Fair and Secure Multi-Two Party Computation and Multi Party Fair Exchange

by

Handan Kılınç

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

Koç University

August 25, 2014

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Handan Kılınç

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Alptekin Küpçü (Advisor), Koç University

_____

Prof. Özgür Barış Akan, Koç University

_____

Assist. Prof. Selçuk Baktır, Bahçeşehir University

Date: _____

To my mother, father and brother.

ABSTRACT

Secure computation cannot be fair in general against malicious adversaries, unless a trusted third party (TTP) is involved, or gradual-release type of costly protocols with super-constant rounds are employed. Existing optimistic fair and secure computation protocols with constant rounds are either too costly to arbitrate (e.g., the TTP may need to re-do almost the whole computation), or require the use of electronic payments or bitcoins. Furthermore, most of the existing solutions were proven secure and fair separately, which, we show, may lead to insecurity overall. Therefore, we propose two new framework for secure two-party and multi-party computation that can be applied on top of both circuit based secure computation protocols to make them fair.

We show that our fairness overhead is minimal with these frameworks, compared to all known existing work, and the TTP never learns the inputs or outputs of the computation. Furthermore, our framework for the two party computation makes a protocol fair even in terms of the work performed by the two parties Alice and Bob. We also prove that the frameworks makes a circuit based secure computation protocol fair and secure simultaneously, through one simulator, which guarantees that our fairness extensions do not leak any information.

The framework for fair and secure multi-party computation includes multi-party fair exchange protocol that we designed. Multi-party fair exchange (MFE) is understudied field of research, with practical importance. We examine MFE scenarios where every participant has some item, and at the end of the protocol, either every participant receives every other participant's item, or no participant receives anything. This is a particularly hard scenario, even though it is directly applicable to protocols such as fair and secure multi-party computation (SMPC) or multi-party contract

signing. We analyze the case where a trusted third party (TTP) is optimistically available, although we emphasize that the trust put on the TTP is only regarding the *fairness*, and our protocols preserve the *privacy* of the exchanged items against the TTP.

We construct two *asymptotically optimal* multi-party fair exchange protocols that require a constant number of rounds, in comparison to linear, and $O(n^2)$ messages, in comparison to cubic, where n is the number of participating parties. In one protocol, we enable the parties to efficiently exchange any item that can be efficiently put into a verifiable escrow (e.g., signatures on a contract). In our other protocol, we let the parties exchange any verifiable item, without the constraint that it must be efficiently put into a verifiable escrow (e.g., a file cannot be efficiently verifiably escrowed, but if its hash is known, once obtained, the file can be verified). We achieve this via use of electronic payments, where if an item is not obtained, the payment of its owner will be obtained in return of the item that we sent. We then generalize our protocols to handle any exchange topology efficiently. Our protocols guarantee fairness in its strongest sense: even if all $n-1$ other participants are malicious and colluding, fairness will hold.

ÖZETÇE

Normalde, iki ve çok kişili ortaklaşa hesaplama protokollerinde güvenilir üçüncü bir şahıs olmadan veya çok maliyetli olan kademeli olarak bırakma (gradual-release) protokolleri kullanılmadan adalet sağlamaz. Güvenilir üçüncü şahıs kullanan iki veya çok kişili adil ortaklaşa hesaplama protokollerinde ya güvenilir kişiye çok iş düşüyor ya da elektronik ödeme ile adalet sağlanıyor. Ayrıca bu protokollerin çoğunda adil ve güvenli oldukları ayrı ayrı ispatlanmış. Bu yüzden bu protokoller bazı güvenlik açıklarına yol açabilirler. Bu sorunları çözmek için devre kullanan, iki veya çok sayıda kişi içeren ortaklaşa hesaplama protokollerini adil yapan iki yeni yapı oluşturduk.

Bu iki yeni yapı iki veya çok kişili ortaklaşa hesaplama protokollerine çok az ek yük getiriyor. Güvenilir üçüncü şahıs hesaplamanın girdi ve çıktılarıyla ilgili hiç bir bilgi öğrenmiyor. Ayrıca ikili ortaklaşa hesaplamayı adil yapan yapımız, partiler arasında eşit yük vererek de adaleti sağlamış oluyor. Her iki yapıyla verdiğimiz adil ve güvenli ortaklaşa hesaplama protokollerinin adil ve güvenli olduklarını birlikte göstererek ispatladık (simülatör kullanarak) ve böylece hiç bir güvenlik açığının olmadığını göstermiş olduk.

Çok kişili ortaklaşa hesaplamayı adil yapan yapımız yeni oluşturduğumuz çok kişili adil takas protokolünü kullanmaktadır. Çok kişili adil takas (MFE), çalışılan bir araştırma alandır ve pratikte önemli bir yere sahiptir. Bizim araştırdığımız senaryo her partinin takas etmek istediği bir nesnesinin olduğu ve takas sonunda her partinin istediği nesneyi aldığı ya da hiç bir partinin hiçbir şey alamadığı senaryodur. Güvenilir üçüncü şahısın (TTP) adaleti sağlamak için zorunlu olduğu durumlarda takas protokolüne girdiği durumu tüm protokollerimizde kullandık. Oluşturduğumuz protokoller güvenilir üçüncü şahısa karşı da gizliliği sağlamaktadır. Sabit sayıda $O(n^2)$ mesaj gerektiren (n kişi sayısı) iki çok kişili adil takas protokolü oluşturduk. Protokol-

lerin birinde, kişiler sadece etkin olarak doğruluğunu gösterebildikleri nesneleri takas edebilmektedirler (örn. bir kontratı ortaklaşa imzalama). Diğer protokolümüzde, kişiler etkin olarak ya da etkin olmadan doğruluklarını gösterebildikleri herhangi bir nesneyi takas edebiler. Bunu elektronik ödeme sistemlerinden yaralanarak başardık. Burada eğer bir kişi istediği nesneyi her hangi bir kişiden alamazsa, onun yerine karşı taraftan bir ödeme yapılıyor. Daha sonra bu iki takas protokolünün herhangi bir takas topolojisine adapte edilebileceğini gösterdik. Takas protokollerimiz $n-1$ tane kişi kötü niyetli olsa bile adil olma özelliğini koruyorlar.

ACKNOWLEDGMENTS

I am using this opportunity to express my gratitude to my advisor Asst. Prof Alptekin Küpçü who supported me throughout my master. I am thankful for him aspiring guidance, invaluably constructive criticism and friendly advice during the project work. I am sincerely grateful to him for sharing his truthful and illuminating views on a number of issues.

I would like to thank to TÜBİTAK under the project number 111E019 for their financial support.

I express my warm thanks to members of Koç University, Cryptography Research Group, especially Buket Yüksel, Mina Namazi and Mohammad Etemad for their friendship and help. I also would like to thank my friend Aurélien Gay with his support and help everytime when I feel hopeless about my studies.

Most importantly, I would like to thank my mother, father and brother for their supports and trusts. My mother, Aynur Kılınç and father, Süleyman Kılınç did everything to make their children happy and gave unconditional love and care. My brother, Orhun Kılınç is my best friend all my life and I love him dearly and thank him because he encourages and supports me during all my life.

TABLE OF CONTENTS

List of Tables **xi**

List of Figures **xii**

Nomenclature **xiii**

Chapter 1: Introduction **1**

1.1 Contribution in MFE 4

1.2 Contribution in Fair and Secure Computation 5

Chapter 2: Related Works **7**

2.1 MFE Related Works 7

2.2 Fair and Secure Computation Related Works 8

Chapter 3: Definitions and Preliminaries **12**

3.1 Secure Computation 12

3.2 Fair and Secure Computation 14

3.3 Other Preliminaries 17

3.4 Notation 20

Chapter 4: MFE **22**

4.1 Multi-Party Fair Exchange Protocol (MFE) 22

4.1.1 Resolve 1 26

4.1.2 Resolve 2 26

4.1.3 Resolve 3 27

4.2 All Topologies for MFE 30

4.3 Coin-based Multi-Party Fair Exchange Protocol (CMFE) 32
4.4 Generalization of MFE and CMFE 41
4.5 Analysis of MFE and CMFE 41

Chapter 5: Fair SMPC **44**
5.1 Efficient Fair Secure Multi-Party Computation 44
5.2 Full Simulation Proof of Fair SMPC 46
5.3 Analysis of Fair SMPC: 47

Chapter 6: Fair, Secure 2PC **49**
6.1 Notation for Two Party Computation 49
6.2 Overview of the Fairness Principles 50
6.3 Making a Secure Protocol Fair 55
6.4 Sub Protocols 63
6.4.1 Resolutions with the TTP 63
6.4.2 Proof of Garbled Construction 64
6.4.3 Proof of Output Gates 65
6.4.4 Proof of Correct Randoms 66
6.5 Security of the Protocol 66
6.5.1 Proof Sketch 67
6.5.2 Full Simulation Proof 67
6.6 Importance of Proving Security and Fairness Together 75
6.7 Analysis of Fair and Secure 2PC: 78
Notes to Chapter 6 80

Chapter 7: Conclusion **81**

Bibliography **83**

LIST OF TABLES

4.1 Desired items by each parties in matrix form 30

4.2 Necessary shares for each party to get the desired items that shown in Table 4.1. 31

4.3 Efficiency comparison with previous works in MFE 42

5.1 Comparison of our fair SMPC solution with previous works 48

6.1 Review of random numbers for fair 2PC framework 55

6.2 Garbled Input and Output Gates for an OR gate constructed by Alice. . . 58

6.3 Fake garbled Normal OR gate of Bob 69

6.4 Comparison of our protocol with previous works in fair 2PC 78

LIST OF FIGURES

4.1 MFE Protocol 23
4.2 Matrix representation of the ring topology. 32
4.3 Graph representation of the ring topology. 32
4.4 Matrix representation of a topology. 32
4.5 Graph representation of a topology. 32
4.6 CMFE Protocol 33

NOMENCLATURE

2PC	**2** (Two) **P**arty **C**omputation
SMPC	**S**ecure **M**ulti **P**arty **C**omputation
MFE	**M**ulti-party **F**air **E**xchange
CMFE	**C**oin-based **M**ulti-party **F**air **E**xchange
MPCS	**M**ulti **P**arty **C**ontract **S**igning
TTP	**T**rusted **T**hird **P**arty
OT	**O**blivious **T**ransfer
COT	**C**omitted **O**blivious **T**ransfer

Chapter 1

INTRODUCTION

Secure multi party computation (SMPC) allows parties to compute a functionality together with their private inputs. The special case of this computation is two party computation (2PC) with two parties. Beside privacy of input and output, one of the problem in secure computation is fairness which guarantees that in the end of the computation all parties will receive their outputs or none of them will receive.

Fairness in 2PC extensively studied but yet there is not an efficient solution. On the other hand, multi party computation does not have any feasible solution for fairness. Therefore, we construct a multi-party fair exchange (MFE) protocol that can be used for a SMPC protocol to make it fair. Then, we design a new framework that is independent from multi party computation for two party computation to add fairness property.

Fair and Secure MPC and 2PC: In two-party computation (2PC), Alice and Bob intend to evaluate a shared function with their private inputs. 2PC is called multi-party computation (MPC) when there are more than two parties in the computation. The computation is called secure when the parties do not learn anything beyond what is revealed by the output of the computation. Yao [87] introduced the concept of secure two-party computation and gave an efficient protocol; but this protocol is not secure against malicious parties who try to learn extra information from the computation by deviating from the protocol. Many solutions [76, 86, 58, 66, 14, 49, 32, 10, 57] are suggested to strengthen Yao's protocol against malicious adversaries.

When one considers malicious adversaries, fairness is an important problem. A fair computation should guarantee if a party learns the output of the function *if and*

only if the other parties learn. This problem occurs since in the protocol one party learns the output earlier than the other party; therefore (s)he can abort the protocol after learning the output, before the other party learns it.

There are two main methods of achieving fairness: using gradual release [81, 60, 82, 54] or a trusted third party (TTP) [22, 63, 15, 2]. The gradual release protocols [36, 20, 16] let the parties gradually (bit by bit, or piece by piece) and verifiably reveal the result. Malicious party will have one bit (or piece) advantage if the honest parties start to reveal the result first. Yet, if the malicious party has more computational power, he can abort the protocol earlier and he can learn the result via brute force, while the honest parties cannot. In this case, fairness is not achieved. Another drawback of this approach is that it is expensive since it requires many rounds of communication.

The TTP approach employs a third party that is trusted by the parties. Of course, a simple solution would be to give the inputs to the TTP, who computes the outputs and distributes fairly. In terms of efficiency and feasibility though, the TTP should be used in the *optimistic* model [5], where he gets involved in the protocol *only* when there is a dispute between the parties because one of them is malicious. It is very important to give the TTP minimum possible workload because otherwise the system will have a bottleneck [22]. Another important concern against the TTP is privacy. In an optimistic solution, if there is not dispute, the TTP should not even know a computation took place, and even with a dispute, the TTP should never learn the identities of the parties, or their inputs or outputs.

Another problem regarding fairness in secure computation is the proof methodology. In previous work [22, 60, 81, 46], fairness and security were proven separately. However, it is important to prove them together to ensure that the fairness solution does not leak any information beyond the original secure computation requirements. Therefore, as in the security of the secure computation, there should be ideal/real world simulation (see Section 3.1) that covers both fairness and security. In other words, **the fairness solution should also be simulatable**. Besides, **the simula-**

tor should learn the output only after the fairness is guaranteed.

Multi-party Fair Exchange: An exchange protocol is a protocol that allows two or more parties to exchange items. The exchange protocol is *fair* when the exchange guarantees that every participant gives an item to the other participants *if and only if* he receives the expected item(s). Examples of such exchanges include signing electronic contracts, certified e-mail delivery, and fair purchase of electronic goods over the Internet. In addition, a fair exchange protocol can be adopted by secure two- or multi-party computation protocols [14, 49, 32, 10, 57, 87, 66] to achieve fairness.

Even in two-party fair exchange scenarios, preventing unfairness completely and efficiently without a trusted third party (TTP) is shown to be impossible [40, 77]. The main reason is that one of the parties will be sending the last message of the protocol, regardless of how the protocol looks like, and may choose not to send that message, potentially causing unfairness. In an *optimistic* protocol, the TTP is involved in the protocol *only* when there is such a malicious behavior [5]. However, it is important not to give a lot of work to the TTP, since this can cause a bottleneck. Furthermore, the TTP is required *only* for *fairness*, and should not learn more about the exchange than is required to provide fairness. In particular, **in our protocols, we show that the TTP does *not* learn the *items*** that are exchanged.

Fair exchange with two parties have been extensively studied and efficient solutions [5, 12, 61] have been proposed. On the other hand, the multi-party case does not have efficient and general solutions. Multi-party fair exchange (MFE) can be described based on *exchange topologies*. For example, a *ring topology* describes an MFE scenario where each party receives an item from previous party in the ring [8, 68, 53, 89, 68]. A common scenario with the ring topology is a customer who wants to buy an item offered by a provider. It is considered as a ring topology because the provider gives the item to the customer, the customer sends a payment authorization to his bank, the customer's bank sends the payment to the provider's bank, and finally the provider's bank credits the provider's account.

Ring topology cannot be used in scenarios like contract-signing and secure multi-

party computation (SMPC), since in such scenarios the parties want items from all other parties. In particular, in such settings, **we want that either every participant receives every other participant's item, or no participant receives anything**. This corresponds to the contract being signed only if everyone agrees, or the SMPC output to be revealed only when every participant receives it. We call this kind of topology a *complete topology*. We can think parties as nodes in a complete graph and the edges between parties show the exchange links. The complete topology was researched mostly in the contract-signing setting [47, 11, 45], with one exception [4]. Unfortunately, all these protocols are inefficient compared to ours (see Table 4.3). Since there was no an efficient MFE protocol that achieves the complete topology, the fairness problem in SMPC protocols still could not be completely solved.

1.1 Contribution in MFE

We suggest two new optimistic multi-party fair exchange protocols that guarantee fairness in a complete topology efficiently. The first one is called MFE, and the other one is called CMFE (for Coined-MFE). The difference between MFE and CMFE is the fairness definition. Fairness in MFE means that either all parties receive the items from all the other parties, or none of them receive any item at the end of the protocol. On the other hand, the fairness in CMFE is more relaxed, in the sense that if our item is sent but an item is not received, the coin of the owner of that missing item will be received to compensate for fairness.

- MFE and CMFE both require only $\mathbf{O(n^2)}$ messages and **constant** number of rounds for n parties, being much more efficient than the previous works (see Table 4.3). These are asymptotically **optimal**, since each party should send his item to all the other parties, even in an unfair exchange. Furthermore, *none* of our protocols necessitate a *broadcast*.
- Our protocols **optimally** guarantee fairness (against honest parties) even when $n-1$ out of n parties are malicious and colluding.
- Our protocols have an easy setup phase, which is employed only **once** for ex-

changing **multiple** sets of items, thus improving efficiency even further for *repeated exchanges* among the same set of participants.

- The TTP for fairness in MFE and CMFE is in the *optimistic* model [5]. The TTP has very low workload, since the parties only employ efficient discrete-logarithm-based sigma proofs to show their honesty. More importantly, the TTP does *not* learn any exchanged item, so **privacy against the TTP** is preserved.
- MFE gives a general multi-party fair exchange solution that can be used in many kinds of exchange protocols including signing electronic contracts. We show how to employ our MFE protocol for **any exchange topology**, with the performance improving as the topology gets sparser.
- CMFE is the **first** protocol to enable multiple parties to exchange verifiable items that cannot be efficiently put in a verifiable escrow. For example, files may be verifiable via their hashes, but still there is no efficient way of putting a file into a verifiable escrow. Yet, our CMFE protocol enables very efficient exchange of even files, by relaxing the fairness definition and employing electronic payments.
- We formulate MFE and CMFE as secure multi-party computation protocols. We then **prove** their security and fairness **via ideal-real world simulation**. To the best of our knowledge, no multi-party fair exchange protocol was proven as an SMPC protocol before.

1.2 Contribution in Fair and Secure Computation

The main achievements are two efficient frameworks for making secure two-party and multi-party computation protocols fair, such that it guarantees fairness and security together, and can work on top of circuit based computation protocols.

- We achieve to build a fair SMPC protocol using our MFE protocol and any circuit based SMPC protocol. Besides, we design new framework only for 2PC that can be adopted by any secure two party computation secure against malicious

adversaries, thereby achieving fairness with little overhead.

- Our frameworks employs a trusted third party (TTP) for fairness, in the optimistic model [5] and TTP's load is very light. If there is no dispute, the TTP does not even know a computation took place, and even with a dispute, the TTP never learns the identities of the parties, or their inputs or outputs. So, a semi-honest TTP is enough in our construction to achieve fairness.
- Our framework is also fair about the work done by the parties, since all of them perform almost the same steps in the protocol.
- We give a simple-to-understand ideal world definition to achieve fairness and security together that guarantee that the fairness extension leaking anything about the inputs, and without necessitating a payment system.
- We show how to prove security and fairness together for the protocols that employs our frameworks according to our simple-to-understand ideal world definition.
- We show that the classic way of **proving security and fairness separately is not necessarily secure** (see Section 6.6).
- We compare our frameworks with related fair secure computation works and show that we achieve better efficiency and security.

Chapter 2

RELATED WORKS

2.1 MFE Related Works

Two-party Fair Exchange: Most of the previous work in the fair exchange setting was done on the two-party case. The interesting case is the optimistic case, where there exists a trusted third party (TTP), but the TTP is not involved if both participants are hosnest [5, 4, 8, 7, 72, 6, 61].

Multi-party Fair Exchange: Franklin and Tsudik [42] classified multi-party fair exchange based on the number of items that a participant can exchange and the dispositions of the participants. Asokan et al. [4] described a generic optimistic fair exchange with a general topology. They define a description matrix to represent the topology, and proposed a fair protocol. The parties are restricted to exchange *exchangeable items*, requiring the TTP to be able to replace or revoke the items, greatly decreasing the applicability of their protocol. In addition, broadcast is used to send the items, making the protocol inefficient.

Ring Topologies: Bao et al. [9] proposed an optimistic multi-party fair exchange protocol based on ring topology. In their protocol, one of the participants is the initiator, who starts the first and second phases of the protocol. The initiator is required to contact the TTP to acknowledge the completion of the first phase of the protocol. Thus, firstly, this is not a strictly optimistic protocol, secondly, there is a necessity of trusting the initiator, and thirdly, there is a passive conspiracy problem [42], which means that a dishonest party may conspire with an honest party without the latter's consent.

Later, Gonzales-Deleito and Markowitch [53] solved the malicious initiator problem of Bao et al. [9]. The problem here is in the recovery protocol, when one of the

participants contacts the TTP, the TTP has to contact the previous participant in the ring. This is not preferable because it is not guaranteed that previous participant will be available.

The protocols in [89, 68] solve the passive conspiracy problem of Bao et al. [9], however the problem in the recovery protocol still remains.

Markowitch and Kremer [69] proposed a non-repudiation protocol where their fairness definition is such that one of the parties sends some information to the other parties, and neither the sender nor others can deny that they participated. However, it does not solve the fairness problem in general multi-party fair exchange.

Complete Topologies: Multi party contract signing indeed corresponds to a complete topology. Garay and Mackenzie [45] proposed the first optimistic multi-party contract signing protocol that requires $O(n^2)$ rounds and $O(n^3)$ messages. Because of its inefficiency, Baum-Waidner and Waidner [11] suggested a more efficient protocol, whose complexity depends on the number of dishonest parties, and if the number of dishonest parties is $n-1$, its efficiency is same as [45]. Mukhamedov and Ryan [73] decreased the round complexity to $O(n)$. Lastly, Mauw et al. [70] gave the lower bound of $O(n^2)$ for the number of messages to achieve fairness. Their protocol requires $O(n^2)$ messages, but the round complexity is not constant. **We achieve both lower bounds ($\mathbf{O(n^2)}$ messages and constant round) for the first time.**

2.2 Fair and Secure Computation Related Works

Secure Two-Party Computation: Yao [87] firstly presented a secure and efficient protocol for two-party computation in the *semi-honest model*, where the adversary follows the protocol's rules but he cannot learn any additional information from his view of the protocol. Since this protocol is not secure against malicious behavior, new methods were suggested based on proving parties' honesty via zero-knowledge proofs [49], which makes the protocol inefficient. Another approach is cut-and-choose modification [66] to Yao's protocol, where we need $O(s)$ circuits for 2^{-s} error where s is security parameter. Furthermore, the number of rounds is not constant and

depends on s. However there are improved versions of this technique [76, 86, 64, 43], as well as non-interactive versions [1]. Some other techniques [22, 58] require parties to efficiently prove gate by gate that their behavior is honest using efficient sigma-proof-based zero-knowledge proofs of knowledge.

Another problem in Yao's protocol is *fairness*. The general solutions are based on gradual release timed commitments [81, 60, 83, 82] and trusted third party (TTP) [22, 63].

Gradual Release: Pinkas [81] presents a complex method in which the evaluation of the garbled circuit is executed on the *commitments* of the garbled values rather than the original garbled values. It uses cut-and-choose method to prevent malicious behavior of the constructor and *blind signatures* [28] for the verification of the evaluator's commitments. However, this protocol is expensive because of the gradual release timed commitments. On the other hand, there is a *majority circuit* evaluation that can reveal the constructor's input values, as shown by Kiraz and Schoenmakers [60], who improve Pinkas's construction by removing the majority circuit computation and the blind signatures, and using OR-proofs [34] instead. Yet, the other inefficiency problems remain. Similarly, Ruan et al. [82] use Jarecki and Shmatikov construction [58], and instead of the proof of the correct garbled circuit, they employ the cut-and-choose technique. Gradual release, which is used to solve the fairness problem, constitutes the weak part regarding the efficiency.

Optimistic Model: Cachin and Camenisch [22] present a fair two-party computation protocol in the optimistic model. The protocol consists of two intertwined verifiable secure function evaluations. In the case of an unfair situation, the honest party interacts with the TTP. Escrowed oblivious transfers and many proofs used cause inefficiency. Even worse, **the job of the TTP can be as bad as almost repeating the whole computation**. Lindell [63] constructs a framework that can be adopted by any two-party functionality with the property that either both parties receive the output, or one party receives the output while the other receives a digitally-signed check (i.e., monetary compensation). The proof is based on a new

ideal world definition that captures the fairness as defined above (i.e., exchanging money in return is considered fair as well [12, 61]). However, one may argue that one party obtaining the output and the other obtaining the money may not always be considered fair, since *we do not necessarily know how valuable the output would be before the evaluation.*

Security Definitions: Pinkas [81] protocol does not have proofs in the ideal-real world simulation paradigm. The importance of ideal-real world simulation is explained by Lindell and Pinkas [67]. Kiraz and Schoenmakers [60] and Ruan et al. [82] protocols have proofs in the ideal-real world simulation paradigm, but these proofs are based on the standard simulation, which is used to prove that the protocol is secure against malicious behavior and allows the adversary to abort. Fairness is then proven separately, without simulation. Therefore, none of these protocols have proofs in ideal-real world simulation paradigm that shows the protocol is both secure and fair. Cachin and Camenisch [22] gives ideal-real world definition includes fairness and security together for protocols that employ a TTP. However, interestingly, they do *not* prove their proposed protocol according to their definition. In Section 6.6, we show that proving fairness and security separately do not necessarily yield to **fair and secure** protocols. Canetti [26] gives general definitions for security for multi-party protocols with same intuition as our security and fairness definition.

Finally, there are very efficient constructions for specific applications [55, 18, 38], but we are interested in computing general functionalities efficiently. We achieve this goal, fairly, in the malicious setting.

Fair Secure Multi-party Computation: Secure multi party computation had an important position in the last decades, but its fairness property did not receive a lot of attention. One SMPC protocol that achieves fairness is designed by Garay et al. [54]. It uses gradual release, which is the drawback of this protocol, because each party broadcasts its output gradually in each round. At each round the number of messages is $O(n^3)$ and there are a lot of rounds due to gradual release.

Another approach is using bitcoin to achieve fairness with using a TTP in the

optimistic model [15, 2]. When one of the parties does not receive the output of the computation, he receives a bitcoin instead of the output. This fairness approach was used by Lindell [63] for the two-party computation case, and by Küpçü and Lysyanskaya [61] and Belenkiy et al. [12] for peer-to-peer systems. However, this approach is not appropriate for multi-party computation since **we do not necessarily know how valuable the output will be before evaluation**. Similarly, there are reputation-based fairness solutions [3], where one can talk about fairness probabilities, rather than complete fairness.

Chapter 3

DEFINITIONS AND PRELIMINARIES

3.1 Secure Computation

In multi-party computation, n parties compute a function $\phi : \{0,1\}^* \times \{0,1\}^* \times ... \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^* \times ... \times \{0,1\}^*$. Each party P_i has the private input w_i where $1 \leq i \leq n$. When they compute the function $\phi(w_1, w_2, ..., w_n) = (\phi_1(w_1, w_2, ..., w_n), \phi_2(w_1, w_2, ...w_n), ..., \phi_n(w_1, w_2, ...w_n)$, each P_i should obtain the corresponding output $\phi_i(w_1, w_2, ...w_n)$. When there are two parties it is called two party computation.

There are important requirements for a secure computation (e.g., privacy, correctness, independence of inputs, guaranteed output delivery, fairness [67]). Security is formalized with the ideal/real simulation paradigm. For every real world adversary, there must exist an adversary in the ideal world such that the execution in the ideal and real worlds are indistinguishable (see e.g., [48]).

Standard Ideal World: Simply, ideal world has a universal trusted third party U and U computes the output for the parties after receiving inputs of them.

Fairness is not concerned in the standard ideal world definition [48, 67] to ease the security proof of the computation protocol. Therefore, the adversary $\mathcal{A}$ in the standard ideal world has a right to stop U before sending the output to the honest parties so only adversary can learn the output of the computation.

The outputs of the parties in an ideal execution between the honest parties and an adversary $\mathcal{A}$ controlling set of corrupted parties C where U computes ϕ is denoted $IDEAL^{std}_{\phi,S(aux)}(w_1, w_2, ...w_n, \lambda)$ where $\{w_i\}_{1 \leq i \leq n}$ are the respective private inputs of the parties, aux is an auxiliary input of $\mathcal{A}$, S is the simulator and λ is the security parameter.

Yao's Two-Party Computation Protocol: We informally review Yao's construction [87], which is secure in the presence of *semi-honest* adversaries. Such adversaries follow the instructions of the protocol, but try to learn more information. The main idea in Yao's protocol is to compute a circuit without revealing any information about the value of the wires, except the output wires.

The protocol starts by agreeing on a circuit that computes the desired functionality. One party, called the *constructor*, generates two keys for every wire except the output wires. One key represents the value 0, and the other represents the value 1. Next, the constructor prepares a table for each gate that includes four double-encryptions with the four possible input key pairs (i.e., representing $00, 01, 10, 11$). The encrypted value is another key that represents these two input keys' output (e.g., for an AND gate, if keys k_0 and k_1 representing 0 and 1 are used, the gate's output key k', which is encrypted under k_0 and k_1, represents the value $0 = 0\ AND\ 1$.). Output gates contain double-encryptions of the output bits (no more keys are necessary, since the output will be learned anyway). All tables together are called the *garbled circuit*.

The other party is the *evaluator*. The constructor and the evaluator perform oblivious transfer, where the constructor is the sender and the evaluator is the receiver, who learns the keys that represent his own input bits. Afterward, the constructor sends his input keys to the evaluator. The evaluator evaluates garbled circuit by decrypting the garbled tables in topological order, and learns the output bits. The evaluator can decrypt one row of each gate's table, since he just knows one key for each wire. Since all he learns for the intermediary values are random keys and only the constructor knows which value these keys represent, the evaluator learns nothing more (other than what he can infer from the output). The evaluator finally sends the output to the constructor, who also learns nothing more than the output, since the evaluator did not send any intermediary values and they used oblivious transfer for the evaluator's input keys.

3.2 Fair and Secure Computation

Optimistic Fair Secure Computation: A group of parties with their private inputs w_i desire to compute a function ϕ [11, 49]. This computation is *secure* when the parties do not learn anything beyond what is revealed by the output of the computation. It is *fair* if either all of the parties learn the output in the end of the computation, or none of them learns the output. For an *optimistic* protocol, the TTP is involved *only* when there is a dispute about fairness between parties. This is formalized by ideal-real world simulations, defined below. Note that all definitions below are also same in two-party computation, so we give definitions over multi-party computation.

To provide *fair* secure computation, we adapt the secure computation definition, and add another player to the system: the trusted third party (TTP). At the end of the computation, we require that either all parties get their output values, or neither of them gets anything useful. For an *optimistic* protocol, the TTP is involved *only* when there is a dispute about fairness between the parties.

Ideal World: It consists of an adversary $\mathcal{A}$ that corrupts the set P_c of m parties where $m \in [1, n-1]$, the set of remaining honest party(s) P_h, and the universal trusted party U (*not the TTP*). The ideal protocol is as follows:

1. U receives inputs $\{w_i\}_{\{i \in P_c\}}$ or the message ABORT from $\mathcal{A}$, and $\{w_j\}_{\{j \in P_h\}}$ from the honest parties. If the inputs are invalid or $\mathcal{A}$ sends the message ABORT, then U sends $\perp$ to all of the parties and halts.

2. Otherwise U computes $\phi(w_1, ..., w_n) =$ $(\phi_1(w_1, ..., w_n), \phi_2(w_1, ..., w_n), ..., \phi_n(w_1, ..., w_n))$. Let $\phi_i = \phi_i(w_1, ..., w_n)$ be the i^{th} output. Then he sends $\{\phi_i\}_{\{i \in P_c\}}$ to $\mathcal{A}$ and $\{\phi_j\}_{\{j \in P_h\}}$ to the each corresponding honest party.

The outputs of the parties in an ideal execution between the honest parties and an adversary $\mathcal{A}$ controlling corrupted parties where U computes ϕ is denoted $IDEAL_{\phi, S(aux)}(w_1, w_2, ...w_n, \lambda)$.

Real World: There is no universally trusted party U for a real protocol π to compute the functionality ϕ. There is an adversary $\mathcal{A}$ that controls the set P_c of corrupted parties, and there is a TTP T who is involved in the protocol when there is unfair behavior. The pair of outputs of the honest party(s) P_h and the adversary $\mathcal{A}$ in the real execution of the protocol π, possibly employing the TTP T, is denoted $REAL_{\pi,T,\mathcal{A}(aux)}(w_1, w_2, ...w_n, \lambda)$, where $w_1, w_2, ...w_n, aux$ and λ are like above.

Note that U and T are not related to each other. T is the part of the real protocol to solve fairness problem when it is necessary, but U is not real (just an ideal entity).

Definition 1 (**Fair Secure Multi-Party Computation**). *Let π be a probabilistic polynomial time (PPT) protocol and let ϕ be a PPT multi-party functionality. We say that π computes ϕ* ***fairly and securely*** *if for every non-uniform PPT real world adversary $\mathcal{A}$ attacking π, there exists a non-uniform PPT ideal world simulator S so that for every $x, y, w \in \{0,1\}^*$, the ideal and real world outputs are computationally indistinguishable:*

$$\{IDEAL_{\phi,S(aux)}(w_1, w_2, ...w_n, \lambda)\}_{\lambda\in\mathbb{N}} \equiv_c \{REAL_{\pi,T,\mathcal{A}(aux)}(w_1, w_2, ...w_n, \lambda)\}_{\lambda\in\mathbb{N}}$$

The standard secure multi-party ideal world definition [67] lets the adversary $\mathcal{A}$ to abort *after* learning his output but *before* the honest party(s) learns her output. Thus, proving protocols secure using the old definition would not meet the fairness requirements. Therefore, we provide the modified definition above, and prove our protocols' security and fairness under this definition. Further realize that since the TTP T does not exist in the ideal world, the simulator should also simulate its behavior.

Canetti [26] gives general definitions for security for multi-party protocols with same intuition as our security and fairness definition.

Hybrid Model: We now present the concept of the *hybrid model*, where parties interact with each other as in the real protocol with having access to some trusted help as in the ideal model. The pair of outputs of the honest party and the adversary $\mathcal{A}$ in a hybrid execution of a protocol π and the universal trusted party U of an ideal world, computing a functionality Γ, is denoted $HYBRID^{\Gamma}_{\pi,\mathcal{A}(aux)}(w_1, w_2, ...w_n, \lambda)$,

where w_i, λ and Γ are like above. Informally, a **Γ-hybrid** proof means the real world protocol we are proving secure has access to the ideal Γ functionality. If we already know a real protocol for Γ that satisfies its ideal world definition, then proving computational indistinguishability between the ideal and hybrid worlds is enough for proving the security of the whole protocol (and thus equivalent to Definition 1) [26]:

$$\{IDEAL_{\phi,S(aux)}(w_1, w_2, ...w_n, \lambda)\}_{\lambda\in\mathbb{N}} \equiv_c \{HYBRID^{\Gamma}_{\pi,\mathcal{A}(aux)}(w_1, w_2, ...w_n, \lambda)\}_{\lambda\in\mathbb{N}}$$

Optimistic Multi-Party Fair Exchange: The participants are $P_1, P_2, ..., P_n$. Each participant P_i has an item f_i to exchange, where $i = \{1, 2, ..., n\}$, and wants to exchange his own item f_i with the other parties' items $\{f_j\}_{j\neq i}$. Thus, at the end, every participant should obtain $\{f_i\}_{1\leq i\leq n}$.

Multi-Party fair exchange is also a multi-party computation where the functionality ϕ is defined via its parts ϕ_i as below:

$$\phi_i(f_1, ..., f_n) = (f_1, f_2, ..., f_{i-1}, f_{i+1}, ..., f_n)$$

Therefore we can use Definition 1 as a security definition of the multi-party fair exchange.

The Resolution Functionality: The resolutions between a party and the fair-exchange TTP generally has the following format: The party proves that (s)he is honest in the protocol. Then (s)he asks to learn the necessary information from the TTP: this happens generally, though not necessarily, via the TTP decrypting a (verifiable) escrow.

Let p be the proof of honest behavior, v be the definition –the (verifiable) escrow– of the information i that the party wants to learn, and k be the secret key that opens v (v is encrypted by the TTP's public key, and thus k is the corresponding secret key of the TTP). Define the resolution functionality $\mathcal{F}_{\mathcal{RES}}((v, p), k) = (i, \lambda)$. The resolving party R employs the functionality with the TTP T and the universal trusted party U as follows:

1. R sends (p, v) and T sends k to U.

2. If p is a valid proof showing that R has behaved honestly until the resolution, U decrypts v and sends the plaintext i to R. Otherwise U sends the message ABORT to R and T.

Adversarial Model: When there is dispute between parties, the TTP resolves the conflict *atomically*. We assume that the adversary cannot prevent the honest party from reaching the TTP before the specified time interval. Secure channels are used to exchange decryption shares and endorsement of an e-coin, and when contacting the TTP. The adversary may control up to $n-1$ out of n parties in the exchange, and is probabilistic polynomial time (PPT). Therefore, $\mathcal{A}$ can control only one adversary in 2PC.

3.3 Other Preliminaries

Threshold Public Key Encryption: In such schemes, encryption is done with a single public key, generated jointly by n decrypters, but decryption requires at least k decrypters to cooperate. It consists of the probabilistic polynomial time (PPT) algorithms [85] *Key Generation, Encryption, Verification* and *Decryption*. We describe these algorithms via *ElGamal* (n, n) threshold encryption scheme as follows:

- *Key Generation:* It generates a list of private keys $SK = \{x_1, ..., x_n\}$ where $x_i \in \mathbb{Z}_p$, public key $PK = (g, h)$ where g is a generator of a prime p-order subgroup of $\mathbb{Z}_q$ with q prime, and $h = g^{\sum x_i}$, and public verification key $VK = \{vk_1, ...,_v k_n\} = \{g^{x_1}, ..., g^{x_n}\}$ where $n \geq 1$. Note that this can also be done in a distributed manner [80].
- *Encryption:* Compute the ciphertext for plaintext m as $E = (e_1, e_2) = (g^r, mh^r)$ where $r \in_r \mathbb{Z}_p$.
- *Verification:* It is between a verifier and a prover. Verifier, using VK, E and the decryption share of the prover $s_i = g^{rx_i}$ outputs *valid* if prover shows that $log_g vk_i$ is equal $log_{e_1} s$. Otherwise it outputs *invalid*.
- *Decryption:* It takes as an input n decryption shares $\{s_1, ..., s_n\}$ where $s_i = g^{rx_i}$, VK, and E. Then it outputs a message m with the following computation (in

$\mathbb{Z}_q$),

$$m' = \frac{e_2}{\prod s_i} = \frac{mh^r}{g^{r\sum x_i}} = \frac{mh^r}{h^r} = m$$

or $\perp$ if the decryption shares are not valid.

Verifiable Encryption: It is an encryption that enables the recipient to verify, using a public key, that the plaintext satisfies some relation, without performing any decryption [25, 23].

Verifiable Escrow: An escrow is a ciphertext under the public key of the TTP. A *verifiable* escrow [5, 25] enables the recipient to verify, using only the public key of the TTP, that the plaintext satisfies some relation, without performing any decryption. A public non-malleable label can be attached to a verifiable escrow [85]. We employ *ElGamal* verifiable encryption scheme [39, 22].

Endorsed E-cash: Endorsed e-cash [24] consists of two pieces: unendorsed coin and endorsement $(coin^u, e)$. $coin^u$ cannot be used as a coin without e. In addition, no one except the owner of $coin^u$ can construct a valid endorsement e for $coin^u$. The endorsement can be efficiently verifiably escrowed and can be verified to endorse the matching unendorsed coin. Note that in our protocols, we can employ any electronic payment scheme, as long as it can be efficiently verifiably escrowed (e.g., electronic checks [29]).

Camenish-Shoup (CS) Encryption Scheme [25]: A trusted party generates (n, g) as a common reference string. $n = pq$ is a safe RSA modulus where $p = 2p' + 1$, $q = 2q' + 1$ with p, q, p', q' all primes, and $g = (g')^{2n}$ with $g' \in \mathbb{Z}^*_{n^2}$. $\alpha = n + 1$ is defined implicitly. Function $abs(a)$ returns a for $a < \frac{n}{4}$ and $n - a$ for $a \geq \frac{n}{4}$.

Key Generation: The private keys $a_1, a_2, a_3 \in [0, \frac{n^2}{4}]$ is chosen. Besides a hash key hk is generated from the key space of a collision resistant hash function family $\mathcal{H}$. The public key of the encryption scheme are $(hk, n, g, g_1, g_2, g_3)$ where $g_i = g^{a_i}$ for $1 \leq i \leq 3$.

Encryption: The encryption of the message $m \in [-\frac{n}{2}, \frac{n}{2}]$ with label L is (u, e, v) where $u = g^r$, $e = g_1^r \alpha^m$ and $v = abs(g_2 g_3^{\mathcal{H}_{hk}(u,e,L)})$, $r \in_R [0, \frac{n}{4}]$.

Decryption: Firstly $abs(v) = v$ and $v^2 = u^{2(a_2+\mathcal{H}_{hk}(u,e,L)a_3)}$ are checked. If the equalities hold then $m' = (e/u^{a_1})^{2t}$ is computed where $t = 2^{-1} \mod n$. If m' is the form of α^m for some $m \in [-\frac{n}{2}, \frac{n}{2}]$, then outputs m.

Simplified Camenish-Shoup (sCS) Encryption Scheme [58]: As above, a trusted party generates (n, g).

Key Generation: The private key $a \in [0, 2^{\frac{|n|}{2}}]$ and public key is $w = g^a$.

Encryption: The encryption of the message $m \in [-\frac{n}{2}, \frac{n}{2}]$ is (u, e) where $u = g^r \mod n^2$ and $e = \alpha^m w^r$ where $r \in_R [0, \frac{n}{4}]$.

Decryption: It is similar to CS decryption. There is no checking process.

sCS commitment is e in the encryption scheme that commits message m.

Double Encryption with sCS Encryption Scheme [58]: The plaintext is first divided into two shares, then the first share is encrypted with sCS encryption using the first wire's key, and similarly the second share is encrypted with the second wire's key for the gate. Thus, the double-encryption consist of these two sCS encryptions, together with the sCS commitment to the first share.

Equality Test [18]:

Equality test is a specific secure 2PC where the parties only learn that if their private input is equal or not in the end of the computation. Below we give the adapted protocol of [18] for the equality test:

- Alice and Bob generate a group G, which has a large prime order q, with generators g_0, g_1, g_2, g_3. Then, they compute the corresponding values of their input bits (for equality test) in $\mathbb{Z}_q$. In our case, these are the hashed values of all left (right) hand sides of equation 6.1. Say Alice's value is θ_a and Bob's is θ_b.

- Alice chooses a random $x_a \in \mathbb{Z}_q$ and computes $g_a = g_1^{x_a}$ and similarly Bob chooses a random $x_b \in \mathbb{Z}_q$ and computes $g_b = g_1^{x_b}$. Then, they send these values

to each other and calculate $g_{ab} = g_a^{x_b} = g_b^{x_a}$. In addition, they prove knowledge of x_a and x_b using Schnorr's protocol [84].

- Alice selects $a \in_R \mathbb{Z}_q$, calculates $(P_a, Q_a) = (g_3^a, g_1^a g_2^{\theta_a})$, and sends (P_a, Q_a) to Bob. Bob does the symmetric version, computing and sending $(P_b, Q_b) = (g_3^b, g_1^b g_2^{\theta_b})$, where $b \in_R \mathbb{Z}_q$. Alice calculates $R_a = (Q_a/Q_b)^{x_a}$. Bob also calculates $R_b = (Q_a/Q_b)^{x_b}$. Then, Alice sends R_a with a proof that $log_{g_1} g_a = log_{Q_a/Q_b} R_a$ to Bob [30].

- Bob can now learn the result of the equality test by checking whether $P_a/P_b = R_a^{x_b}$. Then, he sends same proof as in [30] to show $log_{g_1} g_b = log_{Q_a/Q_b} R_b$ so that Alice also learns the equality test result.

Note that if Bob does not send the last message, he will not obtain the equality signature s_{eq} in our protocol, and the whole protocol will be aborted without anyone learning the actual output.

3.4 Notation

The parties in the protocols are represented by P_i where $i \in \{1, 2, ..., n\}$ and n is the number of parties. P_h is to show the honest parties and P_c is to show the corrupted parties controlled by the adversary $\mathcal{A}$.

VE_i is used to show verifiable encryption and VS_i is used to show verifiable escrow that is prepared by P_i. The descriptive notation for verifiable encryption and escrow is

$$V(E, pk; l)\{(v, \xi) \in R\}.$$

It denotes the verifiable encryption and escrow for the ciphertext E whereby ξ –whose relation R with the public value v can be verified– is encrypted under public key pk (if it is escrow under TTP's public key), and labeled by l.

$PK(v)\{(v, \xi) \in R\}$ denotes the zero-knowledge proof of knowledge of ξ that has a relation R with the public value v. All relations R in our protocols have an honest-

verifier zero-knowledge tree-move proof of knowledge [34], and hence can be implemented very efficiently. The notation ⓩ shows the number z in the Figure 4.1 and 4.6.

Chapter 4

OPTIMALLY EFFICIENT MULTI-PARTY FAIR EXCHANGE

Remember that our aim is to create efficient multi-party fair exchange protocols for the under-studied complete topology. Thus, it is important *not* to let a party learn some item *before* others are *guaranteed* that they will get his item in complete topology MFE. We used this intuition while designing our protocols. Therefore, we oblige **parties to depend on some input from every party in every phase of the protocols**, so even if there is only one honest party, the dishonest ones have to contact and provide their correct values to the honest party so that they can continue with the protocol.

4.1 Multi-Party Fair Exchange Protocol (MFE)

There is a trusted third party (TTP) that is involved in the protocol when a dispute happens between the participants about fairness. His public key pk is known to every participant.

Overview: The protocol has three phases. In the first phase, each party chooses a share and then they jointly generate a public key for the threshold encryption scheme using their shares. This phase needs to be done only once among the same set of participants. In the second phase, they send to each other the threshold encryptions of the items that they want to exchange. In the final phase, they exchange decryption shares for each item. The details of the phases are below (see also Figure 4.1):

Phase 1 (① and ② in Figure 4.1): All participants agree on the prime p-order subgroup $\mathbb{Z}_q$ where q is a large prime and a generator g of this subgroup. Then each P_i does the following [80] (operations are in $\mathbb{Z}_q$):

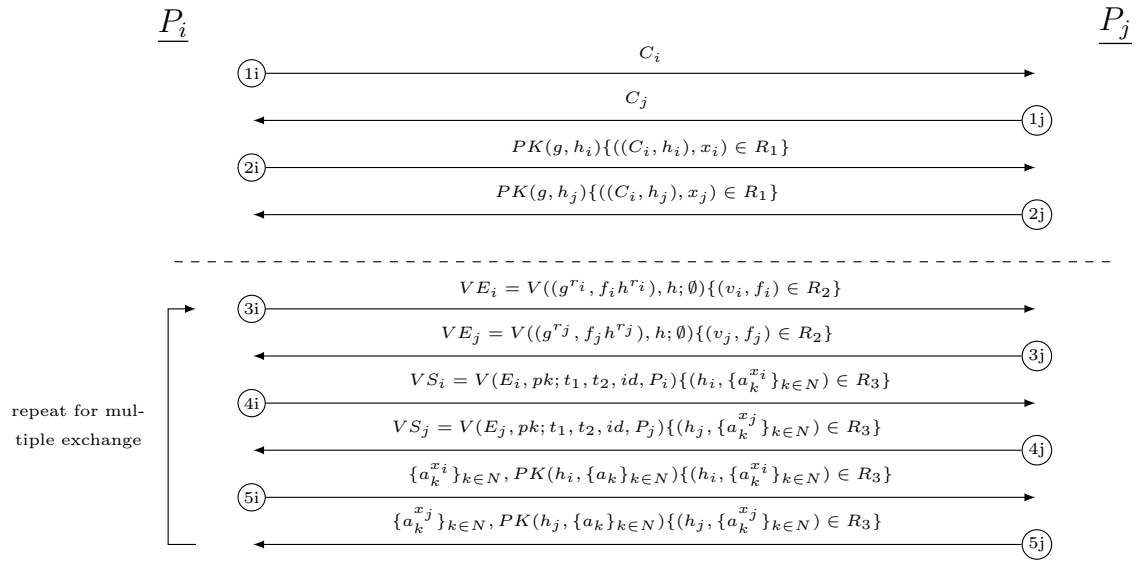

Figure 4.1: MFE Protocol: Here each pair $((i, j))$ can be performed in this order or reverse order or parallel.

- P_i randomly selects his share x_i from $\mathbb{Z}_p$ and computes verification key $h_i = g^{x_i}$. Then he commits to x_i and sends the commitment C_i to other parties [80].
- After receiving all commitments from the other parties, P_i sends verification share h_i and proof of knowledge $PK(g, h_i)\{((C_i, h_i), x_i) \in R_1\}$ where R_1 is the relation showing that $h_i = g^{x_i}$ where x_i is the secret share committed in C_i.
  Note that this must be done after exchanging all the commitments, since otherwise we cannot claim independence of the shares, and then the threshold encryption scheme's security argument would fail. But with the two steps above, the security proof for threshold encryption holds here.
- After receiving all the verification shares successfully, P_i computes the threshold encryption's public key

$$h = h_1 h_2 ... h_n = g^{x_1} g^{x_2} ... g^{x_n} = g^{\sum_i x_i} = g^x.$$

Phase 1 is executed only once. Afterward, the same set of parties can exchange as many items as they want by performing only Phase 2 and Phase 3.

Phase 2 (③ in Figure 4.1): Firstly, they agree on two time parameters t_1 and

t_2, and identification id of the protocol. (Time parameters could have been already agreed in Phase 1.)

Each participant P_i does the following:

- P_i sends a verifiable encryption of his item f_i:

$$VE_i = V((g^{r_i}, f_i h^{r_i}), h; \emptyset)\{(v, f_i) \in R_2\}$$

  where r_i is randomly selected from $\mathbb{Z}_p$. For the notation simplicity, we denote $(a_i, b_i) = (g^{r_i}, f_i h^{r_i})$. VE_i includes the encryption of the item f_i with public key h and it can be verified that the encrypted item f_i and v has the relation R_2. Simply put, P_i proves that he encrypts desired item. Remark that here v is some publicly known value that has a relation R_2 with the item f_i (e.g., if f_i is a signature on a common contract, then v is the signature verification key and the contract, and R_2 is the relation that f_i is a valid signature with respect to v). Refer to [23] for the details of the verifiable encryption scheme.

  Note that without knowing n decryption shares, no party can decrypt any VE_j and learn the items. Therefore, they need a fair exchange protocol after this point to obtain all the decryption shares. This is done in the following phase.

Phase 3 (④ and ⑤ in Figure 4.1): No party begins this phase without completing Phase 2 and receiving all verifiable escrows VE_j correctly.

- P_i sends to other parties another verifiable escrow VS_i that includes decryption shares for each verifiable encryption VE_j. VS_i is computed as

$$VS_i = V(E_i, pk; t_1, t_2, id, P_i)\{(h_i, \{a_k^{x_i}\}_{1 \le k \le n}) \in R_3\}$$

  where E_i is the encryption of $a_1^{x_i}, a_2^{x_i}, ..., a_n^{x_i}$ with the TTP's public key pk. Here, the relation R_3 is:

$$\log_g h_i = \log_{a_k} a_k^{x_i} \text{ for each } k. \tag{4.1}$$

  Simply, the verifiable escrow VS_i includes the encryption of the decryption shares of P_i that will be used to decrypt the encrypted items of all parties. It can be verified that it has the correct decryption shares. In addition, only the

TTP can open it. The label t_1, t_2, id, P_i contains the public parameters of the protocol, and P_i is just a name that the participant chooses. Here, we assume that each party knows the other parties' names.

Remark: The name P_i is necessary to show the VS_i belongs him. It is not beneficial to put a wrong name in a verifiable escrow's label, since a corrupted party can convince TTP to decrypt VS_i by showing P_i is dishonest. The other labels id, t_1, t_2 are to show the protocol parameters to TTP. id is necessary to prevent corrupted parties to induce TTP to decrypt VS_j for another exchange. Consider that some exchange protocol ends unsuccessfully, which means nobody receives any item. The corrupted party can go to TTP as if VS_j is the verifiable escrow of the next protocol, and have it decrypted, if we were not using exchange identifiers. We will see in our resolution protocols that **cheating in the labels do not provide any advantage to an adversary**.

- P_i waits for VS_j from each P_j. If anything is wrong with some VS_j (e.g., verification fails or the label is not as expected), or P_i does not receive the verifiable escrow from at least one participant, he executes **Resolve 1** before t_1. Otherwise, P_i continues with the next step.
- P_i sends decryption shares $(a_1^{x_i}, a_2^{x_i}, ..., a_n^{x_i})$ to each P_j. In addition, he executes the zero-knowledge proof of knowledge showing that these are the correct decryption shares

$$PK(h_i, \{a_k\}_{k\in N})\{(h_i, \{a_k^{x_i}\}_{1\leq k\leq n}) \in R_3\}. \tag{4.2}$$

- P_i waits for $(a_1^{x_j}, a_2^{x_j}, ..., a_n^{x_j})$ from each P_j together with the same proof that he does. If one of the values that he receives is not expected or if he does not receive them from some P_j, he does **Resolve 2** protocol with the TTP, before t_2 and after t_1. Otherwise, P_i continues with the next step.
- After receiving all necessary values, P_i can decrypt each VE_i and get all the items. The decryption for item f_j is below:

$$e_{2_j} / \prod_k a_j^{x_k} = f_j h^{r_j} / g^{r_j \sum_k x_k} = f_j h^{r_j} / (g^{\sum_k x_k})^{r_j} = f_j h^{r_j} / h^{r_j} = f_j$$

4.1.1 Resolve 1

The goal of Resolve 1 is to *record* the corrupted parties that did *not* send their verifiable escrow in ⑤ or ⑥. Resolve 1 needs to be done before t_1. Parties do not learn any decryption shares here. They can just complain about other parties to the TTP. The TTP creates a fresh list *complaintList* for the protocol with parameters id, t_1, t_2. The *complaintList* contains the names of pairs of parties that have a dispute because of the missing verifiable escrow. The **complainant** is the party that complains, whose name is saved as the first of the pair, and the **complainee** is saved as the second of the pair. In addition, the TTP saves *complainee's verification key* given by the complainant; in the case that the complainee contacts the TTP, he will be able to prove that he is the complainee. See Algorithm 1.

Algorithm 1 Resolve 1

1: P_i sends id, t_1, t_2, P_j, h_j to the TTP where P_j is the party that did not send VS_j to P_i. The TTP does the following:
2: **if** $currenttime > t_1$ **then**
3: **send** *msg* "Abort Resolve 1"
4: **else**
5: $complaintList$ = GetComplaintList(id, t_1, t_2)
6: **if** $complaintList$ == NULL **then**
7: $complaintList$ = CreateEmptyList(id, t_1, t_2) // initialize empty list
8: $solvedList$ = CreateEmptyList(id, t_1, t_2) // will be used in Resolve 2
9: **end if**
10: $complaintList$.add$(P_i, (P_j, h_j))$
11: **send** *msg* "Come after t_1"
12: **end if**

4.1.2 Resolve 2

Resolve 2 is the resolution protocol where the parties come to the TTP to ask him to decrypt verifiable escrows and TTP solves the complaint problems in Resolve 1. TTP does not decrypt any verifiable escrow until *complaintList* is empty.

The party P_i who comes for Resolve 2 gives all verifiable escrows that he has already received from other parties and his own verifiable escrow to TTP. TTP uses desired verifiable escrows by the complainants in *complaintList* to save the decryption shares for them. If *complaintList* is not empty in the end, P_i comes after t_2 for Resolve 3. Otherwise, P_i gets all the decryption shares that he requests.

Algorithm 2 Resolve 2

```
1: P_i gives M which is a set of received verifiable escrows by P_i. The TTP does the following:
2: complaintList = GetComplaintList (id, t_1, t_2)
3: for all VS_j in M do
4:     if (*, (P_j, h_j)) ∈ complaintList and CheckCorrectness(VS_j, h_j) is true then
5:         shares_j = Decrypt(sk, VS_j)
6:         solvedList.Save(P_j, shares_j)
7:         complaintList.remove((*, (P_j, h_j)))
8:     end if
9: end for
10: if complaintList is empty then
11:     send msg "Do Resolve 3"
12: else
13:     send msg "Come after t_2"
14: end if
```

CheckCorrectness(VS_j, h_j) returns *true* if the TTP can verify the relation in equation (4.1) using verifiable escrow VS_j and h_j. Otherwise it returns $false$.

4.1.3 Resolve 3

This needs to be executed after t_2; otherwise the TTP denies all Resolve 3 requests. If the *complaintList* still has parties that have a conflict, the TTP answers each resolving party saying that the protocol is **aborted**, which means no body is able to learn any item. If the *complaintList* is *empty*, the TTP decrypts any verifiable escrow that is given to him. Besides, if the complainants in the *solvedList* come, he gives the stored decryption shares. See Algorithm 3.

Algorithm 3 Resolve 3

```
1: P_i gives C which is a set of parties that do not do step (4) or (5) in Figure 4.1 with P_i and V which is the set of some VS that belongs some parties C.
2: complaintList = GetComplaintList (id, t_1, t_2)
3: if complaintList.isEmpty() then
4:     for all P_j in C do
5:         if VS_j ∈ V then
6:             send Decrypt(sk, VS_j)
7:         else
8:             send complaintList.GetShares(P_j) //It returns saved shares of P_j
9:         end if
10:    end for
11: else
12:    send msg "Protocol is aborted"
13: end if
```

Theorem 1. *The MFE protocol above is fair according to Definition 1, assuming that ElGamal threshold encryption scheme is a secure threshold encryption scheme, and the associated verifiable escrow scheme is secure, all commitments are hiding and*

binding, and the discrete logarithm problem is hard (so that the proofs are sound and zero-knowledge).

Proof: Let's assume that adversary $\mathcal{A}$ corrupts $n-1$ parties. The simulator S simulates honest parties in the real world and corrupted parties in the ideal world. S also acts as a TTP in the protocol if any resolution protocol occurs so S publishes his public key pk as a TTP. We assume that the parties $\{P_i\}_{i\in N\setminus\{1\}}$ are corrupted parties and S simulates behavior of the honest party P_1. S does the following:

Phase 1:

S behaves same as Phase 1 in the real protocol. Differently, S behaves as a simulator of the proof of knowledge so that he learns all shares $\{x_i\}_{2\leq i\leq n}$ of the corrupted parties.

Phase 2:

- Phase 2 is almost same in the simulation. Since S does not know the item f_1, he encrypts a random item $\tilde{f_1}$ and sends verifiable encryption $\tilde{VE}_1$ to the other parties.
- S waits verifiable encryption of the corrupted parties $\{VE_i\}_{2\leq i\leq n}$. S does not continue to the next step until he receives all $V^{(1)}$s. When a party sends his verifiable encryption, S behaves as a simulator of it and learns the party's item. In the end of this step, S learns all the items $\{f_i\}_{2\leq i\leq n}$.

Phase 3:

- S behaves as in Phase 3. In the end of t_1, if there are parties that does not send their verifiable escrow, S adds them in *complaintList* because in reality, the honest party should complain them before t_1 in Resolve 1. In addition, if a corrupted party does Resolve 1 even though it is not necessary, S behaves like TTP and add him and his complainee to the *complaintList*.
- After t_1 if S receives all in $\{V_i^{(2)}\}_{2\leq i\leq n}$, it means that fairness is guaranteed because S in the real world is now able to learn all decryption shares from the

corrupted parties or Resolve 2-3. Therefore S sends all items $\{f_i\}_{2\leq i\leq n}$ where he learned in Phase 2 to U. Then U sends f_1 to S. At this point, S should send his decryption shares to the other parties. However, S has sent random values a_1, b_1 as an encryption of f_1 and he does not have decryption shares. Therefore, he calculates following equation to find appropriate decryption shares d_is such that the other parties can get the item f_i from a_i, b_i using d_i.

$$f_i = \frac{b_i}{d_i.a_i^{x_2}...a_i^{x_n}} \tag{4.3}$$

Then S calculates each d_i for each item . Here, d_is represents $a_i^{x_1}$ in the real protocol. He can do this calculation because he learned f_1 from U, $\{f_i\}_{2\leq i\leq n}$ in Phase 2 and $\{a_i^{x_j}\}_{j\in N\setminus\{1\}}$ while the corrupted parties are sending their own $V_i^{(2)}$.

S sends all d_i to all of the parties and simulates the proof of knowledge which shows that d_i is correct. At the same time, S waits decryption shares from the corrupted parties. If all of them send their own decryption shares with valid proof of knowledge that shows that they send correct decryption shares the simulation ends.

If some of the parties does not send decryption shares to S before t_2, S does Resolve 2 as in real protocol and clears the all *complaintList* if necessary because he has all corrupted parties' verifiable escrow.

- If some of the parties do not send $V^{(2)}$s before t_1, S adds these parties to *complaintList*. After he does not send any his decryption shares as in real protocol. If some of the corrupted parties comes for Resolve 2, S behaves exactly as TTP and clears the some parties from *complaintList* according to verifiable escrow that is given to S. Each time when he is clearing *complaintList*, he learns the decryption shares of the complainee party. In the end, if *complaintList* is empty, it means that he learned all decryption shares of the corrupted parties. So S sends all items $\{f_i\}_{2\leq i\leq n}$ to U. Then U sends f_1 to S and then calculates his shares d_i as in Equation (4.3). Therefore when parties comes for Resolve 3,

S can give every share that they want. Simulation ends.

If *complaintList* is not empty in the end of t_2, S sends message ABORT to U and simulation ends.

Claim 1. *The view of adversary $\mathcal{A}$ in his interaction with the simulator S is indistinguishable from the view in his interaction with the honest party.*

Proof: S behaves differently than the real protocol while sending verifiable encryption $\tilde{V}_1^{(1)}$ and verifiable escrow $\tilde{V}_1^{(2)}$. $\tilde{V}_1^{(1)}$ is indistinguishable from the real one because of the security of the ElGamal encryption scheme [39] Similarly, $\tilde{V}_1^{(2)}$ is indistinguishable from the real one because of the security of the verifiable escrow [25, 5].

In addition, S behaves as TTP without any difference so the interaction with the TTP is indistinguishable too.

The output of the parties in the end of the protocol is identical to the real protocol.

This completes the proof of Theorem 1.

4.2 All Topologies for MFE

We introduce the MFE protocol in previous section for the complete topology. Besides our MFE protocol can be adapted to every topology.

	f_1	f_2	f_3	f_4	f_5
P_1		x	x		x
P_2	x			x	x
P_3	x		x		
P_4	x	x	x	x	x

Table 4.1: Desired items by each parties in matrix form. Each party wants the marked items that corresponds his/her row.

If a party desires an item from another party, he should have all the shares of the item as shown in Table 4.2. In general we can say that if a party P_i wants the item

	f_1	f_2	f_3	f_4	f_5
P_1		$\{a_2^{x_i}\}_{1\leq i\leq n}$	$\{a_3^{x_i}\}_{1\leq i\leq n}$		$\{a_5^{x_i}\}_{1\leq i\leq n}$
P_2	$\{a_1^{x_i}\}_{1\leq i\leq n}$			$\{a_4^{x_i}\}_{1\leq i\leq n}$	$\{a_5^{x_i}\}_{1\leq i\leq n}$
P_3	$\{a_1^{x_i}\}_{1\leq i\leq n}$		$\{a_3^{x_i}\}_{1\leq i\leq n}$		
P_4	$\{a_1^{x_i}\}_{1\leq i\leq n}$	$\{a_2^{x_i}\}_{1\leq i\leq n}$	$\{a_2^{x_i}\}_{1\leq i\leq n}$	$\{a_3^{x_i}\}_{1\leq i\leq n}$	$\{a_4^{x_i}\}_{1\leq i\leq n}$

Table 4.2: Necessary shares for each party to get the desired items that shown in Table 4.1.

f_t he should only receive $\{a_t^{x_j}\}_{\{j\in N\}}$ from the corresponding parties $\{P_j\}_{\{j\in N\}}$ where $1 \leq i, t \leq n$. Therefore our MFE can be applied any topology with the same fairness condition which is **all parties will receive all their desired items or none of them receives in the end of the protocol**.

Our strong fairness condition requires that all parties have to depend each other. Even though P_i does not want an item f_j from P_j, getting his desired item have to also depend P_j. Therefore we cannot decrease number of messages even in a simpler (e.g ring) topology.

Let's think a ring topology as in Figure 4.2 and 4.3. Parties want an item from only the previous party. For example, P_2 only wants P_1's item f_1. However, P_2 should contact all other parties because of our fairness condition with using our MFE protocol.

On the other hand, the size of the verifiable escrow which means that number of shares in the verifiable escrow decreases in other topologies. If we represent the topology in a matrix form as in Table 4.1, each party P_i have to add number of x shares corresponding to the party P_j's column to the verifiable escrow that is sent to P_j. For example, each P_i adds one share to the verifiable escrows in the MFE protocol in the ring topology in Table 4.2. We can conclude that total size of the verifiable escrows that a party sends is $O(\#x)$ where x is as in Figure 4.2.

Even though there is no item exchange between the parties as in ring topology (Figure 4.3), we succeed them that either that receive all their desired item(s) or none

of them receives. Besides, we are not limited with a topology that follows a specific pattern as number of parties and items should be equal. For example it is possible to succeed the fairness in thetopology in Figure 4.4 even though P_2, P_3 and P_4 does not have any item to exchange between each other.

	f_1	f_2	f_3	f_4
P_1		x		
P_2			x	
P_3				x
P_4	x			

Figure 4.2: Matrix representation of the ring topology.

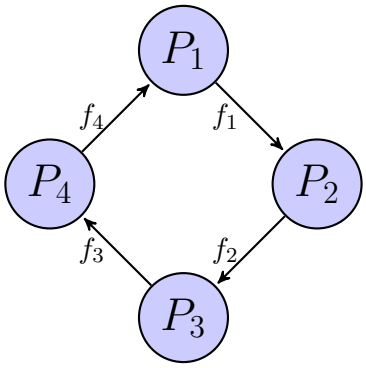

Figure 4.3: Graph representation of the ring topology.

	f_1	f_2	f_3	f_4	f_5
P_1		x	x	x	x
P_2	x				
P_3	x				
P_4	x				

Figure 4.4: Matrix representation of a topology.

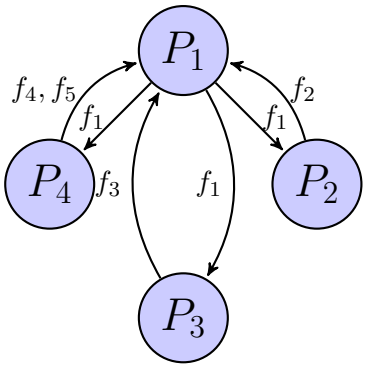

Figure 4.5: Graph representation of a topology.

4.3 Coin-based Multi-Party Fair Exchange Protocol (CMFE)

Our original MFE protocol is efficient if and only if the item that is exchanged can be efficiently verifiably encrypted. This can be very practical, for example, in contract signing scenarios, where a signature in an escrow can be verified easily with the

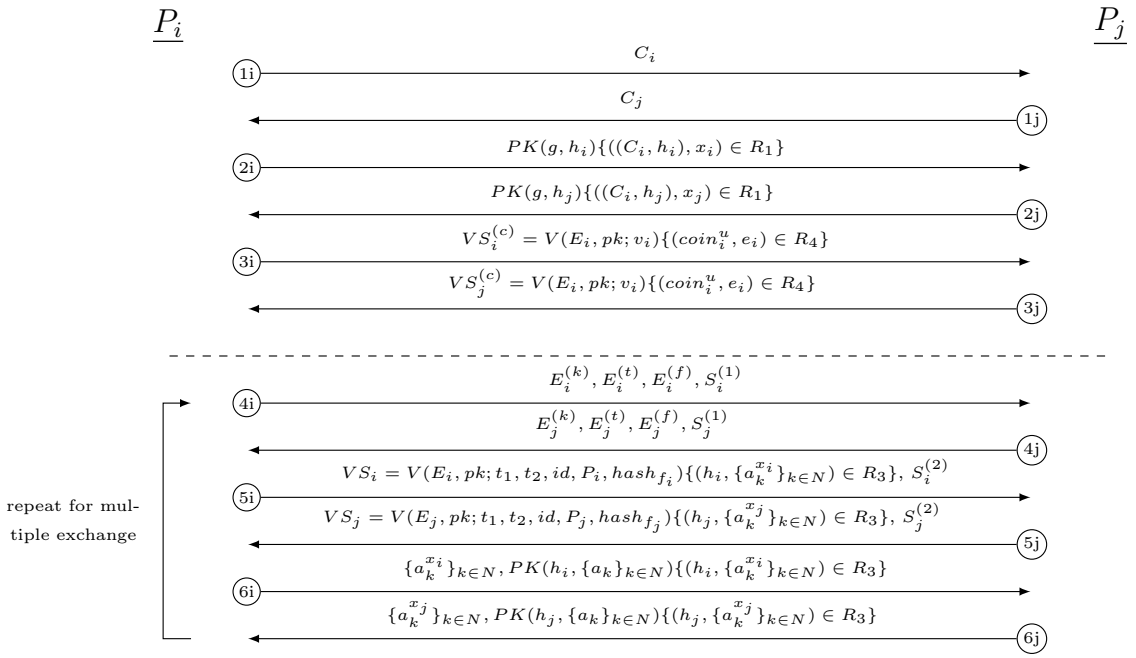

Figure 4.6: CMFE Protocol: Here each pair (i, j) can be performed in this order or reverse order or parallel.

corresponding verification key [5]. Since the verifiable encryption of some items (e.g., files) cannot be performed efficiently, we cannot directly use our MFE protocol. To keep the protocol efficient even for exchanging files, we add electronic payments (e.g., [24, 29]) to the system. The main idea is that, if a party receives a wrong item (because it could not have been verified beforehand efficiently), then he can get the coin of the party that gave the wrong item. Note that, some existing protocols (e.g., [61, 63]) also apply this type of fairness definitions, and consider scenarios such as peer-to-peer file sharing, where the hash of the file to be exchanged is known beforehand (e.g., via a tracker [31]). Therefore we can define functionality φ of CMFE protocol following:

$$\varphi(z_1, z_2, ..., z_n) = (\varphi_1(z_1, z_2, ..., z_n), \varphi_2(z_1, z_2, ..., z_n), ..., \varphi_n(z_1, z_2, ..., z_n)) \tag{4.4}$$

where $z_i = (coin_i, f_i)$ and $\varphi_i(z_i) = \{t_1, ..., t_{i-1}, t_{i+1}, ..., t_n\}$. Each t_j is defined as below:

$$t_j = \begin{cases} f_j, & \text{if } hash(f_j) = hash_{f_j} \\ coin_j, & \text{otherwise.} \end{cases} \tag{4.5}$$

Here, $coin_j$ is the electronic payment and f_i is the item of party P_i. $hash_{f_i}$ is the publicly known hash value of the item f_i, where $hash$ is a universal one-way hash function. Note that, the general condition for t_j values is that the actual item will be obtained as long as it is the desired item (e.g., hashes may be used for verifying files, a public key infrastructure may be used for verifying signatures), and the payment will be obtained otherwise. This condition applies to each t_j independently. Thus, it is possible that some item f_j is obtained from P_j while some payment $coin_i$ is obtained from P_i.

Now that we have relaxed our fairness definition in this way, we want to minimize our changes to the MFE protocol so that we can keep enjoying its efficiency and privacy benefits. One problem is due to the ElGamal threshold encryption scheme we employed. Since it can encrypt only the elements in $\mathbb{Z}_q$, we cannot use it to encrypt any item (e.g., a large file). Therefore, we solve this problem by performing a symmetric key encryption (e.g., AES) of the file, and encrypting the symmetric key with ElGamal.

We assume that the hash of the item $hash_{f_i}$ is publicly known, and for the sake of simplicity of presentation, the underlying electronic payment scheme is Endorsed E-cash [24]. Our CMFE protocol has the following phases:

Phase 1: The parties do the same job as in Phase 1 of the MFE. Additionally, each P_i generates signature signing-verification key pair s_i, v_i. Then each party P_i sends to the other parties a verifiable escrow that includes the endorsement e_i of his own coin, together with the unendorsed coin $coin_i^u$:

$$V_i^{(c)} = VE(E_i, pk; v_i)\{(coin_i^u, e_i) \in R_4\}.$$

Here E_i is encryption of the endorsement e_i with the public key pk of the TTP, and the relation R_4 shows that e_i is the endorsement of $coin_i^u$. The signature verification key v_i is the label of $V_i^{(c)}$. This label is necessary to link the second verifiable escrow (explained below) with this one, so that the TTP can understand that both were sent by the same party. For efficiency, the verifiable escrow used here can be Camenisch-Shoup verifiable escrow [25].

Parties execute Phase 1 only once, as in MFE. They can continue exchanging multiple items using only the following phases. If a party P_r receives the coin of some other party P_s during a dispute, to continue the next item exchange, P_s needs to send a new $V_i^{(s)}$ to only P_r, without the need to fully repeat Phase 1. In some sense, think of every participant putting down some money; if theirs is spent, they need to renew it to continue.

Phase 2: The second phase is again similar to MFE. Since it is hard to encrypt a large item with ElGamal, each party P_i first encrypts their item f_i (e.g., a file) using a symmetric encryption scheme (e.g., AES) with key K_i to obtain $E_i^{(f)} = Enc_{K_i}(f_i)$. Then, P_i encrypts K_i with the threshold ElGamal public key h that they constructed with their shares in Phase 1: $E_i^{(k)} = (g^{r_i}, K_i h^{r_i})$. In addition, he encrypts K_i with the TTP's public key pk. We call this encryption $E_i^{(t)}$. He also signs this ciphertext as $S_i^{(1)} = sign_{s_i}(E_i^{(t)})$. He sends $E_i^{(f)}, E_i^{(k)}, E_i^{(t)}, S_i^{(1)}$ to all parties.The signature $S_i^{(1)}$ helps the TTP to understand that $V_i^{(c)}$ and $E_i^{(t)}$ are encrypted by the same party (remember, $V_i^{(c)}$ contains the signature verification key in its public non-malleable label).

No party continues with the next phase without completing this phase.

Phase 3: This is almost the same as the MFE protocol, except P_i additionally puts $hash_{f_i}$ and $hash_{E_i^{(f)}}$ into the label of $V_i^{(2)}$, and also creates and sends the signature $S_i^{(2)}$ of $V_i^{(2)}$ (again tying $V_i^{(2)}$ to $V_i^{(c)}$). Since $hash_{f_i}$ is public, the other parties can check its correctness. Since they received $E_i^{(f)}$ in the previous phase, they can also verify the correctness of $hash_{E_i^{(f)}}$. If there is a problem with this verifiable escrow or the signature, they do Resolve 1.

Note that Resolve 1-2-3 are the same and can be executed after the same steps as in MFE.

Assume that the protocol is not aborted, and either at the end of the protocol, or in Resolve 2 or 3, P_i obtained the key K_j. It is possible that this key is wrong, since it was not verifiably encrypted. If P_i finds that the key K_j does not result in a file f_j with hash $hash_{f_j}$, then she runs Resolve 4 with the TTP.

Resolve 4

First, P_i requests the key in $E_j^{(t)}$ from the TTP. The TTP decrypts $E_j^{(t)}$ and sends the resulting K_j to P_i (if the *complaintList* is empty). If the K_j turns about to be a wrong key, P_i proves this using the Belenkiy et al. subprotocol [12] below, and obtains the endorsement e_j of the coin in $V_j^{(c)}$ from the TTP. See Algorithm 4.

Algorithm 4 Resolve 4

P_i sends $E_j^{(t)}, V_j^{(c)}, S_j^{(1)}, V_j^{(2)}, S_j^{(2)}$ to the TTP where P_j is the party whose item with hash $hash_{f_j}$ could not be obtained. The TTP does the following:
if $currenttime < t_2$ OR ($S_j^{(1)}$ or $S_j^{(2)}$ is not valid when verified with the signature verification key in the label of $V_j^{(c)}$) **then**
    send *msg* "Abort Resolve 4"
end if
complaintList = GetComplaintList (id, t_1, t_2)
if *complaintList*.isEmpty() **then**
    K_j = Decrypt($E_j^{(t)}, sk$)
    output ← **execute** sub-protocol "Prove Key is not Correct" with P_i // only if requested by P_i
    if *output* is true **then**
        send Decrypt($V_j^{(c)}$, sk)
    else
        send *msg* "Abort Resolve 4"
    end if
else
    send *msg* "Abort Resolve 4"
end if

Belenkiy et al. proof that key is not correct [12]: Belenkiy et al. showed a communication-efficient and privacy-preserving way of proving that the key K_j in $E_j^{(t)}$ fails to decrypt a ciphertext with hash $hash_{E_j^{(f)}}$ to a plaintext with hash $hash_{f_j}$. The TTP knows $hash_{E_j^{(f)}}$ and $hash_{f_j}$ from the label of the verifiable escrow $V_j^{(2)}$ given by P_i (but remember the label is non-malleable). He also knows K_j as he decrypted and obtained from $E_j^{(t)}$. He further verified both signatures $S_j^{(1)}, S_j^{(2)}$ against the same verification key in the label of $V_j^{(c)}$, and hence is sure that all these belong to the same participant. Note that for this subprotocol to work efficiently, the hashes need to be Merkle hashes [71], and the file needs to be encrypted block-by-block with the same key. Hence, $hash_{E_j^{(f)}}$ and $hash_{f_j}$ values are the root hashes of the corresponding Merkle trees. The protocol is as follows:

- The TTP asks for multiple random block indices of the ciphertext $E_j^{(f)}$.
- The requester P_i generates and sends two Merkle hash tree proofs for each block index t requested by the TTP, as follows:

  - One proof for block c_t of $E_j^{(f)}$.
  - One proof for block f'_t of f'_j, where f'_j is the decryption of $E_j^{(f)}$ obtained using the key K_j given by the TTP.
  - She also sends c_t to the TTP.
- The TTP verifies the two proofs separately against their corresponding root hashes. Then, he decrypts the block c_t with the key K_j, and checks if the decryption is the same as f'_t. If these checks hold for every challenged block, he returns true (meaning the requester is right and the key was wrong), otherwise returns false.

Note that throughout the protocol, the TTP only learns some random blocks of the file, and some hash values of the Merkle tree. Thus, even though it is not fully privacy preserving, the protocol does not reveal the whole item either. This protocol was designed for efficiently verifying even large items without the need to transfer them to the TTP, but this privacy comes as a side benefit. Their protocols security analysis can be found in [12, 61].

Fairness of the CMFE Protocol

We give an analysis of the fairness of CMFE protocol and proof of it because it is not possible to prove the security and the fairness of the protocol if symmetric encryption scheme as AES is used for the encryption of an item. On the other hand, if the item is encrypted by a deniable encryption scheme or non-commuting encrytion scheme [27, 75], then we are able to prove the security and fairness of CMFE with the Definition 1.

Theorem 2. *Assuming that ElGamal threshold encryption scheme, the encryption scheme, and the verifiable escrow scheme are secure, all commitments are hiding and binding, the signature scheme and the payment scheme are unforgeable, the hash function is universal one way (required for Belenkiy et al. subprotocol), and the discrete logarithm problem is hard (implying sound and zero-knowledge proofs), the CMFE protocol is fair* against any PPT adversary.*

Note, first of all, that the fair* here refers to the world where either each party received every other parties' item*, or no party received any item*. Here, each item* refers to either the file or the coin of a party. Since the protocol is very similar to MFE, up to the end of Resolve 3, all fairness analysis holds here as well. The modifications, such as encrypting the file with a symmetric key and encrypting the key with ElGamal, and signatures do not affect the analysis in any important manner. The interesting addition is Resolve 4, which we analyze below.

Resolve 4 protocol guarantees that the requesting party can receive the endorsement of the coin if he does not receive the correct item. Note that, the correct item means an item with hash $hash_{f_j}$. There are two things an adversary could try to do in Resolve 4:

1. Try to put both the wrong file, and an invalid coin, thus not giving away anything to the honest party, from whom she received the correct file.
2. Try to obtain both the correct file and valid coin of an honest party.

The first case is impossible, since the endorsement is verifiably encrypted, and the Belenkiy et al. subprotocol is secure. Remember that if the adversary manages to obtain the correct file of the honest party, by the fairness of the MFE protocol, the honest party would also receive the adversary's file (correct or wrong), at the latest in Resolve 3. Thus, if the adversary sent a wrong file, the honest party can prove this via Belenkiy et al. subprotocol, and obtain the verified endorsement from the TTP, thus obtaining the adversary's coin in exchange for his own file.

As for the second case, due to the security of the Belenkiy et al. subprotocol, if the key K_j is correct, the adversary would not be able to prove it wrong to the TTP. Thus, the TTP would never decrypt the verifiably escrowed endorsement of an honest party. At a high level, such an adversary can be used to break the e-cash protocol or the universal one-wayness of the hash function (see a similar reduction in [61]).

Theorem 3. *The CMFE protocol above is fair according to Definition 1, assuming that ElGamal threshold encryption scheme and verifiable escrow scheme are secure, all commitments are hiding and binding, the signature scheme and e-cash scheme are*

unforgeable, the hash function is universal one way (required for Belenkiy et al. sub-protocol), the discrete logarithm problem is hard (implying sound and zero-knowledge proofs) and deniable or non commuting encryption scheme [27, 75] is used for the item encryption.

Proof: Again assume that adversary $\mathcal{A}$ corrupts $n-1$ parties. The simulator S simulates honest parties in the real world and corrupted parties in the ideal world. S also acts as a TTP in the protocol if any resolution protocol occurs so S publishes his public key pk as a TTP. We assume that the parties $\{P_i\}_{2\leq i\leq n}$ are corrupted parties and S simulates behavior of the honest party P_1. S does the following:

Phase 1:

S behaves exactly the same as the simulator of MFE protocol in Phase 1. Additionally, he sends unendorsed coin $coin_1^u$ together with the verifiable escrow $\tilde{V_1^{(c)}}$ that includes fake endorsement $\tilde{e_1}$ and simulates the proof in $\tilde{V_1^{(c)}}$. He labels public key pk and signature verification key v_1. Then he waits the $\{VS_i^{(c)}\}_{\{2\leq i\leq n\}}$ and $\{coin_i^c\}_{\{2\leq i\leq n\}}$ from the corrupted parties. While he is receiving $VS_i^{(c)}$ from the party P_i, he learns the endorsement in $VS_i^{(c)}$ with using simulator of the verifiable escrow. S does not process the next phase until he receives all required values.

Phase 2: S generates all encryptions $E_1^{(k)}, E_1^{(t)}, \tilde{E}_1^{(f)}$ and signature $S_1^{(1)}$ as in the Phase 2 of the CMFE protocol and sends them to the corrupted parties. Here, he encrypts a random item for $\tilde{E}_1^{(f)}$ since he does not know the correct item. Then he waits the same values from the corrupted parties and he does not execute the next step before completing it.

Phase 3:

- S behaves exactly the same as the simulator of MFE protocol in Phase 3.

- S waits the same values from the corrupted parties. While he is receiving VS_i from a corrupted party P_i, he learns the decryption shares of P_i using simulator of the verifiable escrow. If some of them does not send all the required values the

within t_1, he adds them *complaintList*. Also, if some of the corrupted parties does Resolve 1 even though it is not necessary, S replies them like TTP.

- After t_1 if S receives all in $\{VS_i\}_{2\leq i\leq n}$, it means that fairness is guaranteed because S in the real world is now able to learn all decryption share (or coin) from the corrupted parties or Resolve 2-3-4. S decrypts $\{E_i^{(k)}\}_{2\leq i\leq n}$ and learns the keys $\{K_i\}_{2\leq i\leq n}$. Then he decrypts encryptions of the items $\{E_i^{(f)}\}_{2\leq i\leq n}$. He sends all in $\{f_i\}_{2\leq i\leq n}$ to U together with $\{coin_i\}_{2\leq i\leq n\}}$. After U checks the correctness of the each item f_i using their hash values that received by Tracker. If some of them is not correct item, U sends corresponding coins to the participants in ideal world instead of item. Besides, he sends all items or coins and f_1 to S. It is sure that U outputs f_1 for the S because it comes from the honest participant(s) in the ideal world so it cannot be a coin.

  He uses different process to calculate the key K_1. He finds a K_1 such that when a party decrypts $\tilde{E}_1^{(f)}$, he finds the item f_1.

  Note that this process prevent us to use encryption schemes as AES because in this case S cannot find such a key K_1.

  S calculates his shares $\{d_i\}_{2\leq i\leq n}$ as below.

$$K_i = \frac{b_i}{d_i.a_i^{x_2}...a_i^{x_n}} \tag{4.6}$$

  S sends all d_i to all of the parties and simulates the proof of knowledge which shows that d_i is correct and the encryption $E_1^{(f)}$ and the signature $S_1^{(3)}$. Meanwhile, S waits same values from the corrupted parties. If all of them sends the correct values simulation ends.

  If some of the parties does not send decryption shares to S before t_2, S does Resolve 2 as in real protocol and clears the all *complaintList* if necessary because he has all corrupted parties' verifiable escrow.

- If some of the parties do not send Vs before t_1, S adds these parties to *complaintList* and he behaves same as the simulator of MFE in this point.. Additionally, he behaves as TTP in Resolve 4, if necessary. When a party comes for Resolve 4, S checks the same conditions and answers as TTP. He never gives a coin of him because no party pass the condition to make decrypt $V_1^{(c)}$ as in real protocol since S gave the correct decryption shares to the corrupted parties so that they have correct item f_1.

As in Claim 1, the view of adversary $\mathcal{A}$ in his interaction with the simulator S is indistinguishable from the view in his interaction with the honest party.

This completes the proof of Theorem 3.

4.4 Generalization of MFE and CMFE

We used ElGamal threshold encryption to always describe MFE and CMFE but it is not necessary to use this scheme. Instead, any type of threshold encryption scheme as Pailler cryptosystem [78], Franklin and Haber's cryptosystem [41], Damgard-Jurik cryptosystem [37] can be used. If another cryptosystem is used, then Phase 1 of MFE and CMFE should be modified according to it.

In addition, different payment system like electronic checks [29], bitcoins [74] can be used in CMFE instead of endorsed e-cash.

4.5 Analysis of MFE and CMFE

MFE: Each party P_i in MFE prepares one verifiable encryption and one verifiable escrow, and sends them to $n-1$ parties. The verification of these are efficient because the relation that they show can be proven using discrete-logarithm-based honest-verifier zero-knowledge tree-move proof of knowledge [34]. In the end, P_i sends a message that includes decryption shares to $n-1$ parties, again with an efficient proof of knowledge. So, for each party P_i, the number of messages that he sends is $O(n)$. Since there are n parties in MFE, the total message complexity is $O(n^2)$. Table 4.3

	Solution for	**Topology**	**Round Complexity**	**Number of Messages**
[47]	MPCS	Complete	$O(n^2)$	$O(n^3)$
[11]	MPCS	Complete	$O(tn)$	$O(tn^2)$
[73]	MPCS	Complete	$O(n)$	$O(n^3)$
[70]	MPCS	Complete	$O(n)$	$O(n^2)$ ✓
[4]	MFE ✓	Any ✓	$O(1)$ ✓	$O(n^3)$
Ours	MFE ✓	Any ✓	$O(1)$ ✓	$O(n^2)$ ✓

Table 4.3: Efficiency comparison with previous works in MFE. n is number of parties and t is number of dishonest parties.

shows that in comparison to the previous works, our MFE is much more efficient, obtaining **optimal asymptotic efficiency**.

When there is a malicious party or a party suffering from network failure, MFE protocol ends at the latest, immediately after t_2. In the worst case, n parties contact the TTP, so it is important to reduce his workload. TTP's duties include checking some list from his records, verifying efficient zero-knowledge proofs of knowledge from some number of parties (depending on the size of the *complaintList*), and decrypting verifiable escrows. These actions are all efficient.

Moreover, the **privacy against the TTP is preserved**. He just learn some number of decryption shares, but he cannot decrypt the encryption of exchanged items, since he never gets the encrypted items.

CMFE: CMFE is interestingly more efficient than MFE in terms of the number of verifiable escrows used at each repetition, since the parties in CMFE initially send just one verifiable escrow that includes the endorsement of coin. For the following repeated exchanges among the same set of parties, they do not prepare any verifiable escrow until some parties get coins. After phase 1, the parties prepare only *one* verifiable escrow for $n-1$ parties (compared to *two* verifiable escrows in MFE). Overall, the complexity of the number of messages is $O(n^2)$.

When we examine the TTP workload for CMFE, there is an extra resolution

protocol. The TTP does not have to decrypt whole encryption of the potentially very large item; instead he just decrypt some blocks of it to verify the key is not correct (via Belenkiy et al. subprotocol [12]). Therefore, the TTP never needs to download the whole items. Similarly to MFE, CMFE ends right after t_2 in the case of malicious party or network failure problem, in the worst case.

The privacy against the TTP is still preserved, though at a slightly lower level, because the TTP never learns the encryption of the file; rather just learns some random blocks of it. The most important property of it is that **CMFE is the first protocol that can let multiple parties in a complete topology to exchange any verifable item efficiently and fairly, even when the item cannot be efficiently verifiably encrypted**.

Chapter 5

EFFICIENTLY MAKING SMPC FAIR

5.1 Efficient Fair Secure Multi-Party Computation

In this section, we show how to adapt the MFE protocol to **any** secure multi-party computation (SMPC) protocol [14, 49, 32, 10, 88] to achieve fairness.

Assume n participants want to compute a function $\phi(w_1, ..., w_n) = (\phi_1(w_1, ..., w_n), \phi_2(w_1, ..., w_n), ..., \phi_n(w_1, ..., w_n))$ where w_i is the input and $\phi_i = \phi_i(w_1, ..., w_n)$ is the output of party P_i.

- P_i randomly chooses a share $x_i \in \mathbb{Z}_p$.
- Then P_i gives his share and w_i to an SMPC protocol that outputs the computation of the functionality ψ where $\psi_i(z_1, z_2, ..., z_n) = (E_i(\phi_i(w_1, ..., w_n)), \{g^{x_j}\}_{1 \leq j \leq n})$ is the output to, and $z_i = (w_i, x_i)$ is the input of P_i. This corresponds to a circuit encrypting the outputs of the original function ϕ using the shares provided as input, and also outputting verification shares of all parties to everyone. Encryption E_i is done with the key $g^{\sum_{j=1}^{n} x_j}$ as follows:

  $$E_i(\phi_i(w_1, ..., w_n)) = (g^{r_i}, f_i g^{r_i(\sum_{j=1}^{n} x_j)})$$

  where $r_i \in \mathbb{Z}_p$ are random numbers chosen by the circuit (or they can also be inputs to the circuit), similar to the original MFE protocol.

  It is expected that everyone learns the output of ψ before a fair exchange occurs. If some party did not receive his output at the end of the SMPC protocol, then they do not proceed with the fair exchange, and hence no party will be able to decrypt and learn their output.
- If everyone received their output from the SMPC protocol, then they execute the Phase 3 of the MFE protocol above, using g^{x_i} values obtained from the output

of ψ as verification shares, and x_i values as their secret shares. Furthermore, the a_i values are obtained from E_i as the first part of the pair, corresponding to g^{r_i}.

Note that each function output is encrypted with all the shares. But, for party P_i, she need not provide her decryption share for f_i to any other party. Furthermore, instead of providing n decryption shares to each other party as in a complete topology, she needs to provide only one decryption share, $a_j^{x_i}$, to each P_j. Therefore, the Phase 3 of MFE needed here is a much more efficient version. Indeed, the verifiable escrows, the decryption shares and their proofs each need to be only on a *single* value instead of n values.

Phases 1 and 2 of the fair exchange protocol have already been done during the modified SMPC protocol, since the parties get the encryption of the output that is encrypted by their shares. Since the SMPC protocol is secure, it is guaranteed to output the correct ciphertexts, and we do not need further verification. We also do not need to first commit to x_i values, since the SMPC protocol ensures independence of inputs as well. So, the parties only need to perform Phase 3.

In the end of the exchange, each party is able to decrypt only their own output. This is both because the circuit outputs only their encrypted output to them, and because they do not give away their own output's decryption share to anyone else. Indeed, if a symmetric functionality is desired for SMPC, $\psi(z_1, z_2, ..., z_n) = (\{E_i(\phi_i(w_1, ..., w_n)), g^{x_i}\}_{1 \leq i \leq n})$ may be computed, and since P_i does not give the decryption share of f_i to anyone else, each party will still only be able to decrypt their own output. Therefore, *a symmetric functionality SMPC protocol may be employed to compute an asymmetric functionality fairly* using our modification. Note also that we view the SMPC protocol as black box.

Our overhead over performing *unfair* SMPC is minimal. Even though the input and output sizes are extended additionally by $O(n)$ inputs and the circuit is extended to perform encryptions, these are minimal requirements, especially if the underlying SMPC protocol works over *arithmetic circuits* (e.g., [10, 88]). In such a case, per-

forming ElGamal and creating verification values g^{x_i} are very easy. Afterward, we only add two rounds of interaction for the sake of fairness (i.e., Phase 3 of MFE, with smaller messages). Moreover, all the benefits of our MFE protocol apply here as well.

5.2 Full Simulation Proof of Fair SMPC

Theorem 4. *The SMPC protocol above is fair and secure according to Definition 1 for the functionality ϕ, assuming that ElGamal threshold encryption scheme is a secure threshold encryption scheme and the underlying SMPC protocol that computes functionality ψ is secure.*

Proof: Assume that adversary $\mathcal{A}$ corrupts $n-1$ parties, which is the worst possible case. The simulator S simulates the honest party in the real world and the corrupted parties in the ideal world. We assume that the parties $\{P_i\}_{2\leq i\leq n}$ are corrupted parties and S simulates behavior of the honest party P_1. Since the protocol is symmetric, this assumption is there just for the sake of easier presentation of the proof. S does the following:

- Simulator S chooses a random input $\tilde{w}_1$ and share x_1. Then S behaves like the simulator of the underlying SMPC protocol; call that simulator S'. He learns the input of the corrupted parties while he is acting as S', and evaluates the circuit together with the corrupted parties, using $\tilde{w}_1, x_1$ as its input.

  We view S' almost as a black box. The only different behavior we request from S' is that, instead of providing the inputs of the corrupted parties he learned to the universal trusted party U and learning their outputs immediately, we need S' finish its simulation without contacting U and instead using its random input $\tilde{w}_1, x_1$.

  The reason we call this change almost black box is that it will be indistinguishable to the adversary due to the security of the ElGamal threshold encryption scheme. Even though the simulator computed different outputs for the original

functionality ϕ, since the computed functionality ψ outputs encrypted values, these will be indistinguishable. During MFE, the simulator S will learn the real outputs, and fake its share such that the decryptions of ψ_i outputs would result in actual ϕ_i outputs obtained from U. The output verification shares are distributed identically, and thus there is no problem there as well.

We need S' not to contact U due to the fairness simulation requirement of Definition 1. Note that the simulator S is allowed to learn the outputs from U only once it is guaranteed that all parties will learn their outputs.

- Once S' is done, all parties learned some encrypted output together with verification shares. At this point, these encryptions contain random values. Now S continues simulating the Phase 3 of MFE. If during that simulation, it is guaranteed that all parties will learn their outputs, then he contacts U, providing the inputs of the corrupted parties S' extracted, and obtains the outputs ϕ_i from U. Accordingly, S calculates decryption shares d_i for each corrupted party using the equation (4.3). Then he sends (differently from MFE proof) only d_i to P_i.

The behavior of S is indistinguishable since he acts as a simulator of SMPC and MFE. This completes the proof. Table 5.1 compares our fair SMPC solution with others. Our advantage is both efficiency and having no requirement for an external payment mechanism.

5.3 Analysis of Fair SMPC:

The overhead of our fairness solution on top of an existing unfair SMPC protocol is increased input size, and additional computation of encryptions and verification shares. If an arithmetic circuit is used in the underlying SMPC protocol [10, 88, 32], then there are only $O(n)$ additional exponentiations required, which does not extend circuit size a lot. If boolean circuits are used, the size of the circuit increases more than

Solutions	**Technique**	**TTP**	**Number of Rounds**	**Broadcast**	**Proof Technique**
[46]	Gradual Release	No	$O(\lambda)$	Yes	NFS
[15]	Bitcoin	Yes	Constant ✓	Yes	NFS
[2]	Bitcoin	Yes	Constant ✓	Yes	NFS
Ours	MFE	Yes	Constant ✓	No	FS ✓

Table 5.1: Comparison of our fair SMPC solution with previous works. NFS indicates simulation proof given but not for fairness and FS indicates full simulation proof including fairness (λ is the security parameter).

arithmetic circuit would have, but still tolerable, especially considering in comparison to related work.

Table 5.1 shows the comparison of our fair SMPC protocol with existing work. SMPC protocols require many heavy computations for security issues. Therefore it is important to add a minimum workload to SMPC protocol to achieve fairness. [46] uses gradual release for fairness. However, this brings many extra rounds and messages to the protocol. Each round each party releases his item by broadcasting it. Recent, bitcoin-based approaches [15, 2] also require broadcasting in the bitcoin network, which increases message complexity. Our only overhead is a constant number of rounds, and $O(n^2)$ messages. Besides, we prove security and fairness together with using our new ideal/real simulation definition so our fair SMPC protocol has stronger security than the other previous works since we show importance of our proof technique in Section 6.6.

Chapter 6

EFFICIENTLY MAKING SECURE 2PC FAIR

Since two party computation is a sepecial case of multi-party computation, the solution in Chapter 5 can be used to achieve fairness in a secure two party computation protocol too. However, it is not necessary to use previous framework for the simpler case because we can give more efficient solution for the two-party case. Therefore, we introduce a new framework in this chapter for only 2PC to achieve fairness.

6.1 Notation for Two Party Computation

The garbled circuit that is generated by Alice is GC^A and the one generated by Bob is GC^B. We use A and B as a superscript of any notation to show to which garbled circuit they belong to. For simplicity of the presentation, assume Alice and Bob have l-bit inputs and outputs each. Alice's input bits are $x = \{x_1, x_2, ..., x_l\}$ and Bob's input bits are $\{y = y_1, y_2, ..., y_l\}$.

When we say Alice's input wires, it means that Alice provides the input for these wires. Similarly, Alice's output wires means that these wires correspond to the result of the computation learned by Alice. Bob's input and output wires have the matching meaning.

We have three kinds of gates: *Normal Gate*, *Input Gate*, and *Output Gate*. An *Input Gate* is a gate that has an input wire of Alice or Bob. Similarly, an *Output Gate* is a gate that has a wire of Alice's or Bob's output. A *Normal Gate* is a gate that is neither *Input Gate* nor *Output Gate*.

We use sets to group wires and gates. The notation G_I denotes the set of all *Input Gates*, and similarly G_O is for the set of *Output Gates*. The set W_O includes the output *wires* of all *Output Gates* and the set W_I includes the input wires of all

Input Gates. The set of all wires is denoted with W.

Consider some notation $k^A_{w_b}$ in the protocol. The superscript A means that it belongs to the garbled circuit GC^A. w shows to which wire set it belongs. If $w \notin W_O \cup W_I$, b shows that it is the *right-side* wire. Instead of b, if we use a, it shows it is the *left-side* wire. For other wire sets, b shows that it is Bob's input or output wire. If it is a, it shows Alice's input or output wire. Similarly, W_{I_A} and W_{O_A} is the set of input and output wires of Alice, respectively. W_{I_B} and W_{O_B} are Bob's input and output wire sets.

Any small letter notation that have 0 or 1 in its subscript shows that it represents the bit 0 or 1. The capital letters are used for notation of the commitments, and if they have 0 or 1 in their subscript, it shows that they are for some value that represents 0 or 1. E_k shows an encryption with the key k. Therefore, $E_{k_1}E_{k_2}(m_1, m_2)$ means that m_1 and m_2 are both encrypted by the two keys k_1 and k_2.

6.2 Overview of the Fairness Principles

Remember that Yao's protocol is secure against *semi-honest* adversaries. When one considers *malicious* adversaries, care must be taken against the possible actions of the evaluator (E) and the constructor (C):

1. C can construct a circuit that does *not* evaluate the desired function. For example, he can construct a circuit computing $f(x, y) = y$, where x is his input and y is E's input, violating E's privacy.
2. C can use one *wrong* key and one *correct* key in the oblivious transfer phase where E learns his input keys from C. If E aborts, then C infers that E's input bit corresponded the wrong key.
3. E can learn the output without sending it to C, causing *unfairness*.

The first two problems are about malicious behavior, and the last one is about fairness. Solutions to the first two problems are not our contribution; we employ previous work. Our fairness framework can be applied on top of any existing secure two-party

computation solution that is secure against malicious adversaries. For concreteness of presentation, we show our framework as applied to Jarecki and Shmatikov construction [58]. They solve the first problem via *efficient zero-knowledge proofs*, and the second problem via *committed oblivious transfer* (COT) [58, 59]. Note that we can use our fairness extensions on top of cut-and-choose based techniques [86, 64, 81, 66] as well.

Failed Approaches: In the beginning it looks like adding *fairness* in a secure two party computation protocol with TTP does not need a lot of work. However, we should be careful because it can break the *security* of the protocol. Consider a very simple solution regarding constructor C, evaluator E, and the TTP. Assume that C constructs the circuit such that the output is not revealed directly, but instead the output of the circuit is an encrypted version of the real output, and C knows the key. Thus, after evaluation, E will learn this encrypted output, and C and E need to perform a fair exchange of this encrypted output and the key. However, there are multiple problems regarding this idea:

1. C can add wrong encryptions for the outputs of E, and hence E can learn wrong output result. The wrong encryptions can leak information about E's output value to C according to the abortion of E.

2. When there is dispute between E and C, and E goes to TTP for resolution, she cannot efficiently prove to the TTP that she evaluate the C's garbled circuit correctly.

Therefore, E needs to be a constructor too, to remove C's advantage. Tuhs, the idea is to make C and E construct their own circuits and then evaluate the other's circuit [79]. The following problems occur in the two-constructor case:

1. One of the parties can construct a circuit for a different functionality and so the other party may learn a wrong output.

2. Even when both E and C construct the same circuit, they can cause the other one to learn a wrong output by using different inputs for the two circuits.

3. If C and E can check their inputs are the same for both circuits, according to the two outputs from two circuits being equal or not, the malicious party can infer more information.

Besides the problems above, both C and E have to prove that they act correctly during the protocol to learn output results from the TTP when there is dispute. These proofs should be efficient to reduce the workload on the TTP. In addition, the TTP should not learn anything about the inputs and outputs of the computation, as well as the identities of the participants. **Our solution achieves efficiency of and privacy against the TTP**.

Our Solution: We show how to efficiently add fairness property to any secure two party computation protocols with using our principles. Mainly, we explain the principles via the two party computation protocol in [58]. We give almost equal responsibilities to both players Alice and Bob, meaning that Alice acts as both C and E, and similarly for Bob. There will be two garbled circuits: one that Alice constructs and Bob evaluates, and another that Bob constructs and Alice evaluates. Alice and Bob make sure that they both construct the same garbled circuit by using efficient sigma-proof-based zero-knowledge proofs of knowledge as [58]. Remember that cut-and-choose may be used instead, but for the sake of simple presentation, we stick to a particular method. In addition, they show that they use the same inputs for both circuits by employing an *equality test* for inputs (see Section 3.3). Briefly, a secure two party protocol becomes fair as follows:

1. [**Agreement to the Common Knowledge**] Alice and Bob learn the public key of the TTP, and Alice sends her verification key for the signature scheme to Bob. Alice and Bob jointly generate four random elements that are called *equality test constants*. They are designated for the left wire being 0 or 1 and the right wire being 0 or 1.

2. [**Construction of the Garbled Circuit**] Alice and Bob agree on a circuit that evaluates the desired functionality. They first generate two keys for each wire representing 0 and 1, and then commit to them. Afterward, they randomly choose *input gate numbers* for each input gate to generate two *randomized equality test numbers* based on equality test constants for each wire of each input gate. According to their base, randomized equality test numbers represent 0 and 1. Then Alice commits to Bob's input wires' randomized equality test numbers in her circuit, and Bob commits to Alice's input wires' randomized equality test numbers in his circuit. In addition, they choose *random row pairs* for rows of the garbled output gates and commit to them.

   Then, they both construct their garbled circuits separately in the same way as in Yao's protocol [87]. The construction of the input and output gates differ, though. Each row of the garbled *input* table has two encryptions. One of them is an encryption of the gate's output key, and the other one is an encryption of the decommitment of the randomized equality test numbers of the corresponding evaluator's wires. It lets them later prove that the inputs in both circuits are the same. In construction of garbled *output* tables, they encrypt random row pairs instead of output bits. This makes sure that the evaluator cannot learn the output directly, but must perform a fair exchange with the constructor.

   Note that if the underlying protocol uses cut-and-choose techniques, Alice and Bob do not commit to each key. Instead, they generate $O(s)$ circuits, and also different randomized equality test numbers, random row pairs for each circuit, where the gates are constructed in same way as explained above.
3. [**Committed Oblivious Transfer (COT)**] Alice and Bob employ the COT functionality $\mathcal{F}_{\mathcal{COT}}$ [58] for each circuit. In one case the sender is Alice, and in the other the sender is Bob. This way, they learn the keys associated with their input bits, in the garbled circuit the *other* party constructed.

   Note that cut-and-choose oblivious transfer [66] can be used here too.
4. [**Send, Prove and Evaluate**] Alice sends all the necessary values to Bob for

the evaluation of Alice's garbled circuit: all garbled tables, the key commitments, the keys of Alice's input bits, the randomized equality test numbers of Alice's input, and her signatures on the commitment of random row pairs. Similarly Bob sends to Alice the corresponding values for his garbled circuit, except the signatures (Bob does not sign anything in our protocol). Then Alice and Bob each proves that they acted honestly up to this point by showing correctness of the construction and commitments with random values. These proofs are necessary, since otherwise one party aborting due to a mis-constructed gate may let the constructor learn information about the other party's private input. If all proofs are valid, then they evaluate the garbled circuits and learn random row pairs of the other party's output wires. Otherwise, they abort the protocol. If a cut-and-choose based protocol is used, only the way that the proof that shows correctness of the garbled circuit is different. It is done by explicitly revealing and checking about half of the circuits.

5. [**Equality Test**] They execute the equality test in Section 3.3 to learn if they both used the keys that represent the same bit for the input gates. For this purpose, they use the randomized equality test numbers that they learned from evaluation. In the end of evaluation, each party has a pair (representing the right and left wires) of randomized equality test numbers, which represents the same bits as the keys used, for each input gate. Therefore if Alice and Bob used the same input in both garbled circuits, then the equality test succeed. If the test is not successful, then they abort. Otherwise they continue and Alice signs a message that shows equality test is successful, and sends it to Bob.
6. [**Verifiable Escrow**] Alice sends a verifiable escrow to Bob that is encrypted with the TTP's public key. The verifiable escrow includes random row pairs of Bob's output gates that Alice learned from evaluation of Bob's garbled circuit. These verifiable escrow proofs show that Alice escrowed the correct output of Bob, since she learns correct random row pairs only after correct evaluation of the circuit. If there is problem in verification, then Bob aborts. Otherwise, he

Name	**Form**	**Relation**
Equality Test Constants	$e = g^{\rho}$	There are four kind of them where each represents 0 or 1 and right or left.
Input Gate Randoms	u	Each input gate has it. They are private just known by the constructors.
Randomized Equality Test Numbers	$m = e^{u}$	For each input gate u, there are four kind of them where each represents 0 or 1 and right or left according to e.
Random Row Pair	(δ, ε)	They are randomly chosen pairs that represent each row of the garbled output gates.

Table 6.1: Review of random numbers for fair 2PC framework

continues.

7. [**Output Exchange**] Firstly, Bob sends the random row pairs of Alice's output gates to Alice. Because, Bob already has a verifiable escrow, and can resolve with the TTP should Alice choose not to respond. Alice checks if the random row pairs that Bob sent are correct. If they are not correct, she executes *Alice Resolve* with the TTP. Otherwise Alice sends random row pairs of Bob's output gates to Bob. Then Bob checks if the random row pairs that Alice sent are correct. If Alice does not respond or random row pairs are not correct, Bob executes *Bob Resolve* before the timeout. If something wrong goes on after the verifiable escrow phase, resolutions with the TTP guarantee fairness.

6.3 Making a Secure Protocol Fair

Alice and Bob will evaluate a function $f(x, y) = (f_a(x, y), f_b(x, y))$, where Alice has input x and gets output $f_a(x, y)$, and Bob has input y and gets output $f_b(x, y)$, and for simplicity of the presentation we have $f : \{0,1\}^l \times \{0,1\}^l \rightarrow \{0,1\}^l \times \{0,1\}^l$, where l is an integer.

All commitments are Fujisaki-Okamoto commitments [44, 35] unless specified otherwise, and all encryptions are simplified Camenish-Shoup (sCS) encryptions (see Section 3.3). We give a review of the random numbers that are used for fairness in Table 6.1. The protocol steps are described in detail below (underlined parts are directly from previous work [58]):

1. [**Agreement to the Common Knowledge**] The TTP generates a safe RSA modulus n, chooses a random element $g_s = (g')^{2n}$ where $g' \in \mathbb{Z}_{n^2}^*$ (g_s is for sCS encryption scheme). Then TTP selects random high-order elements $g, h \in_R \mathbb{Z}_n^*$. Then, he announces his public key $PK_{TTP} = (n, g_s, g, h)$.

   Alice generates private-public key pair (sk_A, vk_A) for the signature scheme. She shares public signature verification key vk_A with Bob.

   Lastly, they jointly generate four equality test constants $e_{a,0}, e_{a,1}, e_{b,0}$ and $e_{b,1}$. For joint generation, Bob first commits to four random elements from $\mathbb{Z}_n$ and proves knowledge of the committed values and their ranges [21, 17]. Then, Alice chooses four random elements from the same range and sends them to Bob. Finally, Bob decommits. Then they both calculate the multiplication of pairs of these numbers, in mod $\mathbb{Z}_n$, and so jointly agree on random numbers $\rho_{a,0}, \rho_{a,1}, \rho_{b,0}, \rho_{b,1}$. In the end they calculate equality test constants where $\{e_{a,t} = g^{\rho_{a,t}} \ mod \ n, e_{b,t} = g^{\rho_{b,t}} \ mod \ n\}_{t\in\{0,1\}}$. These numbers represent 0 and 1 for the left (a) and right (b) wires of the gates in G_I (input gates).

2. [**Construction of the Garbled Circuit**] Alice and Bob agree on the circuit that computes $f(x, y)$. Then, they begin to construct their garbled circuits separately. The construction is quite similar for Alice and Bob. Therefore, we give details of the construction through Alice. When there is an important difference, we also show Bob's way of doing it.

   Commitments for keys: Alice generates keys $\{k^A_{w_z,t}\}_{z\in\{a,b\},t\in\{0,1\},w\in W\setminus W_O}$ representing left and right and 0 and 1 for each wire except the output gates'

output wires. Then, she computes sCS commitments $\{C^A_{w_z,t}\}_{z\in\{a,b\},t\in\{0,1\},w\in W\setminus W_O}$ for each key as in [58].

Commitments for fairness: Alice chooses randomly one input gate number $u^A_g \in \mathbb{Z}_n$ for each input gate $g \in G_I$. Then she calculates the randomized equality test numbers m by raising equality test constants e to these random input gate numbers u as $\{m^A_{w^g_z,t} = e^{u^A_g}_{z,t} \ mod\ n\}_{g\in G_I,z\in\{a,b\},t\in\{0,1\}}$, where w^g is the wire of gate g. Here, each m value represents left or right wire and values 0 or 1 according to equality test constants used as a base. She prepares two commitments to the input gate number u^A_g for each right wire w^g_b (which corresponds to Bob's side) as $\{D^A_{w^g_b,t} = e^{u^A_g}_{b,t} h^{s_{w^g_b,t}} = m^A_{w^g_b,t} h^{s_{w^g_b,t}}\}_{g\in G_I,t\in\{0,1\}}$, where $s \in \mathbb{Z}_n$ is a random value picked to hide the committed value.

Bob does the same for GC^B, except he commits to the left (Alice's side) wires as $\{D^B_{w^g_a,t} = e^{u^B_g}_{a,t} h^{s_{w^g_a,t}} = m^B_{w^g_a,t} h^{s_{w^g_a,t}}\}_{g\in G_I,t\in\{0,1\}}$.

Remark: There are randomized equality test numbers, which represent 0 and 1 for the right and left wires, for each input gate, but only the evaluator's side ones are committed since the evaluator needs to learn one randomized equality test number that corresponds to his input. These numbers are required later to check if the same input is used for both circuits in the evaluation. Hence, the name is "equality test numbers".

Note that there can be just one input wire of a gate (e.g., NOT gate for negation). In this case there will be two randomized equality test numbers which represent 0 and 1 for this gate. Alternatively, they can agree to construct a circuit using only NAND gates.

Normal Gates: As in the Yao's protocol [87], she begins to construct the garbled tables. First, she picks permutation pairwise bits φ^A_a and φ^A_b for each gate to permute the garbled table. Then she prepares the double encryptions of the keys according to permutation order (see Table 6.2).

Input Gates: The difference in the construction of a *Normal Gate* and an

Input Gate is the following: Alice also doubly-encrypts the partial decommitment of $D^A_{w^g_b,t}$, which is $s_{w^g_b,t}$, for each row, in a corresponding manner. In Bob's construction, he adds double-encryption of the partial decommitment of $D^B_{w^g_a,t}$, which is $s_{w^g_a,t}$, for each row.

Remark: Alice and Bob just encrypt partial decommitments because they only need to learn randomized equality test numbers (m values) that represents their input bits and we do not want to reveal input gate numbers (u values) since it may cause the evaluator to learn equality test constants that correspond to the constructor's input wires representing constructor's input bits.

Output Gates: The garbled tables of *output gates* are constructed differently as well. Alice chooses random row pairs $(\delta^A_i, \varepsilon^A_i)$, each from $\mathbb{Z}_n$, for each row of all output garbled tables, where $i \in \{1, ..., 8l\}$. Then, she commits to all pairs as $S^A_1, ..., S^A_{8l}$ (four commitments per output gate, assuming binary gates). Finally, she can construct the garbled output gates as a normal gate where the difference is that, instead of encryption of a key, there is encryption of random row pairs. In GC^B, similarly Bob encrypts the random row pairs and prepares commitments $S^B_1, ..., S^B_{8l}$. These commitments will be used to prove correct construction of the output gates.

See Table 6.2 for construction details of output and input gates.

Row	t_a, t_b	Output Bit (t)	Garbled Input Gate	Garbled Output Gate
00	$(0 \oplus \varphi^A_a), (0 \oplus \varphi^A_b)$	$(t_a \vee t_b)$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (k^A_{w_{o,t}}, s_{w^g_b,t})$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (\delta^A_i, \varepsilon^A_i)$
01	$(0 \oplus \varphi^A_a), (1 \oplus \varphi^A_b)$	$(t_a \vee t_b)$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (k^A_{w_{o,t}}, s_{w^g_b,t})$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (\delta^A_{i+1}, \varepsilon^A_{i+1})$
10	$(1 \oplus \varphi^A_a), (0 \oplus \varphi^A_b)$	$(t_a \vee t_b)$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (k^A_{w_{o,t}}, s_{w^g_b,t})$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (\delta^A_{i+3}, \varepsilon^A_{i+2})$
11	$(1 \oplus \varphi^A_a), (1 \oplus \varphi^A_b)$	$(t_a \vee t_b)$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (k^A_{w_{o,t}}, s_{w^g_b,t})$	$E_{k^A_{w_{a,t_a}}} E_{k^A_{w_{b,t_b}}} (\delta^A_{i+3}, \varepsilon^A_{i+3})$

Table 6.2: Garbled Input and Output Gates for an OR gate constructed by Alice.

3. [**Committed Oblivious Transfer (COT)**] There are two runs of the COT protocol: one where Alice is the sender and Bob is the receiver, and vice versa. They execute the following for each input bit of the receiver. In the case that Alice is the sender, Bob gives his input bit y_w to the functionality $\mathcal{F}_{\mathcal{COT}}$ [58] and Alice gives the keys $k_{w_b,0}, k_{w_b,1}$ for Bob's input wire to $\mathcal{F}_{\mathcal{COT}}$. Afterward, $\mathcal{F}_{\mathcal{COT}}$ gives the key k_{w_b,y_w} that represents his input bit y_w to Bob. The functionality ensures that the keys match the committed values. When the Bob is the sender, a similar process is executed. As usual, cut-and-choose based techniques can also be applied here.

4. [**Send, Prove, and Evaluate**] In this phase, Alice and Bob send all the necessary information to evaluate the garbled circuits and prove those are built correctly.

   Sending Fairness Values: Alice sends the input gate number commitments $\{D^A_{w_b^g,t}\}_{w_b \in W_{I_b}, g \in G_I, t \in \{0,1\}}$, the randomized equality test numbers of her input wires $\{m^A_{w_a^g,x_w}\}_{g \in G_I}$, and the commitments of random row pairs with their signatures $\{S^A_i, sign_{sk_A}(S^A_i)\}_{i \in \{1,\ldots,4l\}}$. Bob sends the input gate number commitments $\{D^B_{w_a^g,t}\}_{w_a \in W_{I_A}, g \in G_I, t \in \{0,1\}}$, his input's randomized equality test numbers $\{m^B_{w_a^g,y_w}\}_{g \in G_I}$, and the commitments of random row pairs $\{S^B_i\}_{i \in \{4l+1,\ldots,8l\}}$, without signatures.

   Sending Evaluation Values: Alice sends the garbled circuit GC^A to Bob along with the key commitments $\{C^A_{w,t}\}_{t \in \{0,1\}, w \in W \setminus W_O}$, and the keys $\{k^A_{w,x_w}\}_{w \in W_{I_A}}$ that represent Alice's input bits x_w. Similarly, Bob sends the garbled circuit GC^B to Alice along with the key commitments $\{C^B_{w,t}\}_{t \in \{0,1\}, w \in W \setminus W_O}$, and the keys $\{k^B_{w,y_w}\}_{w \in W_{I_B}}$ that represents Bob's input bits y_w.

   After exchanging all garbled values, Alice and Bob prove to each other that they performed the construction and the COT phase honestly. Alice runs the subprotocols as a prover (see Section 6.4):

- *Proof of Garbled Construction* to prove that garbled tables are constructed properly (as in [58]).
- *Proof of Correct Randoms* to prove garbled input gates contain correct decommitment values.
- *Proof of Output Gates* to prove that garbled output gates encrypt the committed random row pairs.

In the case that Bob is the constructor, he runs the same subprotocols as the prover (just replace the names). Such proofs can also be provided via cut-and-choose techniques. If there is a problem in the proofs or the signature verification, they abort. Otherwise, they each evaluate the garbled circuit they were given. At the end of the evaluation, they just learn $2l$ random row pairs, l of them are for Alice's output gates and the other l of them are for Bob's output gates.

Remark: Up to now, the main idea was to prove somehow (via zero-knowledge or cut-and-choose) that the garbled circuits were prepared honestly. Apart from our fairness additions, the underlying protocol remains the same. The evaluation is performed as usual. Note that the constructor does not need to prove the randomized equality test numbers of *her own input wires*, since she has to send the correct ones to be successful in the equality test.

5. [**Equality Test**] *This part is necessary to test whether or not Alice and Bob used the same input bits for both circuit evaluations.* We use the idea by Boudot et al. [18]. There are fair and unfair versions of the equality test in [18], but the unfair version is sufficient for our purpose.

   Alice and Bob want to check, for the input gates in G_I, if $t'_b = t_b$ and $t'_a = t_a$ for the encryptions $E_{k^A_{w_a,t_a}}(E_{k^A_{w_b,t_b}}(k^A))$ and $E_{k^B_{w_a,t'_a}}(E_{k^B_{w_b,t'_b}}(k^B))$, such that the first one was decrypted by Bob and the second one was decrypted by Alice. For this purpose, we will use the randomized equality test numbers

$\{m^A_{w^g_a,t}, m^A_{w^g_b,t}, m^B_{w^g_a,t}, m^B_{w^g_b,t}\}_{g\in G_I, t\in\{0,1\}}$.

Assume Alice decrypted a row for an input gate and learned two randomized equality test numbers $m^B_{g_a}, m^B_{g_b}$ for the input gate g. Also suppose Bob decrypted the same garbled gate and similarly learned $m^A_{g_a}, m^A_{g_b}$. If both used the same input bits for both GC^B and GC^A, then we expect to see the following equation is satisfied:

$$(m^B_{g_a} m^B_{g_b})^{u^A_g} = (m^A_{g_a} m^A_{g_b})^{u^B_g} \tag{6.1}$$

The left hand side of the equation (6.1) is composed of values Alice knows since she learned $m^B_{g_a}, m^B_{g_b}$ values from the evaluation of GC^B and generated u^A_g in the phase 2 of the protocol, and the right hand side values are known by Bob since he learned $m^A_{g_a}, m^A_{g_b}$ from the evaluation of GC^A and generated u^B_g in the phase 2 of the protocol. This equality should hold for correct values since $m^B_{g_a} = e^{u^B_g}_{w_a,t'_a}$, $m^B_g = e^{u^B_g}_{w_b,t'_b}$ and $m^A_{g_a} = e^{u^A_g}_{w_a,t_a}$, $m^A_{g_b} = e^{u^A_g}_{w_b,t_b}$.

After computing their side of these equalities for each input gate, they concatenate the results in order to hash them, where the output range of the hash function is $\mathbb{Z}_q$, where q is a large prime, which is the order of the prime-order group used for the subprotocol *Proof of Equality* in Section 6.3. Then Alice and Bob execute this subprotocol with their calculated hash values as their inputs.

If the equality test succeeds, Alice signs the equality test result ($s_{eq} = sign_{sk_A}(equal)$) and sends it to Bob. If the signature verifies, they continue with the next step.

Remark: Remember that the constructor proved that all randomized equality test numbers in the encryptions are correct and each of them is a randomization of different equality test constants (see Section 6.4.4). However, she did not prove that she added it to the correct row of the encryption table. Suppose that the constructor encrypted the randomized equality test number that represents 0 where the evaluator's encryption key represents 1. In this case, it is sure that the equality test will fail, but the important point is that the constructor *cannot*

understand which row is decrypted by the evaluator, and thus does not learn any information because he cannot cheat just in one row. If he cheat in one row, he has to change one of the row as well otherwise he failed in "Proof of Garbled Construction". Thus, even if the equality test fails, the evaluator might have decrypted any one of the four possibilities for the gate, and thus might have used any input bit. This also means that the equality test can be simulated, and hence reveals nothing about the input.

6. [**Verifiable Escrow**] Alice creates the verifiable escrow by encrypting the random row pairs using the TTP's public key as $V = VE_{TTP}((\delta^B_{g_1^{i_1}}, \varepsilon^B_{g_1^{i_1}}), ..., (\delta^B_{g_l^{i_l}}, \varepsilon^B_{g_l^{i_l}}); vk_A, time)$. Here, g_j^i means that the j^{th} garbled output gate from G_O and i^{th} row of that gate. The part after the semicolon (;) is the public non-malleable label of the escrow. Then, she sends V to Bob. She proves that there are l different decommitments in the escrow that correspond to l of the commitments $S_1^B, ..., S_{4l}^B$ [56, 33, 19]. Since Alice can just decrypt one row for every gate, because she only has one pair of keys for each gate, this proof shows that Alice decrypted Bob's garbled output tables correctly, and the verifiable escrow has the information about which row was decrypted. If V fails to verify, or if the public signature verification key was different from what was used in the previous steps, or the *time* value is not what Bob was expecting (see [61]), then Bob aborts. Else, Bob continues with the next step.

   Remark: We can replace the l out of $4l$ proof above with l proofs that are simply 1 out of 4 each (see [34]). To ensure each 1 out of 4 proof is for a different gate, Bob can use different commitment bases per gate. In this case, TTP's public key for the protocol extends to $O(l)$, and Alice proves, l times, for each output gate of Bob, that she knows one of the decommitments from the 4 commitments of the random row pairs for that gate.

7. [**Output Exchange**] In this part, both Alice and Bob exchange the outputs. Remember that the outputs are indeed randomized, and only the constructor

knows their meaning. Thus, if they do not perform this fair exchange, no party learns any information about the real output (unless they resolve with the TTP, in which case they would learn their outputs). **At this point in the protocol, it is guaranteed that the protocol will be fair**.

First, Bob sends the random row pairs that he learned for Alice's output gates in GC^A to Alice, since he already has a verifiable escrow as insurance. If Alice does not receive a response from Bob, then she runs *Alice Resolve*. Otherwise, she checks if they are indeed random row pairs that she generated. If they are correct, she learns the rows, equivalently the output bits, that correspond to her output.

Then she sends Bob's random row pairs obtained from GC^B to Bob. Bob checks if the random row pairs are correct for his output wires. If they match his garbled construction, then Bob learns his output. If at least one of them is not matching or if Bob does not receive an answer from Alice (until some network timeout), he runs *Bob Resolve*.

Remark: Since both garbled circuits are generated for the same functionality and evaluated using the same inputs, the random row pairs reveal the output. Since Alice only learns her random row pairs in her circuit, she only learns her output and not Bob's. Similar argument applies to Bob's case.

6.4 Sub Protocols

6.4.1 Resolutions with the TTP

Bob Resolve: Bob has to contact the TTP before some well-specified timeout to get an answer [5, 61]. Considering he contacts before timeout (verified using the *time* in the label of the verifiable escrow V), he first sends the verifiable escrow V of Alice, the equality signature s_{eq} to show that equality test was successful, and Alice's l random row pairs $(\delta_i^A, \varepsilon_i^A)$ that he learned from circuit evaluation along with the commitments and signatures of the commitments of these pairs $\{S_i^A, sign_{sk_A}(S_i^A)\}$ to

prove that they are the correct representation of the decrypted row for the output values of Alice. The TTP checks if all signatures were signed by Alice, using the signature verification key vk_A of Alice in the verifiable escrow's label, and that the decommitments are all correct. If there is no problem, then the TTP decrypts the verifiable escrow V and sends the values inside to Bob. Since Bob knows which output wire he put these values in the garbled circuit he constructed, he effectively learns the output. The TTP remembers Alice's random row pair values, given and proven by Bob above, in his database.

Alice Resolve: When Alice contacts the TTP, she asks for her random row pairs. If the TTP has them, then he sends all random row pairs, which have already been verified in Bob Resolve, to Alice. If Bob Resolve has not been performed yet, then Alice should come after the timeout. When Alice connects after the timeout, if the TTP has the pairs, then he sends them to Alice. Otherwise she will not get any output, but she can rest assured that Bob also cannot learn any output values: The protocol is aborted. If Alice obtains random row pairs from the TTP, she can learn her real output results since these random row pairs uniquely define rows of the output gates of the garbled circuit she constructed.[1]

Note that the **TTP learns nothing** about the protocol, since he only sees random row pairs, which are meaningless unless the corresponding bits are known, and a signature. Furthermore, the signature key can be specific to each construction, and thus we do *not* require a public key infrastructure, and it does not give away Alice's identity. Hiding identities and handling timeouts in such resolution protocols are easy, as done by previous work [5, 61].

6.4.2 Proof of Garbled Construction

This proof will be executed for each gate g in the circuit. For gate g, say that the right wire is a, the left wire is b, and the output wire is c. Let $E_{00}, E_{01}, E_{10}, E_{11}$ be the encryptions in the garbled table of g.

Inputs: Common inputs are the commitments to the wire keys $C^A_{w_a,0}, C^A_{w_a,1}, C^A_{w_b,0}$,

$C^A_{w_b,1}, C^A_{w_o,0}, C^A_{w_o,1}$, the garbled gate table of g that has encryptions $E_{00}, E_{01}, E_{10}, E_{11}$, and the commitments $D^A_{w_b,0}, D^A_{w_b,1}$. The prover's private inputs are all the decommitments.

Prover performs following proof systems:

- $\boldsymbol{ZKNotEq}(C^A_{w_a,0}, C^A_{w_a,1}) \wedge \boldsymbol{ZKNotEq}(C^A_{w_b,0}, C^A_{w_b,1}) \wedge \boldsymbol{ZKNotEq}(C^A_{w_o,0}, C^A_{w_o,1})$, where $\boldsymbol{ZKNotEq}(C^A_{w,0}, C^A_{w,1})$ proves that the decommitments of $C^A_{w,0}$ and $C^A_{w,0}$ are not equal. See [58] for the details of the proof.

- $\boldsymbol{CorrectGarble}(E_{00}, E_{01}, E_{10}, E_{11})$ proves that the order of the table is properly formed and the order matches with the commitments of the keys and encryption keys. See [58] for details.

- According to prover's side, he performs one of the proofs below. If she is from the left side (in our protocol it is Alice's side) then she performs ***ZKVerifyEncRight***, otherwise he performs ***ZKVerifyEncLeft***. Each $\boldsymbol{ZKVerifyEnc}(E, D)$ shows that the encryption E contains the decommitment of the commitment D. This proof can be done as in [25].

  - $\boldsymbol{ZKVerifyEncRight} = \boldsymbol{ZKVerifyEnc}(E_{00}, D^A_{w_b,0}) \wedge \boldsymbol{ZKVerifyEnc}(E_{01}, D^A_{w_b,1}) \wedge \boldsymbol{ZKVerifyEnc}(E_{10}, D^A_{w_b,0}) \wedge \boldsymbol{ZKVerifyEnc}(E_{11}, D^A_{w_b,1})$
  - $\boldsymbol{ZKVerifyEncLeft} = \boldsymbol{ZKVerifyEnc}(E_{00}, D^A_{w_b,0}) \wedge \boldsymbol{ZKVerifyEnc}(E_{01}, D^A_{w_b,0}) \wedge \boldsymbol{ZKVerifyEnc}(E_{10}, D^A_{w_b,1}) \wedge \boldsymbol{ZKVerifyEnc}(E_{11}, D^A_{w_b,1})$

6.4.3 Proof of Output Gates

Inputs: Common inputs are the garbled tables of g that contains encryptions $E_{00}, E_{01}, E_{10}, E_{11}$, along with the commitments $S^A_i, ..., S^A_{i+3}$ of the random row pairs of g if g is the output gate of the prover. Prover's private inputs are all decommitments.

Prover performs following, if g is the output gate of the prover:

- $\boldsymbol{ZKVerifyEnc}(E_{00}, S_i^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{01}, S_{i+1}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{10}, S_{i+2}^A) \wedge \boldsymbol{ZKVerifyEnc}(E_{11}, S_{i+3})$, where $\boldsymbol{ZKVerifyEnc}(E, S)$ proves that the encryption E contains the decommitment of S.

6.4.4 Proof of Correct Randoms

Inputs: Common inputs are the commitments $D_{w_b,0}^A, D_{w_b,1}^A$. Prover knows the decommitments. Prover performs the following (details in [58] or appendix of [62]):

- $\boldsymbol{ZKComDL}(D_{w_b,0}^A) \wedge \boldsymbol{ZKComDL}(D_{w_b,1}^A)$ proving knowledge of the decommitments of these commitments.

- $\boldsymbol{ZKEqComDL}$ $(D_{w_b,0}^A, D_{w_b,1}^A)$ showing that the committed messages are equal for both commitments (under different bases, $e_{b,0}$ and $e_{b,1}$).

6.5 Security of the Protocol

Theorem 5. *Let $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$ be any probabilistic polynomial time (PPT) two-party functionality. The protocol above for computing f is secure and fair according to Definition 1, assuming that the subprotocols that are stated in the protocol are all secure (sound and zero-knowledge) [51, 50, 13], the CS and sCS verifiable encryption schemes are secure [25, 58], all commitments are hiding and binding [44, 35], the garbled circuits constructed as in Yao's protocol [87] are secure, and the signature scheme used is existentially unforgeable under adaptive chosen message attack [52].*

Full simulation proof with proof sketch is given in Appendix 6.5. The important point in our proof is that after learning input of malicious adversary, the simulator does not learn the output of the malicious adversary from ideal world TTP U until fairness is guaranteed.

6.5.1 Proof Sketch

Malicious Alice: Simulator S_B prepares a fake circuit with a random input y' and simulates all the proofs, including the equality test. S_B learns the input of Alice x by acting as functionality $\mathcal{F}_{\mathcal{COT}}$ in committed oblivious transfer part. Here, he does not send the input of Alice directly to the ideal world TTP U. He waits until the fairness is guaranteed. If before the verifiable escrow is received properly, the values that Alice sends are not correct or equality test is not successful, he sends message ABORT to U. After S_B receives the correct verifiable escrow, he sends Alice's input to U and receives Alice's output from U. At this point, he sends to Alice random row pairs of Alice's output by choosing the ones that match the output given by U. If he does not receive his correct output results from Alice, he does *Bob Resolve*.

Malicious Bob: Simulator S_A behaves almost the same as S_B, but the point that fairness guaranteed is different. Since S_A does not know the actual output of Bob, she puts random values in the verifiable escrow and sends it to Bob, simulating the proof. Then S_A waits for the correct random row pairs of his output: if Bob sends them, the fairness is guaranteed and S_A sends to U Bob's input that he learned from committed oblivious transfer phase and U sends back Bob's output. Then, S_A sends to Bob the correct random row pairs of Bob using the functionality output. However, if Bob does not send anything or correct random row pairs, the fairness is guaranteed according to the execution of Bob Resolve by Bob. If Bob does Bob Resolve, S_A acts as functionality $\mathcal{F}_{\mathcal{RES}}$. In $\mathcal{F}_{\mathcal{RES}}$, if Bob performs a proper Bob Resolve, then S_A gives the input of Bob to U and learns the output of him. She then answers the resolution request accordingly. If Bob does not perform Bob Resolve until the timeout, S_A sends ABORT to U.

6.5.2 Full Simulation Proof

We prove the theorem by providing simulators for the case that Alice is malicious and the case that Bob is malicious.

Security against Malicious Alice

Lemma 1. *Under the assumptions in Theorem 5, our protocol is secure and fair against malicious Alice.*

Proof. We construct an ideal-model adversary/simulator that interacts with Alice, the universal trusted third party U, and the fairness TTP T. The simulator S_B has to behave like Bob in the interaction with corrupted Alice. Therefore, he builds a fake random garbled circuit that is indistinguishable from the real one and executes the protocol. According to our ideal process definition (see Definition 1), S_B has to send the input x of Alice to U, since he simulates Alice's behaviors in the ideal process. He learns it by using the COT functionality. In the equality test part, S_B has to prove equality properly. For this purpose, he will use zero-knowledge simulator and proof of knowledge extractor. Lastly, he sends random row pairs of Alice according to the output $f(x, y)$ that he learns from U. The detailed behavior of the S_B follows:

1. [**Agreement to the Common Knowledge**] S_B learns the public key PK_{TTP} of the TTP, and $\mathcal{A}$ sends his signature verification key $vk_{\mathcal{A}}$ to S_B. Then, $\mathcal{A}$ and S_B decide on the equality test constants $\{e_{a,t}, e_{b,t}\}_{t\in\{0,1\}}$ that are generated jointly.

2. [**Construction of the Garbled Circuit**]

   - S_B and malicious Alice $\mathcal{A}$ agree on the circuit.
   - S_B generates random keys $\{\tilde{k}_{w,0}, \tilde{k}_{w,1}\}_{w\in W\setminus W_O}$ and commits to them as $\{\tilde{C}^B_{w,0}, \tilde{C}^B_{w,1}\}_{w\in W\setminus W_O}$.
   - S_B picks random row pairs as in the protocol, and commits to them for every row of his garbled output gates. These commitments are formed correctly, as in the protocol.
   - S_B constructs a fake garbled circuit that has a random constant as the output. He first picks random bits $\tilde{\varphi}^B_l, \tilde{\varphi}^B_r$ for every gate. The normal

garbled tables of the fake garbled circuit are as in Table 6.3. For the output gates, S_B adds an encryption of random row pair for every corresponding row of output garbled tables as in the protocol.

Row	t_a, t_b	Encryption
00	$(0 \oplus \tilde{\varphi}_a^B), (0 \oplus \tilde{\varphi}_b^B)$	$E_{\tilde{k}^B_{w_a,t_a}} E_{\tilde{k}^B_{w_b,t_b}}(\tilde{k}_w^B)$
01	$(0 \oplus \tilde{\varphi}_a^B), (1 \oplus \tilde{\varphi}_b^B)$	$E_{\tilde{k}^B_{w_a,t_a}} E_{\tilde{k}^B_{w_b,t_b}}(\tilde{k}_w^B)$
10	$(1 \oplus \tilde{\varphi}_a^B), (0 \oplus \tilde{\varphi}_b^B)$	$E_{\tilde{k}^B_{w_a,t_a}} E_{\tilde{k}^B_{w_b,t_b}}(\tilde{k}_w^B)$
11	$(1 \oplus \tilde{\varphi}_a^B), (1 \oplus \tilde{\varphi}_b^B)$	$E_{\tilde{k}^B_{w_a,t_a}} E_{\tilde{k}^B_{w_b,t_b}}(\tilde{k}_w^B)$

Table 6.3: Fake garbled Normal OR gate of Bob

- In the construction of the input garbled tables, S_B picks $\{\tilde{m}^B_{w_a^g,t}, \tilde{m}^B_{w_b^g,t}\}_{g\in G_I, t\in\{0,1\}}$ randomly instead of randomized equality test numbers.[2] Then he prepares commitments of $\{\tilde{m}^B_{w_a^g,t}\}_{g\in G_{I_a}, t\in\{0,1\}}$ which are $\{\tilde{D}^B_{w_a^g,t}\}_{g\in G_{I_a}}$. Then he constructs fake input garbled tables that have encryption of constant keys and these fake partial decommitments of $\tilde{D}$s .

3. [**Committed Oblivious Transfer (COT)**] S_B activates $\mathcal{A}$ for the COT phase where $\mathcal{A}$ is the sender. He acts as the functionality $\mathcal{F}_{\mathcal{COT}}$ (consider a hybrid model here) and expects proper message that includes the input keys of Bob in Alice's garbled circuit $\{k^A_{w,0}, k^A_{w,b}\}_{w\in W_{I,b}}$ from $\mathcal{A}$. Afterward, the case where Bob is the sender begins, and again S_B behaves like $\mathcal{F}_{\mathcal{COT}}$. He obtains the input of Alice $x = (x_1, ..., x_l)$ from $\mathcal{A}$. Afterward, S_B sends the corresponding keys of the input wires from S_B's garbled circuit.

4. [**Send, Prove and Evaluate**]

- $\mathcal{A}$ sends GC^A with all the necessary commitments, the keys that represent her input along with randomized equality test numbers m^A that represent

her input. Similarly S_B sends the fake garbled circuit $\tilde{GC}^B$ with fake commitments that he generated, and keys along with $\tilde{m}^B$ values that represent his *random inputs*.

- $\mathcal{A}$ performs all proofs in the evaluation case. S_B learns all random row pairs of $\mathcal{A}$'s output gates from the *Proof of Output Gates* via an extractor. If her proofs are not valid, then S_B sends message ABORT to U. Otherwise, he starts to do his proofs. Since he did not construct the circuit correctly, he simulates the proofs. Then, they continue with the evaluation of the garbled circuits.

5. [**Equality Test**] S_B acts as the random oracle, so when $\mathcal{A}$ asks for the hash of her equality test input, S_B answers her randomly and learns the equality test input of Alice, which consist of $\tilde{m}^B$ values. Since these were generated by S_B, he can check if they are the expected ones because S_B knows his inputs and Alice's inputs for S_B's garbled circuit, so he knows which row $\mathcal{A}$ has to decrypt. Then they begin the equality test protocol. If the proofs of $\mathcal{A}$ are not valid or the input values are not the expected ones, then S_B sends ABORT message to U.

   In addition S_B checks if $\mathcal{A}$ used the same input as himself for the garbled circuit evaluation. He knows which row $\mathcal{A}$ should decrypt, as explained above. He learned the input gate numbers $\{u_g\}_{g \in G_I}$ from *Proof of Correct Randoms* in the evaluation phase of the GC^A, using an extractor. So S_B can learn which equality test constants of the row that he decrypted are used, hence he can learn which input $\mathcal{A}$ used in $\mathcal{A}$'s garbled circuit. Finally, S_B can check if the inputs that $\mathcal{A}$ used in both circuits are equal. If they are not equal then he sends ABORT message to U.

   He simulates all zero-knowledge proof of knowledge protocols in the equality test, similarly as in [18]. If the test was supposed to return true, he simulates that way, otherwise he simulates with another random value whose hash is

random. The probability that the adversary queries the random oracle at this random value is negligible.

If the equality test is successful, then $\mathcal{A}$ should send the signature s_{eq}. If S_B does not receive a verifying signature, he sends ABORT message to U.

Remark: Our security theorem intentionally does not mention the random oracle model. As is specified above and in [18], the equality test and the use of the hash function here necessitates the use of the random oracle model. Yet, one may convert this part to be secure in the standard model, with a loss of efficiency. First, all zero-knowledge proofs of knowledge must be done interactively. Second, instead of hashing the values and performing only a single equality test, the values must be compared independently, using $2l$ equality tests. Performing these equality tests independently does not harm security, since one may not cheat in the other party's input, but can only cheat with own input, and therefore learns nothing from the (in)equality.

6. [**Verifiable Escrow**] $\mathcal{A}$ needs to send the verifiable escrow and if verification fails, then S_B sends ABORT message to U. Otherwise, the simulation continues with the next step.

7. [**Output Exchange**] At this point S_B is sure that the exchange will be fair, and thus there is no more need for aborting. Hence, S_B hands Alice's input x to U and U sends $f_a(x, y) = \{f_{w,a}\}_{w \in W_{O,a}}$ to the simulator. He knows all random row pairs of $\mathcal{A}$ from the *Proof of Output Gates* in the evaluation of GC^A. In addition, he learned which bit the row represents from *Proof of Garbled Construction*. Then, he picks random row pairs corresponding to the $\mathcal{A}$'s output result $f_a(x, y) = \{f_{w,a}\}_{w \in W_{O,a}}$ and sends all to $\mathcal{A}$.

   S_B waits for the response from $\mathcal{A}$. If $\mathcal{A}$ does not respond, then he executes *Bob Resolve*. Otherwise, he gets random row pairs corresponding to his output, and checks if they are valid. In the case that they are not correct, he runs *Bob*

Resolve. If they are valid, he outputs whatever $\mathcal{A}$ outputs.

□

Claim 2. *The view of $\mathcal{A}$ in his interaction with the simulator S_B is indistinguishable from the view in his interaction with real Bob.*

Proof. We need to check the behaviors of the S_B that are different from Bob's behavior in the protocol, since these can affect the distribution.

- In [65] it is proven that the fake garbled circuit is indistinguishable in the case that it always returns the functionality result as expected. Note that the output of our real garbled circuit is indeed random numbers. Our fake garbled circuit outputs random numbers as well, and hence this is indistinguishable from the real garbled circuit.

- S_B did not commit to the real randomized equality test numbers as real Bob would do. Since the commitment scheme is hiding, the fake commitments are indistinguishable from the real ones. In addition, S_B simulates the zero-knowledge proofs in the protocol. Their indistinguishably comes from the zero-knowledge simulatability.

- S_B interacts with T in the cases where Bob interacts. In addition, all inputs that S_B gives to T are identical to the inputs Bob would have provided. Thus, the TTP cannot distinguish as well.

- The simulator only aborts when the real Bob would abort. Thus, the abort actions are indistinguishable as well.

We emphasize one more time that a simulation proof for a secure and fair two-party computation protocol needs to make sure that the simulator learns the output only when the protocol becomes fair, and not before that point. In real world, it is sure for honest Bob that he will learn the functionality output after Alice sends the

correct verifiable escrow, because even if Alice does not send the output to him in the fair exchange phase, Bob can learn them from the TTP via *Bob Resolve*. For this reason S_B sends input x to U in ideal world after receiving the valid verifiable escrow. After that Bob and S_B learn their outputs in the ideal world. As mentioned, Bob learns his output certainly in the end of the protocol in the real world after this point. Since Bob is honest, $\mathcal{A}$ learns her output too, in the real world.

Overall, we proved that the joint distribution of $\mathcal{A}$ and the honest Bob's output in the real execution are indistinguishable from the joint distribution of S_B and the honest Bob's output in the ideal model. □

Finally, *Bob Resolve* protocol preserves fairness, as presented in the protocol description. Bob (S_B) can learn his output only by giving Alice's output to the TTP and proving that is the correct output. Therefore, combined with Claim 2, the proof of Lemma 1 is complete.

Security against Malicious Bob

Lemma 2. *Under the assumptions in Theorem 5, our protocol is secure and fair against malicious Bob.*

Proof. The proof is similar to that of Lemma 1. We construct a simulator S_A that interacts as Alice with the adversary $\mathcal{B}$ that controls Bob in the real world, and as Bob with the U in the ideal world. S_A simulates the protocol as following:

1. [**First 5 items in the Protocol]** S_A learns the public key PK_{TTP} of the TTP. Then S_A creates a key-pair for the signature scheme and sends her verification key vk_A to $\mathcal{B}$. Then they agree on the equality test constants. After that, all simulation actions are the same as the simulator S_B above until the end of *equality test*. There is just one difference, which is that S_A additionally sends the signature of the commitments of all the random row pairs to $\mathcal{B}$, as in the protocol. S_A sends the signature s_{eq} to $\mathcal{B}$ (if the equality test was indeed successful).

2. [**Verifiable Escrow**] S_A prepares the verifiable escrow. Since S_A does not know $f_a(x, y)$, she adds random values in the escrow. She labels the escrow with the verification key vk_A of the signature scheme and the correct *time* value. S_A simulates the proof by using verifiable escrow simulator [25].

3. [**Output Exchange**]

   - S_A waits for an answer in the output exchange phase. If $\mathcal{B}$ sends random row pairs, firstly S_A checks if they are random row pairs that she encrypted in the fake garbled circuit. If they are correct, we are guaranteed that the exchange will be fair and the protocol will not be aborted. S_A sends y, which he learned via the committed oblivious transfer functionality $\mathcal{F}_{\mathcal{COT}}$, to U, and gets $f_b(x, y)$ from U. This also means that U sends $f_a(x, y)$ to Alice in the ideal world. Since S_A learned all decommitments of random row pairs of Bob from *Proof of Output Gates*, S_A prepares the random row pairs of $\mathcal{B}$ according to $f_b(x, y)$ and sends them to $\mathcal{B}$.
   - If there was a problem with the random row pairs that $\mathcal{B}$ sent, then resolution needs to take place. If before the timeout $\mathcal{B}$ runs *Bob Resolve*, S_A behaves like the resolution functionality $\mathcal{F}_{\mathcal{RES}}$. First, S_A checks whether or not the verifiable escrow provided by $\mathcal{B}$ is the same as the one S_A sent in step 2, signature s_{eq} given by the adversary verifies under the public key of S_A in the verifiable escrow, and the random row pairs as proven by $\mathcal{B}$ are correct.

     If they are correct, then again fairness is guaranteed, and thus S_A sends y to U and gets $f_b(x, y)$ (and U sends $f_a(x, y)$ to ideal Alice). Then S_A picks the correct random row pairs according to output value $f_b(x, y)$. Then he sends these pairs to $\mathcal{B}$ as if they are the decryption of the verifiable escrow. However, if the $\mathcal{B}$ sends unverified values to $\mathcal{F}_{\mathcal{RES}}$, then S_A sends ABORT message to U and ends the protocol.
   - If $\mathcal{B}$ did not send output values to S_A in step 3, or did not perform *Bob*

Resolve until the time, then S_A sends ABORT message to U.

□

Claim 3. *The view of $\mathcal{B}$ in his interaction with the simulator S_A is indistinguishable from the view in his interaction with real Alice.*

Proof. Honest Alice learns the output of the function if malicious Bob sends it in the output exchange phase, or if Bob successfully performs *Bob Resolve* in the real world. For this reason, the simulator S_A learns the output from U in the ideal world exactly at these points. The abort cases also follow the real world.

As in Claim 2, the joint distribution of $\mathcal{B}$ and honest Alice's output in the real execution is indistinguishable from the joint distribution of S_A and honest Alice's output in the ideal model. □

This completes the proof of Theorem 5.

6.6 Importance of Proving Security and Fairness Together

We give a weird modified version of the protocol in Section 6.3. The modified protocol is almost same as the original. One change is in step 3 in "Sending Fairness Values" part. Here, Alice sends all signatures of the commitments of random row pairs $\{S_i^A, sign_{sk_A}(S_i^A)\}_{i\in\{1,\ldots,8l\}}$ instead of just sending signatures of commitments that corresponds her output wires.

The important changes are in the resolve protocols. In new *Bob Resolve*, Bob contacts the TTP before the timeout and gives all items as in *Bob Resolve*. In addition, he gives all signatures of $\{S_i^A\}_{i\in\{1,\ldots,8l\}}$ (before it was $4l$ signatures) and l random row pairs that correspond to Alice's output along with l **random row pairs that correspond to his output**. In short, he gives information that can be used to identify both his and Alice's outputs as computed from Alice's circuit. Then, the TTP checks the correctness of random row pairs by checking if they are decommitments of $\{S_i^A\}$s. In return, the TTP gives the decryption of the verifiable escrow as in *Bob Resolve*.

New *Alice Resolve* protocol is very similar to the old one. The only difference is that the TTP gives Alice not only Alice's random row pairs but also Bob's random row pairs. Thus, Alice can learn both her and Bob's outputs. This resolution protocol obviously violates privacy.

We try to prove the fairness and security of this protocol with the standard method used in most of the previous work: prove security with standard simulation paradigm first and then show fairness separately. We show that this modified protocol **can be proven secure and fair separately**. Then, we show that such a protocol, and any of its privacy-violating variants, **cannot be proven secure and fair according to our definition**.

Prove Security then Fairness: We firstly prove that the protocol is secure in standard ideal/real world definition. The security proof of this protocol is straightforward, as it is almost the same as our protocol's proof. The simulator learns the input of the adversary in real world in the committed oblivious transfer phase and gives this input to U in the ideal world. Afterwards, the simulator receives output from U and continues the simulation as explained. If the adversary aborts the real protocol, then the simulator sends ABORT message to U.

For fairness of this protocol, note that either both parties receive the output or none of them receives, because if the protocol terminates before the verifiable escrow, no party can learn any useful information. If the verifiable escrow is received, but the last phase of exchanging random row pairs was problematic, then the resolution protocols will ensure fairness such that both parties will learn their corresponding outputs if they resolve. Hence according to this type of separate proofs, this contrived protocol is fair and secure.

Note that the problem arises since the fairness definition is only concerned that either Alice receives her output and Bob receives his, or no one receives anything. This definition of fairness is achieved by the contrived protocol. But, during resolutions, Alice learns some extra information. To prevent learning extra information during fairness extensions, one must also ensure simulation of fairness parts are achievable. If

the simulator is not concerned about fairness, one may prove such a contrived protocol fair, as we did above.

Prove Security and Fairness: Let us try to prove security and fairness of such a contrived protocol using our joint definition. Assume, for simplicity, Alice is malicious. As in the proof in Appendix 6.5.2, the simulator S_B learns the input x via $\mathcal{F}_{\mathcal{COT}}$ and then continues to simulate the view of the protocol using a random input y'. When the fairness is guaranteed, S_B sends Alice's input x to U, who responds back with $f_a(x, y)$.

After this point, S_B sends the corresponding random row pairs for $f_a(x, y)$ and waits for the answer from Alice. In the case that Alice does not answer or does not give the correct random row pairs, S_B performs *new Bob Resolve* with the TTP. Differently, from the original protocol, S_B has to give both random row pairs that correspond to $f_a(x, y)$ and $f_b(x, y)$. However, he cannot do it because he does not know $f_b(x, y)$, or y, or anything that depends on y other than $f_a(x, y)$. The best that S_B can do is to give random row pairs that corresponds $f_b(x, y')$ and $f_a(x, y)$. It simulates the TTP's view since they are just random numbers. But then Alice comes for *new Alice Resolve*, and she gets these random row pairs. Alice now calculates the corresponding bits for each random row pair in her circuit and outputs $f_a(x, y), f_b(x, y')$. Unfortunately, this adversary's output is now distinguishable from the ideal world, since the ideal world of this protocol would correspond to outputting $f_a(x, y), f_b(x, y)$. We can conclude that this protocol is not fair and secure according to our new ideal/real world definition.

In general, using our definition, since the simulator S_B only knows $f_a(x, y)$ and nothing else that depends on the actual y, *no* fairness solution that leaks information on y (other than $f_a(x, y)$) can be simulated.

Consequently, it is risky to prove fairness separate from the ideal/real world simulation. We do not claim that previous protocols [22, 60, 82] has security problems because of their proof techniques, but we want to emphasize that this proof technique does not cover all security aspects of a protocol. Interestingly, Cachin and Camenisch [22] definition also prevents such problems, but they prove their protocol secure using

the old style of proving security and fairness separately.

	[81]	[22]	[60]	[63]	[82]	Ours
Malicious Behavior	CC	ZK	CC	ZK/CC ✓	CC	ZK/CC ✓
Fairness	IGR	EOFE^t	IGR	EOFE^c	IGR	EOFE ✓
Proof Technique	NS	NFS	NFS	FS ✓	NFS	FS ✓

Table 6.4: Comparison of our protocol with previous works in fair 2PC. CC denotes cut-and-choose, ZK denotes efficient zero-knowledge proofs of knowledge, NON denotes not indicated, IGR denotes inefficient gradual release, EOFE denotes efficient optimistic fair exchange, superscript t denotes inefficient TTP, superscript c denotes necessity of using coins, NS denotes no ideal-real simulation proof given, NFS indicates simulation proof given but not for fairness, and finally FS indicates full simulation proof including fairness. A ✓ is put for identifying better techniques.

6.7 Analysis of Fair and Secure 2PC:

Protocols using **gradual release** [81, 60, 82] require much more number of rounds at the final output exchange stage, proportional to the security parameter. In comparison, our protocol adds extra 7 rounds[3] for output exchange, verifiable escrow, and the equality test (See Section 3.3). In the case of an unfair situation, there are 2 extra rounds with the TTP. In comparison, an unfair scenario in the gradual release setting would possibly require extreme computational effort. In addition, when the parties' computational powers are not the same in such a setting, there can be unfairness.

The protocol by Camenisch and Cachin [22] does not use gradual release but uses the TTP inefficiently. In the resolution phase, a party has to send the whole transcript of the protocol to the TTP to show that (s)he behaved honestly. Then, the TTP and the party together run the protocol from where it stopped, which means possibly performing almost the whole protocol with the TTP.

Lindell's protocol [63] uses TTP as in our protocol, however its disadvantage is that one of the parties can get a coin instead of the output of the functionality. Even though this fairness definition presents itself in other contexts [12, 61], it is not suitable

for two-party computation because no party knows the output of the functionality and so it is hard to estimate the value of the result beforehand.

One of the most important contributions of our protocol is that it is proved using our simulation-based definition, where the complete protocol is proven secure, together with its fairness extensions and resolutions. In comparison, the security of the protocols in [81] are not proven with ideal/real world simulation. The protocols in [60, 82] were proven secure according to the standard ideal/real world paradigm definition, and then fairness is proven separately. The simulators in their proofs learn the output from the universal trusted party U of the ideal world before fairness is guaranteed. Since we want that the ideal and real world distributions are the same, these proofs do not satisfy our definition, because it enables one of the parties (the simulator) abort after obtaining the output and before the other party obtains the output. Lindell's protocol [63] is proven to be fair and secure together, but for their own ideal/real world definition based on their protocol. Similarly, [22] has fair and secure ideal world definition, but their proof is done with the old style of proving security and fairness separately. In addition, U in ideal world has to contact the real world TTP T in every case; this harms the indistinguishably property of ideal and real worlds since maybe T never gets involved in the real protocol in an optimistic setting.

Finally, we can replace the zero-knowledge proof-based techniques in our presentation with cut-and choose techniques [86, 64, 81, 66, 43] to prove correct construction of the garbled circuit. Similarly we can employ other oblivious transfer solutions [86, 64, 81, 66]. **Both cut-and-choose and zero-knowledge-proof based techniques have their own advantages and our fairness adaption can be used on top of both types of underlying secure two-party solutions secure against malicious parties**.

Notes to Chapter 6

1 Note that the discussion about the usage of unique identifiers for the TTP to identify different evaluations between some Alice and some Bob (with anonymity and unlinkability guarantees), as well as the timeout not harming the system already exists in the optimistic fair exchange literature [5, 61], and therefore we are not complicating our presentation with such issues.

2 Remember that in the real protocol, he exponentiates equality test constants with input gate numbers to compute randomized equality test numbers.

3 Round means sending one message in our context.

Chapter 7

CONCLUSION

In this thesis, we have presented new frameworks for fair and secure computation and protocols for multi party fair exchange. The new protocols MFE and CMFE (Chapter 4)has strong fairness property which is either all parties receive their desired items or none of them receive. We showed that MFE is more efficient and more general exchange protocol comparing to previous works [9, 42, 4, 45, 73, 70]. In addition we give another exchange protocol CMFE that can be used any item including inefficiently verifiable items. CMFE is the only protocol with this property. Both CMFE and MFE can be adapted by any topology. In addition we prove fairness of our new exchange protocols with using ideal/real world paradigm. This proof technique has never been used in an exchange protocol before. Multi party fair exchange has many real world application. Besides contract signing and multi party computation, it can be used for threshold secret sharing schemes because fairness is one of the problem of the secret sharing scheme protocols.

We described how to adapt MFE protocol to a secure multi party computation protocol to add fairness property in Chapter 4. If fairness property is added to a SMPC protocol with our framework, the only overhead is on circuit size and Phase 3 of MFE protocol. However, if an arithmetic circuit is used the overhead on the circuit size decreases. Still our framework provides more efficiency and security than the previous works (see Table 5.1.

We design new framework for two party computation even though MPC framework can be also used. However, we try to achieve better efficiency here since two party case is simpler case than multi-party case. 2PC framework adds little overhead to the secure two party computation protocol to achieve fairness. The main additions for

fairness are efficient proofs (which can also be done via cut-and-choose), an efficient equality test protocol, an efficient verifiable escrow, and some simple signatures. Our framework efficiently achieves fairness and security in two-party computation against malicious adversaries Table 6.4.

Our other important contributions in this thesis are **we prove security and fairness simultaneously via simulation using our new definition** in exchange protocols and secure computation protocols and **we show that previous proof technique does not completely prove that the computation protocol fair and secure**. This is very important contribution because it shows that previous proofs do not completely guarantee fairness and security according to our result. In addition, in all of our protocols **our TTP workload is very light**, and **TTP does not learn inputs and outputs of the parties**. This shows that our protocols are feasible and semi-honest TTP is enough to achieve fairness.

BIBLIOGRAPHY

[1] A. Afshar, P. Mohassel, B. Pinkas, and B. Riva. Non-interactive secure computation based on cut-and-choose. In *EUROCRYPT*, 2014.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. *IACR Cryptology ePrint Archive*, 2013:784, 2013.

[3] G. Asharov, Y. Lindell, and H. Zarosim. Fair and efficient secure multiparty computation with reputation systems. In *ASIACRYPT (2)*, 2013.

[4] N. Asokan, M. Schunter, and M. Waidner. Optimistic protocols for multi-party fair exchange, 1996.

[5] N. Asokan, V. Shoup, and M. Waidner. Optimistic fair exchange of digital signatures. *IEEE Journal on Selected Areas in Communications*, pages 591–610, 2000.

[6] G. Ateniese. Efficient verifiable encryption (and fair exchange) of digital signatures. In *Proceedings of the 6th ACM Conference on Computer and Communications Security*, 1999.

[7] G. Ateniese and C. Nita-rotaru. Stateless-recipient certified e-mail system based on verifiable encryption. In *In: CT-RSA'02, LNCS 2271*, 2002.

[8] F. Bao, R. H. Deng, and W. Mao. Efficient and practical fair exchange protocols with off-line ttp. In *In Proceedings of IEEE Symposium on Security and Privacy*, pages 77–85, 1998.

[9] F. Bao, R. H. Deng, K. Q. Nguyen, and V. Varadharajan. Multi-party fair exchange with an off-line trusted neutral party. In *DEXA Workshop*, 1999.

[10] C. Baum, I. Damgård, and C. Orlandi. Publicly auditable secure multi-party computation. *IACR Cryptology ePrint Archive*, 2014:75, 2014.

[11] B. Baum-Waidner and M. Waidner. Round-optimal and abuse free optimistic multi-party contract signing. In *ICALP*, 2000.

[12] M. Belenkiy, M. Chase, C. C. Erway, J. Jannotti, A. Küpçü, A. Lysyanskaya, and E. Rachlin. Making p2p accountable without losing privacy. In *In Proc. Workshop on Privacy in the Electronic Society (wpes), Oct. 2007. (Referenced on pages 127 and 130*, 2008.

[13] M. Bellare and O. Goldreich. On defining proofs of knowledge. In *CRYPTO*, 1992.

[14] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1988.

[15] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. *IACR Cryptology ePrint Archive*, 2014:129, 2014.

[16] D. Boneh and M. Naor. Timed commitments (extended abstract). In *CRYPTO*, 2000.

[17] F. Boudot. Efficient proofs that a committed number lies in an interval. In *EUROCRYPT*, 2000.

[18] F. Boudot, B. Schoenmakers, and J. Traoré. A fair and efficient solution to the socialist millionaires' problem. *Discrete Applied Mathematics*, 111(1-2):23–36, 2001.

[19] E. Bresson and J. Stern. Proofs of knowledge for non-monotone discrete-log formulae and applications. In *ISC*, 2002.

[20] E. F. Brickell, D. Chaum, I. Damgård, and J. v. d. Graaf. Gradual and verifiable release of a secret. In *CRYPTO*, 1987.

[21] Brownie cashlib cryptographic library. http://github.com/brownie/cashlib.

[22] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *CRYPTO*, 2000.

[23] J. Camenisch and I. Damgård. Verifiable encryption and applications to group signatures and signature sharing. Technical report, 1999.

[24] J. Camenisch, A. Lysyanskaya, and M. Meyerovich. Endorsed e-cash. In *Proceedings of the 2007 IEEE Symposium on Security and Privacy*, 2007.

[25] J. Camenisch and V. Shoup. Practical verifiable encryption and decryption of discrete logarithms. In *CRYPTO*, 2003.

[26] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13:143–202, 2000.

[27] R. Canetti, C. Dwork, M. Naor, and R. Ostrovsky. Deniable encryption. In *Advances in Cryptology—CRYPTO'97*. 1997.

[28] D. Chaum. Blind signatures for untraceable payments. In *CRYPTO*, 1982.

[29] D. Chaum, B. den Boer, E. van Heyst, S. Mjølsnes, and A. Steenbeek. Efficient offline electronic checks (extended abstract). In *EUROCRYPT*, 1989.

[30] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *CRYPTO*, 1993.

[31] B. Cohen. Incentives build robustness in bittorrent. In *WEPS*, 2003.

[32] R. Cramer, I. Damgård, and J. B. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques: Advances in Cryptology*, 2001.

[33] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology—CRYPTO'94*, 1994.

[34] I. Damgård. On sigma protocols. http://www.daimi.au.dk/ ivan/Sigma.pdf.

[35] I. Damgard and E. Fujisaki. A statistically-hiding integer commitment scheme based on groups with hidden order. In *ASIACRYPT*, 2002.

[36] I. B. Damgård. Practical and provably secure release of a secret and exchange of signatures. *Journal of Cryptology*, 8:201–222, 1995.

[37] I. B. Damgård and M. J. Jurik. A length-flexible threshold cryptosystem with applications. In *In Proceeding of ACISP'03*, 2003.

[38] C. Dong, L. Chen, J. Camenisch, and G. Russello. Fair private set intersection with a semi-trusted arbiter. In *DBSec*, 2013.

[39] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1985.

[40] S. Even and Y. Yacobi. Relations among public key signature scheme. Technical Report 175, Department of Computer Science, TechUnion, Haifa, Israel, 1980.

[41] M. Franklin and S. Haber. Joint encryption and message-efficient secure computation. In *Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, 1994.

[42] M. Franklin and G. Tsudik. Secure group barter: Multi-party fair exchange with semi-trusted neutral parties. In *Lecture Notes in Computer Science*, 1998.

[43] T. K. Frederiksen, T. P. Jakobsen, J. B. Nielsen, P. S. Nordholt, and C. Orlandi. Minilego: Efficient secure two-party computation from general assumptions. In *EUROCRYPT*, 2013.

[44] E. Fujisaki and T. Okamoto. Statistical zero knowledge protocols to prove modular polynomial relations. In *CRYPTO*, 1997.

[45] J. A. Garay, M. Jakobsson, and P. D. MacKenzie. Abuse-free optimistic contract signing. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, 1999.

[46] J. A. Garay, P. MacKenzie, and K. Yang. Efficient and secure multi-party computation with faulty majority and complete fairness. In *In Cryptology ePrint Archive, http://eprint.iacr.org/2004/019*, 2004.

[47] J. A. Garay and P. D. MacKenzie. Abuse-free multi-party contract signing. In *DISC*, 1999.

[48] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[49] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, 1987.

[50] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. *Journal of ACM*, 38:728, 1991.

[51] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18:208, 1989.

[52] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17:281–308, Apr. 1988.

[53] N. González-Deleito and O. Markowitch. An optimistic multi-party fair exchange protocol with reduced trust requirements. In *Proceedings of the 4th International Conference Seoul on Information Security and Cryptology*, 2002.

[54] S. D. Gordon. Complete fairness in multi-party computation without an honest majority. In *In 6th Theory of Cryptography Conference — TCC 2009, volume 5444 of LNCS*, 2009.

[55] S. D. Gordon, C. Hazay, J. Katz, and Y. Lindell. Complete fairness in secure two-party computation. *Journal of ACM*, 58, 2011.

[56] R. Henry and I. Goldberg. Batch proofs of partial knowledge. In *ACNS*, 2013.

[57] M. Hirt and D. Tschudi. Efficient general-adversary multi-party computation. In *ASIACRYPT (2)*, 2013.

[58] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, 2007.

[59] M. S. Kiraz and B. Schoenmakers. A protocol issue for the malicious case of yao's garbled circuit construction. In *In Proceedings of 27th Symposium on Information Theory in the Benelux*, 2006.

[60] M. S. Kiraz and B. Schoenmakers. An efficient protocol for fair secure two-party computation. In *CT-RSA*, 2008.

[61] A. Küpçü and A. Lysyanskaya. Usable optimistic fair exchange. *Computer Networks*, pages 50–63, 2012.

[62] A. Küpçü. *Efficient Cryptography for the Next Generation Secure Cloud: Protocols, Proofs, and Implementation.* Lambert Academic Publishing, 2010.

[63] A. Y. Lindell. Legally-enforceable fairness in secure two-party computation. In *CT-RSA*, 2008.

[64] Y. Lindell. Fast cut-and-choose based protocols for malicious and covert adversaries. In *CRYPTO*, pages 1–17, 2013.

[65] Y. Lindell and B. Pinkas. A proof of yao's protocol for secure two-party computation. *ECCC*, 11:2004, 2004.

[66] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *EUROCRYPT*, 2007.

[67] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *Journal of Privacy and Confidentiality*, 1, 2009.

[68] Y. Liu and H. Hu. An improved protocol for optimistic multi-party fair exchange. In *EMEIT*, 2011.

[69] O. Markowitch and S. Kremer. A multi-party optimistic non-repudiation protocol. In *Proceedings of The 3rd International Conference on Information Security and Cryptology*, 2000.

[70] S. Mauw, S. Radomirovic, and M. T. Dashti. Minimal message complexity of asynchronous multi-party contract signing. In *CSF*, 2009.

[71] R. C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, 1988.

[72] S. Micali. Simple and fast optimistic protocols for fair electronic exchange. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing*, 2003.

[73] A. Mukhamedov and M. D. Ryan. Fair multi-party contract signing using private contract signatures. *Inf. Comput.*, 206:272–290, 2008.

[74] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. page 129, 2008.

[75] J. B. Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In *Advances in Cryptology—Crypto 2002*. 2002.

[76] J. B. Nielsen and C. Orlandi. Lego for two-party secure computation. In *TCC*, 2009.

[77] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, 1999.

[78] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *In Advances in Cryptology- Eurocrypt*, 1999.

[79] M. F. Payman Mohassel. Efficiency tradeoffs for malicious two-party computation. In *PKC*, 2006.

[80] T. P. Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, 1991.

[81] B. Pinkas. Fair secure two-party computation. In *EUROCRYPT*, 2003.

[82] O. Ruan, J. Chen, J. Zhou, Y. Cui, and M. Zhang. An efficient fair uc-secure protocol for two-party computation. *Security and Communication Networks*, 2013.

[83] O. Ruan, J. Zhou, M. Zheng, and G. Cui. Efficient fair secure two-party computation. In *IEEE APSCC*, 2012.

[84] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, pages 161–174, 1991.

[85] V. Shoup and R. Gennaro. Securing threshold cryptosystems against chosen ciphertext attack. *J. Cryptology*, pages 75–96, 2002.

[86] D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. In *EUROCRYPT*, 2007.

[87] A. C. Yao. Protocols for secure computations. In *FOCS*, 1982.

[88] M. Zamani, M. Movahedi, and J. Saia. Millions of millionaires: Multiparty computation in large networks. *IACR Cryptology ePrint Archive*, 2014:149, 2014.

[89] Y. Zhao, Z. Quin, T. Lan, H. Xiong, and F. Zhang. Multi-party fair exchange with an offline semi-trust third party. In *Communications, Circuits and Systems, 2007. ICCCAS 2007. International Conference on*, 2007.