between donors and recipients, we encode the blood group of each donor and recipient as two bit values $\mathsf{bg} \in [0,1]^2$ shown in Table 4.3. All compatible donations in terms of ABO compatibility are listed in Table 2.2.

Table 4.3: Encoding of the different blood groups.

Encoding	Blood Group
00	O
01	A
10	B
11	AB

In this encoding, the donor and recipient are compatible if either of two conditions are true: First, a donor and recipient are ABO compatible if they share the same blood group:

$$\neg\Big(\big(\mathsf{bg_r}[0] \oplus \mathsf{bg_d}[0]\big) \vee \big(\mathsf{bg_r}[1] \oplus \mathsf{bg_d}[1]\big)\Big) \tag{1}$$

Secondly, a donor and recipient are ABO compatible if the least significant bit (LSB) of the donor is greater than the most significant bit (MSB) of the recipient and vice versa:

$$\Big(\big(\mathsf{bg}_\mathsf{r}[1] \wedge \neg \mathsf{bg}_\mathsf{d}[0]\big) \vee \big(\mathsf{bg}_\mathsf{r}[0] \wedge \neg \mathsf{bg}_\mathsf{d}[1]\big)\Big) \tag{2}$$

In case of ABO compatibility, we return the value encoded as $\mathsf{best}_\mathsf{abo}$, otherwise we return 0 because ABO incompatibility does not increase the likelihood of a transplantation being successful, i.e., it does not increase the survival chances of the recipient . Note that $\mathsf{best}_\mathsf{abo}$ is variable and can be configured depending on the importance of ABO compatibility in varying settings.

Sub-protocol 4.4 `evalABO`($\langle \mathsf{bg}_\mathsf{d} \rangle^B$: vector, $\langle \mathsf{bg}_\mathsf{r} \rangle^B$: vector) $\rightarrow \langle \mathsf{int} \rangle^B$

1: $\langle \mathsf{a} \rangle^B \leftarrow \neg\Big(\big(\langle \mathsf{bg}_\mathsf{r} \rangle^B[0] \oplus \langle \mathsf{bg}_\mathsf{d} \rangle^B[0]\big) \vee \big(\langle \mathsf{bg}_\mathsf{r} \rangle^B[1] \oplus \langle \mathsf{bg}_\mathsf{d} \rangle^B[1]\big)\Big)$
2: $\langle \mathsf{b} \rangle^B \leftarrow \big(\langle \mathsf{bg}_\mathsf{r} \rangle^B[1] \wedge \neg\langle \mathsf{bg}_\mathsf{d} \rangle^B[0]\big) \vee \big(\langle \mathsf{bg}_\mathsf{r} \rangle^B[0] \wedge \neg\langle \mathsf{bg}_\mathsf{d} \rangle^B[1]\big)$
3: $\langle \mathsf{sel} \rangle^B \leftarrow \langle \mathsf{a} \rangle^B \vee \langle \mathsf{b} \rangle^B$
4: **return** $\langle \mathsf{sel} \rangle^B$ **?** $\langle \mathsf{best}_\mathsf{age} \rangle^B$ **:** $\langle 0 \rangle^B$

Sub-protocol 4.4 describes the ABO compatibility test.
It takes two two-bit vectors as input: $\mathsf{bg}_\mathsf{d} \in \{0,1\}^2$ is the blood group of the donor and $\mathsf{bg}_\mathsf{r} \in \{0,1\}^2$ is the blood group of the patient. We compute the ABO compatibility by computing an OR gate (Line 3) of Equation (1) (Line 1) and Equation (2) (Line 2).
SMPC Cost. We evaluate 3 Not gates, 2 XOR gates, 2 AND gates, 3 OR gates and one MUX gate. OR gates and Not gates can be realised in terms of XOR and AND gates as mentioned in Subsection 4.3.2. Thus, we evaluate in total 14 XOR gates and 6 AND gates. The depth of the circuit is 4 which is less than the total number of AND gates. Hence, this protocol is most efficient using $\mathcal{B}$.

4.3.5 Age Matching

HLA cross-matching, HLA matching and ABO compatibility are the most common and most impactful factors when it comes to kidney transplantation survival. However, there are other factors that have shown to impact the outcome of transplantation. One of them is the age of the donors and the recipient (cf. Subsection 2.1.3). Based on the results of Waiser et al. [WSB+00], we create two age groups (cf. Subsection 2.1.3) *junior* ($age < 55$) and *senior* ($age \geq 55$) and encode the group each donor and recipient belong to in the variable age. If a person belongs in the junior group, age is set to 0. If they belong in the senior group, age is set to 1. We can place a donor-recipient pair into one of the following three categories and return the weight associated with their respective age-group. The categories are derived from the information mentioned in Subsection 2.1.3:

Same-Group: This category contains possible transplantation between donors and recipients in the same age category, i.e., $\mathsf{age}_\mathsf{d} == \mathsf{age}_\mathsf{r} \rightarrow$ return $\mathsf{best}_\mathsf{age}$.

Junior-Senior: This category contains transplantation from young donors to old recipients, i.e., $\mathsf{age}_\mathsf{d} < \mathsf{age}_\mathsf{r} \rightarrow$ return $\mathsf{good}_\mathsf{age}$.

Senior-Junior: The last category contains transplantation from old donors to young recipients, i.e., $\mathsf{age}_\mathsf{d} > \mathsf{age}_\mathsf{r} \rightarrow$ return 0.

Note that, the Same-Group and the Junior-Senior Group have similar results [WSB+00], however, to avoid that young recipients receive less kidneys as old recipients have the same benefit from young donors, we weigh the Same-Group slightly more than the Junior-Senior. We return 0 as weight for the Senior-Junior group because kidney transplantation between old donors and young recipients do not increase the likelihood of the transplantation being successful, i.e., it does not increase the survival chances of the recipient.

Sub-protocol 4.5 `evalAge`$(\langle \mathsf{age}_\mathsf{d} \rangle^B$: int, $\langle \mathsf{age}_\mathsf{r} \rangle^B$: int$) \rightarrow \langle \mathsf{int} \rangle^B$

1: $\langle \mathsf{same} \rangle^B \leftarrow \langle \mathsf{age}_\mathsf{d} \rangle^B == \langle \mathsf{age}_\mathsf{r} \rangle^B$
2: $\langle \mathsf{ydor} \rangle^B \leftarrow \neg \langle \mathsf{age}_\mathsf{d} \rangle^B \wedge \langle \mathsf{age}_\mathsf{r} \rangle^B$
3: **return** $\langle \mathsf{ydor} \rangle^B$ **?** $\left(\langle \mathsf{same} \rangle^B \text{ ? } \langle \mathsf{best}_\mathsf{age} \rangle^B : \langle \mathsf{good}_\mathsf{age} \rangle^B \right) : \left(\langle \mathsf{same} \rangle^B \text{ ? } \langle \mathsf{best}_\mathsf{age} \rangle^B : \langle 0 \rangle^B \right)$

In Sub-protocol 4.5, we evaluate the age compatibility of donors and recipients weight.
It takes the age groups age_d and age_r of the donor and the recipient respectively as input. In Lines 1 to 2, we evaluate the category of the donor and recipient according to their age groups. The variable same contains the information whether a donor and recipient are of the same age, and ydor contains the information on whether the donor is younger than the recipient. Afterwards, we compute the respective weight of this donor and recipient constellation. (cf. Line 3).
SMPC Cost. Sub-protocol 4.5 contains one comparison, one inversion, one AND gate, and three MUX gates. Since Lines 1 to 2 are independent of each other, as well as the two MUX gates in Line 3, they can be evaluated in parallel. Thus, this sub-protocol is slightly more efficient in $\mathcal{B}$ than in $\mathcal{Y}$ since the amount of AND gates is greater than the depth of the circuit.

4.3.6 Sex Matching

The sex of recipient and donor impacts the outcome of a transplantation survival (cf. Subsection 2.1.4). For that reason we encode the sex of each donor and recipient in a variable sex. In the scope of this work, we encode the sex as follows: If a recipient is female, sex is set to 1, otherwise sex is set to 0. We will not consider intersex, since, to the best of our knowledge, there are no studies on the outcome of kidney transplantation which consider intersex people as well.

We place donors and recipients into one of the following groups based on their sex [ZCH+12]:
Sex Matches:

Same-Sex: This group contains donors and recipients of the same sex. In this case, we return $\mathsf{best}_{\mathsf{sex}}$.
Male-Female: This group contains male donors and female recipients. We return $\mathsf{good}_{\mathsf{sex}}$.
Female-Male: This group contains female donors and male recipients. We return 0.

Sub-protocol 4.6 `evalSex`($\langle \mathsf{sex_d} \rangle^B$: int, $\langle \mathsf{sex_r} \rangle^B$: int) $\rightarrow \langle \mathsf{int} \rangle^B$

1: $\langle \mathsf{same} \rangle^B \leftarrow \langle \mathsf{sex_d} \rangle^B == \langle \mathsf{sex_r} \rangle^B$
2: $\langle \mathsf{fdmr} \rangle^B \leftarrow \langle \mathsf{sex_d} \rangle^B \wedge \neg \langle \mathsf{sex_r} \rangle^B$
3: **return** $\langle \mathsf{fdmr} \rangle^B$ **?** $\left(\langle \mathsf{same} \rangle^B \text{ ? } \langle \mathsf{best}_{\mathsf{sex}} \rangle^B : \langle 0 \rangle^B \right) : \left(\langle \mathsf{same} \rangle^B \text{ ? } \langle \mathsf{best}_{\mathsf{sex}} \rangle^B : \langle \mathsf{good}_{\mathsf{sex}} \rangle^B \right)$

In Sub-protocol 4.6, we compute the weight of the sex of donor and recipients.
It takes two secret shares $\mathsf{sex_d}$ and $\mathsf{sex_r}$ as input which represent the sex of the donor and recipient, respectively. In Lines 1-2, we compute the group of a match. We store the results in the variables same, which corresponds to the Same-Sex group, and fdmr, which corresponds to the male donor/female recipient group. As a final step, the output weight of this donor and recipient constellation is computed, i.e., the optimal combination ($\mathsf{best}_{\mathsf{age}}$) with equal sex receives the highest weight, while a female donor and a male recipient are assigned the lowest weight (0).
SMPC Cost. We evaluate one comparison, one Not gate, one AND gate, and three MUX gates. Lines 1 to 2 are independent as well as the two MUX gates in Line 3 are independent. Thus, the circuit depth is 3 while the number of AND gates is 5. Therefore, Sub-protocol 4.6 is slightly more efficient in $\mathcal{B}$ than in $\mathcal{Y}$. Thus, we use $\mathcal{B}$ for this sub-protocol.

4.3.7 Weight Matching

As a last factor, we consider the weight (cf. Subsection 2.1.5) of recipients and donors. We store the information about the body mass in the variable weight for each donor and each recipient.

We put each pair of donor and recipient into one of the following two groups [MKA+17; EHB+03] based on the difference of their weights:

Fitting Weight: The weight of the donor is greater or equal to the weight of the recipient's kidney. We return $\mathsf{good}_{\mathsf{weight}}$.
Unfitting weight: The weight of the donor is smaller than the weight of the recipient. We return 0.

Sub-protocol 4.7 `evalWeight`($\langle \mathsf{weight_d} \rangle^B$: int, $\langle \mathsf{weight_r} \rangle^B$: int) $\rightarrow$ int

1: **return** $\langle \mathsf{weight_d} \rangle^B < \langle \mathsf{weight_r} \rangle^B$ **?** $\langle 0 \rangle^B$ **:** $\langle \mathsf{good_{weight}} \rangle^B$

In Sub-protocol 4.7, we evaluate the compatibility of a donor and recipient based on their weight. It takes the secret shared weights of the donor and the recipient $\mathsf{weight_d}$ and $\mathsf{weight_r}$ as input. If the donor weighs less than the recipient, the output weight is set to 0, otherwise it is set to $\mathsf{good_{weight}}$.
SMPC Cost. We evaluate only one comparison gate. As the evaluation of a single comparison is more efficient in $\mathcal{Y}$ than in $\mathcal{B}$ [DSZ15], $\mathcal{Y}$ would be more efficient. However, the conversion cost outweighs this benefit which is why $\mathcal{B}$ is used for this sub-protocol as in the previous comparison sub-protocols.

4.3.8 Complexity Assessment

In Table 4.4, the asymptotic complexity for the first part, the compatibility matching, is given along with the space complexity. The most important parameters are the number of HLA (cf. Subsection 2.1.1) $|\mathsf{HLA}|$ and the number of pairs $|\mathsf{pairs}|$. We set $|\mathsf{HLA}|$ to 50 and the number of pairs was benchmarked for $|\mathsf{pairs}| \in \{2, \ldots, 650\}$.

Table 4.4: Complexity Assessment of Part 1 – Compatibility Matching

Protocol	Time Complexity	Space Complexity
Protocol 4.1	$\mathcal{O}(\lvert\mathsf{pairs}\rvert^2 \times \lvert\mathsf{HLA}\rvert)$	$\mathcal{O}(\lvert\mathsf{pairs}\rvert^2)$
Sub-protocol 4.2	$\mathcal{O}(\lvert\mathsf{HLA}\rvert)$	$\mathcal{O}(\lvert\mathsf{HLA}\rvert)$
Sub-protocol 4.3	$\mathcal{O}(\lvert\mathsf{HLA}\rvert)$	$\mathcal{O}(\lvert\mathsf{HLA}\rvert)$
Sub-protocol 4.4	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Sub-protocol 4.5	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Sub-protocol 4.6	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Sub-protocol 4.7	$\mathcal{O}(1)$	$\mathcal{O}(1)$

4.4 Cycle Computation

In the second part of our *EPPKEP*, we determine the number of cycles that exist for the desired cycle length. As discussed in Chapter 3, we recommend cycles with length two or three to increase robustness and to consider the limited availability of medical staff as each exchange comes with at least two simultaneous operations depending on the length of the cycles. Decoupling this part from the previous one allows to compute the number of cycles of different lengths for a given compatibility graph without recomputing the compatibility graph for every cycle length. In addition, the result of this part, namely, the number of existing cycles including duplicates, is used in the following parts to find and filter all existing cycles in our graph. The cycle length cLen and the number of pairs |pairs| are a public parameters, and, thus, will not be used as input for the following protocols. In Subsection 4.4.1, we introduce the main protocol for the cycle computation, in subsection 4.4.2, we introduce the sub-protocol, and in subsection 4.4.3, we give the asymptotic complexities for the introduced (sub-)protocols.

4.4.1 Overview

The number of cycles that exist for a given length can be determined by computing the respective power of the unweighted adjacency matrix. The diagonal of the resulting matrix contains the number of cycles for each vertex. In this subsection, we introduce the main protocol for the second part, the cycle computation, of our *EPPKEP*. The length of cycles is denoted as cLen.

Protocol 4.8 `determineNumberCycles`($\langle \mathsf{compG} \rangle^A$: matrix) $\rightarrow \langle \mathsf{int} \rangle^A$

$\langle \mathsf{compG} \rangle^B \leftarrow \mathsf{a2b}(\langle \mathsf{compG} \rangle^A)$
$\langle \mathsf{uG} \rangle^A \leftarrow$ `computeUnweightedGraph`$(\langle \mathsf{compG} \rangle^B)$ ▷ Sup-protocol 4.9
$\langle \mathsf{cG} \rangle^A \leftarrow \left(\langle \mathsf{uG} \rangle^A\right)^{\mathsf{cLen}}$
$\langle |\mathsf{cycles}| \rangle^A \leftarrow \langle 0 \rangle^A$
for $i = 0, \ldots, |\mathsf{pairs}| - 1$ **do**
  $\langle |\mathsf{cycles}| \rangle^A \leftarrow \langle |\mathsf{cycles}| \rangle^A + \langle \mathsf{cG} \rangle^A[i][i]$
end for
return $\langle |\mathsf{cycles}| \rangle^A$

In Protocol 4.8, we compute the number of exchanges in our graph. It takes the secret shared weighted compatibility graph compG as input. The number of pairs |pairs| and the desired length of cycles cLen are public. We first compute the unweighted adjacency matrix in Line 1 (cf. Sub-protocol 4.9). The entries in the resulting matrix cG indicate how many paths of length cLen start at vertex i and end at vertex j. Thus, for cycles, it holds that $i == j$, where $i, j \in \{0, ..., |\mathsf{pairs}| - 1\}$, i.e., the entries on the diagonal represent the number of cycles starting and ending at any given vertex $i \in \{0, ..., |\mathsf{pairs}| - 1\}$. Following this idea, the sum of the entries of the diagonal is the total number of cycles with the given cycle length cLen.

Note that this number contains duplicates, namely, cycles that include exactly the same edges and vertices in the same order but were found via a different start vertex[3]. We remove the duplicates later in part 3 (cf. Subsection 4.5) in Sub-protocol 4.14.
SMPC Cost. Protocol 4.8 only evaluates arithmetic operations such as ADD and MUL gates, thus, we use $\mathcal{A}$. Using the improved protocols of ABY2.0[PSSY21], we could reduce the communication from $\ell \times (\frac{\ell}{2} + 2 \times \kappa + 4 \times |\mathsf{pairs}|^3 + 1.5)$ bits to $\ell \times (\kappa + 2 \times |\mathsf{pairs}|^3 + 3)$ bits where ell denotes the length of the values and κ denotes the security parameter.

4.4.2 Protocols

In this subsection, we introduce the sub-protocol for computing the unweighted compatibility graph. We compute the unweighted compatibility graph to compute the number of existing exchange cycles in our compatibility graph in Protocol 4.8.

Sub-protocol 4.9 `computeUnweightedGraph`($\langle \mathsf{compG} \rangle^B$: matrix) $\rightarrow \langle \mathsf{matrix} \rangle^A$

1: $\langle \mathsf{uG} \rangle^A \leftarrow matrix \in (\langle 0 \rangle^A)^{|\mathsf{pairs}|}$
2: **for** $i = 0, \ldots, |\mathsf{pairs}| - 1$ **do**
3: **for** $j = 0, \ldots, |\mathsf{pairs}| - 1$ **do**
4: $\langle \mathsf{uG} \rangle^A[i][j] \leftarrow \mathsf{b2a}(\langle \mathsf{compG} \rangle^B[i][j] > \langle 0 \rangle^B \ ? \ \langle 1 \rangle^B : \langle 0 \rangle^B$
5: **end for**
6: **end for**
7: **return** $\langle \mathsf{uG} \rangle^A$

Sub-protocol 4.9 takes the secret shared weighted compatibility Graph compG as input. The number of pairs |pairs| is public. In Line 4, we check if the weight of the current entry in compG is greater than $zero$. If this is the case, we set the respective entry of uG to 1. Otherwise, we set the respective entry to 0. We repeat this for each $i, j \in \{0, \ldots, |\mathsf{pairs}| - 1\}$.
SMPC Cost. We evaluate $|\mathsf{pairs}|^2$ comparisons, MUX gates, and conversion gates, which are all independent of each other. This sub-protocol is most efficient in $\mathcal{B}$ as the depth of the circuit is smaller than the number of AND gates. Using the improved protocols of ABY2.0 [PSSY21], we could reduce the online communication from $\frac{\ell^2+\ell}{2}$ to $2 \times \ell$ for each multiplication where ℓ denotes the length of the values. Additionally, the number of online rounds could be reduced by half.

[3]Cycle (A, B, C) and cycle (B, C, A) are duplicates, but cycle (C, B, A) is not.

4.4.3 Complexity Assessment

In Table 4.5, the asymptotic time and space complexity for the cycle computation is given. The complexities depend on the public inputs |pairs| which denotes the number of pairs in our graph and cLen which denotes the desired length of cycles.

Table 4.5: Complexity Assessment of Part 2 – Cycle Computation

Protocol	Time Complexity	Space Complexity
Protocol 4.8	$\mathcal{O}(\mathsf{cLen} \times \lvert\mathsf{pairs}\rvert^3)$	$\mathcal{O}(\mathsf{cLen} \times \lvert\mathsf{pairs}\rvert^2)$
Sub-protocol 4.9	$\mathcal{O}(\lvert\mathsf{pairs}\rvert^2)$	$\mathcal{O}(\lvert\mathsf{pairs}\rvert^2)$

4.5 Cycle Evaluation

The third part of our *EPPKEP* finds all cycles in the compatibility graph and extracts the duplicates. It builds upon the results of the previous parts, i.e., the compatibility graph compG and the number of cycles |cycles|.
The desired length of cycles cLen, the number of pairs |pairs|, the number of existing cycles |cycles| (computed in Protocol 4.8), and the number of unique cycles $|\text{unique}| = \lfloor \frac{|\text{cycles}|}{\text{cLen}} \rfloor$ are public parameters. In Subsection 4.5.1 we give an overview of the third part of our *EPPKEP*, we introduce the necessary sub-protocols in Subsection 4.5.2, and in Subsection 4.5.3, we show their asymptotic complexities.

4.5.1 Overview

In this subsection, we give an overview of all the sub-protocols in the third part by introducing the main protocol of the third part, the cycle evaluation. Protocol 4.10 shows how the sub-protocols in the third part are connected with each other.

Protocol 4.10 `evaluateCycles`($\langle \text{compG} \rangle^Y$: matrix) $\rightarrow \langle \text{vectoroftuples} \rangle^Y$

1: $\langle \text{allCycles} \rangle^Y \leftarrow \emptyset$
2: $\text{cCycle} \leftarrow \emptyset$
3: $\langle \text{weight} \rangle^Y \leftarrow \langle 0 \rangle^Y$
4: $\langle \text{valid} \rangle^Y \leftarrow \langle 0 \rangle^Y$
5: **for** $i = 0, \ldots, |\text{pairs}| - 1$ **do**
6: cCycle.append(i)
7: $\langle \text{allCycles} \rangle^Y \leftarrow$ `findCycles`($\langle \text{compG} \rangle^Y, \text{cCycle}, \langle \text{allCycles} \rangle^Y, \langle \text{weight} \rangle^Y, \langle \text{valid} \rangle^Y$)
8: cCycle.remove()
9: **end for**
10: $\langle \text{sortedCycles} \rangle^Y \leftarrow$ `kNNSort`($\langle \text{allCycles} \rangle^Y, |\text{cycles}|$)
11: $|\text{unique}| \leftarrow \lfloor \frac{|\text{cycles}|}{\text{cLen}} \rfloor$
12: $\langle \text{filteredCycles} \rangle^Y \leftarrow$ `removeDuplicates`($\langle \text{sortedCycles} \rangle^Y, |\text{unique}|$)
13: **return** $\langle \text{filteredCycles} \rangle^Y$

In Protocol 4.10, we find all existing exchange cycles in the compatibility graph and remove duplicates. It takes the secret shared compatibility graph compG of Sub-protocol 4.1 as input. First, we find all cycles in our compatibility graph using Sub-protocol 4.12 (cf. Lines 1 to 9). Afterwards, we compute the |cycles| cycles with the highest weight using Sub-protocol 4.13 since these cycles are the only valid cycles in the compatibility graph. To reduce the memory consumption in the fourth part, we compute the number of unique cycles |unique| (cf. Line 11)

to remove all duplicates in sortedCycles using Sub-protocol 4.14 (cf. Line 12).
SMPC Cost. In Sub-protocol 4.12, we evaluate $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!}$ comparisons, $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times \mathsf{cLen} \times 4$ ADD gates, and $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times (1 + \mathsf{cLen} \times 2)$ MUX gates. In Sub-protocol 4.13, we evaluate $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times |\mathsf{cycles}|$ comparisons and $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times |\mathsf{cycles}| \times (1 + \mathsf{cLen} \times 2)$ MUX gates. In Sub-protocol 4.14, we evaluate $|\mathsf{cycles}| \times (\mathsf{cLen}^2 + |\mathsf{unique}|)$ comparisons and AND gates, $|\mathsf{cycles}| \times \mathsf{cLen}$ OR gates, $|\mathsf{cycles}| \times (1 + |\mathsf{unique}| \times (2 + \mathsf{cLen} \times 2))$ MUX gates. In total, we evaluate, $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times \mathsf{cLen} \times 4$ ADD gates, $|\mathsf{cycles}| \times (|\mathsf{unique}| \times \mathsf{cLen}^2)$ AND gates, $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times (1 + |\mathsf{cycles}|) + |\mathsf{cycles}| \times (|\mathsf{unique}| + \mathsf{cLen}^2)$ comparisons, $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen})!} \times (1 + \mathsf{cLen} \times 2 \times (1 + |\mathsf{cycles}|)) + |\mathsf{cycles}| \times (1 + |\mathsf{unique}| \times (2 + \mathsf{cLen} \times 2)$ MUX gates, and $|\mathsf{cycles}| \times \mathsf{cLen}$ OR gates. This protocol is most efficient in $\mathcal{Y}$ sharing as we are creating a very deep circuit and each subroutine we are using is most efficient in $\mathcal{Y}$ sharing.

4.5.2 Protocols

In order to avoid leaking any information regarding the compatibility between pairs, we collect all possibly existing cycles in our compatibility graph. The number of possibly existing exchange cycles is determined in Sub-protocol 4.11 and is denoted by |allCycles|. Knowing the number of possible exchange cycles, we search for them in our compatibility graph. A cycle is encoded as a tuple consisting of two entries. The first entry specifies the weight of the cycle and the second entry is a vector containing the vertices of the cycle in order of the traversal. A weight greater than zero indicates that a cycle is valid.

Sub-protocol 4.11 #TotalCycles() → int

1: $|\mathsf{allCycles}| \leftarrow |\mathsf{pairs}|$
2: **for** $i = 1, \ldots, \mathsf{cLen} - 1$ **do**
3: $\quad |\mathsf{allCycles}| \leftarrow |\mathsf{allCycles}| \cdot (|\mathsf{pairs}| - i)$
4: **end for**
5: **return** |allCycles|

In Sub-protocol 4.11, we compute the maximum number of cycles that could possibly exist within our compatibility graph. However, each vertex may appear at most once in a cycle which limits the number of possible cycles. Sub-protocol 4.11 takes no input as the number of pairs |pairs| and the cycle length cLen are public. The computation can be done in plain text as all parameters are public. The result |allCycles| is used in Sub-protocol 4.13 to sort the resulting cycles set of Sub-protocol 4.12. As |allCycles| is public, we will not use it as input for the following protocols.

Sub-protocol 4.12 `findCycles`($\langle$compG$\rangle^Y$: matrix, cCycle: vector of integers, $\langle allCycles \rangle^Y$: vector of vectors, $\langle weight \rangle^Y$: int, $\langle valid \rangle^Y$: int) $\rightarrow$ $\langle$vectoroftuples$\rangle^Y$

```
1: if |cCycle| == cLen then
2:     ⟨weight⟩^Y ← ⟨weight⟩^Y + ⟨compG⟩^Y[cLen − 1][0]
3:     ⟨valid⟩^Y ← ⟨compG⟩^Y[cLen − 1][0] > ⟨0⟩^Y ? ⟨valid⟩^Y + ⟨1⟩^Y : ⟨valid⟩^Y + ⟨0⟩^Y
4:     ⟨addC⟩^Y ← ⟨cLen⟩^Y == ⟨valid⟩^Y
5:     ⟨cWeight⟩^Y ← ⟨addC⟩^Y ? ⟨weight⟩^Y : ⟨0⟩^Y
6:     ⟨allCycles⟩^Y.append(tuple(⟨cWeight⟩^Y, ⟨cCycle⟩^Y))
7:     ⟨weight⟩^Y ← ⟨weight⟩^Y − ⟨compG⟩^Y[cLen − 1][0]
8:     ⟨valid⟩^Y ← ⟨compG⟩^Y[cLen − 1][0] > ⟨0⟩^Y ? ⟨valid⟩^Y − ⟨1⟩^Y : ⟨valid⟩^Y − ⟨0⟩^Y
9: else
10:    for i = 0, ..., |pairs| − 1 do
11:        if cCycle.contains(i) then          ▷ Skip if the next vertex is not new.
12:            continue
13:        else
14:            ⟨weight⟩^Y ← ⟨weight⟩^Y + ⟨compG⟩^Y[−1][i]
15:            ⟨valid⟩^Y ← ⟨compG⟩^Y[−1][0] > ⟨0⟩^Y ? ⟨valid⟩^Y + ⟨1⟩^Y : ⟨valid⟩^Y + ⟨0⟩^Y
16:            cCycle.append(i)
17:            ⟨allCycles⟩^Y ← findCycles(⟨compG⟩^Y, cCycle, ⟨weight⟩^Y, ⟨valid⟩^Y, ⟨allCycles⟩^Y)
18:            cCycle.remove()
19:            ⟨weight⟩^Y ← ⟨weight⟩^Y − ⟨compG⟩^Y[−1][i]
20:            ⟨valid⟩^Y ← ⟨compG⟩^Y[−1][0] > ⟨0⟩^Y ? ⟨valid⟩^Y − ⟨1⟩^Y : ⟨valid⟩^Y − ⟨0⟩^Y
21:        end if
22:    end for
23: end if
24: return ⟨allCycles⟩^Y
```

In Sub-protocol 4.12, we recursively find all possible exchange cycles in the compatibility graph and store them with their respective weight, which also indicates whether a cycle is valid. Essentially, a valid cycle is a cycle where the vertices are connected through edges in the order they are stored in the respective vector. For example, let us assume that (A, B, C) represents a possible cycle of length 3. The cycle is valid if there exists an edge from A to B, B to C, and C to A. The weight of the cycle is the sum of the respective edge weights. If one or more of these edges are missing, the cycle is invalid and its respective weight is set to 0. As input, we receive the compatibility graph compG, a vector cCycle representing the cycle of the current pass-through, a vector of tuples allCycles containing all cycles we have found so far and their respective weights, the weight of the current cycle weight, and a counter (valid) to keep track on the number of edges in cCycle.
Lines 1 to 8 describe the base case of the recursion which is executed when the current cycle has the desired cycle length. We add the weight of the last edge to the current weight of cCycle (cf. Line 2), check whether the last vertex is connected to the first vertex (cf. Line 3), and evaluate whether the current cycle is valid by checking if valid is equal to cLen (cf. Line 4),

i.e., the number of edges is equal to the desired cycle length. We set the weight of cCycle according to cCycle being valid (cf. Line 5). Afterwards, we add the secret shared cCycle with the secret shared weight to allCycles (cf. Line 6). After inserting the current cycle in allCycles, we revert the previous operations to evaluate the next cycle, i.e., we revert the weight and valid to their state prior to starting the base case (cf. Lines 7 to 8).
From Lines 10 to 22, we handle the case when the current cycle does not have the desired length yet. In Line 11, we check whether the new vertex is already used in the current cycle. If it is, we skip this iteration as each vertex may appear at most once within an exchange cycle (cf. Subsection 2.3.2). If the vertex is new, we add the weight of the edge from the previous vertex to the new vertex to the current cycle's weight (cf. Line 14), we increment the counter valid by 1 if there is an edge from the previous vertex to the new vertex, otherwise we do not increment valid (cf. Line 15), and we add the new vertex to cCycle (cf. Line 16). Afterwards, we recursively call Sub-protocol 4.12 to continue the recursion. Once the method call returns, we revert the operations done before calling Sub-protocol 4.12 (cf. Lines 18 to 20).
SMPC Cost. The complexity of Sub-protocol 4.12 depends on the number of all possible cycles $|\mathsf{allCycles}| = \frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen}!)}$. In total, we evaluate $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen}!)}$ comparisons, $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen}!)} \times \mathsf{cLen} \times 4$ ADD gates, and $\frac{|\mathsf{pairs}|!}{(|\mathsf{pairs}|-\mathsf{cLen}!)} \times (1 + \mathsf{cLen} \times 2)$ MUX gates. This protocol is most efficient in $\mathcal{Y}$ as the the MUX gates depend on each other, thus, building a deep circuit.

Sub-protocol 4.13 `kNNSort`($\langle \text{aCycles} \rangle^Y$: vector of tuples, k: int) → vector of cycles

```
1: ⟨sortedW⟩^Y ← ∅
2: ⟨sortedC⟩^Y ← ∅
3: for i = 0 :, ..., k do
4:     ⟨sortedW⟩^Y.append(⟨0⟩^Y)
5:     ⟨vertices⟩^Y ← ∅
6:     for j = 0, ..., cLen − 1 do
7:         ⟨vertices⟩^Y.append(⟨|pairs|⟩^Y)
8:     end for
9:     ⟨sortedC⟩^Y.append(⟨vertices⟩^Y)
10: end for
11: for i = 0, ..., |aCycles| − 1 do
12:     ⟨sortedW⟩^Y[k] ← ⟨aCycles⟩^Y[i][0]
13:     ⟨sortedC⟩^Y[k] ← ⟨aCycles⟩^Y[i][1]
14:     for j = 0, ..., k − 1 do
15:         ⟨sel⟩^Y ← ⟨sortedW⟩^Y[j] > ⟨sortedW⟩^Y[j − 1]
16:         ⟨tmp1⟩^Y ← ⟨sortedW⟩^Y[j]
17:         ⟨tmp2⟩^Y ← ⟨sortedW⟩^Y[j − 1]
18:         ⟨sortedW⟩^Y[j] ← ⟨sel⟩^Y ? ⟨tmp2⟩^Y : ⟨tmp1⟩^Y
19:         ⟨sortedW⟩^Y[j − 1] ← ⟨sel⟩^Y ? ⟨tmp1⟩^Y : ⟨tmp2⟩^Y
20:         for l = 0, ..., cLen − 1 do
21:             ⟨tmp1⟩^Y ← ⟨sortedC⟩^Y[j][l]
22:             ⟨tmp2⟩^Y ← ⟨sortedC⟩^Y[j − 1][l]
23:             ⟨sortedC⟩^Y[j][l] ← ⟨sel⟩^Y ? ⟨tmp2⟩^Y : ⟨tmp1⟩^Y
24:             ⟨sortedC⟩^Y[j − 1][l] ← ⟨sel⟩^Y ? ⟨tmp1⟩^Y : ⟨tmp2⟩^Y
25:         end for
26:     end for
27: end for
28: ⟨result⟩^Y ← ∅
29: for i = 0, ..., |cycles| − 1 do
30:     ⟨result⟩^Y.append(tuple(⟨sortedW⟩^Y[i], ⟨sortedC⟩^Y[i]))
31: end for
32: return ⟨result⟩^Y
```

Sub-protocol 4.13 is a slightly adapted version of the k-nearest neighbour sort protocol in [JLL+19]. It identifies the k most robust cycles, i.e., the cycles with the highest weight and outputs them. It takes a secret shared vector of tuples aCycles as input. The length of cycles cLen is a public parameter, thus, we do not use it as input parameter.
First, we construct the resulting vectors sortedW and sortedC which will later contain the k cycles with the highest weight in decreasing order (cf. Lines 1 to 10), i.e., the most robust cycles. Afterwards, we iterate through all cycles in aCycles and perform an insertion sort like sorting algorithm. Each cycle of aCycles is inserted in sortedW and sortedC if its weight is

currently part of the k highest weights (cf. Lines 11 to 27). Thus, sortedW and sortedC are sorted in decreasing order with respect to the weight of cycles. At the end, we reconstruct the set of the k exchange cycles with the highest weight by storing them in a vector with their respective weights.

SMPC Cost. For this protocol, we evaluate $|\mathsf{aCycles}| \times k$ comparison gates and $|\mathsf{aCycles}| \times k \times (1 + \mathsf{cLen} \times 2)$ MUX gates. In total, we evaluate $|\mathsf{aCycles}| \times k \times (3 + \mathsf{cLen} \times 2)$ AND gates and the depth of the circuit is $|\mathsf{aCycles}| \times k(2 + \mathsf{cLen})$. The depth of the circuit is slightly less than the total amount of AND gates, however, we create a deep circuit, thus, we compute this protocol in $\mathcal{Y}$.

Sub-protocol 4.14 `removeDuplicates`($\langle \mathsf{sortedCycles} \rangle^Y$: vector of tuples) → vector of cycles

```
1:  for i = 0,...,|cycles| − 1 do
2:      ⟨cycle⟩^Y ← ⟨sortedCycles⟩^Y[i][1]
3:      ⟨isDuplicate⟩^Y ← ⟨0⟩^Y
4:      for j = 0,...,i − 1 do
5:          ⟨currentC⟩^Y ← ⟨sortedCycles⟩^Y[j][1]
6:          for k = 1,...,cLen − 1 do
7:              ⟨duplicate⟩^Y ← ⟨1⟩^Y
8:              for l = 0,...,cLen − 1 do
9:                  ⟨same⟩^Y ← ⟨cycle⟩^Y[l] == ⟨currentC⟩^Y[(l + k) mod cLen]
10:                 ⟨duplicate⟩^Y ← ⟨duplicate⟩^Y ∧ ⟨same⟩^Y
11:             end for
12:             ⟨isDuplicate⟩^Y ← ⟨isDuplicate⟩^Y ∨ ⟨duplicate⟩^Y
13:         end for
14:     end for
15:     ⟨sortedCycles⟩^Y[i][0] ← ⟨isDuplicate⟩^Y ? ⟨0⟩^Y : ⟨sortedCycles⟩^Y[i][0]
16: end for
17: return kNNSort(⟨sortedCycles⟩^Y, |unique|)          ▷ Sub-protocol 4.13
```

In Sub-protocol 4.14, we remove all duplicates and extract the k cycles with the highest weight, i.e., the unique exchange cycles to reduce memory consumption during the fourth part of our *EPPKEP*.

It takes a secret shared vector of tuples sortedCycles as input, which represents the result of Sub-protocol 4.13. The number of existing cycles |cycles|, the number of unique cycles |unique|, and the cycle length cLen are public parameters, thus, we do not use them as input. For each cycle in sortedCycles, we check whether the cycle already exists, i.e., if it is a duplicate. Concretely, we check if the cycle at index $i \in \{0, \ldots, |\mathsf{cycles}|\}$ is equal to any cycle at index $j \in \{0, \ldots, i-1\}$ (cf. Lines 1 to 16). In Lines 6 to 13, we check whether two cycles are duplicates. In order to do so, we check if the vertex of cycle at index $l \in \{0, \ldots, \mathsf{cLen}-1\}$ and the vertex of currentC at index $(l + k) \mod \mathsf{cLen}$ are the same for $k \in \{1, \ldots, \mathsf{cLen}-1\}$. The expression $(l + k) \mod \mathsf{cLen}$ represents a shift of the index by k (cf. Line 9). By doing

so, we only need $(\mathsf{cLen}-1)\times\mathsf{cLen}$ comparison gates instead of cLen^2. This approach is correct as each cycle has a unique order per design, thus, duplicates of a cycle are cycles that are shifted by $k \in \{1,\ldots,\mathsf{cLen}-1\}$. In Line 15, we set the weight of the current cycle to 0 if the cycle already exists, otherwise we do not change the cycle's weight. At the end of Sub-protocol 4.14, each duplicate in sortedCycles is marked with 0 as its weight. With the help of Sub-protocol 4.13, we sort sortedCycles and return only the |unqiue| cycles with the highest weight, thus, removing the duplicates as they have a weight of 0. The number of unique cycles |unique| can be determined by dividing the number of cycles that exist in our compatibility graph |cycles| by the length of cycles cLen, i.e., $|\mathsf{unique}| = \lfloor \frac{|\mathsf{cycles}|}{\mathsf{cLen}} \rfloor$.

SMPC Cost. We evaluate $|\mathsf{cycles}| \times \sum_{i=0}^{|\mathsf{cycles}|} (\mathsf{cLen}\times(\mathsf{cLen}-1))$ comparison and AND gates, $|\mathsf{cycles}| \times \sum_{i=0}^{|\mathsf{cycles}|} (\mathsf{cLen}-1)$ OR gates, and |cycles| MUX gates and we evaluate Sub-protocol 4.13. This results in a total amount of $|\mathsf{cycles}| \times (1 + \frac{|\mathsf{cycles}|}{2} * ((\mathsf{cLen}-1)\times(1+\mathsf{cLen}\times 2)) + |\mathsf{unique}| \times (3+\mathsf{cLen}\times 2))$ AND gates, however, the depth of the circuit is slightly less with $|\mathsf{cycles}| \times (1+\frac{|\mathsf{cycles}|}{2} * ((\mathsf{cLen}-1)\times(1+\mathsf{cLen}\times 2)) + |\mathsf{unique}| \times (2+\mathsf{cLen}))$. Even though the depth of the circuit is smaller than the the total amount of AND gates, we evaluate this protocol in $\mathcal{Y}$ because the created circuit is really deep.

4.5.3 Complexity Assessment

In Table 4.6, the asymptotic complexity for this part is listed. The most important parameters are the number of pairs |pairs|, the possible number of cycles |allCycles|, the number of existing cycles |cycles|, the number of unique cycles |unique|, the length of cycles cLen, the number of elements in aCycles |aCycles|, and k. The number of possible cycles depends on the number of pairs and the desired cycle length, i.e., |allCycles|, and the number of unique cycles depends on the number of cycles and the desired cycle length, i.e., $|\mathsf{unique}| = \lfloor \frac{|\mathsf{cycles}|}{\mathsf{cLen}} \rfloor$.

Table 4.6: Complexity Assessment of Phase 3 – Cycle Evaluation

Protocol	Time Complexity	Space Complexity
Protocol 4.10	$\mathcal{O}(\lvert\mathsf{pairs}\rvert^{\mathsf{cLen}})$	$\mathcal{O}(\mathsf{pairs}\rvert^{\mathsf{cLen}+1})$
Sub-protocol 4.11	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Sub-protocol 4.12	$\mathcal{O}(\lvert\mathsf{pairs}\rvert^{\mathsf{cLen}})$	$\mathcal{O}(\mathsf{pairs}\rvert^{\mathsf{cLen}+1})$
Sub-protocol 4.13	$\mathcal{O}(\lvert\mathsf{aCycles}\rvert \times k \times \mathsf{cLen})$	$\mathcal{O}(k \times \mathsf{cLen})$
Sub-protocol 4.14	$\mathcal{O}(\lvert\mathsf{cycles}\rvert^2)$	$\mathcal{O}(\lvert\mathsf{cycles}\rvert \times \mathsf{cLen})$

4.6 Solution Evaluation

In the last part of our *EPPKEP*, we determine the most robust set of disjoint cycles. In this way, our *EPPKEP* outputs a set of disjoint cycles with the highest weight, so that either many exchange cycles are carried out or fewer but more robust cycles, i.e., exchange cycles that are more likely to have a successful transplantation. The cycles in the resulting set have to be disjoint as each pair can donate/receive at most one kidney. Note that we compute a locally optimal solution to reduce memory consumption. The local optimum might differ from the global solution.
The number of pairs |pairs|, the number of unique cycles |unique|, and the desired cycle length cLen are a public parameter, and, thus, will not be used as input for our protocols. In Subsection 4.6.1, we introduce the main protocol of the last part, the solution evaluation, of our *EPPKEP* which combines the sub-protocols introduced in Subsection 4.6.2. Finally, in Subsection 4.6.3, we show the asymptotic complexities of the protocols.

4.6.1 Overview

After having found all valid and unique cycles in the compatibility graph, we construct a set of exchange cycles where all cycles in a set are vertex disjoint. Afterwards, we output the set with the (locally) highest weight, i.e., the set that contains either the most possible exchange cycles or fewer but more robust exchange cycles, i.e., exchange cycles that are more likely to have successful transplantation.

Protocol 4.15 `evalSolution`($\langle \text{cycles} \rangle^Y$: vector of tuples) → tuple(int, vector of vectors)

```
1: ⟨sets⟩^Y ← ∅
2: ⟨weights⟩^Y ← ∅
3: ⟨dummyC⟩^Y ← {⟨|pairs|⟩^Y}^cLen
4: for i = 0,...,|unique| − 1 do
5:     ⟨currentS⟩^Y ← ∅
6:     ⟨currentS⟩^Y.append(⟨cycles⟩^Y[i][1])
7:     ⟨weight⟩^Y ← ⟨cycles⟩^Y[i][0]
8:     cycleCount ← 1
9:     for j = 0,...,|unique| − 1 do
10:        if i == j then
11:            continue
12:        end if
13:        ⟨currentC⟩^Y ← ⟨cycles⟩^Y[j][1]
14:        ⟨disjoint⟩^Y ← disjointCycles(⟨cycles⟩^Y, ⟨currentC⟩^Y, cycleCount)
15:        ⟨vertices⟩^Y ← ∅
16:        ⟨vertices⟩^Y.append(⟨disjoint⟩^Y ? ⟨currentC⟩^Y : ⟨dummyC⟩^Y)
17:        ⟨weight⟩^Y ← ⟨disjoint⟩^Y ? ⟨weight⟩^Y : ⟨0⟩^Y
18:        ⟨currentS⟩^Y.append(⟨vertices⟩^Y)
19:        cycleCount ← cycleCount + 1
20:    end for
21:    ⟨sets⟩^Y.append(⟨currentS⟩^Y)
22:    ⟨weights⟩^Y.append(⟨weight⟩^Y)
23: end for
24: return findMaximumSet(⟨sets⟩^Y, ⟨weights⟩^Y)
```

In Protocol 4.15, we compute sets of *disjoint* exchange cycles and output the set with the highest weight. The weight of each set is determined by the aggregated weights of the cycles in the set and indicates the likelihood that exchange cycles in a set are carried out successfully, i.e., the transplantation being successful. A donor-recipient pair can be involved in only one exchange cycle, thus, the exchange cycles in a set must be disjoint.
It takes a secret shared vector of tuples cycles with all valid unique cycles and their respective weights as input. The number of pairs |pairs|, the number of unique cycles |unique|, and the cycle length cLen are a public variables, thus, we do not pass them as function input. It iterates over each valid cycle currentC and checks if it is disjoint to all other previously analysed disjoint cycles in the set currentS (cf. Line 14). Sub-protocol 4.16 is used for computing whether a cycle is disjoint from all cycles in a set. Only if currentC is disjoint, we add it to currentS and increase the sum of weights of currentS by currentC's weight in Lines 16 to 19. After the analysis of a cycle currentC, the vector sets containing all disjoints sets is updated and the corresponding weight of the set is stored in weights (cf. Lines 21 to 22). After finishing the analysis of all cycles, we return the the set with the highest weight along with its weight by using Sub-protocol 4.17 (cf. Line 24).

SMPC Cost. In total, we evaluate $|\mathsf{unique}|^2$ ADD gates, $|\mathsf{unique}|^2 \times \mathsf{cLen}^2 + |\mathsf{unique}|$ comparisons, $4 \times |\mathsf{unique}|^2 + |\mathsf{unique}|$ MUX gates, and $|\mathsf{unique}|^2 \times \mathsf{cLen}^2$ OR gates. The solution evaluation is most efficient in $\mathcal{Y}$ as there are only few arithmetic operations and mostly comparisons. Additionally, the depth of the circuit is the total amount of AND gates used in this protocol. However, due to memory consumption this protocol has to be executed in $\mathcal{B}$.

4.6.2 Protocols

In this subsection, we introduce the necessary sub-protocols for the fourth part, the solution evaluation, of our *EPPKEP*.

Sub-protocol 4.16 `disjointCycles`($\langle \mathsf{cycles} \rangle^Y$: vector of tuples, $\langle \mathsf{cCycle} \rangle^Y$: vector, cycleCount:int) → Boolean

1: $\langle \mathsf{notD} \rangle^Y \leftarrow \langle 0 \rangle^Y$
2: **for** $i = 0, \ldots, \mathsf{cycleCount} - 1$ **do**
3: $\quad \langle \mathsf{cycle} \rangle^Y \leftarrow \langle \mathsf{cycles} \rangle^Y[i][1]$
4: $\quad$ **for** $j = 0, \ldots, \mathsf{cLen} - 1$ **do**
5: $\quad\quad$ **for** $k = 0, \ldots, \mathsf{cLen} - 1$ **do**
6: $\quad\quad\quad \langle \mathsf{tmp} \rangle^Y \leftarrow \langle \mathsf{cycle} \rangle^Y[j] == \langle \mathsf{cCycle} \rangle^Y[k]$
7: $\quad\quad\quad \langle \mathsf{notD} \rangle^Y \leftarrow \langle \mathsf{notD} \rangle^Y \vee \langle \mathsf{tmp} \rangle^Y$
8: $\quad\quad$ **end for**
9: $\quad$ **end for**
10: **end for**
11: **return** $\neg \langle \mathsf{notD} \rangle^Y$

In Sub-protocol 4.16, we computer whether a cycle cCycle does not join vertices with other cycles of an input set of cycles cycles. It takes as input a secret shared vector of tuples cycles, a secret shared cycle cCycle, and a counter cycleCount which determines the number of cycles in cycles. For each cycle cycle in cycles, we check whether cCycle shares a vertex with any of them (cf. Lines 2-10). If even a single cycle in cycles shares a vertex with cCycle, notD is set to 1 (cf. Line 7). At the end, we invert the result, for further evaluation.
SMPC Cost. In this sub-protocol, we evaluate $\mathsf{cycleCount} \times \mathsf{cLen}^2$ comparisons and OR gates. At the end, we evaluate one XOR gate. As each of the AND gates depend on each other, the circuit depth is the same as the number of AND gates. Thus, this protocol is most efficient in $\mathcal{Y}$ sharing. However, the protocol (cf. Protocol 4.15) that calls this sub-protocol has to be executed in $\mathcal{B}$ due to the occurrence of bugs after conversion requires this protocol to be in $\mathcal{B}$ sharing to avoid conversion cost.
Note that Lines 5 to 8 can be computed as a tree to significantly reduce the depth of the

circuit, which improves the run-time performance of this protocol when computed in $\mathcal{B}^4$.

Sub-protocol 4.17 `findMaximumSet`($\langle$cyclesSets$\rangle^Y$: vector of tuples, $\langle cycleW \rangle^Y$: vector) $\rightarrow \langle$tupleofvectors$\rangle^Y$

1: $\langle$weights$\rangle^Y \leftarrow \emptyset$
2: $\langle$sets$\rangle^Y \leftarrow \emptyset$
3: **for** $i = 0, 1$ **do**
4: $\quad \langle$weights$\rangle^Y.append(\langle 0 \rangle^Y)$
5: $\quad \langle$set$\rangle^Y \leftarrow \emptyset$
6: $\quad$ **for** $j = 0, \ldots |$unique$| - 1$ **do**
7: $\quad\quad \langle$vertices$\rangle^Y \leftarrow \emptyset$
8: $\quad\quad$ **for** $j = 0, \ldots$ cLen $- 1$ **do**
9: $\quad\quad\quad \langle$vertices$\rangle^Y.append(\langle |$pairs$| \rangle^Y)$
10: $\quad\quad$ **end for**
11: $\quad\quad \langle$set$\rangle^Y.append(\langle$vertices$\rangle^{set})$
12: $\quad$ **end for**
13: $\quad \langle$sets$\rangle^Y.append(\langle$set$\rangle^Y)$
14: **end for**
15: **for** $i = 0, \ldots, |$unique$| - 1$ **do**
16: $\quad \langle$weights$\rangle^Y[1] \leftarrow \langle$cycleW$\rangle^Y[i]$
17: $\quad \langle$sets$\rangle^Y[1] \leftarrow \langle$cycleSets$\rangle^Y[i]$
18: $\quad \langle$sel$\rangle^Y \leftarrow \langle$weights$\rangle^Y[1] > \langle weights \rangle^Y[0]$
19: $\quad \langle$tmp1$\rangle^Y \leftarrow \langle$weights$\rangle^Y[1]$
20: $\quad \langle$tmp2$\rangle^Y \leftarrow \langle$weights$\rangle^Y[0]$
21: $\quad \langle$weights$\rangle^Y[1] \leftarrow \langle$sel$\rangle^Y$ **?** $\langle$tmp2$\rangle^Y$ **:** $\langle$tmp1$\rangle^Y$
22: $\quad \langle$weights$\rangle^Y[0] \leftarrow \langle$sel$\rangle^Y$ **?** $\langle$tmp1$\rangle^Y$ **:** $\langle$tmp2$\rangle^Y$
23: $\quad$ **for** $j = 0, \ldots, |$unique$| - 1$ **do**
24: $\quad\quad \langle$tmp1$\rangle^Y \leftarrow \langle$sets$\rangle^Y[1][j]$
25: $\quad\quad \langle$tmp2$\rangle^Y \leftarrow \langle$sets$\rangle^Y[0][j]$
26: $\quad\quad \langle$sets$\rangle^Y[1][j] \leftarrow \langle$sel$\rangle^Y$ **?** $\langle$tmp2$\rangle^Y$ **:** $\langle$tmp1$\rangle^Y$
27: $\quad\quad \langle$sets$\rangle^Y[0][j] \leftarrow \langle$sel$\rangle^Y$ **?** $\langle$tmp1$\rangle^Y$ **:** $\langle$tmp2$\rangle^Y$
28: $\quad$ **end for**
29: **end for**
30: **return** $tuple(\langle$weights$\rangle^Y[0], \langle$sets$\rangle^Y[0])$

In Sub-protocol 4.17, we compute the set of cycles with the highest sum of weights, thus, the set of cycles with the highest probability having successful transplantation. Note that we do not compute a global solution but a local solution. Sub-protocol 4.17 takes a secret shared vector of tuples cyclesSets and a secret shared vector weights as input. cyclesSets contains all sets of disjoint cycles and weights contains the respective weights of the each set.

[4]This protocol was computed in $\mathcal{B}$, however, without this optimisation

The number of pairs |pairs| and the number of unique cycles |unique| are public parameters, thus, we do not pass them as function arguments.
This protocol is a variation of Sub-protocol 4.13 to adapt to the new data structures, i.e., sets of cycles with weights. The parameter k corresponds to 1 since we only need the set with the highest weight.
SMPC Cost. We evaluate |unique|2 comparison gates and $|\mathsf{unique}|^2 + |\mathsf{unique}|$ MUX gates. This protocol is most efficient in $\mathcal{Y}$ sharing as the number of AND gates is equal to the depth of the circuit which is created by AND gates.

4.6.3 Complexity Assessment

Even though all protocols in this part are more efficient in $\mathcal{Y}$, we compute them in $\mathcal{B}$ sharing due to memory consumption. The result of the previous part is stored in $\mathcal{B}$, which would require to convert each sharing to $\mathcal{Y}$ which creates too much memory overhead for our server. However, converting to $\mathcal{Y}$ before executing Sub-protocol 4.17 is possible without causing undefined behaviour.

In Table 4.7, the asymptotic time and space complexity for this part is listed. The most important parameters are the number of unique cycles |unique| and the length of cycles cLen.

Table 4.7: Complexity Assessment of Part 4 – Solution Evaluation

Protocol	Time Complexity	Space Complexity
Protocol 4.15	$\mathcal{O}(\|\mathsf{unique}\|^3 \times \mathsf{cLen}^2)$	$\mathcal{O}(\|\mathsf{unique}\|^2 \times \mathsf{cLen})$
Sub-protocol 4.16	$\mathcal{O}(\|\mathsf{unique}\| \times \mathsf{cLen}^2)$	$\Theta(1)$
Sub-protocol 4.17	$\mathcal{O}(\|\mathsf{unique}\|^2)$	$\mathcal{O}(\|\mathsf{unique}\| \times \mathsf{cLen})$

4.7 Overall Complexity Assessment

In this section, we show the overall time and space complexity of our *EPPKEP* and compare them with current state-of-the-art works [BMWM20; BMW22].
In Table 4.8, we show the total complexities of all four parts and the complexities of the entire protocol. The time complexity of the entire protocol depends on the number of participating pairs |pairs|, the number of observed HLA |HLA|, the length of cycles cLen, and the number of unique cycles that exist in the compatibility |unique|.

Table 4.8: Total Complexity Assessment

Part	Time Complexity	Space Complexity
1 (cf. Section 4.3)	$\mathcal{O}(\|\mathsf{pairs}\|^2 \times \|\mathsf{HLA}\|)$	$\mathcal{O}(\|\mathsf{pairs}\|^2)$
2 (cf. Section 4.4)	$\mathcal{O}(\|\mathsf{pairs}\|^{\mathsf{cLen}})$	$\mathcal{O}(\|\mathsf{pairs}\|^2)$
3 (cf. Section 4.5)	$\mathcal{O}(\|\mathsf{pairs}\|^{\mathsf{cLen}})$	$\mathcal{O}(\mathsf{pairs}\|^{\mathsf{cLen}+1})$
4 (cf. Section 4.6)	$\mathcal{O}(\|\mathsf{unique}\|^3 \times \mathsf{cLen}^2)$	$\mathcal{O}(\|\mathsf{unique}\|^2 \times \mathsf{cLen})$
Total	$\mathcal{O}(\|\mathsf{pairs}\|^2 \times \|\mathsf{HLA}\| + \mathsf{cLen} \times \|\mathsf{pairs}\|^3 + \|\mathsf{unique}\|^3 \times \mathsf{cLen}^2)$	$\mathcal{O}(\mathsf{pairs}\|^{\mathsf{cLen}+1})$

Table 4.9 contains the communication and round complexities of current state-of-the-art [BMWM20; BMW22]. For a detailed complexity analysis of the protocols, we refer to the respective works. In [BMWM20], l denotes the number of parties that participate in the protocol, s denotes the security parameter, and $|\mathcal{G}^{EC}_{(3)}|$ denotes the number of disjoint exchange cycles. Similarly to our *EPPKEP*, the protocol of Breuer et al. 2020 [BMWM20] depends on the number of participating pairs and the number of cycles in the compatibility graph. However, their protocol depends on the number of disjoint exchange cycles while our *EPPKEP* depends on the number of all existing exchange cycles which is greater than or equal to the number of disjoint exchange cycles. The protocol of Breuer et al. 2022 [BMW22] solely depends on the number of pairs, however they only consider cycles of length $L = 2$ (cf. Subsection 3.2.2).

Table 4.9: Complexities of Current State-of-the-Art

Part	Communication Complexity	Round Complexity
[BMWM20]	$\mathcal{O}(l^3 \times s + l^2 \times s \times \|\mathcal{G}^{EC}_{(3)}\|))$	$\mathcal{O}(l^3 + l \times \|\mathcal{G}^{EC}_{(3)}\|)$
[BMW22]	$\mathcal{O}(\|pairs\|^5)$	$\mathcal{O}(\|pairs\|^4)$

5 Evaluation

As performance and real-world feasibility with respect to run-time and communication are included as requirements, we present our performance benchmarks and compare them to current state-of-the-art works by Breuer et al. [BMWM20; BMW22].

Our performance benchmarks were performed on two servers equipped with Intel Core i9-7960X processors and 128GB RAM. The servers are connected via 10 Gb/s LAN with an unmodified average latency of 1.3 ms. Our benchmarks are averaged over 10 runs.

In order to provide meaningful performance benchmarks for varying real-world settings, we used two different network settings for privacy-preserving kidney exchange. The most relevant real-world scenario is a local area network (*LAN*) which is characterised by high bandwidth and low latency. Our *LAN* setting supports a 10 Gb/s connection with an average latency of 1.3 ms. The other possible real-world scenario are wide area networks (*WAN*) which are characterised by a lower bandwidth and higher latency compared to a *LAN* setting. Regional hospitals and residential physicians are more likely to be connected through *WAN*s. Our *WAN* is restricted to 100 Mb/s bandwidth with 100 ms latency. In addition, we reproduced the 1 Gb/s bandwidth 1 ms latency network used in the current state-of-the-art to compare performances.

5.1 Security Discussion

In this section, we lay out why our Efficient and Privacy-Preserving Kidney Exchange Protocol (*EPPKEP*) is secure in the semi-honest security model, i.e., a semi-honest adversary in control of one of the computing parties learns nothing beyond what can be derived from its own input and the outputs it receives from the protocol. Generally speaking, our security follows directly from the provable security of the used SMPC techniques and the security provided by the implementation (cf. Section 2.2.2).

At the beginning of our protocol, the data owners either secret-share their inputs among themselves or in an outsourcing scenario they share their data among two computation parties (cf. Section 2.2.3). Either way, the parties only have access to indistinguishable random secret shares that do not leak any information. The security of our protocol follows

directly from the provable security of the used SMPC techniques: Boolean Sharing [GMW87], Arithmetic Sharing [GMW87], and Yao's Garbled Circuits [Yao86]. The secret shares of the medical health information are never opened at any point of our protocol and only conversions between sharing types are run which are provably secure [DSZ15]. After the first phase, we output the compatibility graph as secret shares, thus, not leaking any information. In the second part, we compute the number of cycles that exist in our graph and open-up the resulting secret share. However, revealing the number of exchange cycles in our graph does not leak significant information on the private inputs since we do not reveal which donor-recipient pairs are part of the exchange cycles. If the compatibility graph is fully connected, revealing the number of exchange cycles leaks that all donor-recipient pairs are compatible but not the degree of compatibility. Hence, it does not leak details of patients' medical information. The output of the third part are all secret-shared cycles with their respective weight excluding duplicates. The secret shared cycles and their respective weights do not leak any information and the number of cycles has already been opened after part two. In the fourth part of our protocol, we output secret shares of the locally most robust set of cycles, which does not leak anything beyond what can be derived from the adversary controlled party's private input and its output. Thus, our *EPPKEP* is secure in the semi-honest security model and the private input remains hidden during the entire protocol.

5.2 Performance Benchmarks

In Figure 5.1, we show the overall run-time of our *EPPKEP* for cycle lengths $L = 2$ and $L = 3$. The full benchmarks are in the appendix (cf. Appendix A.1). During the evaluation of our protocol for cycle length $L = 3$, RAM consumption was the bottleneck for execution. We benchmarked up to 40 pairs for $L = 2$ and up to 18 pairs for $L = 3$. We extrapolated the remaining run-times for cycle length $L = 3$ according to the underlying polynomial complexity. We present the fitted models in Appendix A.2. The sudden increase in run-time for $L = 3$ between 12 and 13 pairs stems from memory swapping.
We observe that there is a polynomial relationship between the number of participating pairs and the run-time of our protocol which is reflected in the power-law development in the semi-log graphs, as expected. For $L = 2$ and 40 pairs, we achieve a run-time of under 4 minutes in our LAN setting and under 15 minutes in our WAN setting. Thus, our *EPPKEP* demonstrates real-world applicable performance in both network scenarios. The run-time increase in the WAN setting compared to the LAN setting is nearly an order of magnitude, however, it still shows that it is theoretically feasible for smaller, local hospitals with residential Internet connections to participate in kidney exchanges using our protocol. For cycle length $L = 3$, the exponent in time complexity[1] increases such that the run-times rise significantly. 25 pairs for cycle length $L = 3$ have a run-time of approximately 1 hour in our LAN setting which is still feasible for real-world applications. However, in our WAN setting cycle length $L = 3$ and 25 pairs have a run-time of approximately 200 hours which is

[1] This shows as a steepen slope in the graphs.

not feasible for real-world applications. This can be mitigated by more frequent re-runs of the protocol with a reduced number of participating pairs.

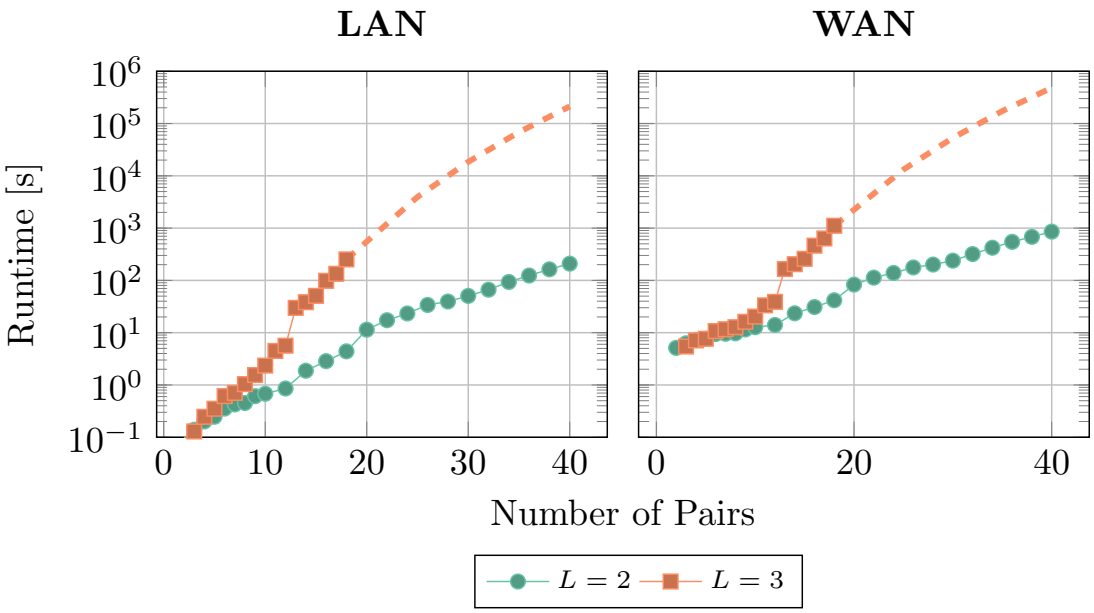

Figure 5.1: Overall run-time of our *EPPKEP* for cycle lengths $L = 2$ and $L = 3$ in both network scenarios. The dashed lines shows the extrapolated power function for $L = 3$. [BHK+22]

In Figure 5.2, we show the overall communication of our protocol for cycle length $L = 2$ and $L = 3$ in mebibyte (MiB). As expected, we observe that the communication overhead for $L = 3$ increases significantly faster than for $L = 2$ such that 40 and 18 pairs have a similar communication overhead for cycle length $L = 2$ and $L = 3$, respectively. Similar to the run-time, we observe a polynomial relationship between the number of participating pairs and the resulting communication cost.

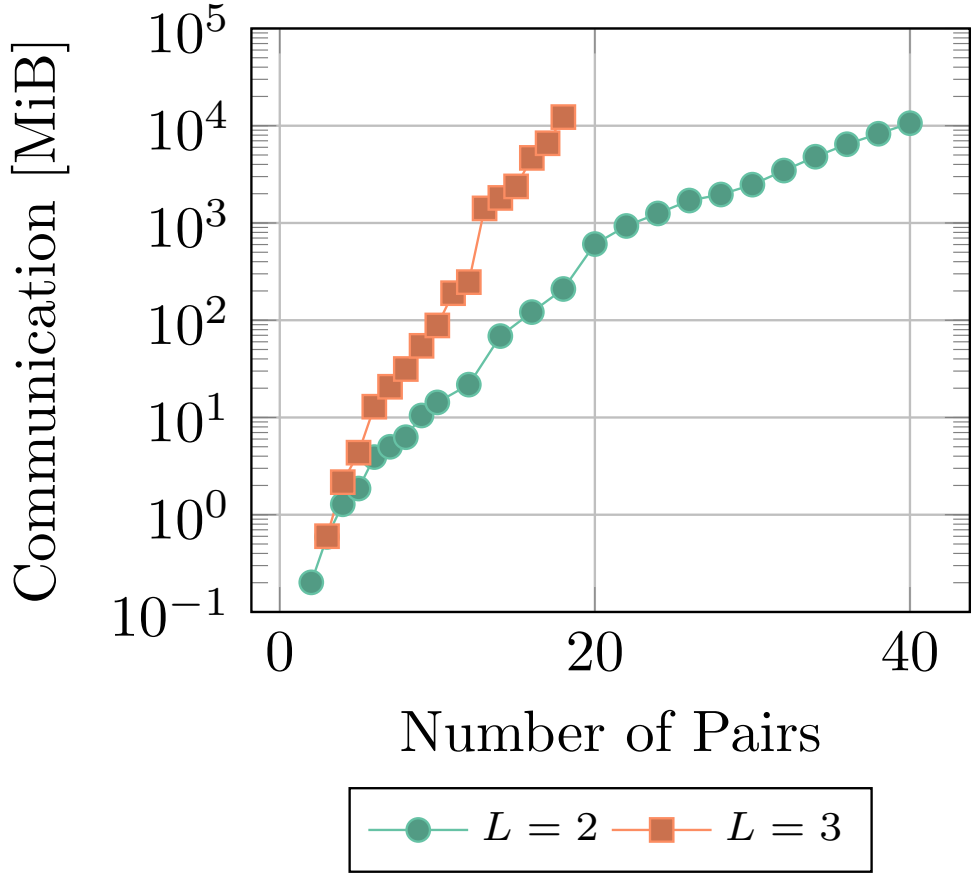

Figure 5.2: Overall communication of our *EPPKEP* for cycle lengths $L = 2$ and $L = 3$.

Figure 5.3 shows the run-times of each part individually for cycle length $L = 2$ in both protocol phases and in both network setting. It shows that the run-times of the computation of the compatibility graph (cf. Section 4.3) and the cycle computation (cf. Section 4.4) become negligible with higher number of pairs compared to the run-times of the cycle evaluation (cf. Section 4.5) and the solution evaluation (cf. Section 4.6). Within the same network setting the setup and online phase of the protocol are in the same order of magnitude.

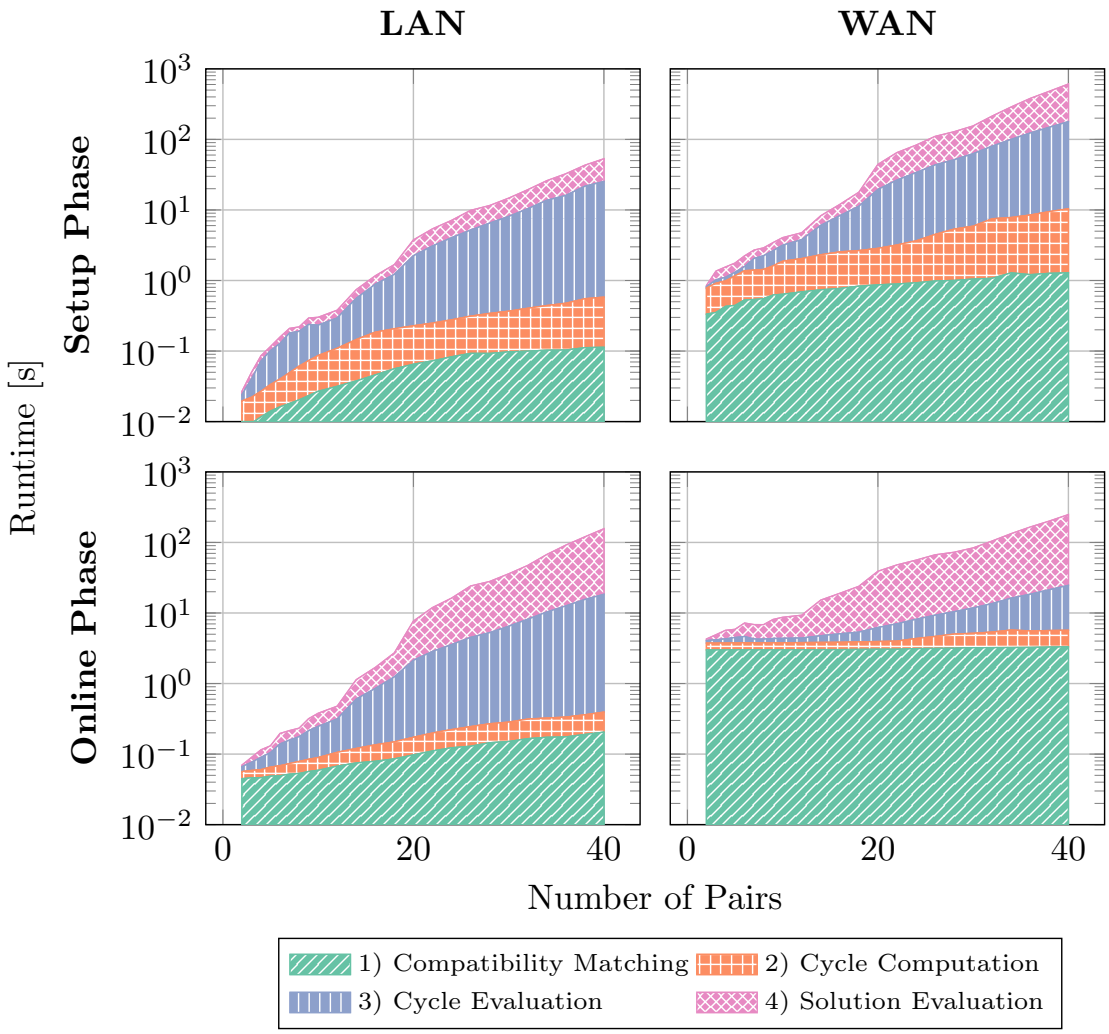

Figure 5.3: Run-time of our *EPPKEP* for $L = 2$ by algorithmic parts, protocol phase, and network setting. [BHK+22]

Figure 5.4 shows the run-times of each part individually for cycle length $L = 3$ in both protocol phases and in both network settings. Similarly to $L = 2$, it shows that the run-times of the computation of the compatibility graph (cf. Section 4.3) and the cycle computation (cf. Section 4.4) become negligible with higher number of pairs compared to the run-times of the cycle evaluation (cf. Section 4.5) and the solution evaluation (cf. Section 4.6). Within the same network setting the setup and online phase of the protocol are in the same order of magnitude. As expected, the run-time increases significantly faster compared to the run-time of $L = 2$. For $L = 3$ with already 18 pairs, we exceed the run-time of $L = 2$ with 40 pairs.

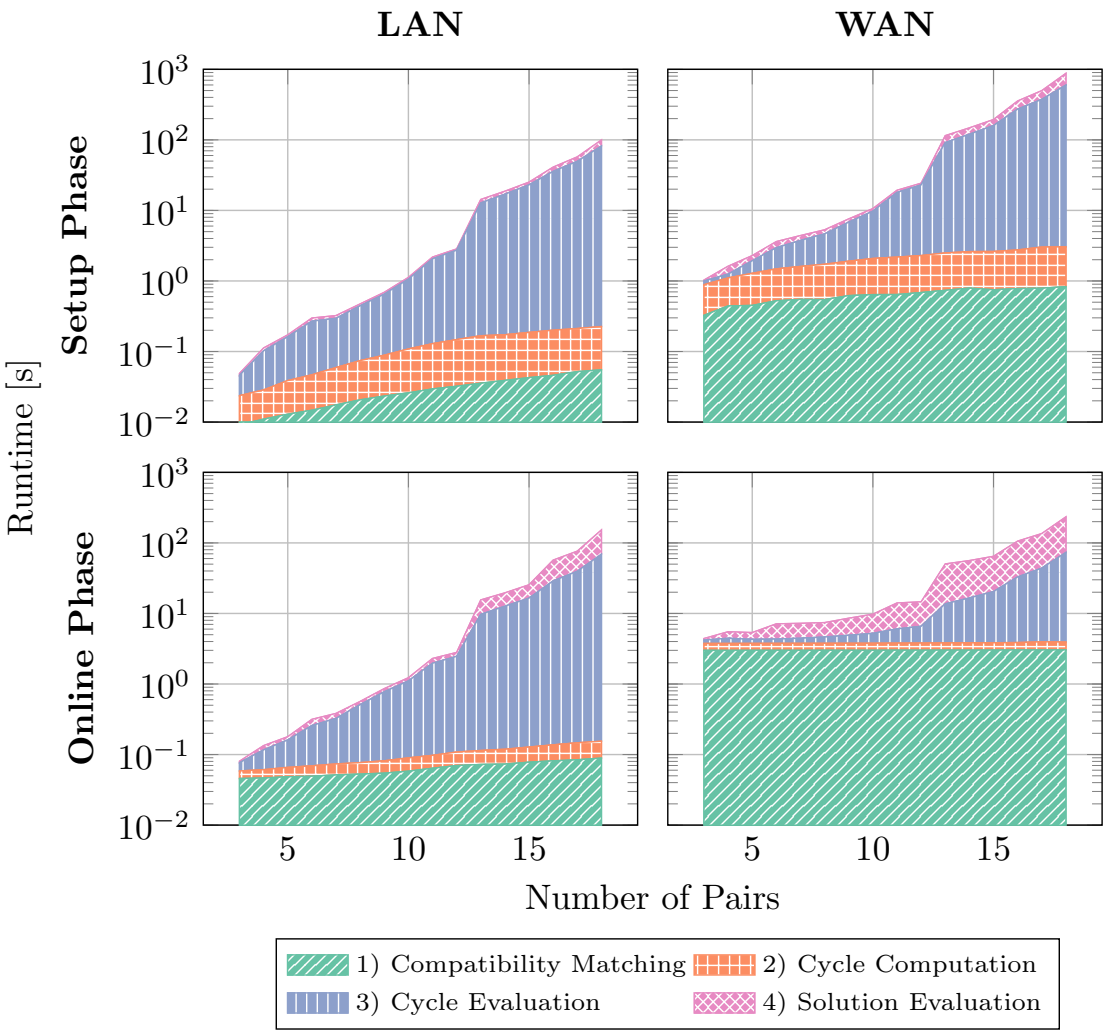

Figure 5.4: Run-time of our *EPPKEP* for $L = 3$ by algorithmic parts, protocol phase, and network setting.

In Figure 5.5, we show the communication of each part for cycle lengths $L = 2$ and $L = 3$. The communication cost for the computation of the compatibility graph and the cycle compute are negligible compared to the communication cost of the cycle and solution evaluation. The strong increase in communication cost from 12 to 13 pairs for $L = 3$ occurs due to memory swapping.

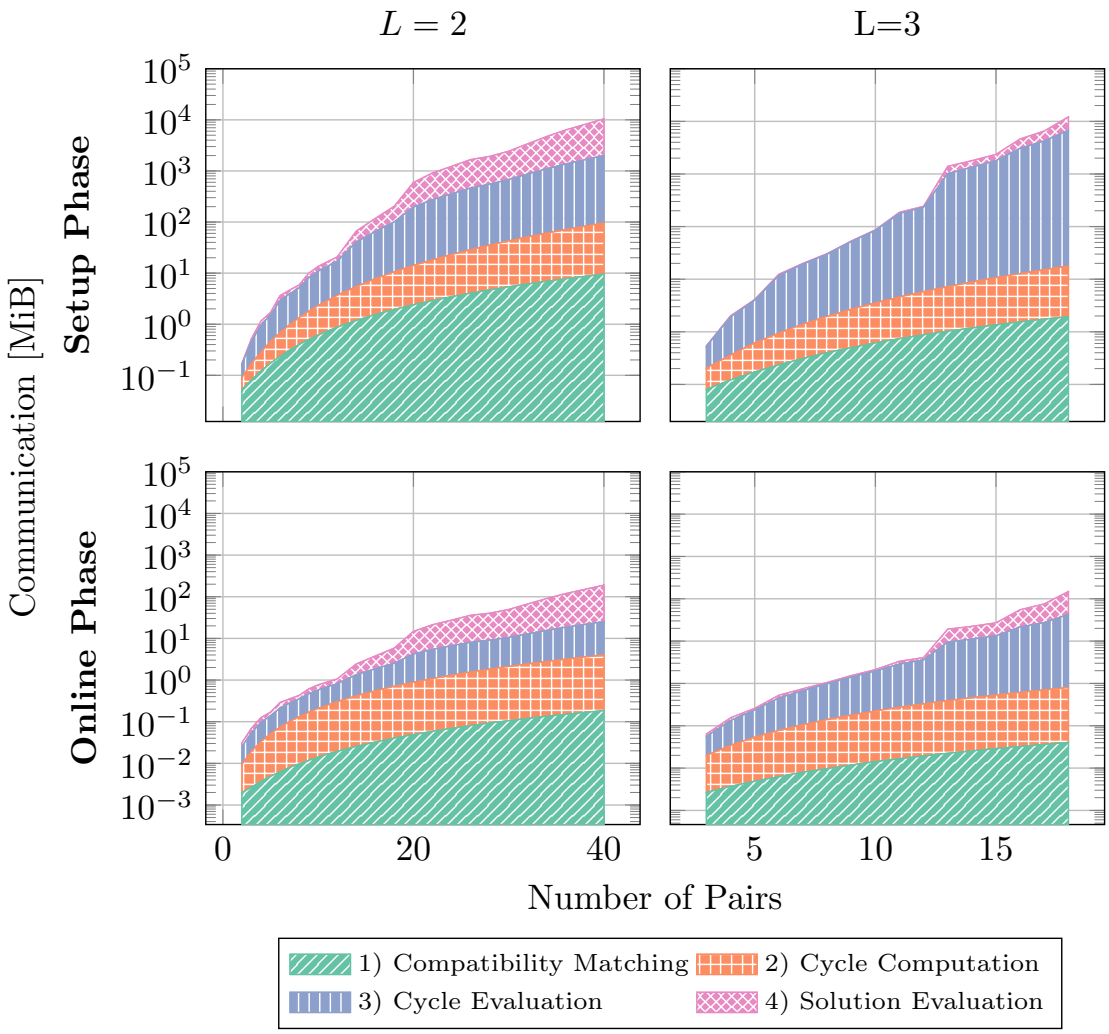

Figure 5.5: Communication of our *EPPKEP* by algorithmic parts, protocol phase, and cycle lengths $L = 2$ and $L = 3$.

To conclude, the run-time performance and the communication show a polynomial relationship with the number of participating pairs. Additionally, the fourth part of our protocol depends on the number of cycles that were found during the third part. Thus, the run-time and the communication can change significantly. Especially, a high occurrence of cycles further exhausts the RAM usage during the fourth part, which could limit the number of participating pairs. Despite the high RAM usage, our *EPPKEP* shows to have real-world applicable run-time performance and communication cost.

5.3 Comparison to State-of-the-Art

In Figure 5.6, we compare the run-time of our protocol for cycle length $L = 2$ and $L = 3$ compared to the two implementations of [BMWM20; BMW22] (cf. Section 3.2). The first implementation [BMWM20] was implemented on top of the SMPC framework SMC-MuSe [NMW13] which implements the *HE* threshold Paillier Cryptosystem. Similar to our protocol, they allow to configure the desired length of cycles, however, they compute all cycles up to the desired cycle length while we only compute cycles of the desired cycle length. They evaluated their protocol using $L = 3$. The second implementation [BMW22] is based on three-party Shamir's Secret Sharing using the MP-SPDZ framework [Kel20]. The authors limit the length of cycles to $L = 2$. The performance data for both implementations are taken from the referenced publications.

Our implementation and the implementation of Breuer et al. 2022 [BMW22] show a polynomial-bound power-law. The run-time of [BMWM20] scales exponentially in the number of pairs. For $L = 3$ and 9 pairs, the maximum number of pairs benchmarked in [BMWM20], our implementation achieves a speedup in run-time by a factor of 29 828. For $L = 2$ and 40 pairs, our implementation shows a 414× speedup in run-time compared to the implementation of [BMW22].

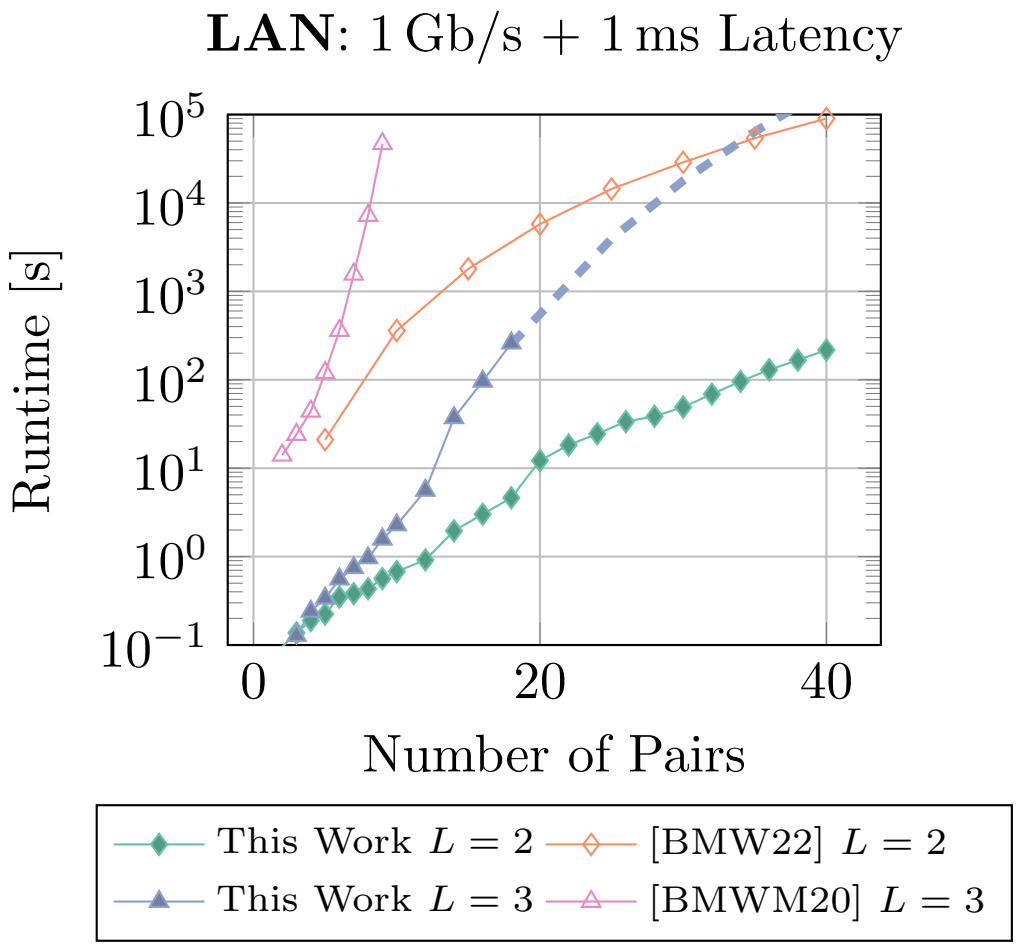

Figure 5.6: Run-time comparison between this work with cycle lengths $L = 2$ and $L = 3$, and Breuer et al. 2022 ($L = 2$) and Breuer et al. 2020 ($L = 3$). [BHK+22]

Figure 5.7 shows a comparison of the communication cost. For $L = 3$, the communication cost increase similarly for up to 7 pairs. However, for 8 and more pairs the communication of the implementation of [BMWM20] increases more rapidly compared to our implementation. In general, our implementation's communication overhead shows to have a polynomial power-law while the implementation of [BMWM20] shows an exponential power-law. For $L = 2$, we observe that for smaller number of pairs there is a transient phase where the communication of [BMW22] increases faster. However, after this transient phase both curves nearly have the same slope.

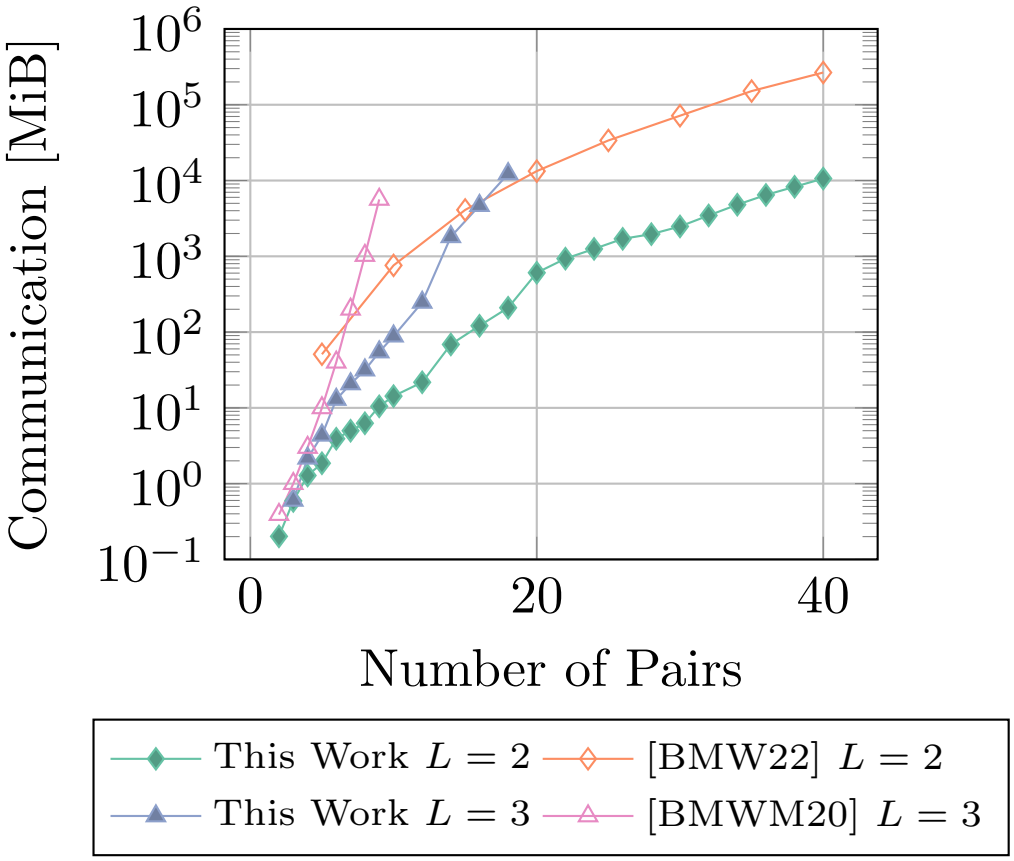

Figure 5.7: Communication comparison between this work with cycle lengths $L = 2$ and $L = 3$, and Breuer et al. 2022 ($L = 2$) and Breuer et al. 2020 ($L = 3$).

We implement additional matching criteria to increase the medical quality of the donor-recipient matching. In Section 5.2, we have seen that the run-time of the medical evaluation is negligible compared to the run-times of the third and fourth part of our protocol. Nevertheless, in Figure 5.8, we compare the run-times of the computation of the compatibility graph for all factors with the run-time of the reduced factors presented in [BMWM20]. We observed 50 HLA (cf. Subsection 2.1.1) in our benchmarks. For smaller number of pairs, the run-time of the full set increases slightly faster than the run-time of the reduced set. For number of pairs larger than 30, both curves assume a nearly similar slope. This shows that using additional medical factors for an increased medical quality of the evaluation leads to an increase of overall run-time but this is increase is manageable.

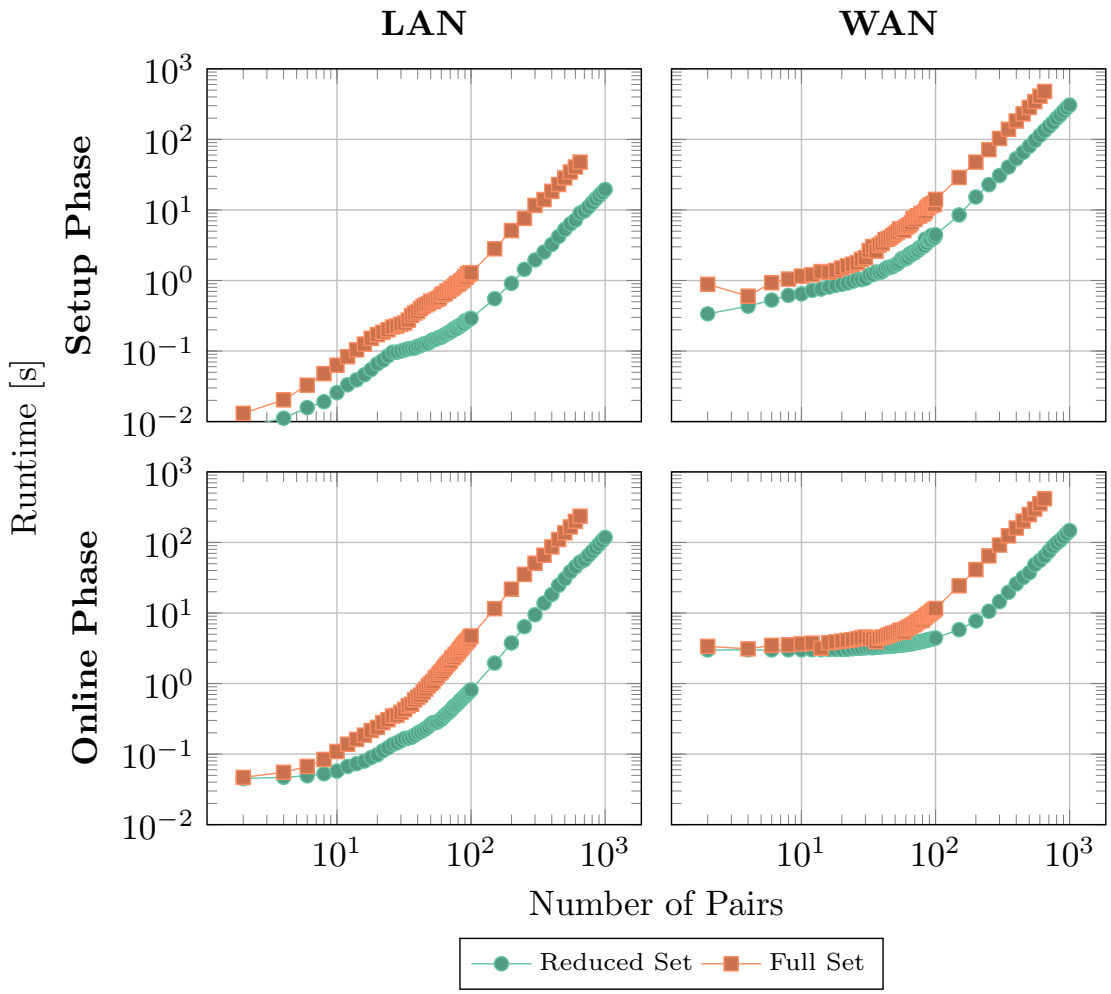

Figure 5.8: Comparison of the compatibility matching (Part 1) performance between the reduced set of criteria [BMWM20] and the full set of this work. The run-times of the remaining protocol parts are independent of the compatibility matching. [BHK+22]

In Figure 5.9, we show the communication of part one of our *EPPKEP* using either all additional factors or the reduced set of medical factors used in [BMWM20]. As expected, the communication in both protocol phases is slightly higher in the full set of criteria than in the reduced set. However, the communication is still in the same order of magnitude which does not affect the overall communication of the protocol.

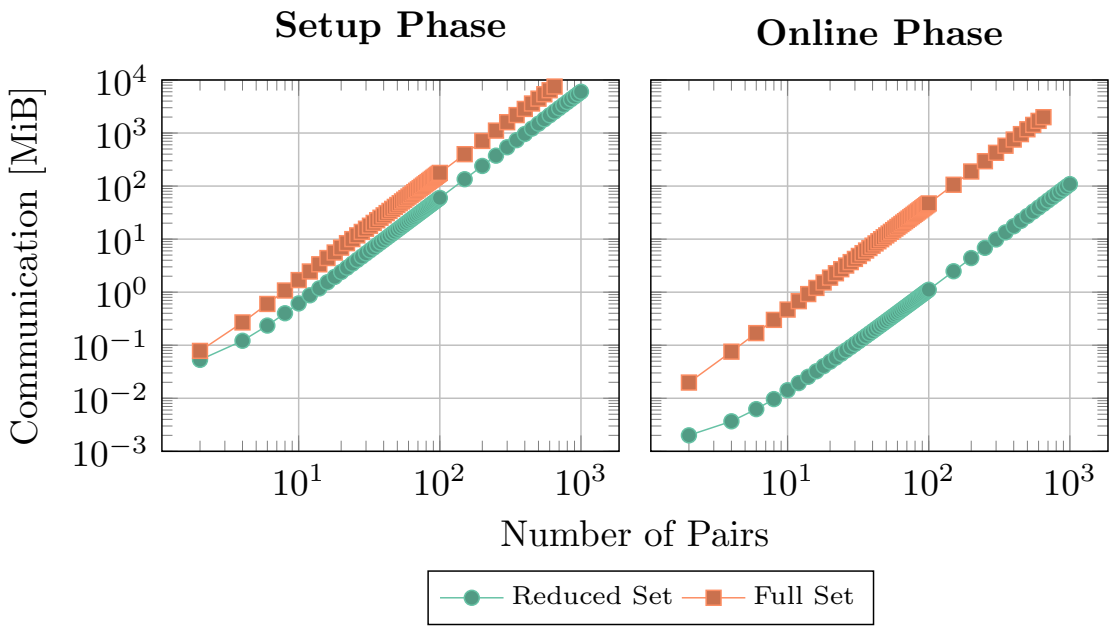

Figure 5.9: Comparison of the communication of the compatibility matching (Part 1) between the reduced set of criteria [BMWM20] and the full set of this work. The communication of the remaining protocol parts are independent of the compatibility matching.

6 Conclusion

We presented an efficient and privacy-preserving solution for the kidney exchange problem. Our Efficient and Privacy-Preserving Kidney Exchange Protocol (*EPPKEP*) provides a highly adaptable medical compatibility matching algorithm by allowing medical experts to choose many parameters regarding the biomedical factors, thus, adapting the compatibility matching algorithm to changes or new advances in the medical field or to specific situational constraints. The compatibility matching protocol is configurable by choosing the considered HLA, as well as the weights of the chosen biomedical factors. Choosing the weight for the biomedical factors allows to highlight certain factors or exclude them entirely. By design, it is also possible to add or remove biomedical factors without re-designing the whole protocol. We also enable flexibility by parameterising the length of cycles. Similar to [BMW22], our protocol can be executed in a dynamic setting in which pairs are periodically drawn of a large pool of donor-recipient pairs. Our protocol allows to do that for cycle lengths $L = 2$ and $L = 3$ in residential network settings in feasible time scales, so that even residential nephrology experts could participate in kidney exchange programmes.

With a run-time of under 4 minutes for 40 pairs at cycle length $L = 2$ (LAN setting) and 1 hour for 25 pairs at cycle length $L = 3$, our *EPPKEP* shows feasible performance for many real-world applications. It is faster than current state-of-the-art works of Breuer et al. 2020 [BMWM20] by a factor of approximately 30 000 for cycle length $L = 3$ and Breuer et al. 2022 [BMW22] by a factor of 400 for cycle length $L = 2$. We also observe slightly reduced communication cost compared to the current state-of-the-art protocols.

The results of this work are summarised in [BHK+22] which is still in submission and an abstract to this work was published on the 33rd Crypto Day [BKMS21].

6.1 Future Work

One shortcoming of our protocol is the high memory consumption during the third and fourth part of our protocol which limits the number of pairs that can participate in our *EPPKEP*. Reducing the memory consumption during those two parts is an interesting direction for future work. Similarly, looking at internal batch processing of graph clusters and the employment of space-optimised data structures might be another direction for improvement. For

part 3, the cycle evaluation, it might be interesting to explore the use of SMPC-based graph analysis for breadth-first search [TAF+21], which scales linearly in the number of vertices, i.e., in our application the number of pairs. Further, instead of naive recursion, it might be interesting to consider the use of shortest-path algorithms to find cycles. For instance, the Floyd-Warshall algorithm allows for computing the shortest distance between two vertices in a weighted graph. Additionally, it might be interesting to explore privacy-preserving Integer Linear Programming or Linear Programming approaches for kidney exchange similar to this non privacy-preserving approach [CKG+20] to further increase the robustness of the solution.
Another shortcoming is that the developed software is designed as a prototype and does not implement widespread medical standards, e.g., **HL7 FHIRE R4**[1], audit- and authentication capabilities, integration in medical research pipelines, creation of deployment packages, and lastly fully (legal) documentation. These aspects must be pursued since they are required for real-world adoption.
Even though we reduce communications compared to the current state-of-the-art [BMWM20; BMW22], our communication cost are still too high to use our protocol in metered or cell data connections. Thus, reducing our communication cost might also be an interesting challenge for future research.
Additionally, more medical research is needed for intersex participants in kidney exchange.

All things considered, this work represents a new state-of-the-art for privacy-preserving kidney exchange protocol. We are certain that increasing the privacy of patients' sensitive data holds merit and we hope that our approach might contribute to reduced organ donation waiting times.

[1] https://www.hl7.org/fhir/R4/

List of Figures

1.1 Cyclic exchange of donors in kidney exchange programmes. 2

2.1 Example of a directed and weighted graph 14
2.2 The cycle in the example graph in Figure 2.1. 15

4.1 High-level overview of our *EPPKEP*. 23
4.2 The ideal functionality for a privacy-preserving *KEP* [BHK+22]. 25

5.1 Overall run-time of our *EPPKEP* for cycle lengths $L = 2$ and $L = 3$ in both network scenarios. The dashed lines shows the extrapolated power function for $L = 3$. [BHK+22] 54
5.2 Overall communication of our *EPPKEP* for cycle lengths $L = 2$ and $L = 3$. . 55
5.3 Run-time of our *EPPKEP* for $L = 2$ by algorithmic parts, protocol phase, and network setting. [BHK+22] 56
5.4 Run-time of our *EPPKEP* for $L = 3$ by algorithmic parts, protocol phase, and network setting. 57
5.5 Communication of our *EPPKEP* by algorithmic parts, protocol phase, and cycle lengths $L = 2$ and $L = 3$. 58
5.6 Run-time comparison between this work with cycle lengths $L = 2$ and $L = 3$, and Breuer et al. 2022 ($L = 2$) and Breuer et al. 2020 ($L = 3$). [BHK+22] . 60
5.7 Communication comparison between this work with cycle lengths $L = 2$ and $L = 3$, and Breuer et al. 2022 ($L = 2$) and Breuer et al. 2020 ($L = 3$). 61
5.8 Comparison of the compatibility matching (Part 1) performance between the reduced set of criteria [BMWM20] and the full set of this work. The run-times of the remaining protocol parts are independent of the compatibility matching. [BHK+22] 62
5.9 Comparison of the communication of the compatibility matching (Part 1) between the reduced set of criteria [BMWM20] and the full set of this work. The communication of the remaining protocol parts are independent of the compatibility matching. 63

List of Tables

2.1 *HLA* split antigens assessed for determining compatibility. 6
2.2 ABO compatibility according to [Blu21] 7
2.3 Garbled AND Gate 12

4.1 Overview of the attributes of a donor that are required to determine compatibility. 26
4.2 Overview of the attributes of a recipient that are required to determine compatibility. 27
4.3 Encoding of the different blood groups. 31
4.4 Complexity Assessment of Part 1 – Compatibility Matching 35
4.5 Complexity Assessment of Part 2 – Cycle Computation 38
4.6 Complexity Assessment of Phase 3 – Cycle Evaluation 45
4.7 Complexity Assessment of Part 4 – Solution Evaluation 50
4.8 Total Complexity Assessment 51
4.9 Complexities of Current State-of-the-Art 51

A.1 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the aggregated total costs [BHK+22]. 77
A.2 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the individual costs of Part 1 (Compatibility Matching) [BHK+22]. 78
A.3 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the individual costs of Part 2 (Cycle Computation) [BHK+22]. 79
A.4 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the individual costs of Part 3 (Cycle Evaluation) [BHK+22]. 80
A.5 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 2$. This table contains individual costs of Part 4 (Solution Evaluation) [BHK+22]. 81

A.6 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the aggregated total costs [BHK+22]. 82
A.7 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 1 (Compatibility Matching) [BHK+22]. 83
A.8 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 2 (Cycle Computation) [BHK+22]. 84
A.9 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 3 (Cycle Evaluation) [BHK+22]. 85
A.10 Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 4 (Solution Evaluation) [BHK+22]. 86
A.11 Comparison of the setup and online run-times of *EPPKEP* for the reduced medical factor compatibility matching and the full set in the two main networking configurations A: LAN + 10 Gb/s, C: WAN [BHK+22]. 87

List of Abbreviations

ABMR Antibody Mediated Rejection

EPPKEP Efficient and Privacy-Preserving Kidney Exchange Protocol

HE Homomorphic Encryption

HLA Human Leukocyte Antigens

KEP Kidney Exchange Problem

MHC Major Histocompatibility Complex

SIMD Single Instruction Multiple Data

SMPC Secure Multi-Party Computation

TTP Trusted Third Party

Bibliography

[ABL+04] M. ATALLAH, M. BYKOVA, J. LI, K. FRIKKEN, M. TOPKARA. **"Private Collaborative Forecasting and Benchmarking"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2004.

[AJL+02] B. ALBERTS, A. JOHNSON, J. LEWIS, M. RAFF, K. ROBERTS, P. WALTER. **"The Adaptive Immune System."** In: *Molecular biology of the cell 4th Edition*. Garland Science, 2002.

[ALR+17] V. B. ASHBY, A. B. LEICHTNAM, M. A. REES, P. X.-K. SONG, M. BRAY, W. WANG, J. D. KALBFLEISCH. **"A Kidney Graft Survival Calculator that accounts for Mismatches in Age, Sex, HLA, and Body Size"**. In: *Clinical Journal of the American Society of Nephrology*. American Society of Nephrology (ASN), 2017.

[BCF+14] J. BRINGER, H. CHABANNE, M. FAVRE, A. PATEY, T. SCHNEIDER. **"GSHADE: faster Privacy-Preserving Distance Computation and Biometric Identification"**. In: *Information Hiding and Multimedia Security (IH&MMSec)*. ACM, 2014.

[BCS21] L. BRAUN, R. CAMMAROTA, T. SCHNEIDER. **"A Generic Hybrid 2PC Framework with Application to Private Inference of Unmodified Neural Networks (Extended Abstract)"**. In: *Privacy in Machine Learning Workshop (NeurIPS 2021)*. 2021.

[Bet21] BETTERHEALTH. **"Immune System Explained"**. https://www.betterhealth.vic.gov.au/health/conditionsandtreatments/immune-system. Accessed: 2021-05-29. 2021.

[Bet22] BETTERHEALTH. **"Dialysis"**. https://www.healthline.com/health/dialysis. Accessed: 2022-03-11. 2022.

[BFS+17] G. A. BÖHMIG, J. FRONEK, A. SLAVCEV, G. F. FISCHER, G. BERLAKOVICH, O. VIKLICKY. **"Transplant International"**. In: Wiley Online Library, 2017.

[BGM+20] P. BIRÓ, M. GYETVAI, R. S. MINCU, A. POPA, U. VERMA. **"IP Solutions for International Kidney Exchange Programmes"**. In: *Central European Journal of Operations Research 29*. Springer, 2020, pp. 403–423.

[BHA+18] P. Biró, B. Hasse-Kromwijk, T. Andersson, E. I. Ásgeirsson, T. Baltesová, I. Boletis, C. Bolotinha, G. Bond, G. Böhmig, L. Burnapp, K. Cechlárová, P. D. Ciaccio, J. Fronek, K. Hadaya, A. Hemke, C. Jacquelinet, R. Johnson, R. Kieszek, D. R. Kuypers, R. Leishman, M.-A. Macher, D. Manlove, G. Menoudakou, M. Salonen, B. Smeulders, V. Sparacino, F. Spieksma, M. O. S. Valentín, N. Wilson, J. v. d. Klundert. **"Building Kidney Exchange Programmes in Europe—An Overview of Exchange Practice and Activities"**. In: *Transplantation*. The Transplantation Society, 2018.

[BHK+22] T. Birka, K. Hamacher, T. Kussel, H. Möllering, T. Schneider. **"SPIKE: Secure and Private Investigation of the Kidney Exchange problem"**. In: 2022.

[BKM+21] P. Biró, J. v. d. Klundert, D. Manlove, W. Pettersson, T. Andersson, L. Burnapp, P. Chromy, P. Delgado, P. Dworczak, B. Haase, A. Hemke, R. Johnson, X. Klimentova, D. Kuypers, A. N. Costa, B. Smeulders, F. Spieksma, M. O. Valentín, A. Viana. **"Modelling and optimisation in european kidney exchange programmes"**. In: *European Journal of Operational Research*. Elsevier, 2021.

[BKMS21] T. Birka, T. Kussel, H. Möllering, T. Schneider. **"Efficient and Practical Privacy-Preserving Kidney Exchange Problem Protocol"**. In: 2021.

[Blu21] Blutspenden. **"Rund ums Blut"**. https://www.blutspenden.de/rund-ums-blut/blutgruppen/. Accessed: 2021-05-29. 2021.

[BMW22] M. Breuer, U. Meyer, S. Wetzel. **"Privacy-Preserving Maximum Matching on General Graphs and its Application to Enable Privacy-Preserving Kidney Exchange"**. In: *Conference on Data and Application Security and Privacy (CODASPY)*. ACM, 2022.

[BMWM20] M. Breuer, U. Meyer, S. Wetzel, A. Mühlfeld. **"A Privacy-Preserving Protocol for the Kidney Exchange Problem"**. In: *Workshop on Privacy in the Electronic Society (WPES)*. ACM, 2020.

[CCA13] P. Cruz-Tapias, J. Castiblanco, J.-M. Anaya. **"Chapter 10 major Histocompatibility Complex: Antigen Processing and Presentation"**. In: *Autoimmunity: From Bench to Bedside*. Center for Autoimmune Diseases Research, 2013.

[CH10] O. Catrina, S. d. Hoogh. **"Secure Multiparty Linear Programming Using Fixed-Point Arithmetic"**. In: *Proceedings of the 15th European Symposium on Research in Computer Security*. ACM, 2010.

[Cho07] S. Y. Choo. **"The HLA System: Genetics, Immunology, Clinical Testing, and Clinical Implications"**. In: *Yonsei Medical Journal*. Yonsei University College of Medicine, 2007.

[CKG+20] M. Carvalho, X. Klimentova, K. Glorie, A. Viana, M. Constantino. **"Robust Models for The Kidney Exchange Problem"**. In: *Informs Journal on Computing*. INFORMS, 2020.

[DK11] J. DREIER, F. KERSCHBAUM. **"Practical Secure and Efficient Multiparty Linear Programming Based on Problem Transformation"**. In: *Technical Report] IACR Cryptology ePrint Archive*. HAL, 2011.

[DSZ15] D. DEMMLER, T. SCHNEIDER, M. ZOHNER. **"ABY – A Framework for Efficient Mixed-Protocol Secure Two-Party Computation"**. In: *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2015.

[EHB+03] A. E. EL-AGROUDY, N. A. HASSAN, M. A. BAKR, M. A. FODA, A. A. SHOKEIR, A. B. S. EL-DEIN. **"Effect of Donor/recipient Body Weight Mismatch on Recipient and Graft Outcome in Living-Donor Kidney Transplantation"**. In: *American Journal of Nephrology*. American Society of Nephrology, 2003.

[Eur21a] EUROTRANSPLANT. **"About us"**. https://www.eurotransplant.org/about-eurotransplant/eurotransplants-aims/. Accessed: 2021-08-09. 2021.

[Eur21b] EUROTRANSPLANT. **"History and timeline"**. https://www.eurotransplant.org/about-eurotransplant/history-and-timeline/. Accessed: 2021-05-10. 2021.

[Eur21c] EUROTRANSPLANT. **"Kidney"**. https://www.eurotransplant.org/organs/kidney/. Accessed: 2021-08-09. 2021.

[Eur21d] EUROTRANSPLANT. **"Recipients: Eurotransplant Manual Chapter 10"**. https://www.eurotransplant.org/wp-content/uploads/2020/01/H10-Histocompatibility.pdf. Accessed: 2021-08-06. 2021.

[Eur21e] EUROTRANSPLANT. **"Recipients: Eurotransplant Manual Chapter 4"**. https://www.eurotransplant.org/wp-content/uploads/2020/01/H4-Kidney.pdf. Accessed: 2021-06-17. 2021.

[Eur22a] EUROTRANSPLANT. **"Active Kidney-Only Waiting List (at year end) in 2021, by Country, by Characteristics"**. https://statistics.eurotransplant.org/index.php?search_type=waiting+list&search_organ=&search_region=by+country&search_period=2021&search_characteristic=&search_text=. Accessed: 2022-03-04. 2022.

[Eur22b] EUROTRANSPLANT. **"Deceased Kidney Donors Used in 2021, by Donor Country, by Characteristic"**. https://statistics.eurotransplant.org/index.php?search_type=donors&search_organ=&search_region=by+country&search_period=2021&search_characteristic=&search_text=. Accessed: 2022-03-04. 2022.

[FGHW14] M. K. FUNG, B. J. GROSSMAN, C. D. HILLYER, C. M. WESTHOFF. **"ABO, H, and Lewis Blood Groups and Structurally Related Antigens"**. In: *Technical Manual 18TH Edition*. Association for the Advancement of Blood & Biotherapies(AABB), 2014.

[FPS00] P.-A. FOUQUE, G. POUPARD, J. STERN. **"Sharing Decryption in the Context of Voting and Lotteries"**. In: *Financial Cryptography*. Springer, 2000.

[GMW87] O. GOLDREICH, S. MICALI, A. WIGDERSON. **"How to Play any Mental Game or a Completeness Theorem for Protocols with Honest Majority"**. In: *Symposium on Theory of Computing (STOC)*. ACM, 1987.

[JLL+19] K. JÄRVINEN, H. LEPPÄKOSKI, E.-S. LOHAN, P. RICHTER, T. SCHNEIDER, O. TKACHENKO, Z. YANG. **"PILOT: Practical Privacy-Preserving Indoor Localization using Outsourcing"**. In: *EuroS&P*. IEEE Computer Society, 2019.

[Kel20] M. KELLER. **"MP-SPDZ: A Versatile Framework for Multi-Party Computation"**. In: *Computing Classification System (CCS)*. ACM, 2020.

[KKC+05] M. D. KLERK, K. M. KREIZER, F. H. J. CLASS, M. WITVLIET, B. J. J. M. HAASE-KROMWIJK, W. WEIMAR. **"The Dutch National Living Donor Kidney Exchange Program"**. In: *American Journal of Transplantation*. Wiley Online Library, 2005.

[KR11] S. KAMARA, M. RAYKOVA. **"Secure Outsourced Computation in a Multi-Tenant Cloud"**. In: *IBM Workshop on Cryptography and Security in Clouds*. 2011.

[KS08] V. KOLESNIKOV, T. SCHNEIDER. **"Improved Garbled Circuit: Free XOR Gates and Applications"**. In: *Automata, Languages and Programming*. Springer, 2008.

[KSS14] F. KERSCHBAUM, T. SCHNEIDER, A. SCHRÖPFER. **"Automatic Protocol Selection in Secure Two-Party Computations"**. In: *Applied Cryptography and Network Security*. Springer, 2014.

[LCC+12] W. H. LIM, S. J. CHADBAN, P. CLAYTON, C. A. BUDGEON, K. MURRAY, S. B. CAMPBELL, S. COHNEY, G. R. RUSS, S. P. MCDONALD. **"Human Leukocyte Antigen Mismatches associated with increased Risk of Rejection, Graft Failure, and Death independent of initial Immunosuppression in Renal Transplant Recipients"**. In: *Clinical Transplantation*. Wiley Online Library, 2012.

[LLH+10] C. LEFAUCHEUR, A. LOUPY, G. S. HILL, J. ANDRADE, D. NOCHY, C. ANTOINE, C. GAUTREAU, D. CHARRON, D. GLOTZ, C. SUBERBIELLE-BOISSEL. **"Preexisting Donor-Specific HLA Antibodies Predict Outcome in Kidney Transplantation"**. In: *Journal of American Society of Nephrology*. American Society of Nephrology (ASN), 2010.

[LPT+18] N. LEEAPHORN, J. R. A. PENA, N. THAMCHAROEN, E. V. KHANKIN, M. PAVLAKIS. **"HLA-DQ Mismatching and Kidney Transplant Outcomes"**. In: *Clinical Journal of the American Society of Nephrology*. American Society of Nephrology (ASN), 2018.

[MKA+17] A. J. MILLER, B. A. KIBERD, I. P. ALWAYN, A. ODUTAYO, K. K. TENNANKORE. **"Donor-Recipient Weight and Sex Mismatch and the Risk of Graft Loss in Renal Transplantation"**. In: *Clinical Journal of the American Society of Nephrology*. American Society of Nephrology (ASN), 2017.

[NHK01] M.-C. NGUYEN, K. HEINDL-RUSAI, L. KALTENEGGER. **"Evaluation of HLA typing data and transplant outcome in pediatric renal transplantation"**. In: 20201.

[NIK+11] I.-S. A. NTOKOU, A. G. INIOTAKI, E. N. KONTOU, M. N. DAREMA, M. D. APOSTOLAKI, A. G. KOSTAKIS, J. N. BOLETIS. **"Long-Term Follow Up for Anti-HLA Donor Specific Antibodies Postrenal Transplantation: High Immunogenicity of HLA Class II Graft Molecules"**. In: *Transplant International*. Wiley Online Library, 2011.

[NMW13] G. NEUGEBAUER, U. MEYER, S. WETZEL. **"SMC-MuSe: A framework for Secure Multi-Party Computation on MultiSets"**. In: *Informatik angepasst an Mensch, Organisation und Umwelt*. Springer, 2013.

[OD12] G. OPELZ, B. DÖHLER. **"Association of HLA Mismatch with Death with a Functioning Graft after Kidney Transplantation: a Collaborative Transplant Study Report"**. In: *American Journal of Transplantation*. Wiley-Blackwell, 2012.

[Ope97] G. OPELZ. **"Impact of HLA Compatibility on Survival of Kidney Transplants from Unrelated Live Donors"**. In: *Transplantation*. The Transplantation Society, 1997.

[OPS13] J. OWEN, J. PUNT, S. STRANFORD. **"The Major Histocompatibility Complex and Antigen Presentation"**. In: *Kuby Immunology (Seventh Edition)*. W. H. Freeman and Company, 2013.

[PBS12] P. PULLONEN, D. BODANOV, T. SCHNEIDER. **"The Design and Implementation of a Two-Party Protocol Suite for SHAREMIND 3"**. In: Technical report, CYBERNETICA Institute of Information Security, 2012.

[PC80] U. PAPE, D. CONRADT. **"Maximales Matching in Graphen"**. In: *Ausgewählte Operations Research Software in FORTRAN*. R. Oldenbourg Verlag, 1980.

[PCCS18] L. PANSART, H. CAMBAZARD, N. CATUSSE, G. STAUFFER. **"Kidney Exchange Problem: Models and Algorithms"**. In: *congrès annuel de la Société Francaise de Recherche Opérationelle et d'Aide á la Décision (Le congrès annuel - ROADEF)*. ROADEF, 2018.

[PSSY21] A. PATRA, T. SCHNEIDER, A. SURESH, H. YALAME. **"ABY2.0: Improved Mixed-Protocol Secure Two-Party Computation"**. In: *30. USENIX Security Symposium (USENIX Security'21)*. USENIX, 2021.

[SCM+14] C. SANTOS, R. COSTA, J. MALHEIRO, S. PEDROSO, M. ALMEIDA, L. S. MARTINS, L. DIAS, S. TAFULO, A. C. HENRIQUES, A. CABRITA. **"Kidney Transplantation across a Positive Crossmatch: a Single-Center Experience"**. In: *Transplantation Proceedings*. Elsevier, 2014.

[TAF+21] TOSHINORI, ARAKI, J. FURUKAWA, K. OHARA, B. PINKAS, H. ROSEMARIN, H. TSUCHIDA. **"Secure Graph Analysis at Scale"**. In: *Proceedings of the 2021 SIGSAC Conference on Computer and Communications Security*. ACM, 2021, pp. 610–629.

[Tof09] T. TOFT. **"Solving Linear Programs Using Multiparty Computation"**. In: *Financial Cryptography and Data Security*. Springer, 2009.

[WB18] A. E. d. WEERD, M. G. H. BETJES. **"ABO-Incompatible Kidney Transplant Outcomes: A Meta-Analysis"**. In: *Clinical Journal of the American Society of Nephrology*. American Society of Nephrology (ASN), 2018.

[WSB+00] J. WAISER, M. SCHREIBER, K. BUDDE, L. FRITSCHE, T. BÖHLER, I. HAUSER, H.-H. NEUMAYER. **"Age-Matching in Renal Transplantation"**. In: *Nephrology Dialysis Transplantation*. Oxford University Press, 2000.

[Yao82] A. C. YAO. **"Protocols for Secure Computations"**. In: *Symposium on Foundations of Computer Science*. IEEE Computer Society, 1982.

[Yao86] A. C.-C. YAO. **"How to Generate and Exchange Secrets"**. In: *Foundations of Computer Science (FOCS'86)*. IEEE Computer Society, 1986, pp. 162–167.

[ZCH+12] J.-Y. ZHOU, J. CHENG, H.-F. HUANG, Y. SHEN, Y. JIANG, J.-H. CHEN. **"The Effect of Donor-Recipient Sex Mismatch on Short- and Long-Term Graft Survival in Kidney Transplantation: a Systematic Review and Meta-Analysis"**. In: *Clinical Transplantation*. Wiley Online Library, 2012.

[ZRE15] S. ZAHUR, M. ROSULEK, D. EVANS. **"Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits using Half Gates"**. In: *Advances in Cryptology - EUROCRYPT 2015*. Springer, 2015.

A Appendix

A.1 Full Benchmarks

In Tables A.1 to A.5, we provide the full benchmarks for the communication overhead and for the setup and online phase in all three described network settings (A: LAN + 10 Gb/s, B: LAN + 1 GB/s, C: WAN) for cycle length $L = 2$. Tables A.6 to A.10 show the full benchmarks for cycle length $L = 3$. In Table A.11, we provide the full benchmarks of both the reduced set of medical compatibility factors and the full set of the medical compatibility factors in our two main network settings A and C.

Table A.1: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the aggregated total costs [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Total								
2	0.2	0	0.027	0.027	0.85	0.07	0.068	4.3
3	0.5	0.1	0.05	0.045	1.4	0.09	0.092	4.9
4	1.2	0.1	0.086	0.066	1.6	0.11	0.12	5.7
5	1.7	0.2	0.12	0.081	1.8	0.13	0.14	5.9
6	3.6	0.3	0.16	0.13	2.2	0.2	0.22	7.2
7	4.6	0.4	0.21	0.15	2.7	0.22	0.23	6.8
8	5.8	0.4	0.22	0.19	2.9	0.23	0.24	6.8
9	9.9	0.6	0.29	0.24	3.4	0.32	0.33	8.1
10	13.5	0.8	0.3	0.3	4	0.38	0.38	8.7
12	20.7	1.1	0.38	0.4	4.7	0.48	0.51	9.3
14	66.3	2.4	0.74	0.79	8.3	1.1	1.2	15
16	117.4	3.8	1.2	1.3	12	1.7	1.8	19
18	203.6	5.9	1.7	2	18	2.7	2.7	24
20	592.4	14.7	3.8	4.6	44	7.7	7.6	39
22	910.4	21.2	5.4	6.5	65	12	12	48
24	1,225.3	27.4	7.1	8.4	83	16	16	56
26	1,665.6	35.8	9.8	11	110	24	22	67
28	1,921.5	40.2	12	13	130	28	26	72
30	2,428.2	49.3	15	16	160	36	33	84
32	3,392.7	66.9	19	22	210	47	47	100
34	4,709.2	90.5	26	29	290	67	67	130
36	6,333.6	119.2	33	38	380	91	91	170
38	8,119.2	150.1	43	48	480	120	120	200
40	10,424.9	189.7	53	62	610	160	160	250

Table A.2: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the individual costs of Part 1 (Compatibility Matching) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 1: Compatibility Matching								
2	0	0	0.01	0.01	0.45	0.012	0.012	0.75
3	0.1	0	0.013	0.013	0.56	0.013	0.013	0.75
4	0.2	0	0.015	0.015	0.56	0.014	0.014	0.76
5	0.3	0	0.019	0.018	0.72	0.017	0.016	0.76
6	0.4	0.1	0.024	0.023	0.85	0.019	0.018	0.76
7	0.7	0.1	0.031	0.029	0.87	0.021	0.021	0.76
8	0.9	0.1	0.041	0.038	0.88	0.026	0.023	0.77
9	1.3	0.2	0.05	0.044	0.99	0.028	0.027	0.77
10	1.7	0.2	0.06	0.054	1.2	0.03	0.03	0.78
12	2.8	0.3	0.078	0.073	1.4	0.038	0.036	0.79
14	4.3	0.4	0.11	0.11	1.6	0.045	0.041	0.8
16	6.2	0.5	0.14	0.14	1.8	0.056	0.053	0.82
18	8.6	0.7	0.15	0.15	1.8	0.064	0.063	0.85
20	11.6	0.8	0.16	0.16	2	0.076	0.074	0.88
22	15.3	1	0.18	0.19	2.3	0.088	0.089	0.94
24	19.6	1.2	0.19	0.21	2.7	0.1	0.094	1.2
26	24.6	1.5	0.22	0.25	3.5	0.12	0.12	1.5
28	30.5	1.7	0.25	0.29	4.4	0.12	0.14	1.9
30	37.2	2	0.27	0.33	4.8	0.13	0.15	1.9
32	44.8	2.3	0.3	0.39	6.4	0.15	0.16	2.1
34	53.4	2.6	0.34	0.45	6.5	0.15	0.18	2.5
36	63	3	0.37	0.5	7.3	0.16	0.18	2.3
38	73.7	3.4	0.44	0.58	8.2	0.17	0.2	2.4
40	85.6	3.8	0.47	0.67	9.1	0.19	0.22	2.4

Table A.3: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the individual costs of Part 2 (Cycle Computation) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 2: Cycle Computation								
2	0	0	0.01	0.01	0.45	0.012	0.012	0.75
3	0.1	0	0.013	0.013	0.56	0.013	0.013	0.75
4	0.2	0	0.015	0.015	0.56	0.014	0.014	0.76
5	0.3	0	0.019	0.018	0.72	0.017	0.016	0.76
6	0.4	0.1	0.024	0.023	0.85	0.019	0.018	0.76
7	0.7	0.1	0.031	0.029	0.87	0.021	0.021	0.76
8	0.9	0.1	0.041	0.038	0.88	0.026	0.023	0.77
9	1.3	0.2	0.05	0.044	0.99	0.028	0.027	0.77
10	1.7	0.2	0.06	0.054	1.2	0.03	0.03	0.78
12	2.8	0.3	0.078	0.073	1.4	0.038	0.036	0.79
14	4.3	0.4	0.11	0.11	1.6	0.045	0.041	0.8
16	6.2	0.5	0.14	0.14	1.8	0.056	0.053	0.82
18	8.6	0.7	0.15	0.15	1.8	0.064	0.063	0.85
20	11.6	0.8	0.16	0.16	2	0.076	0.074	0.88
22	15.3	1	0.18	0.19	2.3	0.088	0.089	0.94
24	19.6	1.2	0.19	0.21	2.7	0.1	0.094	1.2
26	24.6	1.5	0.22	0.25	3.5	0.12	0.12	1.5
28	30.5	1.7	0.25	0.29	4.4	0.12	0.14	1.9
30	37.2	2	0.27	0.33	4.8	0.13	0.15	1.9
32	44.8	2.3	0.3	0.39	6.4	0.15	0.16	2.1
34	53.4	2.6	0.34	0.45	6.5	0.15	0.18	2.5
36	63	3	0.37	0.5	7.3	0.16	0.18	2.3
38	73.7	3.4	0.44	0.58	8.2	0.17	0.2	2.4
40	85.6	3.8	0.47	0.67	9.1	0.19	0.22	2.4

Table A.4: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN, and for cycle length $L = 2$. This table contains the individual costs of Part 3 (Cycle Evaluation) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 3: Cycle Evaluation								
2	0.1	0	0.0045	0.005	0.039	0.0097	0.0079	0.31
3	0.3	0	0.018	0.013	0.11	0.018	0.02	0.42
4	0.7	0.1	0.045	0.026	0.12	0.03	0.037	0.54
5	1	0.1	0.068	0.036	0.14	0.04	0.054	0.72
6	2.2	0.1	0.088	0.061	0.2	0.071	0.1	0.86
7	2.9	0.2	0.13	0.08	0.65	0.084	0.1	0.5
8	3.8	0.2	0.13	0.11	0.81	0.098	0.11	0.49
9	6.3	0.3	0.17	0.12	1	0.13	0.14	0.57
10	8.6	0.4	0.15	0.15	1.3	0.16	0.17	0.58
12	13.4	0.5	0.19	0.21	1.7	0.21	0.25	0.62
14	35	0.9	0.41	0.42	3.7	0.48	0.54	0.93
16	57.3	1.3	0.7	0.66	5.7	0.72	0.79	1.2
18	90.2	1.8	1	1	8.6	1.1	1.1	1.5
20	181.2	3.3	2	2	17	2	1.9	2.3
22	255.2	4.4	2.8	2.8	23	2.6	2.5	3
24	332.8	5.3	3.8	3.7	30	3.3	3.3	3.8
26	431.8	6.5	4.9	4.9	39	4.3	4.2	4.7
28	514.4	7.2	6.2	5.8	46	5	4.9	5.4
30	635.1	8.4	7.7	7.3	57	6.2	6.1	6.6
32	815.8	10.4	10	9.3	73	7.8	7.7	8.2
34	1,037.4	12.8	13	12	92	10	9.8	11
36	1,292.4	15.4	16	15	110	12	12	13
38	1,567.8	18	21	18	140	15	15	16
40	1,894.4	21.2	25	22	170	18	18	19

Table A.5: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 2$. This table contains individual costs of Part 4 (Solution Evaluation) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 4: Solution Evaluation								
2	0	0	0.0027	0.0021	0.027	0.0035	0.0036	0.22
3	0.1	0	0.0096	0.0095	0.34	0.013	0.013	0.75
4	0.2	0	0.014	0.013	0.44	0.025	0.025	1.4
5	0.2	0	0.014	0.013	0.46	0.026	0.025	1.4
6	0.7	0.1	0.03	0.027	0.66	0.057	0.055	2.6
7	0.7	0.1	0.029	0.027	0.65	0.057	0.056	2.6
8	0.7	0.1	0.029	0.028	0.66	0.056	0.056	2.6
9	1.8	0.1	0.055	0.05	0.8	0.1	0.098	3.7
10	2.6	0.2	0.068	0.069	0.88	0.13	0.12	4.3
12	3.6	0.2	0.084	0.086	0.95	0.16	0.16	4.9
14	25.9	1.1	0.18	0.23	2.2	0.53	0.51	10
16	52.4	1.9	0.28	0.41	3.7	0.84	0.84	14
18	102.8	3.3	0.46	0.72	6.5	1.5	1.5	18
20	397.1	10.5	1.5	2.3	25	5.5	5.5	33
22	637	15.8	2.3	3.4	38	9.1	9.1	41
24	869.5	20.8	3.1	4.4	49	13	13	48
26	1,205.1	27.7	4.7	6.1	67	20	18	57
28	1,372	31.2	5	6.8	77	23	20	62
30	1,750.6	38.8	6.6	8.4	92	29	26	72
32	2,526	54.1	8.9	12	130	39	39	91
34	3,611.5	75	12	17	190	57	56	120
36	4,970.5	100.6	16	23	260	79	79	150
38	6,469	128.5	21	30	330	100	100	180
40	8,435.3	164.5	28	39	430	140	140	220

Table A.6: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the aggregated total costs [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Total								
3	0.5	0.1	0.049	0.041	1	0.08	0.086	4.4
4	2	0.2	0.11	0.084	1.6	0.13	0.16	5.5
5	4.1	0.3	0.17	0.15	2.3	0.18	0.19	5.4
6	12.4	0.5	0.3	0.24	3.6	0.32	0.31	7.1
7	20	0.7	0.32	0.32	4.4	0.38	0.42	7.2
8	30.5	1.1	0.47	0.42	5.3	0.57	0.55	7.4
9	53	1.5	0.69	0.7	7.6	0.86	0.87	8.6
10	86.2	2.1	1.1	1.1	11	1.2	1.2	9.7
11	186.4	3.3	2.2	2.8	19	2.3	2.8	14
12	242.5	4.1	2.8	18	24	2.8	19	15
13	1,393.3	19.1	14	43	110	16	53	50
14	1,787.2	22.3	19	110	150	20	150	56
15	2,350.3	27	25	–	190	25	–	65
16	4,614.1	54.9	41	–	360	57	–	110
17	6,527.4	75.3	57	–	500	76	–	140
18	12,121.4	148	99	–	880	150	–	230

Table A.7: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 1 (Compatibility Matching) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 1: Compatibility Matching								
3	0.1	0	0.0091	0.0095	0.33	0.046	0.046	3
4	0.1	0	0.011	0.011	0.44	0.047	0.047	3
5	0.2	0	0.013	0.012	0.45	0.048	0.048	3
6	0.2	0	0.015	0.015	0.52	0.049	0.049	3
7	0.3	0	0.017	0.017	0.55	0.051	0.051	3
8	0.4	0	0.021	0.018	0.54	0.053	0.052	3
9	0.5	0	0.023	0.021	0.62	0.054	0.055	3
10	0.6	0	0.026	0.024	0.64	0.058	0.059	3
11	0.7	0	0.029	0.029	0.64	0.064	0.068	3
12	0.9	0	0.032	0.037	0.69	0.07	0.074	3
13	1	0	0.035	0.045	0.74	0.072	0.082	3
14	1.2	0	0.039	0.052	0.79	0.073	0.087	3
15	1.4	0	0.043	–	0.75	0.078	–	3
16	1.5	0	0.046	–	0.78	0.081	–	3
17	1.7	0	0.052	–	0.79	0.084	–	3
18	1.9	0	0.055	–	0.83	0.089	–	3

Table A.8: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 2 (Cycle Computation) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 2: Cycle Computation								
3	0.1	0	0.015	0.014	0.56	0.013	0.013	0.76
4	0.2	0	0.018	0.017	0.66	0.015	0.014	0.76
5	0.4	0	0.026	0.023	0.83	0.017	0.016	0.76
6	0.7	0.1	0.032	0.029	0.96	0.02	0.018	0.77
7	1.1	0.1	0.042	0.039	1.1	0.022	0.021	0.77
8	1.6	0.1	0.054	0.047	1.2	0.024	0.022	0.77
9	2.2	0.2	0.065	0.061	1.3	0.027	0.024	0.77
10	2.9	0.2	0.083	0.077	1.4	0.032	0.03	0.78
11	3.8	0.3	0.1	0.11	1.5	0.033	0.036	0.78
12	4.9	0.3	0.12	0.14	1.6	0.039	0.045	0.79
13	6.2	0.4	0.13	0.15	1.8	0.042	0.053	0.8
14	7.6	0.4	0.13	0.18	1.8	0.046	0.067	0.81
15	9.3	0.5	0.14	–	1.9	0.05	–	0.82
16	11.2	0.6	0.15	–	2	0.056	–	0.83
17	13.4	0.7	0.16	–	2.2	0.062	–	0.93
18	15.8	0.8	0.17	–	2.2	0.066	–	0.89

Table A.9: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 3 (Cycle Evaluation) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 3: Cycle Evaluation								
3	0.3	0	0.022	0.015	0.11	0.018	0.024	0.43
4	1.6	0.1	0.073	0.046	0.15	0.055	0.078	0.69
5	3.4	0.2	0.12	0.11	0.65	0.095	0.11	0.54
6	10.8	0.4	0.22	0.17	1.5	0.19	0.19	0.57
7	17.9	0.6	0.24	0.24	2.1	0.25	0.3	0.69
8	27.8	0.8	0.37	0.33	2.9	0.43	0.42	0.86
9	49.1	1.2	0.56	0.58	4.9	0.69	0.71	1.1
10	80.6	1.7	0.95	0.93	7.7	1	0.98	1.4
11	172.4	2.6	1.9	2.6	16	1.9	2.4	2.2
12	227.3	3.3	2.6	15	21	2.4	12	2.8
13	1,005.9	9.1	13	36	89	9.6	28	9.9
14	1,329.9	10.7	17	83	120	13	69	13
15	1,773.8	12.8	23	–	160	16	–	17
16	3,058.3	20.8	36	–	270	29	–	29
17	4,213.3	26.5	49	–	370	39	–	40
18	6,735.8	42	81	–	590	69	–	69

Table A.10: Comparison of the communication costs and setup and online run-times of *EPPKEP* for the three networking configurations A: LAN + 10 Gb/s, B: LAN + 1 Gb/s, C: WAN and for cycle length $L = 3$. This table contains the individual costs of Part 4 (Solution Evaluation) [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]			Online Phase [s]		
Pairs	Setup	Online	A	B	C	A	B	C
Part 4: Solution Evaluation								
3	0	0	0.0027	0.0027	0.032	0.0036	0.0038	0.22
4	0.1	0	0.01	0.011	0.37	0.017	0.017	1.1
5	0.1	0	0.0099	0.0098	0.35	0.018	0.017	1.1
6	0.7	0.1	0.029	0.027	0.65	0.057	0.056	2.8
7	0.7	0.1	0.028	0.026	0.64	0.057	0.057	2.8
8	0.7	0.1	0.028	0.026	0.66	0.057	0.055	2.8
9	1.3	0.1	0.039	0.038	0.76	0.083	0.082	3.7
10	2.1	0.2	0.06	0.054	0.84	0.12	0.12	4.5
11	9.4	0.5	0.12	0.12	1.3	0.33	0.32	8
12	9.4	0.5	0.12	2.4	1.3	0.33	6.8	8
13	380.2	9.6	1.5	7.5	23	5.8	24	37
14	448.5	11.1	1.7	25	28	6.9	83	40
15	565.8	13.6	2.1	–	34	9	–	44
16	1,543.1	33.4	5.3	–	83	28	–	73
17	2,298.9	48.1	7.7	–	120	37	–	92
18	5,367.9	105.2	18	–	280	83	–	160

Table A.11: Comparison of the setup and online run-times of *EPPKEP* for the reduced medical factor compatibility matching and the full set in the two main networking configurations A: LAN + 10 Gb/s, C: WAN [BHK+22].

Pairs	Comm. [MiB]		Setup Phase [s]		Online Phase [s]	
Pairs	Setup	Online	A	C	A	C
Reduced Medical Factor Set						
2	0.1	0	0.0084	0.34	0.045	3
50	14.9	0.3	0.14	1.7	0.26	3.4
100	59.8	1.1	0.29	4.4	0.81	4.4
150	134.7	2.5	0.55	8.5	1.9	5.8
200	239.5	4.4	0.91	15	3.8	7.7
250	374.4	6.9	1.4	23	6.4	11
300	539.2	9.9	2	31	9.4	14
350	734	13.4	2.5	41	14	20
400	958.8	17.5	3.2	53	18	26
450	1,213.6	22.1	4.2	65	25	32
500	1,498.3	27.3	5.3	80	31	37
550	1,813.1	33	6.3	96	38	48
600	2,157.8	39.3	7.2	110	45	56
650	2,532.5	46.1	9	130	53	64
Full Medical Factor Set						
2	0.1	0	0.013	0.88	0.047	3.4
50	44	11.8	0.51	4.6	1	5.2
100	177.1	47.1	1.3	14	4.7	12
150	399.2	105.9	2.8	29	12	24
200	710.5	188.3	5.1	48	22	41
250	1,110.9	294.3	7.6	71	35	64
300	1,600.4	423.8	12	100	51	92
350	2,179.1	576.8	14	140	66	120
400	2,846.8	753.4	18	180	86	160
450	3,603.7	953.5	23	230	110	200
500	4,449.6	1,177.2	28	280	140	250
550	5,384.7	1,424.4	35	340	170	300
600	6,408.9	1,695.2	41	410	200	350
650	7,522.2	1,989.5	48	480	240	420

A.2 Fitting Models

We present the models we used to extrapolated the missing measurements. For all network settings we used the general model:

$$f(x) = a \cdot x^b + c$$

The parameters were computed in Matlab 2021a (9.10.0.1602286) with the Trust-Region algorithm with a maximum of 400 iterations.

LAN10:
Coefficients (with 95% confidence bounds):

$$a = 5.437 \cdot 10^{-6}\ (-1.1162 \cdot 10^{-5}, 2.249 \cdot 10^{-5})$$
$$b = 8.484\ (7.392, 9.576)$$
$$c = 1721\ (-3373, 7815)$$

Goodness of fit:

SSE: $6.681 \cdot 10^8$
R-square: 0.9907
Adjusted R-square: 0.9893
RMSE: 7169

LAN1:
Coefficients (with 95% confidence bounds):

$$a = 1.032 \cdot 10^{-5}\ (-9.383 \cdot 10^{-7}, 2.159 \cdot 10^{-5})$$
$$b = 8.285\ (7.906, 8.663)$$
$$c = 329.3\ (-1314, 1973)$$

Goodness of fit:

SSE: $3.889 \cdot 10^7$
R-square: 0.9994
Adjusted R-square: 0.9993
RMSE: 2079

WAN:
Coefficients (with 95% confidence bounds):

$$a = 0.0002595\ (-0.000441, 0.00096)$$
$$b = 7.658\ (6.718, 8.598)$$
$$c = 1.335 \cdot 10^4\ (-8887, 3.3559 \cdot 10^4)$$

Goodness of fit:

SSE: $1.203 \cdot 10^{10}$
R-square: 0.9915
Adjusted R-square: 0.9902
RMSE: $3.042 \cdot 10^{4}$