Secure and fair two-party computation

Secure and Fair Two-Party Computation

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op woensdag 27 augustus 2008 om 16.00 uur

door

Mehmet Sabir Kiraz

geboren te Siirt, Turkije

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr.ir. H.C.A. van Tilborg
en
prof.dr. T. Lange

Copromotor:
dr.ir. L.A.M. Schoenmakers

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN
Kiraz, Mehmet Sabir
Secure and Fair Two-Party Computation/ door Mehmet Sabir Kiraz. –
Eindhoven : Technische Universiteit Eindhoven, 2008. –
Proefschrift. – ISBN 978-90-386-1369-7
NUR 919
Subject headings : Cryptology
Printed by Printservice Technische Universiteit Eindhoven.
Cover design: CreanzaMedia (www.creanza.nl).

To my mother...

Contents

1 Introduction

1.1 Background and Motivation

Consider a set of parties who do not trust each other. Nevertheless the parties wish to correctly compute some given function of their local inputs, while keeping their local inputs as private as possible. A popular example is the *millionaires' problem* [Yao82]. In this problem, there are two millionaires who wish to find out who is richer in such a way that their own fortunes remain private, but the correctness of the result can be checked by both of them. Another well-known example is from the *love game* (also known as *dating problem*) [CDG87]: Alice and Bob are very shy and would like to find out whether they are interested in each other. However, if only Alice is interested, then she does not want to let Bob know that she is interested in him. The same holds if only Bob is interested. In other words, if a party is not interested then it should not be possible to find out the other party's decision. The study of these kinds of problems belongs to the research area of *secure multiparty computation*.

It is easy to solve the problem if we assume the existence of a trusted third party which collects the inputs, computes the function and distributes the result to all participating parties. However, the problem becomes very challenging if we assume that there is no trusted third party available and parties can misbehave arbitrarily, i.e., they can send wrong messages or fail to send messages at all. Even then, the parties must be ensured that the protocol correctly and securely computes the function "as if" a trusted third party were available.

The concept of secure multiparty computation was first introduced by Andrew Yao in the early 1980s. Since then, many secure *cryptographic protocols* have been developed to provide various solutions for secure multiparty computation problems. Some well-known examples of such problems are: electronic voting, electronic auctions, contract signing, and private information retrieval schemes. Consider, for now, just the voting and auctions schemes. The requirements for a voting protocol ensure that no parties learn anything about the individual votes of other parties and no coalition of parties can influence the outcome of the election beyond just voting for their preferred candidate. In the same way, in an auction protocol, the requirements ensure that only the winning bid is revealed, and the highest bidder is indeed the party to win (and so the auctioneer, or any other party, cannot influence the outcome in an unfair way). This thesis will be concerned

with arbitrary functions instead of a particular function like those examples, and the main motivation of the thesis is thus to contribute to improve the existing solutions for general secure multiparty computation problems.

In general, a party involved in an execution of a secure multiparty protocol is considered to be either honest, semi-honest, or malicious. Honest parties always follow the protocol specifications and do nothing else. A semi-honest party also follows the protocol specifications without deviation, except for maintaining records of the entire internal messages during the flow of the protocol in order to later derive additional information. A malicious party, on the other hand, may deviate arbitrarily from the protocol at any time. The protocols explored in this thesis are all secure within either the semi-honest model or the malicious model, provided that the parties are computationally bounded.

The protocol by Yao [Yao82] only considers semi-honest parties. Goldreich, Micali and Wigderson [GMW87] generalized Yao's description to secure multiparty computation by allowing malicious parties. They show that for all multiparty problems there exists a secure protocol, when assuming an honest majority. In its most general form, there are n parties who wish to compute an agreed task on their private inputs without revealing these inputs, even if some of the parties are malicious. Secure multiparty computation can be obtained when any collection of parties is malicious as long as a majority is honest. Hence, the protocols for secure multiparty computation do not work if at least half the parties are malicious or trying to get information [Cle86]. The case when there are two parties is called *secure two-party computation* [Yao86]. In this thesis, we mainly focus on secure two-party computation, instead of general secure multiparty computation. In secure two-party computation at most one party is assumed to be malicious which is not a majority. Therefore, one of the major points for secure two-party computation is that fairness is a special property that one must either consider or ignore. Intuitively, fairness means that, if one party obtains its desired output, then the other party should also receive his desired output. In this thesis, fairness will be one of the major goals. In fact, the point for fairness is that one can weaken the requirements a bit (like using *gradual release*, or use trusted parties) to make fairness possible in a certain sense.

When designing a protocol, the main issues are the security the protocol and its complexity. That is, does the protocol maintain the security requirements? or, how is the running-time of the protocol? Protocol security measures the amount of information that can be gained during the protocol by any party. Complexity is a measure of efficiency which is often analyzed by three measures of complexity: *communication complexity*, *computational complexity*, and *round complexity*. Firstly, communication complexity of the protocol is the number of bits that needs to be exchanged between the two parties. Secondly, computational complexity is a measure of the amount of computational work that is necessary in order to complete the execution of the protocol. Finally, round complexity is the minimum number of rounds of interaction required to complete the protocol.

In the general case, the goal of a two-party protocol (for computing a function) is to compute a circuit, which is a representation of the function, as securely and as efficient as possible. A circuit can be an arithmetic or a Boolean circuit. The complexity of protocols depend on the size of the circuit. The depth of a circuit is also another important

characteristic for secure multi-party computation. Clearly, if the size of a circuit for the function is very large (or, if the depth is very large), the circuit evaluation protocols could not be efficient and hence not be practical. However, if the function is simple enough, using the circuit evaluation protocols can be practical. Since a circuit is by far not uniquely determined for a given function, and depending on the method of secure computation used, one may use different circuits.

The general secure multiparty computation problem (for any function) is solvable using circuit evaluation protocols. The following two types of circuit evaluation protocols are common in the literature [Yao82, GMW87, Gol04].

- **Protocols with interaction at the gate level (gate by gate):** In this case, there is an interaction between two parties where the evaluation of an arithmetic (or a Boolean) circuit is performed gate by gate, from input gates to output gates. When entering each step, the parties hold encrypted (or shared) values of the input wires of the circuit, and when the step is completed they hold encrypted values of the output wires of the circuit. And, during the evaluation of any basic step, no information is disclosed except at the final step. In the final step, the parties decrypt their encrypted values of the output wires together to compute the final results. Therefore, privacy of the input is achieved.

- **Protocols with interaction at the circuit level only:** This type of protocols is based on the *garbled circuit* (also called *circuit-scrambling*) technique introduced by Yao in [Yao82]. In this paper, Yao presented the first general protocol for secure two-party computation considering only the semi-honest model. And later it has been extended to the malicious case [Lin01, Pin03, MNPS04, FM06, Woo06, LP07]. Yao's protocol uses the underlying primitives (pseudorandom generator and oblivious transfer) as black-boxes which leads to efficient results in the computational setting, where Alice and Bob are assumed to be bounded to polynomial-time computation. Let Alice have private input x and Bob have private input y, and let f be the function that they wish to compute. For the sake of simplicity, let $f(x, y)$ be public output. In Yao's protocol, the function $f(x, y)$ is first represented by a Boolean circuit. Next, one party (say Bob) is the constructor who "encrypts" this circuit, and sends it to the other party Alice. This encrypted circuit is usually known as garbled circuit. Alice is the evaluator who is going to "decrypt" it.

In this thesis, we will improve upon existing protocols for secure two-party computation based on Yao's garbled circuits. Note that the garbled circuit is constructed in such a way that it reveals no information in its encrypted form and therefore Alice learns nothing from this stage. However, Alice can obtain the output $f(x, y)$ by "decrypting" the circuit. In order to ensure that nothing is learned beyond the output itself, this decryption must be privately performed by Alice who must only reveal the result $f(x, y)$. Without going into details, this is achieved by Alice obtaining a series of keys corresponding to her input x such that, given these keys and the circuit, the output value $f(x, y)$ (and only this value) is

obtained. Of course, Alice must obtain these keys without revealing anything about x, and this can be done by running a secure *1-out-of-2 oblivious transfer* protocol for each input bit (e.g., [EGL85]). Yao's approach is attractive since it is viable and leads to potentially practical protocols. The existing protocols in this category are already quite good (but with room for improvement).

1.2 Overview of This Thesis

After the brief introduction of the context and motivation of the thesis we are ready to give an overview of the thesis and our contributions.

Overview of the research:

This thesis presents efficient and secure two-party computation protocols that are secure in the semi-honest and the malicious models. Yao's garbled circuit is used as a main building block in the design of our protocols. The simplest secure two-party protocol with respect to semi-honest behavior was first considered in [Yao82]. The security of this protocol was proved formally by Lindell and Pinkas [LP04]. Therefore, in this thesis, we mainly concentrate on problems with respect to malicious behavior and construct a two-party protocol in the malicious case using suitable building blocks, and finally expand it to achieve an efficient fair and secure protocol.

Yao's garbled circuit is described in detail in this thesis, and it is first followed by a protocol in the semi-honest model which has a slight modification to Yao's original protocol. Yao's protocol is modified for the following reasons. Firstly, it gives a clear insight into our protocols for the malicious case, and it will be easily expanded to the malicious case in a modular fashion. Secondly, the modified version is analyzed according to the real/ideal simulation-based definition. When considering the protocols in each model all the necessary building blocks are described. For the malicious case, the possible weaknesses are discussed when the composition of such blocks are performed. In particular, we show that the protocols [Pin03, MNPS04, FM06] are vulnerable to a certain type of attack with the use of *Oblivious Transfer* (OT) in the malicious case. We discuss how this attack works, and how to modify the scheme to make it more secure. In this thesis, we also consider fairness for a two-party protocol using Yao's garbled circuit. The protocol by Pinkas [Pin03] is the only protocol in the literature which is based Yao's garbled circuit discussing fairness. In this thesis, we will show another subtle problem in [Pin03] which is in fact related to the fairness. We also describe this problem in detail, and finally propose a more efficient scheme that achieves fairness.

All the protocols in this thesis are analyzed according to the simulation-based definition. The security analysis is based on the results of [LP07]. We also present an improvement on the security proof of [LP07] by presenting a slight modification to the circuit so that the failure probability of the simulator in [LP07] is reduced.

Throughout this study, we will review a number of cryptographic primitives that are

used to solve secure two-party computation problems. For the sake of completeness, we also cover briefly several well-known approaches to secure two-party computation problems.

Apart from presenting two-party protocols based on Yao's garbled circuit approach, we also present an efficient and secure protocol for a primitive known as *Committed Oblivious Transfer*, which is a variation of OT that uses commitments for inputs and outputs. Our protocol is different for the following reasons. All the protocols implementing committed OT functionality in the literature allows transferring only bits rather than bit-strings, and therefore our protocol is the first one that allows transferring bit-strings. On the other hand, it is still comparable to the most efficient protocol in the literature when bits are transferred. Finally, we show that our protocol is very useful since it can also fix some subtle flaws like the attack described above with the use of OT.

Roadmap of the thesis and our main contributions:

Chapter 2: Preliminaries

In this chapter, we present notation, definitions and basic cryptographic primitives that will be used throughout the thesis.

Chapter 3: Secure Two-Party Computation

We outline secure two-party computation in general. Before we present the protocols using Yao's garbled circuit in the next chapters, we give an overview of several known approaches here to solve secure two-party computation problems.

Chapter 4: Combining Oblivious Transfer and Commitments

We present an efficient protocol for committed OT of bit strings. There are several protocols which use OT in the semi-honest model transferring bit-strings. We note that we may not achieve security when extending the same protocols simply to the malicious model, even if OT is secure in this model. We show that committed OT solves this problem immediately. However, there are several protocols for committed OT which transfer only bits whereas we consider the transfer of an arbitrary string of bit while the efficiency remains comparable to when bits are transferred. Our protocol is the first one that is efficient and transfers bit-strings. Also, we present a new variant of OT and commitments, the so-called *Committing Oblivious Transfer* which is a weaker version of committed OT—but may lead to more efficient schemes.

Parts of this chapter are based on [KSV07] (joint work with Berry Schoenmakers and José Villegas)

Chapter 5: Secure Two-Party Computation in the Semi-Honest Model

We give a detailed description of Yao's garbled circuit [Yao82] in order to use it later for our protocols. We start by presenting a protocol using Yao's garbled circuit on the basic level of security, namely in the semi-honest model. This protocol is a slight modification of Yao's protocol in order to be able to simulate according the real/ideal simulation-based definition, and to be able to extend it to resist the malicious behaviors of the malicious party in a modular way. We conclude this chapter by analyzing the security of this protocol and its complexity.

Parts of this chapter are based on [KS06b] (joint work with Berry Schoenmakers).

Chapter 6: Secure Two-Party Computation in the Malicious Model

Here, we look in detail at the malicious case, which is the situation where a malicious party exists. We describe a number of building blocks which are needed for the malicious case like *cut-and-choose techniques*, *commitment schemes*, *the equality-checker* scheme [FM06] and *majority circuit computation* [Pin03, LP07].

We address specific problems with previous protocols in this chapter. In this respect, we show a protocol issue with the use of OT in some of the protocols in the literature (e.g., [Pin03, MNPS04, FM06]). The protocol issue is rather serious since it is possible for a malicious party to learn the input of the honest party which compromises the overall protocol. We show that this issue can be fixed efficiently using a committing OT protocol. We also describe a problem related to "majority circuit" with a previous version of our protocols [KS06b, KS08] which is an observation by Pinkas [Pin08]. We note that this problem also appears for the protocol [Pin03]. We finally present a protocol which fixes these problems for the malicious case borrowing some well-known techniques. We note that the protocol which is presented in the semi-honest model (in Chapter 5) is extended to the malicious case in this chapter. We analyze the security of our protocol according to the ideal/real simulation-based definition and analyze its performance. The security analysis is similar to the one of [LP07]. Also, we show how to slightly modify the circuit such that the failure probability of the simulator in [LP07] is reduced.

Parts of this chapter are based on [KS06a, KS06b, KS08] (joint work with Berry Schoenmakers). Other parts of the chapter are based on [KSV07] (joint work with Berry Schoenmakers and José Villegas).

Chapter 7: Fair Secure Two-Party Computation

In this chapter, we present a fair and secure two-party protocol. There are not many papers based on Yao's garbled circuit discussing the fairness issue, and in fact the protocol by Pinkas [Pin03] is the only one up to now. However, we show that there is a subtle problem

with this protocol, compromising the overall protocol. Furthermore, Pinkas' protocol uses blind signatures in order to achieve fairness which makes the protocol rather complex and inefficient. We provide an alternative solution using OR-Proofs of Cramer et al.[CDS94] which leads to a more efficient design. While the security analysis of Pinkas' protocol is rather informal, ours is analyzed according to the real/ideal simulation-based definition. We finally analyze its performance.

Parts of this chapter are based on [KS06a, KS06b, KS08] (joint work with Berry Schoenmakers).

Chapter 8: Conclusions

We conclude with a summary of our work and suggestions for future research.

2
Preliminaries

In this chapter, we present the notation and the definitions that will be used throughout the thesis. We also review the basic cryptographic primitives needed to develop our two-party protocols. Additional definitions and preliminaries, which are specific to some parts of the thesis, appear within the following chapters.

2.1 Indistinguishability

For ease of reference, we include the three standard notions of indistinguishability, which are defined in terms of negligible functions and probability ensembles. Denote by $\mathbb{N}$ the set of integers.

Definition 2.1 (Negligible function) *A function*

$$\begin{aligned} \epsilon : \mathbb{N} &\mapsto [0,1] \\ k &\mapsto \epsilon(k) \end{aligned}$$

is negligible *if, for all positive polynomials $p(\cdot)$, there exists $k_0 \in \mathbb{N}$ such that, for any $k > k_0$ $\epsilon(k) < \frac{1}{p(k)}$.*

A very common example of a negligible function is the inverse exponential $\epsilon(k) = 2^{-k}$. We will define security against any feasible adversary and require that the probability it gets any information is a negligible function in a security parameter k. We note that nowadays for most applications, a value of about $k = 80$ should provide an adequate level of security. We can also say that if an event occurs with *overwhelming probability* if the probability that it does not happen is negligible.

Definition 2.2 (Probability ensemble) *Let X_w be a random variable on the set of strings of length polynomial in $|w|$ for $w \in \{0,1\}^*$. A* probability ensemble *$X = \{X_w\}_{w \in \{0,1\}^*}$ is an infinite set of random variables which ranges over strings of length polynomial in $|w|$.*

In general, there exists three standard variants of indistinguishability, namely *perfect, statistical* and *computational*. The security is referred to unconditional and computational according to the notion of one of these variants being achieved. For example, unconditional

security is considered if either perfect or statistical indistinguishability is achieved, and computational security is considered if computational indistinguishability is achieved.

First of all, two ensembles are called *computationally indistinguishable* if, given a sample from a probability distribution from the ensemble, no probabilistic polynomial-time algorithm can tell which ensemble that sample came from. More formally,

Definition 2.3 (Computational indistinguishability) *Two probability ensembles* $X = \{X_w\}_{w\in\{0,1\}^*}$ *and* $Y = \{Y_w\}_{w\in\{0,1\}^*}$ *are called* computationally indistinguishable *if, for every probabilistic polynomial-time algorithm A (called the "the Distinguisher"),*

$$|\Pr[A(X_w) = 1] - \Pr[A(Y_w) = 1]| \text{ is a negligible function in } |w|.$$

If X *and* Y *are computationally indistinguishable then we write* $X \stackrel{\mathcal{C}}{\equiv} Y$.

We note here that, the computational security generally relies on certain unproved assumptions about the hardness of some computational problem, like the hardness of factoring or discrete logarithm problem. In order to compute a function f correctly using a two-party protocol, we will compare different output distributions of this function.

Next, two ensembles are called *statistically indistinguishable* if, given a sample, there is no algorithm (even with unbounded computational power) which can tell with non-negligible probability to which ensemble that sample belongs. More formally,

Definition 2.4 (Statistical indistinguishability) *Let* $\{X_w\}_{w\in\{0,1\}^*}$ *and* $\{Y_w\}_{w\in\{0,1\}^*}$ *be two ensembles of probability distributions, where for each* w *both* X_w *and* Y_w *are defined over the same range* $\mathcal{D}$. *We say that two probability ensembles* X_w *and* Y_w *are* statistically indistinguishable *if the* statistical distance *between them is a negligible function in* $|w|$, *i.e.*

$$\triangle(X_w, Y_w) = \frac{1}{2} \cdot \sum_{\alpha\in\mathcal{D}} |\Pr([X_w = \alpha]) - \Pr([Y_w = \alpha])| < \epsilon$$

where $\epsilon = \epsilon(|w|)$ *is a negligible function.*

Definition 2.5 (Identical distribution) *We say that two probability ensembles* $X = \{X_w\}_{w\in\{0,1\}^*}$ *and* $Y = \{Y_w\}_{w\in\{0,1\}^*}$ *are* identically distributed *if, for every* w, $\triangle(X_w, Y_w) = 0$. *We write* $X \equiv Y$ *to denote identical distributions.*

We note that identical distributions are statistical indistinguishable, and statistical indistinguishability implies computational indistinguishability. The converse is not true in general.

2.2 Σ-Protocols

In this section we present Σ-*protocols* which are introduced by Cramer, Damgård and Schoenmakers in [CDS94, Cra97]. These protocols are used throughout the protocols in the malicious model in order to ensure the honest parties that the malicious parties cannot

violate during the protocol execution in order to gain advantage over the parties. Roughly speaking, in these protocols a prover convinces a verifier that a statement is true or on the knowledge of something, without revealing any additional information. For example, for homomorphic ElGamal encryptions and for Paillier encryptions [Pai99], there are efficient Σ-protocols for the relation $R_{\mathsf{enc}} = \{(e; m, r) : e = \mathsf{Enc}(m, r)\}$, proving knowledge of the message m and randomness r for a given encryption $e = \mathsf{Enc}(m, r)$ [CDS94].

Given access to a random oracle, a Σ-protocol can be transformed into a non-interactive Σ-protocol by using the Fiat-Shamir heuristic where the verifier does not need an interaction to achieve a proof of knowledge [FS86].

2.3 Non-Interactive Commitment Schemes

We next give an informal definition of a *non-interactive commitment scheme*. Roughly speaking, a commitment scheme between a committer and a receiver is a protocol in two phases, a commit phase and a reveal phase. In the commit phase, the committer chooses a private value m (a bit or a bit string) and computes $c = \text{commit}(m; r)$ where r is some randomness, and sends c to the receiver. In the reveal phase, the committer reveals the values m and r, where m is the bit string in the commitment c and r is the randomness used to form the commitment c. The receiver checks whether c is a valid commitment to m using the randomness r. In this thesis, we shall use the notation $\text{commit}_P(m; r)$ to denote a commitment to a message m using a random number r generated by the party P, and leave out P (or r) when this is clear.

A commitment scheme must satisfy the following security properties. Firstly, the commitment scheme must be *hiding* which means that a receiver cannot learn any information from the commitment c before the reveal phase. Next, the commitment scheme must be *binding* which means that it should be impossible for a committer to change the underlying value of the commitment c, i.e. find a way to open the commitment to another value which is different from m. To summarize, the hiding property protects against a malicious receiver while the binding property protects against a malicious committer.

Depending on the cryptographic protocols, the binding and the hiding properties can be required with either information-theoretic or computational security. It is well-known that a commitment scheme cannot be both perfectly binding and perfectly hiding. Typical commitment schemes in the literature are either information-theoretically binding and computationally hiding, or computationally binding and information-theoretically hiding. For example, efficient information-theoretic binding commitment schemes are described by Naor in [Nao91] where they are based on pseudo-randomness (no public key operations). Efficient information-theoretic hiding commitment schemes can be based on number theoretic assumptions, and use $O(1)$ exponentiations (see, e.g., [GMR88, Ped91]). For simplicity, non-interactive commitment schemes will be called commitments throughout the rest of the thesis.

2.4 Oblivious Transfer

In this section we examine *Oblivious Transfer (OT)*. OT was first introduced by Rabin [Rab81] and later many papers discuss possible extensions, variants and applications. Intuitively, Rabin's OT is an efficient protocol in which two parties, the sender (Bob) and the chooser (Alice) want to achieve the following. Bob has a private bit, and sends it to Alice who receives the bit with probability 1/2 while Bob does not know whether the secret has been received or not. Later, Even, Goldreich and Lempel [EGL85] presented 1-out-of-2 OT, where Alice has a private bit b, and Bob has two private bits s_0 and s_1. The protocol is correct when Alice and Bob follow the protocol, Alice outputs s_b with probability close to 1. Bob, on the other hand, does not output anything.

Crépeau [Cre87] later showed that Rabin's OT is actually equivalent to 1-out-of-2 OT. Namely, he showed that 1-out-of-2 OT can be implemented from Rabin's OT in polynomial number of steps. We note that today there are several OT protocols (e.g. [EGL85, NP01]) in the literature which can transfer not only bits but also bit strings which are profitable for many applications.

Note that 1-out-of-2 OT can also be generalized to 1-out-of-n OT where the sender has n messages $s_1, \ldots, s_n \in \{0,1\}$ and the chooser has a value $t \in \{1, \ldots, n\}$. At the end of 1-out-of-n OT the chooser obtains only s_t, and the sender does not know which one has been chosen. In this thesis, we are also interested in 1-out-of-2 OT which transfers bit strings. Unless stated, for notational simplicity we use the term OT instead of 1-out-of-2 OT throughout the rest of the thesis.

In general OT protocols require "public key" operations like *one-way trapdoor permutations* which are implemented using modular exponentiations. To give some examples, the security of Rabin's protocol for OT is based on the factoring problem, the security of the OT protocol by Even, Goldreich and Lempel is based on one-way trapdoor permutations [EGL85], and the security of the protocol by Naor and Pinkas [NP01] is based on the *Decision Diffie-Hellman* (known as DDH) assumption.

2.5 Threshold Homomorphic Cryptosystems

A threshold homomorphic cryptosystem is a specific type of public key cryptosystem, combining the properties of threshold cryptosystems and homomorphic cryptosystems, which we will briefly introduce below. A threshold scheme enables a private key to be shared among a set of n parties providing each party with a private key share. The scheme has a threshold t if combining the shares of i parties, $t \leq i \leq n$, enables the private key to be recovered and less than t parties can not uniquely determine the private key. We will use the notation (t, n) to refer to such a scheme. Let $\mathcal{M}$ (resp., $\mathcal{C}$) denote the set of the plaintexts (resp., ciphertexts). A (t, n)-*threshold cryptosystem* consists of the following components.

- A *distributed key generation protocol* allows parties $P_1, \ldots, P_n$ to, respectively, gen-

erate private key shares $sk_1, \ldots, sk_n$, and a public key pk.

- An *encryption algorithm* takes as input the public key pk and a plaintext $m \in \mathcal{M}$; it outputs an encryption $c = \mathsf{Enc}_{pk}(m)$ where $c \in \mathcal{C}$ and Enc_{pk} is an encryption function under public key pk.

- Based on the inputs public key pk, private key share sk_i and a ciphertext c, a *decryption protocol* allows parties $P_1, \ldots, P_n$ to, respectively, output a decryption share c_i with a proof of validity. Based on the ciphertext c and a list $c_1, \ldots, c_n$ of decryption shares, each party i then outputs either a plaintext m or "fails".

In the initialization phase of a (t, n)-*threshold cryptosystem*, the distributed key generation protocol is used to generate a public/private key pair (pk, sk) in such a way that the i-th party holds a private key share sk_i of the overall private key sk where $1 \leq i \leq n$. As in the basic public-key cryptosystem the public key pk allows anyone to encrypt plaintexts by running the encryption algorithm. For decryption of a ciphertext c, at least t parties must cooperate to jointly decrypt a ciphertext, whereas any collusion of less than t parties cannot get any information about the plaintext. To decrypt a ciphertext c, using their secret keys sk_i, each party runs the decryption protocol and outputs a decryption share c_i with a proof of validity. Finally, the parties recover the plaintext if enough decryptions are valid. An encryption scheme is said to be *homomorphic* if there exists an algorithm, which is the encryption function Enc_{pk} under a public key pk satisfies

$$\forall m_1, m_2 \in \mathcal{M}, \qquad \mathsf{Enc}_{pk}(m_1 \oplus_{\mathcal{M}} m_2) = \mathsf{Enc}_{pk}(m_1) \otimes_{\mathcal{C}} \mathsf{Enc}_{pk}(m_2)$$

for some operators $\oplus_{\mathcal{M}}$ in $\mathcal{M}$ and $\otimes_{\mathcal{C}}$ in $\mathcal{C}$. If $(\mathcal{M}, \oplus_{\mathcal{M}})$ and $(\mathcal{C}, \otimes_{\mathcal{C}})$ are groups, we have a group homomorphism. A scheme is called *additively homomorphic* if $\oplus_{\mathcal{M}}$ is addition, and *multiplicatively homomorphic* if $\oplus_{\mathcal{M}}$ is multiplication. If the homomorphic encryption scheme satisfies the above threshold properties then it is called *threshold homomorphic encryption*. Homomorphic encryption schemes can be very useful because it allows a third party to operate on encrypted plaintexts x, y based on the encryptions $\mathsf{Enc}_{pk}(x), \mathsf{Enc}_{pk}(y)$ without the knowledge of the plaintexts x, y or the decryption key. As a consequence, if the encryption scheme is additively homomorphic, given $\mathsf{Enc}_{pk}(m)$ and any constant c, encryption of cm can be efficiently computed as $\mathsf{Enc}_{pk}(m)^c$.

There are various instances of threshold homomorphic cryptosystems. The most widely used are (based on) ElGamal or Paillier. Threshold additively homomorphic ElGamal has the drawback of only allowing decryption of values belonging to a relatively small set, for which it is feasible to compute discrete logs. On the other hand, Paillier does not have this problem and allows decryption of encrypted values in an arbitrarily large set (e.g., 1024-bit integers). However, the distributed key generation protocol for threshold Paillier is computationally very expensive compared to threshold ElGamal.

3

Secure Two-Party Computation

In this chapter, we study the basic definitions for secure two-party computation. For the sake of completeness, we review several approaches for secure multiparty (in particular, two-party) computation. We refer to the textbook of Goldreich [Gol04] for a broader introduction to secure two-party computation.

3.1 Introduction

A *polynomial-time algorithm* is one that can be executed by a Turing machine in polynomial-time, proportional to the size of the input. The number of computational steps of a polynomial-time algorithm is always bounded by a polynomial function of the size of the input which is usually represented in binary.

A party executing a two-party protocol is modeled by an *interactive Turing machine* [GMR89]. Informally speaking, an interactive Turing machine is a probabilistic Turing machine with a communication tape that runs in polynomial-time in the length of its input. In general, running time is measured in terms of the number of elementary computations required as a function of input length. Therefore, it is independent of the platform.

A party which is involved in an execution of a protocol may be *honest* or *dishonest*. Honest parties always follow the protocol exactly as what is prescribed by the protocol and do nothing else. A dishonest party in a protocol, however, is modeled by assuming a central adversary who may corrupt a party, and get all the information known to this party. The computational power of the adversary is modeled by a *probabilistic polynomial-time* machine, namely an interactive Turing machine whose running time is bounded by a polynomial in a *security parameter*. Such a machine uses a source of randomness, such as a coin-flip, to basically determine the next move by randomly choosing one of the two possible alternatives. (Running such a machine multiple times on the same input may result in different outputs.)

An adversary can be either *passive* (also known as *semi-honest*) or *active* (also known as *malicious*). If an adversary is semi-honest then corrupted parties follow exactly the protocol specifications but attempt to learn extra information by examining the transcript of messages that it received during the protocol execution. If an adversary is malicious then he takes full control over corrupted parties. These corrupted parties behave arbitrarily

according to the instructions of the adversary. In other words, a malicious adversary may follow an arbitrary "feasible" strategy; that is, any strategy implementable by probabilistic polynomial-time algorithms.

Also, an adversary may be *static* or *adaptive* (dynamic). A static adversary has to determine which parties to corrupt before the execution of the protocol starts, and these corrupted parties remain the same through the protocol execution. Adaptive adversaries on the other hand can choose the parties dynamically during the execution of the protocol, namely at any point during the computation. This may depend on the information it records before deciding which party it is going to corrupt. In this thesis, we only consider static adversaries which can corrupt only one party.

In this thesis, we express complexity of our protocols in terms of the input length and a security parameter n (which is usually defined as a function of n rather than $\log n$). And the efficiency will be measured by considering communication, computational and round complexity. The communication complexity of the protocol is the number of bits sent to each other. That is, it is the total number of bits the parties exchange. One of the goals of the parties is to compute a public function with a two-party protocol with the least amount of communication between them. The computational complexity is a measure of the amount of computational work that is necessary in order to complete the execution of the protocol. As a computational complexity measure, we use the number of modular exponentiations (or public key operations) necessary for a protocol to be executed for all the parties. By the round complexity we mean the number of exchanged messages, or rounds of interaction in a protocol [1]. A very important note to keep in mind is that if some tasks can be performed in parallel by parties, then they can decrease the overall number of rounds. In this thesis, we give a comparison of our proposed solutions with the existing solutions to secure two-party computation considering these above measurements.

We are now ready to review the required background knowledge on secure two-party computation.

3.2 Secure Two-Party Computation

In secure two-party computation, there are two parties, Alice and Bob, where Alice holds a private input x and Bob holds a private input y. Both Alice and Bob wish to jointly and securely perform a computation based on their inputs that maps pairs of inputs (one input for each party) to pairs of outputs (one output for each party). Such a process is called as the desired functionality, and denoted

$$f : \{0,1\}^* \times \{0,1\}^* \rightarrow \{0,1\}^* \times \{0,1\}^* \text{ where } f = (f_1, f_2).$$

Note that the term functionality is used rather than function, since the parties of a protocol have separate inputs, and, in general, wish to compute different functions of the common

[1] We call each sending of a message by a party a *round*. In some works it is called a *move*, and two consecutive moves are said to make a round. So, according to this terminology, our two-round protocol would be a two-move or one-round protocol.

input. At the end of the computation, both parties should know the output of f. That is, for every pair of inputs (x, y), the desired output-pair is $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. Alice holds an input x, wishing to obtain $f_1(x, y)$ and Bob holds an input y, wishing to obtain $f_2(x, y)$ (but there might be only a common output $(f_1(x, y) = f_2(x, y))$, or only one party that has a private output).

In this setting, a two-party protocol is considered to implement f, and an adversary (a semi-honest, or a malicious) is only allowed to corrupt Alice or Bob, and cannot corrupt both Alice and Bob simultaneously. Let (x, y) be an input, f be as above and Π be a two-party protocol computing f. By saying Π for computing f we mean that Π correctly computes f when both parties honestly follow the protocol specifications. The protocol Π is a secure two-party protocol if the following security properties hold:

- *Correctness.* The output that the parties receive is guaranteed to be correct. In other words, no party can influence the output of the computation other than by changing its own input.
- *Privacy.* No party should learn anything more than what is implied by the function f and its own input. In other words, no party can obtain information about the honest party's input and what it sees from the protocol execution except what it can derive from its own input and the output.
- *Fairness.* A corrupted party should receive its output if and only if the honest party also receives its output. In general, if a party is corrupted, then the adversary may learn the output before the honest party obtains it and then may decide to abort the protocol. This may result in an unfair situation.

Note that we consider $f(x, y)$ to be a deterministic function in this thesis, rather than probabilistic, for each input (x, y). For the more general setting in which $f(\cdot, \cdot)$ is a probabilistic function, please see [Gol04].

3.3 Formal Definitions

The security definition for two-party computation varies depending on whether the adversary is semi-honest, or malicious. The security definition we present refers to the stand-alone setting where only a single protocol execution is run in isolation. We note that security in this setting does not imply security in the setting of arbitrary or concurrent composition, where many protocol executions take place simultaneously. Such a definition of security in a composable setting is known as the Universally Composable (UC) Security paradigm of Canetti [Can01]. In this thesis, we will not cover UC framework, rather we only analyze the security of protocols in the stand-alone setting.

The security of a protocol is analyzed by comparing the adversarial behavior in a real protocol and in the ideal world that is secure by definition. In the ideal world an incorruptible *trusted third party* to whom the parties send their inputs computes the function

on the inputs and returns to each party their respective output. Informally speaking, a protocol is secure if any adversarial behavior in the real protocol (where no trusted third party exists) can be carried out in the ideal world. Demonstrating the provable security of a cryptographic protocol against a malicious adversary is usually a difficult task since some malicious behavior cannot be prevented like refusing to participate in the protocol or entering a different input. These kinds of behavior are also included in the ideal model.

We shall now discuss this in greater detail. Without loss of generality, we assume that the inputs x and y are both of the same length. If this is not the case, padding may be applied. This assumption allows us to use the input length as a security parameter (see discussion in [Gol04, Section 7.2]).

3.3.1 Execution in the Ideal Model

Ideally, each party sends its input to a trusted third party over a private channel, who will compute the result and send it to both parties. The goal of a secure two-party computation is to eliminate the need for a trusted third party, while preserving the security as in the ideal scenario even in the presence of malicious parties.

Ideal scenario in the semi-honest model.

We first describe the ideal scenario in the semi-honest model. Let Alice and Bob be two parties with private inputs x and y respectively. Let $\mathcal{A}$ be an adversary with auxiliary input z, and $\mathcal{F}$ be the trusted third party. More formally,

- *Inputs:* Each party has an input, say x for Alice, and y for Bob. A party always sends its input to $\mathcal{F}$.
- *Trusted party answers back:* $\mathcal{F}$ (for computing f), replies to both parties with $f_1(x, y)$ and $f_2(x, y)$, respectively.
- *Outputs:* An honest party always outputs the message received from $\mathcal{F}$. A corrupted party may output an arbitrary (polynomial-time computable) function of its initial input and the message received from $\mathcal{F}$.

Note that the adversary here attempts to learn something from the party's view which consists only of its local input and output.

Ideal scenario in the malicious model.

Next we describe the ideal scenario in the malicious model. Note that there are three things we cannot avoid for no matter what protocol we use. First of all, the definition below does not preserve fairness (early aborting) since this cannot be prevented because of the impossibility result of Cleve [Cle86]. Secondly, we cannot avoid parties refusing to participate in the protocol when the protocol is first invoked. Thirdly, substituting a

different input rather than their local input is also unavoidable. Therefore, the ability of the adversary for these three actions is also included into the definition of the ideal scenario.

- *Inputs:* Each party has an input, denoted ω ($\omega = x$ for an honest Alice, and $\omega = y$ for an honest Bob).
- *Sending inputs to the trusted party:* An honest party always sends ω to the trusted party $\mathcal{F}$ (for computing f). A corrupted party may, depending on ω, either abort or send some $\omega' \in \{0,1\}^{|\omega|}$ to $\mathcal{F}$ [2].
- *Trusted party answers first party:* In case $\mathcal{F}$ (for computing f) has obtained an input pair (x, y), it first replies to the first party with $f_1(x, y)$. Otherwise (i.e., in case it receives only one valid input), $\mathcal{F}$ replies to both parties with a special symbol $\perp$.
- *Trusted party answers second party:* In case the first party is malicious it may, depending on its input and $\mathcal{F}$'s answer, decide to "stop" $\mathcal{F}$ by sending it $\perp$ (which denotes an error symbol) after receiving its output. In this case $\mathcal{F}$ sends $\perp$ to the second party. Otherwise (i.e., if not stopped), $\mathcal{F}$ sends $f_2(x, y)$ to the second party.
- *Outputs:* An honest party always outputs the message received from $\mathcal{F}$. A corrupted party may output an arbitrary (polynomial-time computable) function of its initial input (auxiliary input and random-tape) and the message received from $\mathcal{F}$.

3.3.2 Execution in the Real Model

We now consider the real model in which a real two-party protocol is executed where there exists no trusted third party. Let f be as above and let Π be a two-party protocol for computing f. Furthermore, let $\overline{M} = (M_1, M_2)$ be a pair of probabilistic polynomial-time algorithms (representing the two parties in the real model). Such a pair $\overline{M} = (M_1, M_2)$ is admissible (for the real semi-honest or malicious model) if for at least one $i \in \{1, 2\}$ we have that M_i is honest (i.e., follows the strategy specified by Π). Then, the execution of Π under $\overline{M}$ in the real model (on input pair (x, y)), denoted $\mathrm{REAL}_{\Pi,\overline{M}}(x, y)$, is defined as the output pair of M_1 and M_2 resulting from the protocol interaction between M_1 and M_2.

3.3.3 Security as Emulation of a Real Execution in the Ideal Model

Having defined the ideal and the real models, we can now present security definitions of protocols in the semi-honest and in the malicious models. Loosely speaking, the definition

[2] Note that this restriction is done between w and w' because the trusted party is assumed to accept the length of w.

states that a secure two-party protocol (in the real model) emulates the ideal model (in which a trusted party exists). This is formulated by saying that admissible adversaries in the ideal model are able to simulate admissible adversaries in an execution of a secure protocol in the real model.

The term *perfect security* is used if the view of the adversary in the real world *and* the simulated view of the adversary in the ideal world are equally distributed, the term *statistical security* is used if they are statistically indistinguishable, and the term *computational security* is used if they are computationally indistinguishable.

Security in the semi-honest model.

We first present the security definition in the semi-honest model. We begin with the following notation: let $f = (f_1, f_2)$ be a function and let Π be a two-party protocol for computing f.

Definition 3.1 (Security in the Semi-Honest Model.) *Let f and Π be as above. The protocol Π is said to be statistically (computationally) secure in the semi-honest model if, for every probabilistic polynomial-time pair of algorithms $\overline{R} = (R_1, R_2)$ that is admissible for the real model (of Section 3.3.2, Semi-honest), there exists a probabilistic polynomial-time pair of algorithms $\overline{S} = (S_1, S_2)$ (called simulator) that is admissible for the ideal model (of Section 3.3.1, Semi-honest), such that*

$$\{IDEAL_{f,\overline{S}}(x,y)\}_{x,y \text{ with } |x|=|y|} \stackrel{\mathcal{S}}{\equiv} \{REAL_{\Pi,\overline{R}}(x,y)\}_{x,y \text{ with } |x|=|y|}$$

holds (in case of statistical security), or

$$\{IDEAL_{f,\overline{S}}(x,y)\}_{x,y \text{ with } |x|=|y|} \stackrel{\mathcal{C}}{\equiv} \{REAL_{\Pi,\overline{R}}(x,y)\}_{x,y \text{ with } |x|=|y|}$$

holds (in case of computational security).

Note that the above definition implies that the parties already know the input lengths (by the requirement that $|x| = |y|$). Observe that the difference between honest and semi-honest parties is simply in their actions on the corresponding local views. An honest party outputs only the output part of the view, whereas a semi-honest party may output an arbitrary ("feasible" computable) function of the view.

Security in the malicious model.

We now consider the security definition in the in the malicious model. In the real model, a two-party protocol is executed, and there exists no trusted third party. In this case, a corrupted party may follow an arbitrary feasible strategy, that is, any strategy implementable by a probabilistic polynomial-time algorithm (which gets an auxiliary input). In particular, the malicious party may abort the execution at any point in time, and when this happens prematurely, the other party is left with no output.

The security definition in the malicious model is similar to Definition 3.1, where the polynomial-time pair of algorithms $\overline{R} = (R_1, R_2)$ that is admissible for the real model are defined as in the malicious case of Section 3.3.2, and the pair of algorithms $\overline{S} = (S_1, S_2)$ that is admissible for the ideal model are defined as in the malicious case of Section 3.3.1.

3.3.4 Hybrid Model

In the hybrid model, parties run an arbitrary protocol like in the real model, but have access to a trusted party that computes some function like in the ideal model. Namely, the hybrid model is a straightforward mix of the real and ideal models. It is exactly the same as the real model except that all parties also have access to an ideal function f. Inputs for the function can be influenced by the surrounding protocol. Hence, a protocol that is designed for a hybrid model contains two types of messages: real messages that are sent directly between the parties, and ideal messages that are sent between the parties and the trusted third party. In the real model, parties run an arbitrary protocol simultaneously instead of using the trusted party. A protocol is secure if any attack on the real model can be carried out in the hybrid model.

The protocols in Chapter 5, 6 and 7 use a secure oblivious transfer (OT) protocol as a subprotocol. In [Can01, AL07] it is shown that it is sufficient to analyze the security of such a protocol in a hybrid model in which the parties interact with each other and assumed to have access to a trusted party that computes an OT protocol for them. Thus, in the security analysis of our protocol the simulator will play the role of the trusted party for the OT functionality when simulating the corrupted party.

3.4 Fairness

In this section, we review the fairness property of secure two-party computation protocols. We refer to [Pin03] for a list of references on fairness.

3.4.1 Complete Fairness

A two-party protocol (computing a function f) is said to be fair if either both parties learn the output of f, or no party learns the output. That is, whenever a party aborts the protocol prematurely, it should not gain any advantage on the output over the other party. This is also called *complete fairness*- in the sense of both parties obtaining the outputs at the same time. It is an important requirement for secure two-party computation problems since aborting a protocol prematurely is an unavoidable issue in the presence of malicious behavior.

It is shown by Cleve [Cle86] that complete fairness is actually impossible to achieve for any two-party protocol. The reason of this impossibility result is roughly as follows. Assume that in a protocol parties start exchanging messages back-and-forth to get their

outputs. Observe that whenever a party sends the last piece of critical secret information and since he is not going to receive any critical secret information anymore at this time, he must already have received his secret information. He may not complete the protocol by not sending his last piece of critical private information to the other party which leads to an unfair situation.

3.4.2 Relaxed Fairness

Because of the impossibility result of Cleve for complete fairness, researchers are interested to achieve alternative (relaxed) solutions. In this thesis, we will be interested in the so-called *gradual release* approach to achieve fairness in which no trusted third party is involved to complete the protocol. In this approach, two parties interactively run a protocol to control the unfair situation. The parties take turns to release their secrets in a gradual fashion. Assume that the secrets are from a uniform distribution, so that any party does not know more information about the secret than what has been already revealed. By gradually releasing the secrets, the computational effort to reconstruct the respective message without help of the other party decreases. For example, suppose there are two parties, each possessing a secret. Suppose further that both secrets are important contents to the other party, and that they are therefore willing to "trade" the secrets against each other. If the two secrets are represented as bit strings of the same length, this can be solved by exchanging the secrets bit by bit; if this is done honestly, no party will be more than one bit ahead of the other at any time during the exchange. Therefore, if one party aborts the protocol, this party will have only a limited computational advantage over the other party. Informally speaking, early aborting gives only a negligible advantage to the corrupted party, and the honest party in a fair protocol may run longer than a corrupted one. Because of the impossibility result of Cleve this advantage is unavoidable. We note that an honest party may have given away its secret correctly and that, in return, he has been given "garbage" instead of correct bits of the secret. Hence, it is also necessary that the other party can verify for each part of the released secrets that it has been given correct information.

One of the main ideas of a secure two-party protocol to achieve fairness is to compute commitments (or encryptions, shared values) to output values. Basically, Alice will hold commitments for Bob's output, and, vice versa, Bob will hold commitments for Alice's output. Finally, the parties gradually open the commitments. Note that these commitments (or encryptions, shared values) are of special type in order to apply gradual release. That is, not any commitment scheme is applicable to a gradual release approach. For example, in the protocol by Pinkas [Pin03, Appendix A], *timed commitment* of Boneh and Naor [BN00] is used. Informally speaking, a timed commitment is a commitment that, in addition, the receiver of the commitment can find the committed value by running a computation of 2^k steps where k is a security parameter (i.e. called "forced opening"). If the parties interactively run the opening of the timed commitments, then they can open in only k steps. However, if at some point a corrupted party aborts then the honest party

has to invest at most twice computationally effort than the corrupted party in order to obtain its output.

The crucial and necessary point is now that both parties are convinced of the correctness of the values contained in the commitments held by the other party. For this we need some special protocol techniques. For example, in [Pin03], blind signatures are used as a building block to achieve correctness of these commitments.

3.4.3 How to Prove Fairness

Recently, Garay et al. [GMPY06] presented the notion of *resource-fair* protocols which is a relaxation of fairness. Informally speaking, it states that if one party learns the output of the protocol, then so can all other parties, as long as they use roughly the same amount of resources. Their definition follows the standard simulation paradigm in the UC framework of Canetti [Can01], and works in a model that allows any party to request additional resources to deal with malicious parties that may prematurely abort.

The authors in [GMPY06] start by presenting a complete fair function $\mathcal{F}_f$ for computing a function f. A critical part of $\mathcal{F}_f$ is that there are certain messages that $\mathcal{F}_f$ sends to both parties such that both of them must receive the message in the same round of communication. (For this it is necessary that the adversary in the ideal model cannot block messages from $\mathcal{F}_f$ to the honest parties.) However, in order to define fairness formally, the authors in [GMPY06] provided a relaxation of $\mathcal{F}_f$ that can be realized in terms of resource fairness (in their work, the resource is defined as time). Omitting details, the authors used a fairly "wrapped" version of $\mathcal{F}_f$ (which they call *wrapper functionality* denoted by $\mathcal{W}(\mathcal{F}_f)$) in order to fairly send the messages to each party. Namely, both parties will not receive the outputs in the same round (in case one party aborts prematurely) but it will be quantified such that the honest party is ensured to receive its output. In the real model, the wrapper functionality is in fact implemented using the gradual release approach.

We remark that the wrapper functionality is what makes actually a protocol resource-fair. Roughly speaking, the wrapper functionality allows the adversary to make "deals" of the following kind: even if $\mathcal{F}_f$ requires a message to be simultaneously delivered to all parties, the adversary can "invest" computational resources and obtain the message from $\mathcal{W}(\mathcal{F}_f)$ in an earlier communication round. However, in this case, $\mathcal{W}(\mathcal{F}_f)$ will offer a "fair deal" to the honest parties: each of them will be given the option of obtaining its message by investing (at most) the same amount of computational resources as invested by the adversary.

In the same paper, they later define a particular functionality called *commit-prove-fair-open functionality* ($\mathcal{F}_{\text{CPFO}}$), which allows each party to commit to a value, prove relations about the committed value, and more importantly, open all committed values simultaneously to all parties. They then design an efficient resource-fair protocol that securely realizes it with (the fairly "wrapped" version of) this functionality. The wrapped version of this functionality ($\mathcal{W}(\mathcal{F}_{\text{CPFO}})$) takes care of the opening phase, and makes it fair.

The authors in [GMPY06] finally show that by using the $\mathcal{F}_{\text{CPFO}}$ functionality, many

existing secure protocols can be transformed into resource-fair protocols while preserving their security. For a particular example, they showed how to use it to add fairness to the result of the "gate-by-gate" approach in the setting of [CDN01]. For the protocol in [CDN01], they first modified the cryptosystem to (n, n)-threshold scheme so that all parties must cooperate in order to decrypt the encrypted message. As we said in Section 2.5, the protocol in [CDN01] contain an "output" phase, in which every party broadcasts its decryption share with a proof of validity, and once all shares are broadcasted, each party performs the last steps of the decryption on its own. Finally, [GMPY06] modified the joint decryption phase in order to achieve fairness. That is, instead of the parties revealing their secrets directly in the output phase, the parties invoke the wrapper functionality $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$ to receive their secrets fairly, i.e. the parties first commit to their decryption shares and prove the correctness of them. Once the correctness of the commitments is guaranteed, parties will gradually open these commitments.

A consequence of [GMPY06] is that, the resulting fair protocol is simulatable. The protocol of [GMPY06] is attractive since it is also secure against *parallel attacks*: having many more computers does not give any advantage to the malicious parties over having a single computer.

In line with this, to show that our protocol in Chapter 7 is fair we propose to use as a subroutine the protocol of [GMPY06]. The main idea of our protocol is first to show the correctness of the commitments to outputs at the end, and then apply the gradually release phase to achieve directly fairness. For this gradual release phase, we will use the protocol of [GMPY06] implementing the wrapper functionality $\mathcal{W}(\mathcal{F}_{\text{CPFO}})$, which is actually shown to be simulatable. We will use the results of [GMPY06] in a black-box way (in the hybrid model).

3.5 Probabilistic vs. Deterministic Functions

In this section, we discuss some issues related to probabilistic and deterministic functions.

Deterministic functions.

In our thesis, we only show how to securely compute *deterministic functions*. Having secure protocols for deterministic functions suffices to obtain secure protocols for arbitrary probabilistic functions. We show this in the following theorem of [Gol04].

Let $x, y \in \{0,1\}^*$ be input of Alice and Bob respectively, and f be a probabilistic function. Let $f((x,y),r)$ denote the value of $f(x,y)$ when random bit strings r of length polynomial in $|x|$ are used. In other words, $f(x,y)$ is a probabilistic function consisting of selecting random bit strings r polynomial in $|x|$, and deterministically computing $f((x,y),r)$. Next, define a deterministic function $\hat{f}((x,r_1),(y,r_2)) \stackrel{\text{def}}{=} f((x,y),(r_1 \oplus r_2))$ where r_1, r_2 are random bit strings polynomial in $|x|$.

Theorem 3.2 ([Gol04]) *Let f be a probabilistic function, and $\hat{f}$ be a deterministic func-*

tion as defined above. If, for any $\hat{f}$, there is a secure protocol $\widehat{\Pi}$ which securely computes it then it is possible to construct a secure protocol Π for computing any f.

Intuitively, assume that we have a secure protocol $\widehat{\Pi}$ for computing $\hat{f}$. Now, the following is a secure protocol Π for computing the function f. Alice and Bob choose bit strings r_1 and r_2 uniformly at random, respectively. They then invoke the protocol $\widehat{\Pi}$ for securely computing $\hat{f}$ in order to obtain $\hat{f}((x, r_1), (y, r_2)) = f((x, y), (r_1 \oplus r_2))$.

The formal proof that the protocol Π securely computes the probabilistic function f is given in [Gol04, Section 7.3]. Note that the size of the circuit computing f is of the same order as the size of the circuit computing $\hat{f}$. The only difference is that the circuit for $\hat{f}$ has $|r|$ additional exclusive-or gates, where $|r|$ is the length of f's randomness.

Therefore, for the sake of simplicity we present two-party protocols only for deterministic functions in this thesis.

A simpler definition for deterministic functions.

As we said before, to analyze the security of a protocol we have to construct a simulator in such a way that the view of a corrupted party in the real model can be simulated by a probabilistic polynomial-time algorithm in the ideal model. We here note that it is not sufficient for a simulator to generate a transcript indistinguishable from view of the corrupted party in the case that the function f is probabilistic. The *joint distribution* of the simulator's output *and* the function output $f(x, y)$ should also be considered, and that must be indistinguishable from view of the corrupted party *and* the output of the protocol. On the other hand, if the function f is deterministic, it is sufficient to show that a simulator generates the view of a party, without considering the joint distribution with the output. The reason is that when f is deterministic the output of the protocol must be equal to $f(x, y)$.

3.6 Approaches to Secure Computation

All the current solutions to general secure two-party computation problems are either based on arithmetic circuits or Boolean circuits implementing the function f. In this section, we review the most common approaches for secure-two party computation.

3.6.1 The Gate-by-Gate Approach

Most protocols for general secure multiparty computation (in particular, secure two-party computation) are based on the "gate-by-gate" approach where each gate of the circuit C_f is computed securely by performing a secure protocol. In this section, we review some of the main solutions using the "gate-by-gate" approach for solving secure two-party computation problems.

Using Oblivious Transfer.

OT is a powerful primitive for secure multiparty computation since it has been shown by Kilian [Kil88] to be complete for multiparty computation problems. That is, it is possible to securely evaluate any polynomial-time computable function based only on an implementation of OT.

We now show that using any 1-out-of-4 OT protocol every function f, which has polynomial-size circuits, one can construct a secure computation protocol. We describe only the two-party protocol in detail, and explain briefly how this protocol can be easily extended to the case where the number of parties is more than 2.

The function f that the parties wish to compute is first represented as a Boolean circuit C_f which consists of only AND and XOR gates. We here note that any Boolean circuit can be rewritten using only AND and XOR gates. Next, we show how the parties compute AND and XOR gates.

Before the computation starts the parties share the values of each wire. More precisely, let x_A, y_A be shares of Alice and x_B, y_B be shares of Bob where $x = x_A \oplus x_B$ and $y = y_A \oplus y_B$ where they want to compute $x \wedge y$ and $x \oplus y$ respectively as follows. Observe that AND is the multiplication and XOR is the addition modulo 2.

- *AND gate.* We first show how the parties securely compute an AND gate, i.e. they wish to compute the value $z = x \cdot y \mod 2$. Alice first chooses z_A randomly.

  She then prepares a four-tuple by computing for each pair of values x_B and y_B a value z_B by means of

  $$z_B = ((x_A + x_B) \cdot (y_A + y_B)) + z_A \mod 2. \text{ (see Table 3.1)}$$

(x_B, y_B)	(0,0)	(0,1)	(1,0)	(1,1)
z_B	$x_A y_A + z_A$	$x_A(y_A + 1) + z_A$	$(x_A + 1)y_A + z_A$	$(x_A + 1)(y_A + 1) + z_A$

Table 3.1: 1-out-of-4 OT Input/output

  Alice and Bob then run a 1-out-of-4 OT protocol where Bob chooses the z_B value based on his private shares x_B and y_B. From the security of the OT protocol, Bob learns no additional information about Alice's shares x_A, y_A, z_A, and similarly Alice learns nothing about Bob's shares x_B, y_B, and z_B.

- *XOR gate.* We next show how the parties securely compute an XOR gate where they wish to compute the value $z = x + y \mod 2$. In this case, Alice computes $z_A = x_A + y_A \mod 2$ and Bob computes $z_B = x_B + y_B \mod 2$ without any interaction. It is easy to see that z_A, z_B are already shares of the XOR gate's output, and since there is no interaction a corrupted party cannot learn any extra information from the other party.

At the end, each party holds a share of each output wire. To obtain the desired output, every party sends its share of each output wire to all parties. Then each party sums up all the shares to obtain the desired output.

As we mentioned earlier, the above solution can be easily extended to secure multiparty computation. In short, it is given as follows. Suppose that there are n parties and let x_i and y_i denote the shares of the i-th party where $x = x_1 \oplus \ldots \oplus x_n$ and $y = y_1 \oplus \ldots \oplus y_n$. To compute the XOR of bit x and y, each party i where $1 \leq i \leq n$ simply computes $z_i = x_i \oplus y_i$ as in the two-party case. To compute the AND of bit x and y, we follow the approach in [Gol04]:

$$(x_1, y_1), \ldots, (x_n, y_n) \rightarrow (z_1, \ldots, z_n) \text{ where } \sum_{i=1}^{n} z_i = \left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} y_i\right) \mod 2$$

Actually, the solution comes from the following observation:

$$\left(\sum_{i=1}^{n} x_i\right) \cdot \left(\sum_{i=1}^{n} y_i\right) = n \cdot \sum_{i=1}^{n} x_i \cdot y_i + \sum_{1 \leq i < j \leq n} (x_i + x_j) \cdot (y_i + y_j) \mod 2$$

(see [Gol04, Section 7.5.2] for the proof of this equality). Therefore, each party i can compute $n \cdot x_i \cdot y_i$ without any interaction, and parties i and j can jointly compute $(x_i + x_j) \cdot (y_i + y_j) \mod 2$ as in the two-party case described above using a 1-out-of-4 OT.

Using threshold homomorphic cryptosystems.

We now review the use of *threshold homomorphic cryptosystems* for secure two-party computation. There are several protocols in the literature that use this setting based on using an arithmetic circuit or a Boolean circuit. In the case of arithmetic circuits, the evaluation is performed from the input gates to output gates. We note that the addition gates can be computed without any interaction and without decrypting any value due to the homomorphic property of the cryptosystem. Addition gates can be evaluated without having to decrypt any value because of the homomorphic property of the underlying cryptosystem. Multiplication gates, however, are computed via a *secure multiplication* protocol by an interaction where decryption is required even in the semi-honest model. In the malicious model, multiplication gates additionally require the use of zero-knowledge proofs (e.g., [CDN01, ST04]).

Jakobsson and Juels [JJ00] also presented a protocol for secure multiparty computation. It is based on homomorphic ElGamal cryptosystem, using a so called *mix-and-match* scheme. In this protocol, the function that the parties wish to compute is first represented as a Boolean circuit. Each party encrypts the entries of the standard truth table for every gate using a homomorphic ElGamal cryptosystem, replaces the entries with the corresponding ciphertexts and randomly permutes rows of the table. In general, the mixing uses the homomorphic property that allows a party to change the encryption without knowing the correct decryption. For each gate, the parties compare the encrypted values

they hold with the encrypted values in the blinded permuted truth table to determine the correct row. When the correct row in the table is found, the parties obtain an output ciphertext from the third column. Since the values are encrypted and permuted by each party, nobody learns the plaintext corresponding to the output value. The ciphertext for the output of the gate is used as input to the truth table of the next gate. The parties do this computation from input gates to output gates. After the truth table of the output gate is computed parties decrypt the final ciphertext. We note that this construction is similar to Yao's garbled circuit except that the evaluation of the circuit is done gate by gate based on threshold cryptosystems.

3.6.2 Yao's Garbled Circuit Approach

A well-known alternative solution to the "gate-by-gate" approach, is Yao's *garbled circuit* approach. In his seminal paper, Yao [Yao86] designs a method for generating an encrypted circuit and uses it to obtain a general secure two-party protocol which is secure in the presence of semi-honest adversaries. In this protocol, there is a constructor (Bob) and an evaluator (Alice). On a very high level, the method consists of three procedures:

1. The function f is first represented as a Boolean circuit. Bob uses an algorithm to "encrypt" this circuit. The encrypted circuit is called as *garbled circuit*.

2. An interactive protocol is run between Alice and Bob in order to send the correct decryption keys to Alice.

3. Based on the decryption keys Alice uses an algorithm to "decrypt" the garbled circuit.

The original implementation of the protocol, due to Yao [Yao86], makes a black-box use of the underlying primitives a pseudorandom generator and OT. As we said, to decrypt the garbled circuit Alice needs to learn the decryption keys. To do so, a 1-out-of-2 OT is run to receive the correct decryption keys. Alice then decrypts the garbled circuit with no further interaction with Bob and sends his output back.

Yao's garbled circuit approach is quite different than the "gate-by-gate" approach. In the garbled circuit approach, the "encryption" of the whole logical circuit is done by Bob, and the "decryption" of the whole logical circuit is done by Alice (with no interaction). Additionally, by using the garbled circuit approach it is always possible to achieve a constant number of rounds (only a few rounds). However, in the "gate-by-gate" approach the parties evaluate the circuit which describes the function to be evaluated gate by gate which in general results in polynomial number of rounds depending on the depth of the circuit. Another difference is that the computational setting. Any secure private-key encryption scheme can be used to construct a garbled circuit, and in this way the garbled circuit is efficiently "encrypted" and "decrypted". However, the only additional (major) computational cost is needed for OT protocol.

Yao's garbled circuit is usually limited to two parties, but has been extended to handle multiparty inputs using similar settings. For example, Chaum, Damgård and van de

Graaf [CDG87] presented a circuit scrambling technique that is similar to Yao's garbled circuit to achieve secure multiparty computation. However, this protocol does not run in constant number of rounds and the rounds depend on the size of the circuit. Also, Beaver, Micali and Rogaway later [BMR90] generalized Yao's original protocol [Yao86] to the case of multiparty computation with an honest majority while preserving the constant round complexity. In the construction of [BMR90] the circuit encryption process can be done in parallel for every gate in the circuit which yields a constant-round protocol for secure multiparty computation. At a very high level, the idea is to construct a common garbled circuit interactively. Then, each individual party evaluates the garbled circuit, without interacting with other parties. The intermediate information that this garbled circuit does not reveal any information to the parties except the output of the circuit and is guaranteed to be correct. Recently, Damgård and Ishai [DI05] observed a problem in [BMR90] with the encryption scheme for the garbled circuit technique. Briefly, the parties need to provide zero-knowledge proofs for statements that involve the computation of a pseudorandom generator for computing encryptions on which Yao's protocol relies. In general, this requires generic zero-knowledge techniques, which means we no longer make a black-box use of a pseudorandom generator, and also this leads to very inefficient result. In [DI05], the authors propose a new protocol using a different encryption scheme as a solution for secure multiparty computation.

In this thesis, we will focus on the use of Yao's garbled circuit which will be described in detail in Chapter 5. We will present new protocols for secure two-party computation based on Yao's garbled circuit considering different security levels.

4

Combining Oblivious Transfer and Commitments

There are several ways of combining oblivious transfer with commitments. One of the strongest known combinations is *Committed Oblivious Transfer*. Known protocols for committed oblivious transfer, however, only cover the transfer of single bits (one bit per execution of the protocol). We present committed oblivious transfer protocols for transferring bit strings. Our constructions work for are any (2,2)-threshold homomorphic cryptosystem, and perform favorably in comparison to the most efficient committed oblivious transfer protocols for single bits. We also introduce *Committing Oblivious Transfer*, which is a somewhat weaker combination than committed oblivious transfer, and therefore allows for potentially more efficient protocols (see Chapter 6).

Parts of this chapter are based on [KSV07] (joint work with Berry Schoenmakers and José Villegas) and based on [KS06a] (joint work with Berry Schoenmakers).

4.1 Introduction

Oblivious transfer (OT) is a powerful primitive in modern cryptography. As we said in Section 2.4, in a 1-out-of-2 OT protocol, the sender has two messages s_0 and s_1, and the chooser has a bit b, and the chooser wishes to receive s_b, without the sender learning b, while the sender wants to ensure that the chooser receives only one of the two messages (see Figure 4.1 for 1-out-of-2 OT functionality).

Sender		**Chooser**
Private Input: s_0, s_1	**1-out-of-2 OT** $\longleftrightarrow$	**Private Input:** b
Private Output: $\perp$		**Private Output:** s_b

Figure 4.1: 1-out-of-2 oblivious transfer

OT protocols are usually used in the semi-honest model. Now suppose that we are in the malicious model and the inputs s_0, s_1 are critical messages that actually come from

some previous constructions, or the chosen output s_b will again be used in some further constructions. Therefore, we want to be sure whether the parties will use their inputs/outputs correctly. How do we ensure that both parties are using the correct inputs/outputs? In general, in order to solve these kinds problems, commitments to inputs or to outputs are used. Namely, the cases where not only s_0 and s_1 are committed before the protocol starts but also a commitment to s_b is committed at the of the protocol. In the next section, we discuss several variants of combinations of OT with commitments.

4.2 Variants of Oblivious Transfer with Commitments

Committed oblivious transfer is obtained as a natural combination of OT and bit commitments. This notion was first introduced by Crépeau [Cre90] under the name *Verifiable Oblivious Transfer*. In short, at the start of committed OT, the sender is committed to bits s_0 and s_1 and the chooser is committed to bit b; at the end of computation the chooser learns s_b and is committed to s_b, and knows nothing about s_{1-b}, while the sender learns nothing about b. Later, Crépeau, Graaf and Tapp [CGT95] presented a more efficient committed OT protocol and showed that from committed OT one can construct a protocol for general secure multi-party computation in the malicious model. Garay, MacKenzie and Yang [GMY04] present the most efficient committed OT protocol to date realizing committed OT functionality in the universal composable (UC) framework of Canetti [Can01]. However, this protocol also works only for bits. In this chapter, we will present an efficient protocol for committed OT which allows transferring arbitrary bit strings. We will only consider a stand-alone setting, noting that the efficiency of our protocols improves the efficiency of the UC-protocols of [GMY04] when reduced to a stand-alone setting (replacing, e.g., the use of Ω-protocols [GMY06] by Σ-protocols).

Sender	**Common Input:** $\text{commit}_C(b; r)$, $\text{commit}_S(s_0; r_0)$, $\text{commit}_S(s_1; r_1)$	**Chooser**
Private Input: s_0, s_1, r_0, r_1		**Private Input:** b, r
	Committed OT $\longleftrightarrow$	
Private Output: $\perp$	**Common Output:** $\text{commit}_C(s_b; u)$	**Private Output:** s_b, u

Figure 4.2: Committed oblivious transfer

We are now ready to present a formal definition of committed OT.

Definition 4.1 (Committed OT protocol) *A committed OT protocol is run between the sender S and the chooser C. S has as private input the bits or strings s_0 and s_1 and also randomly chosen strings r_0 and r_1, and C has as private input bit b and also randomly chosen string r. Both have public commitments $commit_S(s_0; r_0)$, $commit_S(s_1, r_1)$,*

and $\text{commit}_C(b, r)$ *as common input. At the end of the protocol,* C *receives* s_b *and a fresh commitment* $\text{commit}_C(s_b, u)$ *becomes publicly available.* S *learns nothing about* b *while* C *has no information about* s_{1-b}*. (See Figure 4.2)*

If the parties are committed to the inputs of the OT protocol but there is no commitment to chooser's output we refer to this variant as verifiable OT (see Figure 4.3). In this direction, Cachin and Camenisch [CC00] as well as Jarecki and Shmatikov [JS07] present protocols for verifiable OT in 2 rounds. These protocols can be converted into committed OT by requesting the chooser to recommit to its received value and to prove the validity of this commitment with respect to the commitments for the inputs. In general, this leads to one extra communication round.

Sender	**Common Input:** $\text{commit}_C(b; r)$, $\text{commit}_S(s_0; r_0)$, $\text{commit}_S(s_1; r_1)$	**Chooser**
Private Input: s_0, s_1, r_0, r_1		**Private Input:** b, r
	Verifiable OT $\longleftrightarrow$	
Private Output: $\perp$		**Private Output:** s_b

Figure 4.3: Verifiable oblivious transfer

Now we point out the difference between committed OT and verifiable OT in more detail. Committed OT and verifiable OT are identical except that in verifiable OT the commitment by the chooser to its selected value s_b is not required. Keeping this in mind, we notice that [Cre90, CGT95, CD97, GMY04] are papers that present committed OT (of bits). Instead, in [CC00, JS07] only verifiable OT protocols are presented. However, the different use of these terms causes some confusion in the literature: Crépeau [Cre90] introduces committed OT under the name of verifiable OT, Jarecki and Shmatikov [JS07] present protocols for verifiable OT, while they use the term committed OT. In the latter paper, they present a UC-secure verifiable OT protocol (which for them is a committed OT), modifying the definition of the ideal functionality for committed OT by Garay *et al.* [GMY04] to make it into verifiable OT. It is straightforward to see that in line with our definitions committed OT implies verifiable OT by just ignoring the output commitment.

As a final variation, assume now that the inputs are not committed in the beginning of the protocol. Let's now commit to the inputs and invoke committed OT functionality. In this respect, we introduce a new notion called *Committing Oblivious Transfer*. Namely, the sender and the chooser start as in a plain oblivious transfer (without any commitments for their inputs). Upon completion of a committing OT protocol, however, the sender and the chooser are committed to (some of) the actual inputs used by them in the protocol run. Possibly, the receiver is committed to its output as well. More formally,

Definition 4.2 (Committing OT protocol) *Committing OT is a protocol between two parties, the sender* S *and the chooser* C*. The private input of* S *consists of bit strings* s_0 *and* s_1 *and the private input of* C *is a bit* b*. The private output for* S *consists of bit strings*

r_0 and r_1, the private output for C consists of the bit string s_b, and the common output for S and C is $commit_C(s_b; u)$,$commit_S(s_0; r_0)$ and $commit_S(s_1; r_1)$ (see Figure 4.4).

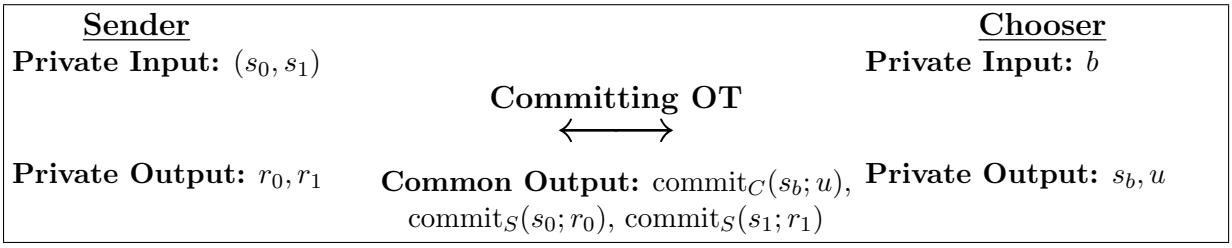

Figure 4.4: Committing oblivious transfer

An OT protocol in the literature that fits with committing OT is presented by Lipmaa in [Lip03]. That protocol is presented under the name *Verifiable Homomorphic Oblivious Transfer*. Here, homomorphic means that it is based on homomorphic encryption and commitment schemes, and verifiability is defined in the sense that at the end of the protocol there will be some commitments to all inputs of OT. Also, the OT protocol by Bellare and Micali based on ElGamal encryption [BM90] can also be seen as an instance of committing OT, as the sender outputs two encryptions for his respective inputs. Another instance of committing OT is presented by Naor and Pinkas which is the first two-round efficient OT without using random oracles [NP01]. Note that in all these examples, the chooser is not committed to its output (i.e., s_b).

In Section 4.5, for the sake of completeness, we will propose a concrete protocol for committing OT which is a derived protocol that is close to our committed OT protocol. We remark that committing OT will be used in Chapter 6 to allow more efficient protocols than using committed OT.

4.3 Basic Concepts and Definitions

Before we present our protocol for committed OT, we discuss in this section some important properties of the cryptographic tools that we are going to use.

4.3.1 (Non-Interactive) Public and Private Threshold Decryption

In a (t, n)-threshold cryptosystem, given a ciphertext, any combination of at least t parties can decrypt and produce shares of the decryption, based on their respective shares of the secret key. These shares are broadcasted and with this, everyone can simply recover the plaintext by using a reconstruction algorithm. Putting this more formally, given a ciphertext c=$\mathsf{Enc}(m)$ where Enc denotes the encryption algorithm, at least t parties broadcast m_i=$\mathsf{Dec}_{sk_i}(c)$, where Dec denotes the decryption algorithm under the secret key

share sk_i for the i-th party. Afterwards, everyone can reconstruct m as $m = R(m_1, \ldots, m_t)$ where R denotes the public reconstruction algorithm.

In order to withstand malicious adversaries, parties have to prove that the decryption share m_i is correctly computed. For this, they use a Σ-protocol for the relation

$$R_{\mathsf{tdec}} = \{(m_i, c; sk_i) : m_i = \mathsf{Dec}_{sk_i}(c)\}.$$

For the security of this process, and for later use in our security analysis, we assume that if $t-1$ parties are corrupted, then there is a simulator that on inputs e=$\mathsf{Enc}(m)$, the message m, and the $t-1$ shares of the private key for the corrupted parties, the simulator can produce a statistically indistinguishable view of the decryption protocol. The concrete details on how to do this depend on the specific threshold encryption scheme used. To give some examples, see [ST04] for the homomorphic threshold ElGamal, and [DJ01] for the threshold Paillier cryptosystem.

In our protocol, we consider a variant of the threshold decryption protocol: the so-called *private threshold decryption* [CDN01] where only one of the t parties will recover the secret. This is easily achieved: all $t-1$ other parties follow the protocol, and broadcast their shares (along with the proofs of correctness). The party who will learn the plaintext proceeds with the decryption process privately, collects all decryption shares from the $t-1$ other parties, and privately reconstructs the message. The remaining parties will not get any information about this message by the threshold construction.

4.3.2 Encryptions as Commitments

We note that a probabilistic public-key encryption scheme can be used as a non-interactive commitment scheme. One party commits to a message by encrypting it. The opening is done by disclosing the message and the randomness used. However, in this scenario we have to be careful: the holder of the private key can *always* see the contents of any commitment of this type and, depending on the encryption scheme used, this party might recover the randomness and therefore virtually open *any* commitment.[1] This compromises the hiding property for the committers that do not know the secret key. We can resolve this issue with the following two possible actions: using the encryption scheme as a commitment without allowing any of the parties to know the secret key; another suitable alternative could be to set up a threshold encryption scenario. In this way, the ability to decrypt can be distributed in a threshold fashion (possibly letting the threshold be the total number of parties).

Given a commitment e=$\mathsf{Enc}(m, r)$, its committer in this scenario is the party that knows both the message m and the randomness r. Note that each party can perform private threshold homomorphic decryption with respect to one party to retrieve the message behind e, but this does not always allow the recipient to obtain the randomness used in e

[1] For Paillier encryptions, one is able to recover both the plaintext and the randomness used if one knows the private key. Whereas, for ElGamal encryptions recovering the randomness is impossible under the DL assumption.

(e.g., ElGamal), and therefore this party will not be able to open e as a commitment. If party P is the committer of e=$\mathsf{Enc}(m, r)$ we denote it by $e = \text{commit}_P(m, r)$.

4.3.3 Security Definitions

The security analysis of our committed OT protocol is done by means of the simulation paradigm which guarantees privacy following the lines by Lipmaa in [Lip03]. Note that the definition in [Lip03] is a weaker form than the fully simulatable ideal/real simulation paradigm. Although the privacy for both parties is computational (as the commitments in our protocol are public key encryptions), we show a simulation which produces a statistically indistinguishable view of the committed OT protocol for both parties. Hence, the committed OT protocol does not divulge any information beyond what can be inferred from the encryptions (which are used as computationally hiding commitments).

The main security requirement is to show that our protocol achieves the privacy requirements for committed OT. There are protocols in the literature that achieve unconditional privacy for one of the parties (e.g., [NP01, Tze02, Lip03]) while the privacy for the other party relies on a computational assumption. As our commitments are encryptions of the underlying threshold public key cryptosystem, only computational privacy is achieved for both parties. However, our protocol achieves more than computational privacy: we show that for any corrupted party (the sender or the chooser) there exists a simulator that produces a view of the protocol which is statistically indistinguishable from the view of the corrupted party executing a real instance of the protocol. This has clear consequences in the framework of [CDN01]: a successful attacker to our protocol is an attacker to the security of underlying cryptosystem without loss in its success probability. This allows modular security proofs of higher level protocols that use our committed OT as a subroutine.

To carry out such simulations, we proceed as follows. Assuming that one party is corrupted, we build an efficient simulator that has access to the public input, private secret shares of secret key and, as done in [Lip03], the private output in the case that the chooser is corrupted. Besides, the simulator knows the public output.

4.4 A Committed Oblivious Transfer Protocol

In this section, we will present our committed OT protocol which is based on a (2,2)-threshold homomorphic cryptosystem. We assume that the cryptosystem has already been set up (see 4.4.2). This implies that the parties must run a multiparty computation to get a public key and their shares of the private key. Although this require some work from the two parties involved, it may pay off if the committed oblivious transfer protocol has to be run multiple times. Let Enc denote the encryption algorithm of this cryptosystem, and as explained above we also use Enc as a non-interactive commitment scheme.

Let e_0=$\mathsf{Enc}(s_0, r_0)$ and e_1=$\mathsf{Enc}(s_1, r_1)$ be the commitments to the sender's input strings s_0 and s_1, and e=$\mathsf{Enc}(b, r)$ be the commitment to the chooser's selection bit b.

We note that if a message m_1 is private to one of the parties, this party can compute

e' as $e' = (e_2)^{m_1} \cdot \mathsf{Enc}(0, r')$ where e_2 is an encryption of a message m_2 and r' is some randomness, and is able to generate a proof that e' is correctly computed. This is called a *private-multiplier gate* (see, e.g., [ST04]). In this case, e clearly encrypts $m_1 m_2$ because of homomorphic properties.

For later use, the relation for the proof given in the private-multiplier gate is denoted by $R_{\mathsf{pm}} = \{(e_1, e_2, e'; m_1, r_1, r') : e_1 = \mathsf{Enc}(m_1, r_1) \wedge e' = (e_2)^{m_1} \cdot \mathsf{Enc}(0, r')\}$. Also, for later use in the simulation of our protocols, given $\mathsf{Enc}(m_1)$, $\mathsf{Enc}(m_2)$ and $\mathsf{Enc}(m_1 m_2)$, the private-multiplier gate can be statistically simulated when there are at most $t-1$ corrupted parties in a (t, n)-threshold homomorphic cryptosystem. For details, see [CDN01, DJ01, ST04, DN03].

Using the general approach to secure multiparty computation of [CDN01], the committed OT protocol corresponds to the secure evaluation of an arithmetic circuit given by $t = b(s_1 - s_0) + s_0$ which clearly returns s_0 if $b = 0$ and s_1 when $b = 1$. This approach is so general that even s_0, s_1 and b need not be known to any party. Note that the output of the evaluation will be an encryption $e' = \mathsf{Enc}(t) = \mathsf{Enc}(s_b)$. If inputs (s_0, s_1) and/or b are known to the respective parties then one can securely compute e' using a private-multiplier gate (instead of a secure multiplication gate), resulting in a more efficient protocol.

Once e' is obtained, according to one of the committed OT requirements, only the chooser must be able to recover the plaintext. For this, we use *private decryption*, where the chooser is the one who will learn the plaintext inside e'.

To complete the committed OT protocol, the chooser needs to commit to the received value s_b, and prove that she does so correctly. In principle, this can be done using proofs of knowledge. However, we will use the fact that our commitments are encryptions for a threshold cryptosystem: to prove that a fresh commitment e'' to output s_b is correct, we observe that this proof is equivalent to showing that e''/e' is an encryption of 0. The latter statement is proved by actually decrypting e''/e'.

As a final remark, we see that if the chooser starts the protocol by producing e', it turns out that she has to wait for the decryption share of e' from the sender, so that she later can produce the fresh commitment as just explained. This results in at least 3 rounds of communication. However, if the sender starts, he produces e' and, at the same time the decryption share for e', which reduces the overall strategy to at least 2 rounds of communication. In both cases, the computational cost is actually the same. For this reason, we only go into the details of this second approach, as it results in a committed OT with one round less.

We are now ready to present our protocol for committed OT in full detail. This protocol has two rounds and it is quite efficient compared to the state of the art. In the beginning of the protocol, we take advantage of the fact that the values for the commitments e_0, e_1 and e are known to the respective parties. The protocol is as follows (see also in Figure 4.5).

Step 1. The sender produces $e' = \mathsf{Enc}(b(s_1 - s_0) + s_0) = e^{(s_1 - s_0)} \cdot e_0 \cdot \mathsf{Enc}(0, r')$ and a Σ-proof (see Section 2.2) for relation R_{pm} on $(e, e_1/e_0, e'/e_0; s_1 - s_0, r_1 - r_0, r')$. The sender also produces its decryption share s_S of e', along with a Σ-proof for relation R_{tdec} on $(e', s_S; sk_S)$.

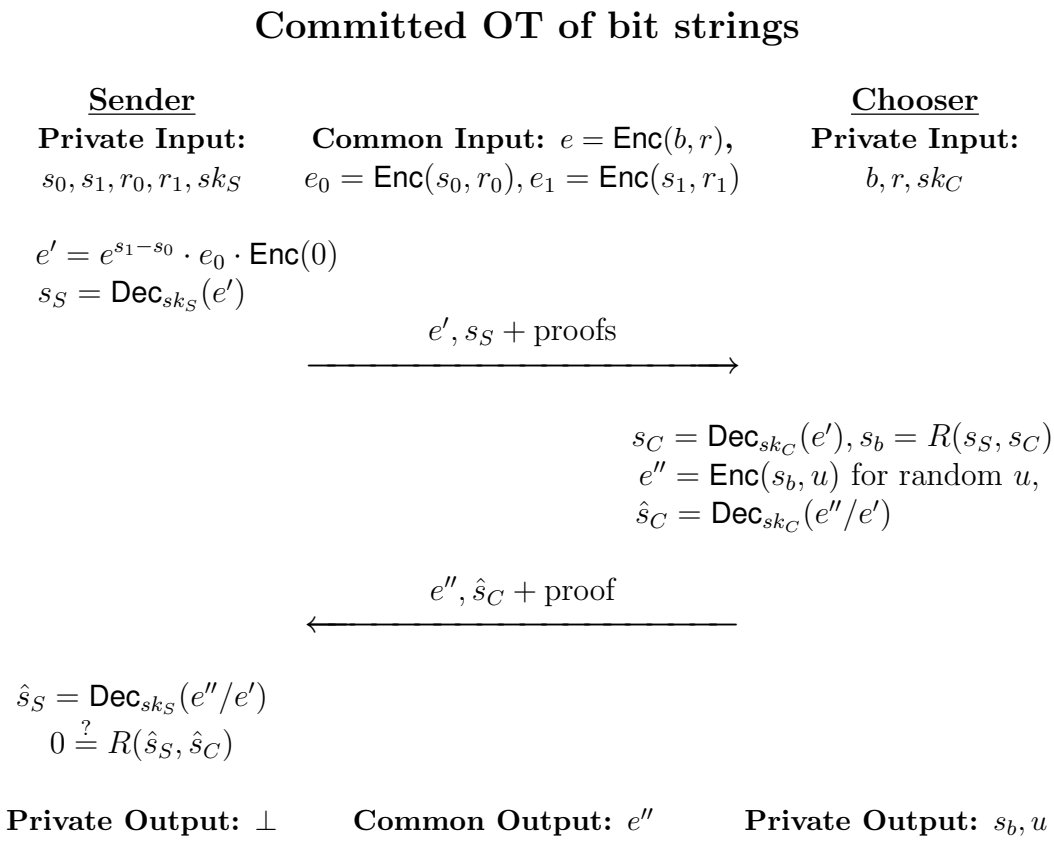

Figure 4.5: The committed OT protocol

Step 2. After checking the two proofs given by the sender, the chooser produces the corresponding decryption share for e', denoted here by s_C. Combining s_S and s_C, the chooser gets s_b. Immediately, the chooser produces a fresh encryption $e'' = \mathsf{Enc}(s_b, u)$ for a fresh random u, and generates her decryption share for e''/e', denoted by $\hat{s}_C$. Then, e'' and $\hat{s}_C$ are sent along with Σ-proofs for R_{enc} and R_{tdec} on inputs $(e''; s_b, u)$ and $(e''/e', \hat{s}_C; sk_C)$ respectively.

Step 3. Finally, upon receiving e'', the sender produces its decryption share for e''/e', denoted by $\hat{s}_S$. This is combined with $\hat{s}_C$ to check whether the resulting decrypted value is 0. If so, the sender accepts e'' as a valid commitment for the chooser's output. Otherwise, the sender rejects.

The value s_b denotes the output of the chooser after privately decrypting e'. When this value has been computed, a fresh commitment to s_b (denoted as e'') by the chooser has to be sent in order to fulfill the committed OT requirement that the chooser's output must be committed. Notice here that without the fresh commitment to s_b the protocol fulfills the verifiable OT requirement in one round only.

4.4.1 Security Analysis

For the security analysis, we are going to prove that this protocol fulfills the privacy requirements for committed OT. We are going to show that given a corrupted party, there exists a simulator that can produce a view which is statistically indistinguishable from the view of that party interacting with the other honest party.

As we mentioned earlier, before the simulation is run the simulator already knows the shares of the secret key of the corrupted party. The reason is that the threshold cryptosystem is set up before the protocol starts, and therefore we assume that the simulator extracts this information when the distributed key generation protocol is run (see Section 2.5).

Also, in case the chooser is corrupted, we use the approach in [Lip03]: the simulator will be given access to the value received by the chooser. From this and available public information, we construct a simulator that produces an indistinguishable view for the adversary with respect to the view in the real execution.

Finally, we recall that the protocol gives computational privacy to both parties, the sender and the chooser, because of the semantic security of the underlying cryptosystem. But, as we said before, in the next theorem we are going further than computational privacy, namely we show that the protocol is simulatable for both parties and those simulations produce views which are statistically indistinguishable from the views in the real protocol execution.

Theorem 4.3 *On the sender's inputs s_0, s_1 (and randomness r_0 and r_1) and the chooser's private selection bit b (and randomness r), where public commitments to the parties' inputs $e_0 = \mathsf{Enc}(s_0, r_0)$, $e_1 = \mathsf{Enc}(s_1, r_1)$, and $e = \mathsf{Enc}(b, r)$ are available, the committed OT protocol privately gives s_b (and a fresh randomness u) to the chooser, along with a public commitment $e'' = \mathsf{Enc}(s_b, u)$.*

Proof. We will separately consider the case that the chooser is corrupted, and the case that the sender is corrupted. Based on public information and the corrupted party's private decryption share, we show a simulation which produces a view to the adversary that is statistically indistinguishable from the view in the real protocol execution.

In both cases, a set of valid public inputs is available: e is a commitment to the chooser's selection bit, and e_0, e_1 are respective commitments to the sender's inputs. Also, the simulator is assumed to get the public output commitment e'' which is a "valid" commitment to the value received by the chooser.

Case 1- The chooser is corrupted.

We first analyze the security for the case that the chooser is corrupted. The simulator has the chooser's private key share sk_C, and received value s_b, apart from the public commitments. From this information, the simulator constructs a view for the chooser which is statistically close to the one when interacting with the honest sender, as follows:

1. The simulator computes $e' = e'' \cdot \mathsf{Enc}(0)$, and generates a simulation of the private-multiplier gate (over multiplicands e and e_1/e_0 and result e'/e_0).

2. At the same time, the decryption share s_S can be simulated given e', the plaintext of e' (which is s_b) and the share of private key sk_C of the chooser. All proofs at this stage are also simulated.

This completes the simulation for the malicious chooser. The simulated transcript is consistent with the view of the chooser and statistically indistinguishable when interacting with the honest sender.

Case 2- The sender is corrupted.

We next analyze the security for the case that the sender is corrupted. The simulator has only sender's private key share sk_S and all public information as described above. From this information, the simulator constructs a view for the sender which is statistically close to the one when interacting with the honest chooser, as follows:

1. The simulator waits until the sender produces the encryption e' and the decryption share for e'. The simulator checks all the proofs as if the honest chooser would check in the real protocol execution. If all proofs are passed, the simulator goes on, otherwise it aborts.

2. Now, simulator prepares e'' as $e' \cdot \mathsf{Enc}(0)$ and outputs it along with a simulated proof of knowledge. Also, it simulates $\hat{s}_C$ calling the simulator to the decryption process on inputs e''/e', plaintext 0 and the sender's secret key share sk_S.

This completes the simulation for the malicious sender. The transcript is consistent and statistically indistinguishable from the sender's view when interacting with the honest chooser. □

4.4.2 Complexity Analysis and Comparison

Our protocol involves only a constant number of computational, communication and round complexities. We remark that our protocol, unlike previous protocols, allows transferring arbitrary bit strings. When studied in similar frameworks where only bits are transferred, our protocols are as efficient as the committed OT protocol by Garay *et al.* [GMY04] which is the most efficient up to now. We note that the protocol of Garay *et al.* works in a stronger model, namely in the universal composability (UC) framework in the common reference string (CRS) model. We will adapt their protocol to our framework to be able to carry out a comparison.

In the following, we present the precise description for the complexity of our protocol. For a concrete result we use the (2,2)-threshold ElGamal cryptosystem by considering offline computations. The protocol by Garay *et al.* needs a special type of zero-knowledge

protocols for the proofs of knowledge, namely, Ω-protocols [GMY06], which are variants of the Σ-protocols. For simplicity, we reduce them to the simpler Σ-protocols. This is done to be able to make a reasonable comparison.

Committed OT protocol by Garay et al.

We first give a global description of their protocol. The CRS consists of the pair (g, h), where nobody knows the discrete log x of h to the base g, i.e. $h = g^x$. The protocol uses Pedersen commitments. Therefore, let $E_0 = g^{r_0}h^{s_0}$ and $E_1 = g^{r_1}h^{s_1}$ denote the commitments to sender's inputs s_0 and s_1. Let also $E = g^r h^b$ denote the commitment to chooser's input b. The protocol has the following two main steps:

1. The sender "re-encrypts" E_0 and E_1 under the 'keys' E and E/h respectively. Let E'_0 and E'_1 denote the resulting encryptions. Note that E_b will be re-encrypted with the key g^r. The sender also proves that this is done correctly.

2. The chooser can "decrypt" the message in E'_b as she knows the secret exponent r, recovering s_b. On the other hand, the chooser cannot decrypt E'_{1-b} unless the discrete-log of h to the base g is known. To finish, the chooser has to recommit to the received value s_b and prove that this is correctly computed.

See [GMY04] for more details. In the first step, for the reencryption, 4 exponentiations are computed by the sender (2 of them can be done off-line). The proofs at that step cost 16 exponentiations (8 of them can be done off-line). As for the second step, the chooser needs only 1 on-line exponentiation to retrieve the chosen value. To finish, the chooser computes a fresh commitment which costs 1 off-line exponentiation. The proof of knowledge at the end costs 8 exponentiations (4 can be off-line). In total, there are 15 on-line and 15 off-line exponentiations.

Verifiable OT by Jarecki and Shmatikov.

We now sketch the verifiable OT protocol in [JS07]. The input commitments are encryptions under a homomorphic public key cryptosystem (the public key is part of the CRS). The chooser first sends a new public key together with the encryption of its selection bit under this new cryptosystem, proving that this is done correctly. Later, the sender encrypts its inputs under this new public key, combining them with the encryption for the selection bit. Finally, the chooser can decrypt both ciphertexts, but only one of them contains the selected value, and the other one is random.

To convert it into committed OT, the chooser must recommit to its received value, producing a proof that the value encrypted is consistent with previous commitments. This protocol results in 3 rounds.

This scheme virtually works for any homomorphic encryption. When instantiated to additively homomorphic ElGamal, for the sake of our comparison, the protocol is slightly less efficient than that of [GMY04]: around 17 on-line and 16 off-line exponentiations

	Online	**Offline**
Private-multiplier gate	3	5
Private threshold decryption	4	2
Recommitment	1	3

Table 4.1: Number of exponentiations of building blocks used in our committed OT protocol for (2,2)-threshold ElGamal setting

(mainly due to the generation of the new cryptosystem, the recommitment of the selection bit and the respective proofs of knowledge). Meanwhile, for the verifiable OT protocol, the cost is 13 on-line and 10 off-line exponentiations.

Our committed OT protocol.

Now we present the computational cost for our protocol in the case of (2,2)-threshold ElGamal. In Table 4.1 we summarize the computational complexity of the building blocks used in our protocol. For the private-multiplier gate, we include the costs of producing the output and the Σ-proof for relation R_{pm}. In the case of the private threshold decryption, we include the costs for generating the decryption shares and for the Σ-proof for R_{tdec}. We conclude by considering the recommitment at the last step. In the case of e'', the chooser has to encrypt the received value and the Σ-proof for the knowledge of the randomness used in that encryption. This suffices and if the chooser passes this proof and e'/e'' decrypts to 0, it implies that she knows the plaintext in e''. We divide the complexities analysis again into on-line and off-line computations.

To get the total number of exponentiations, we note that our protocol requires one private-multiplier gate at the first step (to produce e'), two private threshold decryptions (for decrypting e' and e''/e') and one encryption at the last step (to generate e''). Therefore, we have in total 12 on-line and 12 off-line exponentiations.

Observe that the way of proving that the fresh commitment is correct in our protocol is different from, yet equally efficient as in the proof in [GMY04]. The protocol in [GMY04] needs 9 exponentiations to recommit and prove. Ours needs 9 exponentiations as well: produces e'' and one threshold decryption.

If we restrict ourselves to a verifiable OT protocol, removing the recommitment step, we can see that our protocol is really much more efficient than the current the state-of-the-art protocols. It requires 7 on-line and 7 off-line exponentiations (against 11 and 10 resp. for Garay *et al.*'s protocol) and also involves only one round of interaction. Ours easily generalizes to any (2,2)-threshold homomorphic cryptosystem at the cost of a distributed key generation protocol at the beginning.

4.5 A Committing Oblivious Transfer Protocol

In this section, we will present a two-round protocol for committing OT which is a derived version of the protocol presented in Section 4.4. Let $e_0 = \mathsf{Enc}(s_0, r_0)$, $e_1 = \mathsf{Enc}(s_1, r_1)$ and $e = \mathsf{Enc}(b, r)$ be as above.

As a remark, if the sender starts the protocol by producing e_0 and e_1 then the chooser blinds one of e_0 and e_1 and sends to the sender so that the sender can produce a share for the blinded commitment. The chooser also compute its share and later can obtain s_b. The common output would be e_0 and e_1. This results in three rounds of communication. However, if the chooser starts, he produces e, which reduces the overall strategy to two-rounds of communication. We only go into the details of the two-round protocol.

We are now ready to present our protocol for committing OT in full detail. The protocol is as follows (see also in Figure 4.6).

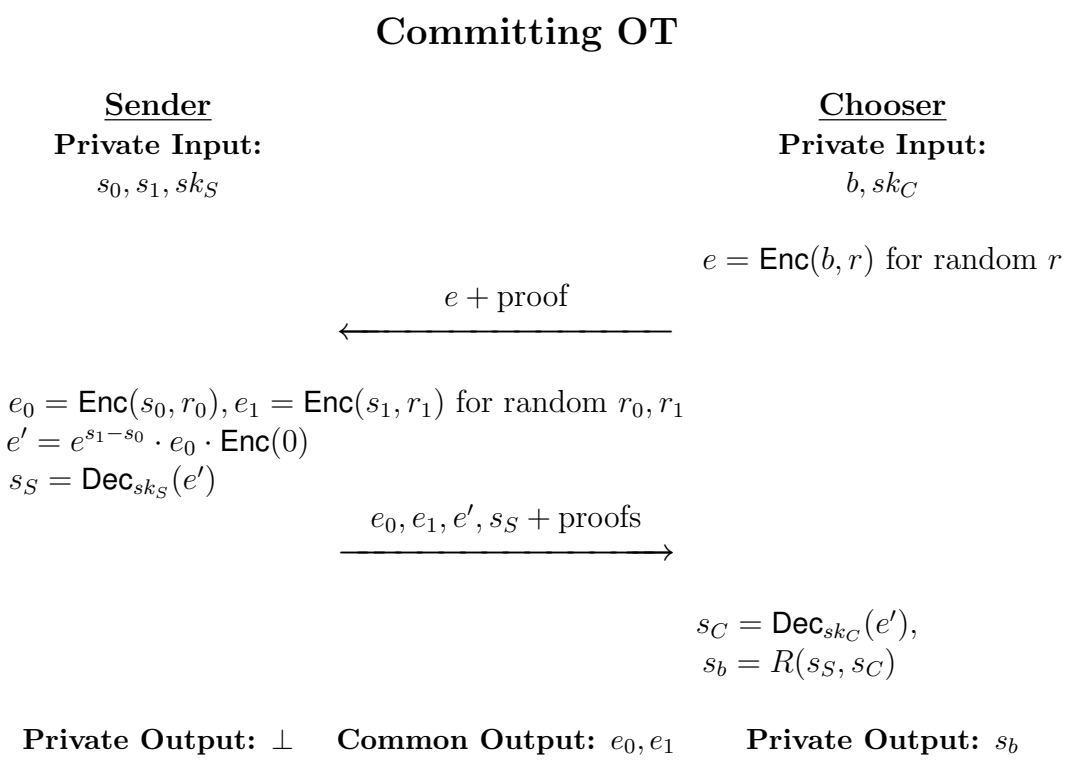

Figure 4.6: A committing OT protocol

Step 1. The chooser produces $e = \mathsf{Enc}(b, r)$ for a random r and a Σ-proof (see Section 2.2) for relation $R = \{(e; b, r) : e = \mathsf{Enc}(m, r)\}$.

Step 2. After checking the proof given by the chooser the sender generates $e_0 = \mathsf{Enc}(s_0, r_0)$ and $e_1 = \mathsf{Enc}(s_1, r_1)$ for random r_0, r_1, produces a fresh encryption $e' = e^{(s_1 - s_0)} \cdot e_0 \cdot \mathsf{Enc}(0)$, and generates the corresponding decryption share for e'. Then, e_0, e_1, e' and

s_S are sent along with Σ-proofs for R_{enc} and R_{tdec} on inputs $(e_0, e_1, e'; s_0, s_1)$ and $(e', s_S; sk_S)$ respectively.

Step 3. Finally, upon receiving e_0, e_1, e' and s_S, the chooser also produces its decryption share for e', denoted by s_C. Combining s_S and s_C, the chooser gets s_b. The common output is e_0 and e_1.

Note that as a further computation if the chooser also committed to s_b at Step 3 then our committing OT protocol would be its most general case as defined in Section 4.2. However, it is not necessary to have the final commitment step in order to fix the protocol issue which will be presented in Chapter 6.

4.5.1 Security Analysis

The security analysis of the committing OT protocol is similar to the one for the committed OT protocol. We note that we prove the security in the ideal/real simulation paradigm. As before the protocol is run there is a setup phase (to get public keys, and shares of the secret key), we will work in the hybrid model assuming that this setup phase in securely computed. Therefore, we will assume that the simulator has access to the corrupted party's share of the private key. With this, we show a simulation which produces a view that is computationally indistinguishable to the view of a real life execution.

In the case that the chooser is corrupted, the simulator extracts the bit b from the given proof in the first round. Once the simulator learns b it sends to the ideal functionality that implements committing OT, and receives s_b back. From the information chooser's private key share sk_C and received value s_b, the simulator constructs a view for the chooser which is indistinguishable to the one when interacting with the honest sender, as follows: the simulator chooses a random s_{1-b} and computes $e_b = \mathsf{Enc}(s_b, r_b)$ and $e' = \mathsf{Enc}(s_b, r_b^*)$, and $e_{1-b} = \mathsf{Enc}(s_{1-b}, r_{1-b})$. Then the simulator generates a real proof for the private-multiplier gate. At the same time, the decryption share s_S can be simulated given e', the plaintext of e' (which is s_b) and the share of private key sk_C of the chooser. All proofs at this stage are also simulated which completes the simulation for the corrupted chooser. The simulated transcript is consistent with the view of the chooser and computationally indistinguishable when interacting with the honest sender.

In the case that the sender is corrupted, the simulator has the sender's private key share sk_S. From this information, the simulator constructs a view for the chooser which is computationally close to the one when interacting with the honest sender, as follows: the simulator chooses an arbitrary bit b and computes $e = E(b, r)$ for random r, and generates a real proof of knowledge of b. The simulated view is computationally indistinguishable when interacting with the honest sender. Note that the views are computationally indistinguishable since the contents inside e cannot be known by the simulator as it happens in the committed OT protocol above.

5

Secure Two-Party Computation in the Semi-Honest Model

Yao's protocol for secure two-party computation based on garbled circuits was already introduced at the end of Chapter 3. In this chapter we present and analyze protocols for secure two-party protocol assuming semi-honest adversaries, following the same line as Yao's original protocol. The security analysis is performed according to the ideal/real simulation paradigm. Furthermore, our protocol can be extended to the malicious case in a modular fashion, as will be shown in Chapter 6.

Parts of this chapter are based on [KS08] (joint work with Berry Schoenmakers).

5.1 Yao's Garbled Circuit

In this section, we describe Yao's garbled circuit in full detail. Informally speaking, a garbled circuit is an encrypted version of a Boolean circuit in which the Boolean values are replaced by encrypted values. In this section we will describe Yao's garbled circuit construction, opening and evaluation steps in detail (see also, e.g., [Yao86, GMW87, Rog91]).

5.1.1 Preparation of a Garbled Circuit

Bob generates a circuit C_f that computes a function f. A circuit consists of gates and wires. A wire is either an internal wire, an input wire or an output wire. Each internal wire connects two gates.

As we mentioned earlier, Bob is the constructor who generates the garbled circuit and Alice is the evaluator who evaluates it. Bob garbles the circuit in two phases. Let k be the security parameter.

- **Phase 1:** For each wire in C_f, say wire W_i, Bob generates two random k-bit strings $v_{i,0}$, $v_{i,1} \in_R \{0,1\}^k$, and a random bit $p_i \in_R \{0,1\}$, and computes $w_{i,0}$ and $w_{i,1}$ as follows:
  $w_{i,0} = v_{i,0} \parallel (0 \oplus p_i)$, $w_{i,1} = v_{i,1} \parallel (1 \oplus p_i)$. In our setting, the values $w_{i,0}$ and $w_{i,1}$ will be called the *garbled values* for the i-th wire. The garbled value $w_{i,0}$ will correspond to the bit 0, and $w_{i,1}$ will correspond to the bit 1.

- **Phase 2:**

  Let $b_{i''} = g_{i''}(b_i, b_{i'}) \in \{0,1\}$ where b_i and $b_{i'} \in \{0,1\}$. Let a 4-tuple of bits $< g_{i''}(0,0), g_{i''}(0,1), g_{i''}(1,0), g_{i''}(1,1) >$ represent the logical table of a gate $g_{i''}$, with input wires W_i, $W_{i'}$ and output wire $W_{i''}$ (see Figure 5.1).

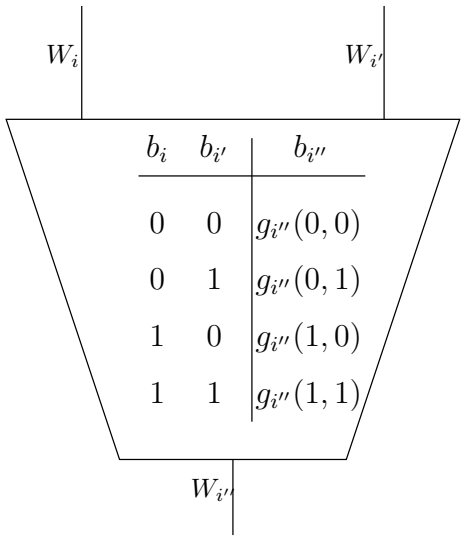

Figure 5.1: Truth table of a gate $g_{i''}$

Then for each gate $g_{i''}$, Bob performs the following transformations on this logical table.

  - **Garbling Phase:**
    Bob replaces $g_{i''}(b_i, b_{i'})$ with $w_{i',b_{i''}}$ for $b_i, b_{i'}, b_{i''} \in \{0,1\}$. The result is the garbled-4-tuple $\langle w_{i'',g_{i''}(0,0)}, w_{i'',g_{i''}(0,1)}, w_{i'',g_{i''}(1,0)}, w_{i'',g_{i''}(1,1)} \rangle$.
  - **Encryption Phase:**
    Bob replaces $w_{i'',g_{i''}(b_i,b_{i'})}$ of the garbled-4-tuple by its encryption $\mathsf{Enc}_{w_{i,b_i},w_{i',b_{i'}},i''}(w_{i'',g_{i''}(b_i,b_{i'})})$ using the encryption key $(w_{i,b_i}, w_{i',b_{i'}}, i'')$ where Enc is a private-key encryption function. The result is the encrypted-garbled-4-tuple $\langle \mathsf{Enc}_{w_{i,0},w_{i',0},i''}(w_{i',g_{i''}(0,0)}),\ \mathsf{Enc}_{w_{i,0},w_{i',1},i''}(w_{i'',g_{i''}(0,1)}),$ $\mathsf{Enc}_{w_{i,1},w_{i',0},i''}(w_{i'',g_{i''}(1,0)}),\ \mathsf{Enc}_{w_{i,1},w_{i',1},i''}(w_{i'',g_{i''}(1,1)}) \rangle$. For notational simplicity, let $\langle E_{i,0,0}, E_{i,0,1}, E_{i,1,0}, E_{i,1,1} \rangle$ represent the encrypted-garbled-4-tuple for a gate g_i.
  - **Permutation Phase:**
    Let $\pi_{0,0} = id$ denote the identity permutation of $\{1,2,3,4\}$. Let us define the permutation functions $\pi_{0,1}$, $\pi_{1,0}$ and $\pi_{1,1}$ as follows:

$$\pi_{0,1} = (1,2)(3,4)\,, \quad \pi_{1,0} = (1,3)(2,4)\,, \quad \pi_{1,1} = \pi_{0,1} \circ \pi_{1,0} = (1,4)(2,3)\,.$$

Bob applies permutation $\pi_{p_i,p_{i'}}$ to the components of encrypted-garbled-4-tuple, resulting in the permuted-encrypted-garbled-4-tuple

$$\langle E_{i'',0\oplus p_i,0\oplus p_{i'}}, E_{i'',0\oplus p_i,1\oplus p_{i'}}, E_{i'',1\oplus p_i,0\oplus p_{i'}}, E_{i'',1\oplus p_i,1\oplus p_{i'}}\rangle.$$

The garbled circuit GC_f for the function f then consists of:

- The *permuted-encrypted-garbled-4-tuples* (PEG-4-Tuple) for all gates,
- The *ordered pairs* $OP_{i''} = (w_{i'',0}, w_{i'',1})$ for each output wires $W_{i''}$.

This garbled circuit GC_f is sent to Alice.

5.1.2 Opening of a Garbled Circuit

The opening of a garbled circuit as follows: For each wire W_i in the circuit, Bob sends Alice the garbled strings $v_{i,0}$, $v_{i,1}$ and the bits p_i. For each gate $g_{i''}$ Alice generates a PEG-4-Tuple using the generated garbled strings. She also prepares the ordered pairs for her output wires. Then she verifies that all PEG-4-Tuples and ordered pairs are the same as the values Bob already sent to her.

5.1.3 Evaluation of a Garbled Circuit

In this section, we describe how Alice is going to evaluate a garbled circuit. To be able to evaluate the garbled circuit the garbled input values for all input gates must be known. Here, Bob with private input y simply sends his garbled values for each of his input wires to Alice. Alice, on the other hand, receives her garbled input values via oblivious transfer (OT) [Rab81, GM86]. To be more precise, Alice and Bob run a single OT for every bit of Alice's input bit to make them garbled. Bob is the sender and Alice is the chooser. In the beginning, Alice has input bits x_i where $x = x_1 \dots x_\ell$. and Bob has garbled string values. For each of Alice's i-th input wires the protocol runs as follows:

1. Bob knows two garbled values $w_{i,0}$, $w_{i,1}$ where $w_{i,0}$ is the garbled value corresponding to the input bit 0 and $w_{i,1}$ is the garbled value corresponding to the input bit 1. Alice has input bit x_i.
2. The protocol runs in such a way that Alice chooses w_{i,x_i} and she does not learn anything about $w_{i,1-x_i}$ and Bob learns no information about x_i.

Knowing the garbled inputs and with the help of a PEG-4-Tuple for every gate it is possible to evaluate the circuit gate by gate and to produce a garbled output for each output wire without learning any information about the input, including the intermediate values. The circuit is evaluated on a gate level, namely from the input wires to the output wires. More precisely, for each gate $g_{i''}$ she does the following:

1. Let the right-most bits of w_{i,b_i} and $w_{i',b_{i'}}$ be $b_i \oplus p_i$ and $b_{i'} \oplus p_{i'}$, respectively. Alice uses $b_i \oplus p_i$ and $b_{i'} \oplus p_{i'}$ to find out the right position in the PEG-4-Tuple. In this way, Alice determines the correct and unique intended decryption corresponding to the output b_i and $b_{i'}$.
2. The decryption key is computed by the knowledge of w_{i,b_i}, $w_{i',b_{i'}}$ and i''.
3. Alice computes $w_{i'',b_{i''}} = \mathsf{Dec}_{w_{i,b_i},w_{i',b_{i'}},i''}(\mathsf{Enc}_{w_{i,b_i},w_{i',b_{i'}},i''}(w_{i'',g_{i''}(b_i,b_{i'})}))$ where $b_{i''} = g_{i''}(b_i, b_{i'})$ and finds out the garbled output value of the gate $g_{i''}$.

After evaluation of all gates, Alice computes the garbled values of all the output wires. She uses ordered pairs to find out her actual output bits of the output wires. She also has Bob's garbled output values and sends them back to Bob. Finally, Bob computes his actual output bits which he can do because he knows how he had made them garbled.

Note that Yao's garbled circuit uses a private-key encryption scheme that has indistinguishable encryptions for multiple messages, i.e. that are secure for multiple messages. This means that, for every (possibly known) messages m_0 and m_1 if an encryption of a message m_0 and an encryption of a message m_1 are given then no polynomial-time adversary can distinguish the encryption of m_0 from the encryption of m_1. For example, in [LP04] they present a simple construction based on using a *pseudorandom generator* [Nao91].

5.2 An Example: AND Gate

For a better understanding, we shall demonstrate the construction, opening and evaluation of a garbled circuit for an AND gate. We shall use the one-time pad as example for the private-key encryption. The secret key is a uniformly chosen sequence of t bits, and an t-bit ciphertext is produced by XORing the plaintext with the key. The plaintext is computed from the ciphertext in the same way.

In principle, given a hash function it is possible to construct a one-time pad, and XOR the one-time pad with the message. Let pk be the private key and m be the message to be encrypted. The encryption is done by first computing $\mathsf{Hash}(pk)$ using SHA-1 and XORing the outcome with the message m where we assume that the length of $\mathsf{Hash}(pk)$ and m are equal (so the outcome is $\mathsf{Hash}(pk) \oplus m$). The decryption can be performed only with knowledge of the key pk. In our setting, we use symmetric encryption scheme that first hashes the garbled inputs, and then XORes the computed hash values with the garbled output values.

Assume Bob is the constructor who generates a garbled circuit that only consists of one AND gate with two input wires W_i, $W_{i'}$ and one output wire $W_{i''}$ where i, i' and i'' denotes the indices of the wires (see Table 5.1).

Garbling an AND gate.

Bob generates two random values for each of wires. Let $v_{i,0}, v_{i,1}$ denote the garbled values for wire W_i, $v_{i',0}, v_{i',1}$ for wire $W_{i'}$, and $v_{i'',0}, v_{i'',1}$ for wire $W_{i''}$ where $v_{i,0}, v_{i',0}, v_{i'',0}$

W_i	$W_{i'}$	$W_{i''}$
0	0	0
0	1	0
1	0	0
1	1	1

Table 5.1: Logical table representing an AND gate

corresponds to 0 and $v_{i,1}, v_{i',1}, v_{i'',1}$ corresponds to 1, respectively. For each wire, Bob also generates random bits, say p_i, $p_{i'}$ and $p_{i''}$ for the wires W_i, $W_{i'}$ and $W_{i''}$ respectively. Suppose for instance that $p_i = 0$, $p_{i'} = 1$ and $p_{i''} = 1$. Then he puts

$$\begin{aligned} w_{i,0} &= v_{i,0}||0, & w_{i,1} &= v_{i,1}||1, \\ w_{i',0} &= v_{i',0}||1, & w_{i',1} &= v_{i',1}||0, \\ w_{i'',0} &= v_{i'',0}||1, & w_{i'',1} &= v_{i'',1}||0. \end{aligned}$$

Bob replaces all four entries of the logical table of the AND gate by the corresponding garbled values input (see Table 5.2).

x	y	z
$w_{i,0}$	$w_{i',0}$	$w_{i'',0}$
$w_{i,0}$	$w_{i',1}$	$w_{i'',0}$
$w_{i,1}$	$w_{i',0}$	$w_{i'',0}$
$w_{i,1}$	$w_{i',1}$	$w_{i'',1}$

Table 5.2: Garbled AND gate

Next, Bob encrypts the garbled values for the output wire $W_{i''}$ as follows. The encryption is done by hashing $v_{i,x}||i''||(x \oplus p_i)||(y \oplus p_{i'})$ and $v_{i',y}||i''||(x \oplus p_i)||(y \oplus p_{i'})$ using SHA-1 for each entry $x, y \in \{0, 1\}$, and XORing them with the corresponding garbled output values $w_{i'',0}$ and $w_{i'',1}$, and the encrypted-garbled-4-tuple $(E_{00}, E_{01}, E_{10}, E_{11})$ is computed is as follows (see Table 5.3).

$$\begin{aligned} E_{00} &= \mathsf{Hash}(v_{i,0}||i''||0||1||v_{i',0}||i''||0||1) \oplus w_{i'',0} \\ E_{01} &= \mathsf{Hash}(v_{i,0}||i''||0||0||v_{i',1}||i''||0||0) \oplus w_{i'',0} \\ E_{10} &= \mathsf{Hash}(v_{i,1}||i''||1||1||v_{i',0}||i''||1||1) \oplus w_{i'',0} \\ E_{11} &= \mathsf{Hash}(v_{i,1}||i''||1||0||v_{i',1}||i''||1||0) \oplus w_{i'',1}. \end{aligned}$$

W_i	$W_{i'}$	$W_{i''}$
$w_{i,0}$	$w_{i',0}$	$\mathsf{Hash}(v_{i,0}\|\|i''\|\|0\|\|1\|\|v_{i',0}\|\|i''\|\|0\|\|1) \oplus w_{i'',0}$
$w_{i,0}$	$w_{i',1}$	$\mathsf{Hash}(v_{i,0}\|\|i''\|\|0\|\|0\|\|v_{i',1}\|\|i''\|\|0\|\|0) \oplus w_{i'',0}$
$w_{i,1}$	$w_{i',0}$	$\mathsf{Hash}(v_{i,1}\|\|i''\|\|1\|\|1\|\|v_{i',0}\|\|i''\|\|1\|\|1) \oplus w_{i'',0}$
$w_{i,1}$	$w_{i',1}$	$\mathsf{Hash}(v_{i,1}\|\|i''\|\|1\|\|0\|\|v_{i',0}\|\|i''\|\|1\|\|0) \oplus w_{i'',1}$

Table 5.3: Encrypted garbled AND gate

Next, Bob permutes the order of the encrypted-garbled-4-tuple so that Alice cannot learn the order of the logical table. Since

$$\pi_{0,1} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix},$$

the PEG-4-Tuple is $\langle E_{01}, E_{00}, E_{11}, E_{10}\rangle$. Then the resulting garbled circuit for the AND gate is as follows:

$$GC_{\text{AND}} = \{\langle E_{01}, E_{00}, E_{11}, E_{10}\rangle, OP_{i''}\} \text{ where } OP_{i''} = (w_{i',0}, w_{i',1}).$$

Evaluating an AND gate.

Bob sends GC_{AND} to Alice for evaluation. Suppose that Bob has input 1 and Alice has input 0. Therefore, he sends his garbled input $w_{i',1}$ to her. For reasons of simplicity, at the end of the computation we consider the case that only Alice is going to learn the desired output.

Alice and Bob run OT so that Alice will get the garbled input value corresponding to her input bit. Namely, Alice has input bit 0 and Bob has two garbled values $w_{i,0}$ and $w_{i,1}$. At the end of OT Alice learns only $w_{i,0}$ and obtains no information about $w_{i,1}$. Bob does not know which value is chosen by Alice.

Now, Alice has $w_{i,0}$ and $w_{i',1}$ which are of the form $w_{i,0} = v_{i,1}||0$ and $w_{i',1} = v_{i',1}||0$. Since she sees the last bit of $w_{i,0}$ and $w_{i',1}$, namely 0 and 0 respectively, she decrypts the first value of $\langle E_{01}, E_{00}, E_{11}, E_{10}\rangle$, namely E_{01}. That is, Alice computes

$\mathsf{Hash}(v_{i,0}||i''||1||1||v_{i',0}||i''||1||1) \oplus E_{01}$ which results in $w_{i'',0}$.

She matches with the ordered pair $OP_{i''}= (w_{i'',0}, w_{i'',1})$ and learns her output as 0. Notice that she cannot conclude whether Bob's input is 0 or 1 when her input is 0.

5.3 A Two-Party Protocol in the Semi-Honest Model

In this section, we present a two-party protocol in the semi-honest model which is a slight modification of Yao's original protocol. We slightly modify Yao's protocol in order to analyze its security following the real/ideal simulation paradigm.

We note that the security analysis we present is in the ideal/real simulation paradigm. However, in the case of semi-honest adversaries, the ideal/real simulation definition is equivalent to the (simpler) definition used in [LP04]. In fact, we give such a proof for only consistency and completeness of the thesis.

We also highlight that this modified protocol will allow private output for both parties, just like its extended version for the malicious model in Chapter 6 and in Chapter 7. We note that this modified protocol will be easily extended to the malicious case (see Chapter 6).

Let the function f that Alice and Bob want to compute be represented by a circuit C_f. Let also I_A, O_A denote the sets of Alice's input/output wires, and I_B, O_B denote the sets of Bob's input/output wires.

Bob then computes the garbled circuit GC_f denoted by

$$\begin{aligned} GC_f &= \langle \langle \text{PEG-4-Tuple}_i : 1 \leq i \leq |C_f| \rangle, \langle OP_i : i \in O_A \rangle \rangle \\ &= \langle \langle t_i : 1 \leq i \leq |C_f| \rangle, \langle u_i : i \in O_A \rangle \rangle , \end{aligned}$$

where t_i denotes i-th PEG-4-Tuple, $|C_f|$ denotes the number of gates in the circuit C_f and u_i denotes the ordered pair for Alice's output wires O_A. Note that ordered pairs for Bob are not included in GC_f, since in that case Alice could evaluate GC_f and could learn Bob's output bits once she has computed the garbled circuit. This should not be possible.

We now describe a two-party protocol, which we will denote by Π_f. See Figure 5.2 for the illustration and notation of the protocol Π_f. On a high level, there are four phases in the protocol Π_f. We now describe the protocol Π_f in full detail. Let k be a security parameter which denotes the length of the garbled string.

1. Bob generates garbled strings $w_{i,b} \in_R \{0,1\}^k$ for $i \in I_A$ and $b \in \{0,1\}$. These garbled input strings are for Alice's input wires and are generated to be able to perform OT.

2. OT is run in order for Alice to learn her garbled input values. Bob is the sender with private input $w_{i,0}, w_{i,1}$ for $i \in I_A$ which are generated in Phase 1, and Alice is the chooser with private input $x_i \in \{0,1\}$. At the end of OT Alice receives w_{i,x_i} for her input bit x_i and Bob gets no information about which one is chosen.

3. In this phase Bob prepares the garbled circuit GC_f such that the garbled strings $w_{i,0}, w_{i,1}$ for $i \in I_A$ which are generated in Phase 1 are used for the corresponding wires. He sends the circuit and his garbled inputs to Alice.

4. Alice evaluates the circuit GC_f and computes the garbled output values. She sends the garbled output values for Bob's output wires back. Note that neglecting this round allows only one private output.

A Secure Two-Party Protocol

Notation:

- I_A, O_A denote the sets of Alice's input/output wires. Similarly, I_B, O_B denote the sets of Bob's input/output wires. C_f is the Boolean circuit which computes the function f. $|C_f|$ denotes the number of gates in circuit C_f.
- Alice's ordered pair is denoted by $OP_i = (w_{i,0}, w_{i,1})$ for $i \in O_A$. The garbled circuit is denoted by $GC_f = \langle \langle \text{PEG-4-Tuple}_i : 1 \leq i \leq |C_f| \rangle, \langle OP_i : i \in O_A \rangle \rangle$.

Common Input: f
Compute: $f(x, y) = (f_1(x, y), f_2(x, y))$

Alice	**Bob**
Private Input: $x = \langle x_i \in \{0,1\}, i \in I_A \rangle$	**Private Input:** $y = \langle y_i \in \{0,1\}, i \in I_B \rangle$

Phase 1: Generation of bit-strings.

Generate $w_{i,b} \in_R \{0,1\}^k, i \in I_A, b \in \{0,1\}$

Phase 2: OT Run in parallel, for $i \in I_A$.

Receiver		**Sender**
Private Input: x_i		**Private Input:** $\langle w_{i,0}, w_{i,1} \rangle$
	OT subprotocol $\longleftrightarrow$	
Private Output: $\langle w_{i,x_i} \rangle$		**Private Output:** $\perp$

Phase 3: Construction.

Compute GC_f s.t. for all $i \in I_A$ $\langle w_{i,0}, w_{i,1} \rangle$ are used for the corresponding wires in GC_f

$GC_f, \langle w_{i,y_i} : i \in I_B \rangle$ $\longleftarrow$

Phase 4: Evaluation.

Evaluate GC_f

$\langle w_{i'} : i' \in O_B \rangle$ $\longrightarrow$

Match w_i with OP_i for $i \in O_A$	Match $w_{i'}$ with $OP_{i'}$ for $i' \in O_B$
Private Output: $f_1(x, y)$	**Private Output:** $f_2(x, y)$

Figure 5.2: A secure two-party protocol in the semi-honest model

5.4 Additional Modifications for the Security Analysis

Before we analyze the security of the protocol Π_f in the next section we present the following two additional modifications over the circuit GC_f. They are necessary to be able to simulate the protocol Π_f in the real/ideal simulation paradigm. We want to point out that the following modifications are also going to be used for the protocols in malicious model, namely in Chapter 6 and in Chapter 7.

- **Modification 1.** We modify the input wires of Bob in circuit C_f in the following way: for each input wire of Bob (say W_B), we add an AND and an OR gate as shown in Figure 5.3 in such a way that the AND gate has one new input wire for Alice (say W_A) and the original input wire from Bob (W_B). This composition of gates always reproduces the value of the wire W_B independently of the value of W_A (i.e. $(W_A \wedge W_B) \vee W_B = W_B$).

  This modification is applied so that the simulator is able to learn the input of the corrupted Bob. This will be used in the security analysis. On a higher level, if the simulator knows the garbled circuit, two garbled values for each of Alice's input wires (together with their corresponding bit values) and a garbled value for each of Bob's input wires (of which he does not know the corresponding bit value) then it is possible to compute the bit value of the garbled value for each of Bob's input wires.

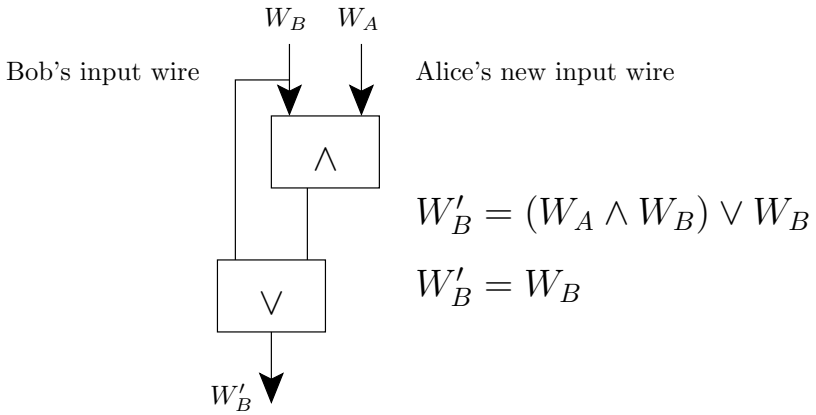

Figure 5.3: Additional gates for each of Bob's input wires

We now proceed with the details. Let $w_{A,i,0}$ and $w_{A,i,1}$ denote Alice's garbled input values (for 0 and 1) for $i \in I_A$, and $w_{B,i,b}$ Bob's garbled input value for $i \in I_B$ and for some $b \in \{0,1\}$. If the garbled values $w_{A,i,0}, w_{A,i,1}, w_{B,i,b}$ and the garbled circuit are given then by evaluating the garbled AND gate with the garbled inputs $(w_{A,i,0}, w_{B,i,b})$ and $(w_{A,i,1}, w_{B,i,b})$ it is possible to decide on the bit value of $w_{B,i,b}$ (i.e. learn the bit value b). Namely, if after these two evaluations the same garbled string is obtained, then this means that Bob's garbled input corresponds to bit 0; otherwise, Bob's input

bit is 1. We note that this deduction process does not work for an arbitrary Boolean gate, and this is the reason why we modified the circuit in such a way that the input gates are AND gates. For example, evaluation of an XOR gate (has Alice's input wire and Bob's input wire) using $(w_{A,i,0}, w_{B,i,b})$ and $(w_{A,i,1}, w_{B,i,b})$ would always result in two different garbled values from which one cannot conclude the input bit of the corrupted Bob. Note that for efficiency reasons W_A could be the same for all W_B wires.

- **Modification 2.** We modify the output wires of Bob in circuit C_f in the following way: for each output wire of Bob, we add the construction presented in Modification 1 and add two XOR gates as shown in Figure 5.4 in such a way that a new input wire for Alice is added. This composition of gates always reproduces the original output bit of Bob in the garbled circuit independently of the value of Alice's additional input (i.e. the bit value of wires (i), (ii), (iii) in Figure 5.4 is the same regardless of Alice's input.). The simulator has to learn the garbled output values of the corrupted Bob together with their corresponding bit (we discuss this more in the security analysis), and by this modification the simulator would be able to learn them.

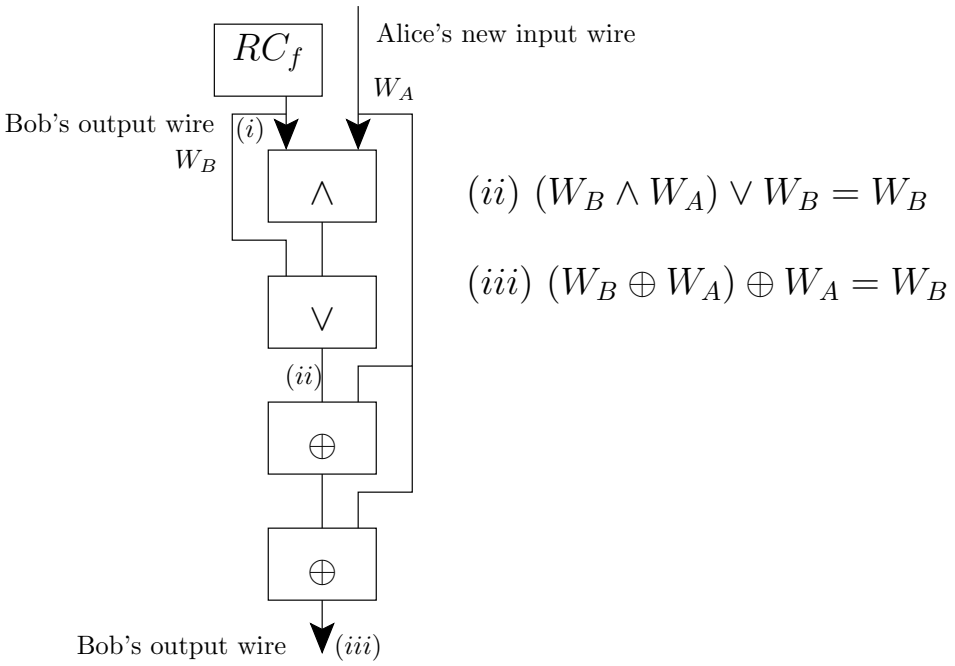

Figure 5.4: Additional gates for each of Bob's output wires

If the garbled values $w_{A,i,0}, w_{A,i,1}, w_{B,i,b}$ and the garbled circuit are given then by evaluating the modification in Figure 5.4 it is possible to compute the garbled values for each of Bob's output wires together with their corresponding bit. More precisely, as described above, by evaluating the garbled AND gate with the garbled inputs

$(w_{A,i,0}, w_{B,i,b})$ and $(w_{A,i,1}, w_{B,i,b})$ it is possible to compute the output bit value of Bob's garbled output value $w_{B,i,b}$ for wire (i) in Figure 5.4 (i.e. learn the bit value b). We here note that the bit value of (ii) is the same as the bit value of (i), so the bit value of (ii) is also known.

We next show that it is possible to compute both garbled output values for the output wires of XOR gates from the second construction (for the wires (ii) and (iii) in Figure 5.4), together with their corresponding bit. Let $\hat{w}$ be the evaluated garbled output value of the OR gate for the wire (ii). By evaluating the garbled XOR gate with the garbled inputs $(w_{A,i,0}, \hat{w})$ and $(w_{A,i,1}, \hat{w})$ where the bit value of $\hat{w}$ is known one can learn both garbled values for the output wires of XOR gates ((ii) and (iii)), and the corresponding bit values. Namely, these two evaluations always result in two different garbled strings from which it is easy to learn the corresponding bits.

We stress that the protocol Π_f is applied to this final modified circuit together with the above modifications.

5.5 Security Analysis

We are now ready to analyze the security of the protocol Π_f. The security analysis of the protocol Π_f is done according on the real/ideal simulation paradigm which is similar to the one in [LP07] (see Section 3.3.3 for a definition of security). Since we only consider the semi-honest model, any OT protocol that is secure in the presence of semi-honest adversaries is suitable to be used as a black-box in Π_f. To analyze the security of Π_f we show a simulation by assuming the cases where either Bob or Alice is corrupted. Informally speaking, to see that Π_f is secure when Alice is corrupted, we observe that without the knowledge of the garbled values of the input wires of a gate, the garbled output values of the gate look random. Therefore, the knowledge of one garbled value of each of the input wires of a gate discloses only a single garbled value of the output wire of the gate and she cannot distinguish the other garbled output value from random. Also, the OT protocol ensures that Alice learns only a single garbled value for each input wire. Therefore, inductively, Alice can compute only a single garbled output value of each gate, and in particular of the garbled output values of the output wires of the circuit. As for the security when Bob is corrupted, we note that Bob receives messages only during the OT protocol, and by definition of OT, he does not learn any information from the OT phase.

We now show a simulation by considering the cases where either party is corrupted. In both cases, we construct a simulator that has internal simulated communication with the adversary, and external communication with the trusted party of its ideal model.

Assume Bob is corrupted.

On a high level, Alice receives her garbled input values from OT where Bob does not learn which of the garbled strings has been received. Then she computes the circuit where she can decrypt only one value, and computes only one garbled string for each output wire. Since the circuit and the input are correct, correctness is ensured.

Let R_B be an adversary corrupting Bob. We shall now construct a simulator S_B for R_B. Since we assume that the OT protocol is secure, we analyze the security of the protocol Π_f in the hybrid model with a trusted party computing the OT functionality (see Section 3.3.4).

The simulator.

1. The simulator S_B chooses a fixed input $x' = 0$ for Alice and uses it only in the beginning of the protocol Π_f to be able to start the protocol (namely, to run the OT phase) but it is not used later on.

2. S_B invokes R_B and extracts the garbled input values $w_{i,0}$ and $w_{i,1}$ for $i \in I_A$ from the OT subprotocol. Note that S_B can extract this information since we analyze the security of the protocol in the hybrid model with a trusted party computing the OT functionality. Namely, Bob sends the garbled input values $w_{i,0}$, $w_{i,1}$ to the trusted party, and so the simulator S_B obtains them directly.

3. S_B receives the garbled circuit GC_f from R_B together with its garbled input values to S_B.

4. Now the input of R_B will be extracted as follows. The simulator S_B receives the garbled values that correspond to Bob's input. Let w_i be Bob's garbled input value for $i \in I_B$.

   S_B obtains the input for Bob because of Modification 1. More precisely, the simulator knows $w_{i,0}$, $w_{i,1}$ for $i \in I_A$ and w_i for $i \in I_B$, and by Modification 1 the simulator can learn the input bit of Bob for each w_i for $i \in I_B$. (In the real case, this does not happen since Alice learns only one garbled input value from OT for each of her input wires.) Then, S_B sets y to be the value and sends it to the trusted party. The trusted party replies with $f_2(x, y)$ to S_B.

   Now the simulator knows the private output of the corrupted party from the trusted party but it has to convert the output into the corresponding garbled values [1]. The garbled values of the corrupted Bob together with the corresponding bits are achieved as follows.

   The simulator S_B first computes the circuit as in the real protocol Π_f and obtains garbled output values. We know that once the simulator computes the circuit it can compute a single garbled value per output wire. However, the evaluated garbled

[1] Note that in the real case Alice sends Bob's garbled output values for the output wires.

output values are not necessarily the correct ones since the simulator computes the garbled circuit in the case that $x' = 0$. Therefore, Modification 2 has been applied in order to learn both garbled output values of Bob, and the corresponding bits. As we described above, the simulator learns the output bit of w_i for $i \in I_B$ from the AND gate in Figure 5.4 (for wire (i)). This bit value is the same as the bit value for wire (ii) in Figure 5.4. Then, by decrypting the XOR gates, the simulator learns both garbled values, and their corresponding bits. As we mentioned before, this does not happen in the real case since Alice learns only one garbled input value from OT for each of her input wire.

Hence, since the simulator knows the private output of the corrupted party and the corresponding garbled output values it can send the correct garbled outputs to R_B.

Analysis. We claim that the view of R_B with S_B is statistically close to its view in a hybrid execution of the protocol Π_f with a trusted party computing the OT protocol. (Note that the protocol Π_f is not statistically secure since the simulation is in the hybrid model for OT functionality, and it depends on the implementation of the OT subprotocol).

We know that the circuit and the garbled values that Bob sends are correct since we are in the semi-honest model. We now show that the simulated view of R_B is identically distributed to its view in an execution of the protocol Π_f. Actually, they are identical since S_B just runs the honest Alice's instructions when interacting with the corrupted Bob. Since S_B follows Alice's instructions each time, the above process results in a distribution that is identical to the view of R_B in a real execution with Alice.

Assume Alice is corrupted.

The security analysis when Alice is corrupted is very similar to the one of [LP07]. During the protocol Π_f Alice sees the circuit and they run a secure OT where she gets only the garbled values corresponding to her input bits. On a high level, the simulator first extracts the input of Alice from OT functionality in the hybrid model and sends it to the trusted party and learns the output value. Based on the output value, the simulator constructs a new garbled circuit which always outputs the output value which is received from the trusted party. We refer to [LP07] for details.

5.6 Performance Analysis

The protocol Π_f is efficient for several reasons. Firstly, it has only a constant number of rounds (in fact only a few rounds) and it is independent of the size of the circuit. Secondly, since we are in the semi-honest model it is sufficient for Bob to construct only a single garbled circuit and for each input wire only one OT is sufficient to run.

The computational complexity of Π_f consists of running an OT subprotocol for every input wire of the circuit and computing a pseudorandom function for every gate in the circuit. The communication complexity is dominated by sending a garbled circuit GC_f.

More precisely, GC_f consists of an ordered pair, and for every gate the output of the pseudorandom function is linear in the security parameter. Therefore, the communication complexity is linear in the size of the circuit. Thus, Yao's protocol is efficient if and only if the circuit representation of f and the input of one of the parties are not very large.

6

Secure Two-Party Computation in the Malicious Model

In this chapter, we present a two-party protocol in the presence of malicious adversaries. We start by describing the necessary building blocks to be able to construct our protocol. Furthermore, we also address a protocol issue in the malicious model that arises with the use of oblivious transfer. We explain this issue for the case of several two-party computation protocols based on garbled circuits. The security of our protocol is analyzed according to the real/ideal simulation paradigm, as Lindell and Pinkas did for the malicious case. We end this chapter by discussing how to reduce the failure probability of the simulator in one part of their proof by applying Modification 1, presented in Section 5.4. We remark that in Chapter 7 we will add fairness to this protocol in a modular way.

Parts of this chapter are based on [KS06a, KS06b, KS08] (joint work with Berry Schoenmakers) and based on [KSV07] (joint work with Berry Schoenmakers and José Villegas).

6.1 Introduction

In general, constructing a protocol against malicious adversaries is more difficult than against semi-honest adversaries since a malicious adversary may deviate from the protocol in an arbitrary way. The overall goal of a protocol in the presence of malicious adversaries is to enforce the malicious adversary to follow the protocol specifications like in the semi-honest model. Any malicious behavior should not be successful without being detected.

Typically, a secure two-party protocol which is constructed in the semi-honest model can be transformed into a secure two-party protocol against malicious adversaries by applying some standard modular expansion. This transformation can often be implemented using generic zero-knowledge techniques [GMW86], however, this approach is usually highly inefficient and impractical to use. In this chapter, we use several special protocol techniques to avoid these expensive proofs.

As mentioned earlier, Yao's garbled circuit approach originally dealt with passive adversaries [Yao86]. Now consider what happens if Yao's protocol is run when the adversary is malicious. To start with, we have no guarantee that the constructor (Bob) generated

the garbled circuit correctly. Namely, the circuit may not implement the function f. This may clearly affect the privacy of the evaluator's (Alice's) inputs and the correctness of its outputs. Secondly, the OT protocol must satisfy the requirements for secure two-party computation in the case of malicious adversaries, and must also preserve its security when run in parallel. We discuss these kinds of misbehavior in greater detail in the next section. In this chapter, we also show a protocol issue that arises when using OT in the malicious case. We describe this issue for a protocol by Pinkas [Pin03] and for the Fairplay protocol [MNPS04], and we discuss why this issue still persists for a suggested modification of the Fairplay protocol [FM06]. We propose a solution which uses committed oblivious transfer (OT) instead of (plain) OT. We also propose using committing OT which leads to an alternative and more efficient solution.

6.2 Building Blocks for Malicious Model

6.2.1 The Cut-and-Choose Technique

The *cut-and-choose technique* is an interactive proof technique which was first introduced by Rabin [Rab78]. As we said before, when considering malicious behavior the garbled circuit may not have been constructed correctly by Bob (the constructor). To gain efficiency, the correctness of the garbled circuit is often proved by using the cut-and-choose technique instead of using generic the expensive zero knowledge proofs (e.g., [Pin03, MNPS04, FM06, LP07]).

Malkhi et al. [MNPS04] presented a two-party protocol based on Yao's garbled circuit technique, which is the first implemented application for secure two-party computation. This protocol uses a basic cut-and-choose technique called the *1-out-of-m technique*. Roughly speaking, Bob constructs m garbled circuits where m is a security parameter, and sends them to Alice. Alice chooses one circuit at random for evaluation. Bob then opens all the remaining $m-1$ circuits to assure Alice that the circuits are correctly prepared. Alice verifies that these $m-1$ circuits were correctly prepared. Then, an OT protocol is run in order for Alice to receive her garbled input values for the evaluation-circuits. After Bob sends his garbled input values to Alice she computes the chosen circuit without further interaction with Bob. She computes all the garbled output values of the chosen circuit and sends only Bob's garbled output values to him. Finally, they compute their respective output. Thus, security comes from the fact that Bob does not know which garbled circuit will be evaluated. Note that with this technique Bob has the capability to cheat with the construction of the garbled circuit with probability at most $1/m$.

Pinkas [Pin03] presented a more advanced protocol against malicious behavior. It uses the *$m/2$-out-of-m technique* to achieve exponentially small cheating probability. The $m/2$-out-of-m technique can be briefly described as follows: Bob constructs m garbled circuits as usual and sends them to Alice. Alice chooses $m/2$ circuits at random for evaluation and tells Bob to open the remaining ones. Let the evaluated circuits be called *evaluation-circuits* and the opened circuits be called *check-circuits*. She verifies whether the check-circuits were

correctly prepared. If so, they run an OT protocol. Bob also sends his garbled input values to Alice. She evaluates the evaluation-circuits and computes a garbled value per output wire. As we said before, with this technique Bob cannot cheat successfully except with exponentially small probability (in m). The reason is that if there are at least $m/4$ garbled circuits which are not correct and none of them is chosen by Alice during the challenge phase then a wrong input is received. However, this event occurs with probability at most $\binom{3m/4}{m/2}/\binom{m}{m/2} \leq 2^{-m/4}$ (see [FM06, Woo06] for more details.).

Observe that not only the correctness of garbled circuits can be proved using the cut-and-choose technique but also the correctness of commitments can be proved. In this case, there is a prover who will generate *many* commitments (like the garbled circuits above) and there is a verifier who is allowed to choose a random set of these commitments using the cut-and-choose technique. The prover reveals the chosen commitments, and the half of the other commitments is ensured to be correct to the verifier with very high probability by the same reasoning as above.

Our protocol in the malicious model is going to use the $m/2$-out-of-m technique in order to ensure a negligible cheating probability for malicious adversaries. For security reasons, the randomness, which is used to select the circuits, must be computed by both parties, as observed by [LP07]. The reason is that, when analyzing the security of the protocol when Alice is corrupted the simulation is run in such a way that incorrect circuits are evaluated. This is solved using rather standard techniques, like choosing the circuits to be opened via a coin-tossing protocol.

We finally remark that Franklin and Mohassel in [FM06] presented a "relaxed" protocol which allows leaking at most one bit of input of the honest party. In this way, there is no need to generate many garbled circuits, and actually it is sufficient to construct only two of them. Roughly speaking, Alice constructs a garbled circuit and sends it to Bob. Similarly, Bob also constructs a garbled circuit and send it to Alice. OT is run for each of them, one after the other. The first OT is run for the first garbled circuit where Alice is the sender and Bob is the receiver. Similarly, the second OT is run for the second circuit where the roles of Alice and Bob are changed. They compute their respective circuits, and compute the output values. They then run a new secure protocol comparing the outputs to check whether they are equal. In this way the protocol becomes more efficient since there is no need to generate many garbled circuits, and therefore the cut-and-choose technique will not be necessary. However, a corrupted party may learn at most one bit of information about the honest party's input (during the comparison of the outputs for checking the equality). For example, assume that Bob is corrupted and he is using an input bit 0 in the oblivious transfer for obtaining the garbled value for evaluating the circuit constructed by Alice, and use 1 for the same wire for the circuit he constructed (and to be evaluated by Alice). Then he may learn something from the fact that the output values of the two evaluations match or not. For example, suppose Alice and Bob agreed to run the millionaires protocol. Assume that a malicious Bob constructs his garbled circuit correctly. Assume also that a malicious Bob has input $B = 100$ and an honest Alice has input $A = 110$. A malicious Bob can send the wrong input $B' = 101$ in garbled form for his "correct" circuit and can use the correct input $B = 100$ for Alice's circuit. The output of two circuits would be

the same (namely that Alice is richer) and she cannot detect that Bob cheated, so Alice would think that everything ran correctly. However, Bob learnt that Alice's input value is bigger than 101 instead of learning only bigger than 100. (For example, only the values 110 and 111 are greater than 101 and therefore, the second bit of Alice's input must be 1.) Hence, different inputs might result in extra information for a malicious participant without detection.

In this chapter, we consider full privacy rather than leaking some private information of the honest party.

6.2.2 Majority Circuit Computation

We consider protocols which let Alice decide (at random) for each m garbled circuit, whether such a circuit must be opened (for checking the correctness), or whether the circuit will be evaluated. Exactly $m/2$ circuits will be checked by Alice, and she will evaluate the remaining half. Among the evaluated circuits there may still be a few incorrect circuits (not computing function f), but with high probability the majority of the evaluated circuits is correct. Before we proceed with the details, we explain why Alice should to send the output values only for a majority circuit, rather than sending Bob the output values for all evaluated circuits. (e.g., [Pin03, LP07]). Note that if there is only one output for Alice but not for Bob, then the majority circuit for Alice is straightforward. However, if there is also a private output for Bob then we have to ensure that Bob does not get any further information, and this is done by sending outputs of only the correct one. In this chapter, we consider the case where both parties receive respective private outputs.

For example, if Bob is malicious then he may construct $m-1$ garbled circuits that compute the function $f(x,y) = (f_1(x,y), f_2(x,y))$ and a single circuit which simply outputs Alice's input values. Since Alice evaluates $m/2$ out of m circuits, the probability would be $1/2$ that this single circuit is selected by Alice. Therefore, Bob receives the outputs of the incorrect circuit from Alice with probability $1/2$ at the end of the evaluation. It follows that, Alice should determine a correct circuit, the so-called *majority circuit* out of all evaluation-circuits. We actually need to deal with two types of majority circuits. One is related to Bob's output wires (which we will indicate by index r throughout the chapter; to ensure that Bob cannot get any information on Alice's inputs), and one is related to Alice's output wires (to ensure that Alice is guaranteed to get correct output values). These majority circuits can be characterized as follows. First, the *majority value* for an output wire is the bit value that occurs most frequently for that wire when considered over all $m/2$ evaluated circuits (ties can be broken arbitrarily). Further, an output wire is called a *majority wire* if its value is equal to the majority value. And, finally, a circuit is called a *majority circuit for Alice resp. Bob* if all of Alice's resp. Bob's output wires are majority wires.

Lindell and Pinkas [LP07] presented a solution when two-parties receives respective outputs, dealing with majority. This works as follows (see Figure 6.1). Let $\mathcal{D}$ be a field that contains the range of values $f_2(x,y)_{x,y\in\{0,1\}^*}$, and let p, a, b be randomly chosen elements in

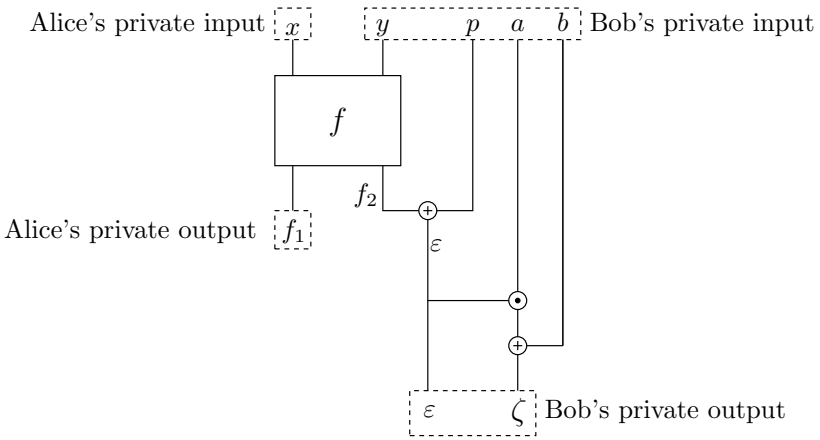

Figure 6.1: The function $g(x,(p,a,b,y)) = (f_1, \varepsilon, \zeta)$.

$\mathcal{D}$. A new function g is constructed from $f(x,y) = (f_1, f_2)$ by $g(x,(p,a,b,y)) = (f_1, \varepsilon, \zeta)$. Bob's new input will be y, p, a, b in such a way that the output of Bob is ε, ζ where $\varepsilon = p + f_2(x,y)$, $\zeta = a \cdot \varepsilon + b$ is a one-time pad encryption of Bob's output, and ζ is an information-theoretic message authentication tag of ε. Both Alice and Bob compute the new function g. At the end of the protocol, Alice sends ε and ζ to Bob. Bob checks first that $\zeta = a \cdot \varepsilon + b$; if yes, he outputs $\varepsilon - p$, and otherwise he outputs $\perp$. Note here that addition gates needs $O(\ell)$ boolean gates, and multiplication needs $O(\ell^2)$ boolean gates where $\ell = |\mathcal{D}|$.

In our protocol, we will present a different technique than the one described-above. Recall that when Alice evaluates the circuit she learns garbled values for every output wire. She obtains her output, and sends Bob's output back to him. Our protocol is run in such a way that Alice sends the bits of the majority circuit together with an OR-proof. Bob is going to reject, of course, if the proof is unsuccessful. In this way, we will not use the modification described-above.

6.2.3 The Equality-Checker Scheme

Recall that only one circuit is constructed by Bob for the protocol in the semi-honest model, and in that case he simply sends his garbled input to Alice. However, in the malicious model we have to be more careful. There are many circuits to be evaluated by Alice when the m/2-out-m technique is used, and one may wonder how to ensure that Alice and Bob use the same input for all the circuits.

From security point of view, if a corrupted party provides different inputs for garbled circuits, it can learn more information than what follow directly from the desired output of the function. Suppose, for example, we want to compute millionaires' problem (to find out who is richer) using Yao's garbled circuit with the m/2-out-of-m technique where m denotes the number of garbled circuits. Assume also that m denotes the length of the

inputs. A corrupted Alice could provide the inputs $\langle 10\ldots0\rangle$, $\langle 010\ldots0\rangle$,...,$\langle 0\ldots01\rangle$, and in this way learns Bob's input easily.

We now briefly show how the input can be guaranteed to be the same for every garbled circuit. That is, we have to ensure that both parties use the same input for all the garbled circuits. A solution to show the consistency of Bob's input is presented in the protocol by Pinkas [Pin03] that uses the proofs of partial knowledge technique of Cramer et al. [CDS94] (which uses public key operations). Later, the authors in [FM06, LP07] presented new alternative schemes which are based on using only symmetric operations (no public key operations). For efficiency reasons, we will also use the *equality-checker scheme* of Franklin and Mohassel in our protocol for the proof of consistency of Bob's input (later, slightly improved by Woodruff [Woo06]).

The basic structure of the equality-checker scheme is as follows: let

$$B_{i,j,j',b} = \text{commit}(w_{i,j,b} \| w_{i,j',b}; \gamma_{i,j,j',b})$$

be the commitment to the garbled values $w_{i,j,b}$ and $w_{i,j',b}$ for every i-th input wire and for every $b \in \{0,1\}$ in the j-th and the j'-th garbled circuit such that $1 \leq j < j' \leq m$ [1]. The idea is that a correctly built commitment binds the two garbled values that correspond to the same bit value for the same input wire, but in two different circuits. An equality-checker scheme between these circuits is the collection of the commitments $B_{i,j,j',b}$ (see [FM06] for more details). When Alice asks to open the check-circuits Bob also opens these commitments for the j-th and j'-th check circuits. Note that the number of commitments $B_{i,j,j',b}$ is $m(m-1)$ for m garbled circuits.

On the other hand, Alice's garbled input values for all the circuits must also be the same, and that follows directly by definition of OT. Namely, Bob is the sender with private input $(\langle w_{i,1,0},\ldots,w_{i,m,0}\rangle, \langle w_{i,1,1},\ldots,w_{i,m,1}\rangle)$ and Alice is the receiver with private input bit x_i for every input wire i of Alice where $x_i \in \{0,1\}$. At the end of OT, Alice receives $\langle w_{i,1,x_i},\ldots,w_{i,m,x_i}\rangle$, i.e, she receives the set of all garbled input values for all the circuits at once. Therefore, OT not only gives Alice her garbled input but also proves consistency of her input.

6.3 A Protocol Issue Regarding Oblivious Transfer Protocol

The problem that we address in this section stems from the particular way OT is used in the cut-and-choose protocols mentioned above. A malicious Bob cannot simply be assumed to send the right garbled strings to Alice during the OT protocol. Without any further protection, Alice's privacy and the correctness of her outputs may have been affected by Bob's behavior. This protocol issue is not related to OT per se, but rather to the interaction between OT and the surrounding protocol. A similar point is made in recent paper by Franklin and Mohassel [FM06] (of which our work is independent). Their discussion and proposed solution focuses on Fairplay [MNPS04], but as we will point out, Bob will still be able to cheat in a critical way.

[1] Note that the commitments here are only based on symmetric operations.

Recall that at some stage in Yao's protocol, OT is used to let Bob provide Alice the garbled strings corresponding to her input bits. To discuss the problem it suffices to consider a single OT in which Alice uses an input bit b as her private input, say, and Bob uses garbled strings w_0 and w_1 as his private input, where w_0 corresponds to bit value 0 and w_1 represents bit value 1. Alice can only evaluate the corresponding circuit correctly if she indeed gets the bit string w_b at the end of the OT protocol. The problem with the extensions to the malicious case of Yao's protocol as presented in [Pin03, MNPS04] is that a malicious Bob is not stopped from using other values than w_0 and w_1.

Therefore, Bob can follow several strategies to compromise correctness and/or privacy of Yao's protocol in the malicious case. We mention a few obvious deviations of the protocol, resulting in wrong values $\tilde{w}_0, \tilde{w}_1$:

- Bob may interchange the values of w_0 and w_1, putting $(\tilde{w}_0, \tilde{w}_1) = (w_1, w_0)$;
- Bob may duplicate either of the values w_0 and w_1, putting either $(\tilde{w}_0, \tilde{w}_1) = (w_0, w_0)$ or $(\tilde{w}_0, \tilde{w}_1) = (w_1, w_1)$;
- Bob may replace either or both of the values with a bogus value, denoted by $*$, putting $(\tilde{w}_0, \tilde{w}_1) = (w_0, *)$, or $(\tilde{w}_0, \tilde{w}_1) = (*, w_1)$, or even $(\tilde{w}_0, \tilde{w}_1) = (*, *)$.

The consequences for Alice's security are as follows in these cases.

- If Bob interchanges the values of w_0 and w_1 then clearly Alice will receive an incorrect garbled string for her input wire, but she will continue to evaluate the circuit without noticing anything (but obtaining wrong output values). Similarly, if Bob duplicates either of the values w_0 and w_1 then Bob is sure which input value Alice is using, and Alice possibly gets a wrong output (without her noticing anything).
- If Bob uses $(w_0, *)$ in OT where $* \neq w_1$ then depending on Alice's input bit b, she will either, if $b = 0$, be able to evaluate the circuit and produce an output for Bob (and not notice anything) or, if $b = 1$, she will notice that she cannot evaluate the circuit and cannot produce an output for Bob. Clearly, this way Bob learns the value of Alice's input b, and Alice cannot do anything about it.

To eliminate these problems, Bob must be forced to use the correct values in OT.

The repair of Franklin and Mohassel does not work.

We now review the proposed modification for Fairplay [MNPS04] by [FM06] and then present the reasons why the problem is not completely eliminated.

For the sake of simplicity, we assume that the garbled circuit has only one input wire for Alice. The more general case of multiple input wires can be dealt with by a simple extension of this case. The modification proposed in [FM06] is as follows. When Bob sends the garbled circuits to Alice, he also generates commitments to the garbled strings $w_{j,0}$ and $w_{j,1}$ used for Alice's input wire in the j-th circuit. Let these commitments be denoted by

$\text{commit}_B(w_{j,0}; r_{j,0})$ and $\text{commit}_B(w_{j,0}; r_{j,0})$, respectively where $r_{j,0}$ and $r_{j,1}$ are random bit strings. When verifying the $m-1$ check-circuits, Bob also opens these commitments and Alice verifies that these commitments are correct and whether the committed garbled strings $w_{j,0}$ and $w_{j,1}$ correspond to the garbled strings used in the j-th circuit.

If verification succeeds, Alice is sure with overwhelming probability that the evaluation-circuit and the corresponding commitments are correct as well. For the evaluation-circuit they run OT for Alice's input wire where Bob is the sender holding two witnesses and two garbled strings and Alice is the receiver holding her input bit. At the end of OT Alice receives one of the witnesses and the corresponding garbled string.

As a consequence, with probability at least $1-1/m$, Alice is ensured that the committed values for the evaluation-circuit are correct. Therefore, with probability at least $1-1/m$, Alice will notice when she gets a wrong garbled string for her input wire (e.g., because Bob interchanged or duplicated the garbled strings), and by using the $m/2$-out-of-m technique, this probability will be close to 1.

The problem with this proposal, however, is that it does not consider what happens once a cheating Bob sends a bogus value in the OT protocol, following the same scenario as described above. A cheating Bob will simply construct all the garbled circuits and all the commitments correctly. Therefore, Alice will certainly accept the check-circuits and the corresponding commitments. But during OT Bob may cheat as described above. Suppose Bob decides to use $((w_{j,0}, r_{j,0}), (*, *))$ as input to OT. Then Alice's input bit is 0 and she will obtain correct values $(w_{j,0}, r_{j,0})$, which pass all verifications, and she will be able to evaluate the circuit without any problems. But if Alice's input bit is 1, she gets bogus values, which she will notice, and she is unable to hide this fact from Bob. Clearly, Bob is able to tell what Alice's input bit is, thereby compromising her privacy.

6.4 Our Modification to Fix the Protocol Issue

In this section, we propose two schemes to fix the issue described in Section 6.3. We first describe how we can repair the issue with the use of *committed oblivious transfer*.

6.4.1 Using Committed Oblivious Transfer

We describe how we can fix the issue in case of the 1-out-of-m technique, by using committed OT which works for bit strings. Note that since the inputs are garbled strings, an implementation of committed OT that allows to transmit bit-strings is necessary (which is presented in Chapter 4). A protocol using the 1-out-of-m technique and committed OT runs as follows.

Phase 1. Before Bob sends the garbled circuits to Alice, he also generates commitments to the garbled strings for her inputs. He then sends the garbled circuits together with the corresponding commitments to Alice.

Phase 2. When Alice asks Bob to open the circuits he also opens all the corresponding commitments.

Phase 3. Alice is now almost (i.e., with probability $1/m$) sure that the commitments for the evaluations-circuits are correct. As required for committed OT, Alice commits to her input bits. Then for the evaluation-circuit they run committed OT for every input wire of Alice from which Alice learns her corresponding garbled input values.

Now, any attempt by Bob to use other values than the correct garbled strings in committed OT will be detected by Alice, and she can abort the protocol at this point. The point is that she can abort without revealing any information on her input bits, as committed OT will only finish successfully (by definition) if Bob's private inputs correspond to the commitments. As desired, the cheating probability of Bob is thus $1/m$ (which can be made exponentially small by using committed OT in combination with the m/2-out-of-m technique).

6.4.2 Using Committing Oblivious Transfer

As described above, the use of committed OT also requires Alice to send commitments to her input bits, and one may wonder why this would be needed. In fact, the use of committed OT may be too much for securing the protocols which use cut-and-choose methods. Therefore, as a concrete example, we like to propose using committing OT which conceptually sit between plain OT and committed OT (see Section 4.2 for committing OT).

Before Alice chooses which circuits she wants to evaluate, Alice and Bob run committing OT for all garbled circuits. This will be done using only one committing OT per input wire of Alice. Thus Alice receives the garbled strings for her input values, together with commitments on all garbled strings associated with her input wires. She will check the commitments for the check-circuits, ensuring her that the garbled strings for the evaluation-circuits are valid as well, with high probability. More concretely, a protocol using a 1-out-of-m technique with committing OT may run as follows.

Phase 1. Bob sends garbled circuits GC_j for $j \in \{1, \ldots, m\}$ to Alice.

Phase 2. For each input wire of Alice the following is done. Let x_i denote Alice's input bit for the i-th input wire. Alice and Bob run committing OT where Bob is the sender holding the bit strings $(w_{i,1,0}, \ldots, w_{i,m,0})$ and $(w_{i,1,1}, \ldots, w_{i,m,1})$, consisting of the garbled strings for Alice's input wires in all GC_j, $j \in \{1, \ldots, m\}$. Alice is the receiver holding her input bit x_i. At the end of committing OT, Alice gets $(w_{i,1,x_i}, \ldots, w_{i,m,x_i})$. The commitments by Bob to his input strings are common output.

Phase 3. Alice randomly chooses $r \in_R \{1, \ldots, m\}$ and sends r to Bob.

Phase 4. Bob opens the circuits GC_j and the corresponding commitments $\text{commit}_B(w_{i,j,b})$ for $j \in \{1, \ldots, m\} \backslash r$ and $b \in \{0, 1\}$.

Phase 5. Alice first checks whether the check-circuits are correct, and then she verifies all committing OTs. If all are correct, Alice evaluates circuit GC_r as before.

The cheating probability of Bob is now $1/m$. A difference with the protocols based on committed OT of the previous section is that in this protocol committing OT is applied to all circuits, whereas in the previous protocols committed OT is applied to the evaluation-circuits only. Note, however, that using the $m/2$-out-of-m technique the number of evaluation-circuits is exactly half of the total number of circuits. Hence, for the general case when Bob's cheating probability should be exponentially small, the protocol based on committing OT is more efficient than one based on committed OT.

6.5 A Modification to the Circuit

Alice needs to be able to determine a majority circuit for Bob, but at the same time she should not learn Bob's actual output values. Let C_f denote a circuit computing function $f(x, y)$. We extend circuit C_f to a *randomized circuit* RC_f as follows (see Figure 6.2. Hence, for each output wire W_i of Bob, a new input wire W'_i is added as well as a new output wire W''_i, such that $W''_i = W_i \oplus W'_i$. This will require as many XOR gates and additional input wires of Bob as the number of Bob's output wires of the original circuit C_f. This modification was suggested to us by Pinkas to resolve a subtle problem for the protocol of [Pin03], which we communicated to him [Pin05]. We remark that this modification is going to be used in our protocols in this chapter and in Chapter 7.

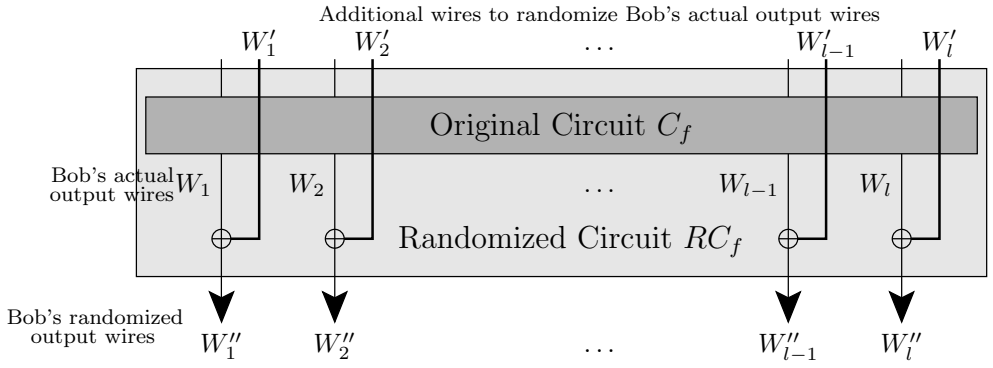

Figure 6.2: The randomized circuit RC_f made from the original circuit C_f.

6.6 A Two-Party Protocol in the Malicious Model

In this section we present a formal description of our two-party protocol against active adversaries. The object of the protocol is to evaluate a function of the form $f(x, y) =$

$(f_1(x,y), f_2(x,y))$ securely, where Alice holds input x and gets output $f_1(x,y)$ and Bob holds input y and gets output $f_2(x,y)$. For simplicity, we assume that these inputs and outputs are all of equal length, i.e., $f : \{0,1\}^\ell \times \{0,1\}^\ell \to \{0,1\}^\ell \times \{0,1\}^\ell$, for some integer value ℓ.

Before we go into details for our protocol we remark that a protocol computing two private outputs (even with fairness) could possibly work as follows: First modify a protocol allowing one private output for the evaluator. The idea is to run this protocol twice one after the other, by interchanging the roles. Namely, Alice will be the evaluator in the first protocol execution, and Bob will be the evaluator in the second protocol execution. In the first execution the parties compute $f'(x,y) = (f_1(x,y), \perp)$ where Alice learns $f_1(x,y)$ (in this case, Alice is the evaluator) and in the second execution the parties compute $f''(x,y) = (f_2(x,y), \perp)$ where Bob learns $f_2(x,y)$ (in this case, Bob is the evaluator). Roughly speaking, if a protocol is run twice for the functions $f'(x,y)$ and $f''(x,y)$ then, in order to analyse the security, we only need to ensure that the circuits for the functions are correct and their inputs are the same for both protocols. The correctness of the circuits for the functions $f'(x,y)$ and $f''(x,y)$ can be proven easily by the cut-and-choose technique. Next, both parties has to be sure that the second protocol execution uses exactly the same inputs as in the first protocol execution. This can be also done, for example, by adding an additional round as a first round in which parties commit to their input bits and sent to each other. Then start running the protocols. Finally, the parties need to prove that the inputs they are using are the same for both protocol executions. We leave the investigation of this approach as an interesting future problem.

Let's now present our protocol in greater detail. Let RC_f denote the randomized boolean circuit for function f, see Figure 6.2. Let I_A denote the set of Alice's input wires and I_B the set of Bob's input wires. Similarly, O_A denotes the set of Alice's output wires and O_B the set of Bob's output wires. Furthermore, we use I'_B to denote the additional input wires for Bob, used in the construction of RC_f from C_f. Note that $|I_A| = |I_B| = |I'_B| = |O_A| = |O_B| = \ell$. Accordingly, we write $x = \langle x_i \in \{0,1\} : i \in I_A \rangle$ for Alice's input, $y = \langle y_i \in \{0,1\} : i \in I_B \rangle$ for Bob's input, and $z = \langle z_i \in \{0,1\} : i \in I'_B \rangle$ for Bob's random input to RC_f. Further, $|RC_f|$ denotes the number of gates in the circuit RC_f, and we $\mathcal{W}$ denote the set of all wires in the circuit RC_f. Hence, $I_A \cup I_B \cup I'_B \cup O_A \cup O_B \subseteq \mathcal{W}$.

Commitments to Bob's ordered pairs $OP_{i,j} = (w_{i,j,0}, w_{i,j,})$ for his i-th output wire and for the j-th garbled circuit are denoted by $COP_{i,j} = (\text{commit}_B(w^0_{i,j,0}; \beta_{i,j,0}),$ $\text{commit}_B(w_{i,j,1}; \beta_{i,j,1}))$. Bob then converts the *randomized circuit* RC_f into *garbled randomized circuits* GRC_j for $j = 1, \ldots, m$ as follows:

$$\begin{aligned} GRC_j &= \big\langle \langle \text{PEG-4-Tuple}_{i,j} : 1 \le i \le |RC_f| \rangle, \langle OP_{i,j} : i \in O_A \rangle, \langle COP_{i,j} : i \in O_B \rangle \big\rangle \\ &= \langle \langle t_{i,j} : 1 \le i \le |RC_f| \rangle, \langle u_{i,j} : i \in O_A \rangle, \langle v_{i,j} : i \in O_B \rangle \rangle \end{aligned}$$

where $t_{i,j}$ denotes the permuted-encrypted-garbled-4-tuple for the i-th wire in the j-th circuit, $u_{i,j}$ denotes the ordered pair for Alice's output wires I_A and $v_{i,j}$ denotes the committed ordered pair for Bob's randomized output wires O_B.

When Bob constructs the garbled circuits GRC_j's in Phase 3 he chooses secretly a bit b at random for each of his additional input wires W'_i (see Figure 6.2). He then generates

garbled values to the additional input wires W_i' and to the additional output wires W_i'' as in the original garbled circuit. In the same way as his original inputs, he then commits to them. Bob is required to prove that all garbled inputs for each circuit of the same wire are consistent by the equality-checker scheme [FM06]. Hence, he must also choose the same bit b for each of the additional wires in the garbled circuits. The $m/2$-out-of-m technique assures Alice that, with high probability, at least half of the circuits compute the correct function and receive the same input values from Bob for all the circuits.

We remark that there is an observation by Pinkas which improved our protocol [Pin08]. This observation applies to a previous version of our protocols [KS06b, KS08], and to the protocol in [Pin03], and any other protocols of similar nature. The problem occurs when Bob is corrupted and is described as follows: At the end of our protocol Alice sends the garbled values of Bob's output wires for the majority circuit. This majority circuit computes the correct function f. However, informing Bob that a certain circuit is a majority circuit allows leaking some information about Alice's input. Let's consider, for example, the following attack by Bob: He generates $n-1$ garbled circuits which compute f correctly and one garbled circuit that computes f correctly only if Alice's first input bit is 0, and which can not be computed at all otherwise. This can be done by putting some garbage values in the permuted-garbled-4-tuple of a certain gate which uses Alice's first input bit (see Section 5.1.1). With probability 1/2 this circuit will not be selected during the Opening & Checking phase. If Alice's first bit is 0, then, with probability $2/n$, the garbled values of this circuit will be sent to Bob. If Alice's first input bit is 1, then Alice cannot compute that garbled circuit. Therefore, it will not be a majority circuit and this event will occur with probability 0. We highlight that any protocol which reveals the index of a majority circuit to Bob will have this problem. Therefore, we at the end of the protocol only the bit values of the majority circuit are revealed to Bob.

In our protocol we use two types of commitments, namely homomorphic ("asymmetric") commitments, e.g., Pedersen commitments [Ped91], and other ("symmetric") commitments, e.g., constructed from pseudorandom generators [Nao91]. We let $\text{commit}_P(m;r)$ denote a symmetric commitment to a message m using randomness r generated by party P, and we use $\text{commit}^h_P(m;r)$ to denote homomorphic commitments.

There are 8 phases in our protocol (see Figure 6.3 for an illustration of the protocol).
Description of our protocol.

Phase 1. [**Generation of garbled input strings**] Bob generates garbled strings $w_{i,j,b} \in \{0,1\}^k$ for Alice's input wires $i \in I_A$ and for circuit indices $j = 1, \ldots, m$.

Phase 2. [**Committing OT**] Committing OT is run in order to let Alice learn her garbled input values. Bob is the sender with private input $w_{i,j,0}, w_{i,j,1}$ which are generated in Phase 1, and Alice is the receiver with private input $x_i \in \{0,1\}$ for $i \in I_A$. At the end of committing OT Alice receives w_{i,j,x_i} for $j = 1, \ldots, m$, and Bob gets no information about which one is chosen. The common output is $A_{i,j,b} = \langle \text{commit}_A(w_{i,j,b}; \alpha_{i,j,b}) \rangle$ for $i \in I_A$, $j = 1, \ldots, m$ and $b \in \{0,1\}$.

Phase 3.[Construction] Bob does the following:

- He computes the garbled randomized circuits GRC_j for $j = 1, \ldots, m$ in such a way that the garbled strings from Phase 1 are used for Alice's input wires. Note that the $COP_{i,j}$'s are necessary to prevent Bob from cheating so that Alice could determine a correct majority circuit. Indeed, Alice needs to be ensured that the garbled values in $COP_{i,j}$ are in order (i.e., the first value corresponds to 0 and the second value 1) so that she can determine a correct majority circuit.
- He also generates the commitments $B_{i,j,j',b} = \text{commit}_B(w_{i,j,b} \| w_{i,j',b}; \gamma_{i,j,j',b})$ for $i \in I_B \cup I'_B$ and j, j' such that $1 \leq j < j' \leq m, b \in \{0,1\}$.

He sends these circuits and the corresponding commitments to Alice. The commitment sets $B_{i,j,j',0}$ and $B_{i,j,j',1}$ are sent in random order so that Alice cannot link the committed values to 0 or 1.

Phase 4. [Challenge] Alice and Bob run the challenge phase which results in a random bit-string $\ell = \ell_1 \ldots \ell_m$ for $\ell_i \in_R \{0,1\}$ as common output. She asks Bob to open the circuits GRC_j for j such that $\ell_j = 1$ which are called as check-circuits. She also asks Bob to open the corresponding commitments for j such that $\ell_j = 1$.

Phase 5. [Opening & Checking] Bob sends the following:

- The opening sets

  $$\widetilde{GRC}_j = \langle \langle w_{i,j,b} : i \in \mathcal{W}, b \in \{0,1\} \rangle, \langle \alpha_{i,j,b} : i \in I_A \rangle, \langle \beta_{i,j,b} : i \in O_B, b \in \{0,1\} \rangle \rangle$$

  for j such that $\ell_j = 1$. These will open the circuits GRC_j for which $\ell_j = 1$ and the corresponding commitments $A_{i,j,b}$ and $COP_{i,j}$. Note that once GRC_j is opened the corresponding committed ordered pairs $COP_{i,j}$ for $i \in O_B$ and $\ell_j = 1$ are also opened since it is a part of the construction of GRC_j.
- The set $\widetilde{B}_{j,j',b} = \langle \gamma_{i,j,j',b} : i \in I_B \cup I'_B \rangle$ for j, j' such that $\ell_j = \ell_{j'} = 1$, $1 \leq j < j' \leq m, b \in \{0,1\}$ to open the corresponding commitments $B_{i,j,j',b}$.
- The opening set $\widetilde{B}^{input}_{j,j'} = \langle\ w_{i,j,y_i}, w_{i,j',y_i}, \gamma_{i,j,j',y_i}, w_{i',j,z_{i'}}, w_{i',j',z_{i'}}, \gamma_{i',j,j',z_{i'}} : i \in I_B, i' \in I'_B\ \rangle$ for j, j' such that $\ell_j = \ell_{j'} = 0$, $1 \leq j < j' \leq m$. Note that $\widetilde{B}^{input}_{j,j'}$ contains Bob's garbled input values for the evaluation-circuits.

Alice verifies the circuits and the commitments. She can detect wrong values easily. Note that the consistency check of Bob's input is now done by the equality-checker scheme with the commitment sets $\widetilde{B}_{j,j',b}, \widetilde{B}^{input}_{j,j'}$. Committing OT is verified as follows: Bob opens the commitments $\langle\ \text{commit}_B(w_{i,j,b}; \alpha_{i,j,b})\ \rangle$ for $i \in I_A, \ell_j = 1$ and $b \in \{0,1\}$. Alice checks that her garbled input values $w_{i,j,0}, w_{i,j,1}$ used in committing OT are equal to the corresponding circuit garbled input values, and the first garbled input value $w_{i,j,0}$ corresponds to 0 and the second value $w_{i,j,1}$ corresponds to 1. If any of the verification fails Alice aborts the protocol.

Phase 6. [**Evaluation**] Alice evaluates the evaluation-circuits GRC_j for j such that $\ell_j = 0$ using the set $\widetilde{B}^{input}_{j,j'}$ (that contains garbled values of Bob's input wires). She then commits to garbled values of Bob's output wires, denoted by $C_j = \text{commit}^h_A(\langle w_{i,j} : i \in O_B\rangle; \delta_j)$ for j such that $\ell_j = 0$. If Alice cannot compute a circuit, she will commit to some garbage values (not to reveal extra information to Bob). She sends $\langle C_j : j \text{ s.t. } \ell_j = 0\rangle$ to Bob. Note that the commitments C_j are generated to assure Bob that the committed values in C_j are really circuit values.

For efficiency reasons, the committed value inside C_j can be $M_j = \text{Hash}(\langle w_{i,j} : i \in O_B\rangle)$. We note that instead of a full random oracle the property of collision-resistance is sufficient for this case. For example, if we use Pedersen commitments then C_j will be equal to $g^{M_j}h^{\delta_j}$ where g is a generator of group G and h is a random element of G assuming that nobody knows the discrete logarithm of h to the base g [Ped91].

Phase 7. [**Opening of** $COP_{i,j}$] After Bob receives the commitments C_j he opens the committed ordered pairs $COP_{i,j}$ for $i \in O_B$ and $\ell_j = 0$ by sending the set $\widetilde{COP}_{j,b} = \langle\ w_{i,j,b}, \beta_{i,j,b} \text{: } i \in O_B\ \rangle$ for j such that $\ell_j = 0$, $b \in \{0,1\}$.

Phase 8. [**Decision of majority circuit**] Alice determines a majority circuit GRC_r for some r such that $\ell_r = 0$. She then sends the output $\langle b_i : i \in O_B, b_i \in \{0,1\}\rangle$ of GRC_r together with an OR-proof that this output agrees with at least one circuit output. More precisely, the OR-proof is simply generated as follows:

She will prove that one of the C_j's commits to M_j where the relation is

$$R_{\text{output}} = \{(g, h, \langle M_j, C_j : j \text{ s.t. } \ell_j = 0\rangle; \delta) : \exists_{j \text{ s.t. } \ell_j = 0}\ h^\delta = C_j / g^{M_j}\}$$

where both parties can compute $M_j = \text{Hash}(\langle w_{i,j,b_i} : i \in O_B\rangle)$ for j such that $\ell_j = 0$ and $\langle b_i : i \in O_B, b_i \in \{0,1\}\rangle$ is the output of GRC_r. In this way, Alice will ensure that at least one of the circuit output agrees with the output of the majority circuit.

Note that she can determine the majority circuit GRC_r without further interaction with Bob since the additional input values that were used to randomize Bob's output wires are all identical for the same wire in each of the circuits GRC_j where $\ell_j = 0$. She finally matches her outputs with the $OP_{i,r}$ where $i \in O_A$ and learns $f_1(x, y)$.

Bob also computes his output $f_2(x, y)$ by XORing his randomized output wire for the circuit GRC_r with the corresponding additional wires.

A Two-Party Protocol in the Malicious Model

Common Input: f
Compute: $f(x,y) = (f_1(x,y), f_2(x,y))$

Alice	**Bob**
Private Input: $x = \langle x_i \in \{0,1\}, i \in I_A \rangle$	**Private Input:** $y = \langle y_i \in \{0,1\}, i \in I_B \rangle$

Phase 1: Generation of garbled input strings.

Generate $w_{i,j,b} \in_R \{0,1\}^k, i \in I_A, j = 1, \ldots, m, b \in \{0,1\}$.

Phase 2: Committing OT. Run in parallel, for $i \in I_A$.

Receiver	**Sender**
Private Input: x_i	**Private Input:** $\langle w_{i,j,0}, w_{i,j,1} : j = 1, \ldots, m \rangle$
Committing OT subprotocol $\longleftrightarrow$	
Private Output: $\langle w_{i,j,x_i} : j = 1, \ldots, m \rangle$	**Private Output:** $\langle \alpha_{i,j,0}, \alpha_{i,j,1} : j = 1, \ldots, m \rangle$

Common Output:
$A_{i,j,b} = \langle\ \mathrm{commit}_B(w_{i,j,b}; \alpha_{i,j,b})\ \rangle$ for $j = 1, \ldots, m, b \in \{0,1\}$

Phase 3: Construction.

Compute GRC_j for $j = 1, \ldots, m$ s.t. for all $i \in I_A$ $\langle w_{i,j,0}, w_{i,j,1} : j = 1, \ldots, m \rangle$ are used for the corresponding wires in GRC_j.

Compute $B_{i,j,j',b} = \mathrm{commit}_B(w_{i,j,b} \| w_{i,j',b}; \gamma_{i,j,j',b})$ for $i \in I_B \cup I'_B$ and j, j' s.t. $1 \le j < j' \le m, b \in \{0,1\}$.

$\langle GRC_j : j = 1, \ldots, m \rangle,$
$\langle B_{i,j,j',b_{i,j,j'}}, B_{i,j,j',1-b_{i,j,j'}} : i \in I_B, 1 \le j < j' \le m, b_{i,j,j'} \in_R \{0,1\} \rangle$
$\longleftarrow$

Phase 4: Challenge.

Secure coin-flipping subprotocol $\longleftrightarrow$

Common Output: $\ell = \ell_1 || \ldots || \ell_m, \ell_i \in_R \{0,1\}$

Phase 5: Opening & Checking.

$\widetilde{GRC}_j = \langle \langle w_{i,j,b} : i \in \mathcal{W}, b \in \{0,1\} \rangle, \langle \alpha_{i,j,b} : i \in I_A, b \in \{0,1\} \rangle, \langle \beta_{i,j,b} : i \in O_B \rangle \rangle$, for j s.t. $\ell_j = 1$.

$\widetilde{B}_{j,j',b} = \langle \gamma_{i,j,j',b} : i \in I_B \cup I'_B \rangle$ for j, j' s.t. $\ell_j = \ell_{j'} = 1$, $1 \le j < j' \le m, b \in \{0,1\}$.

$\widetilde{B}^{input}_{j,j'} = \langle w_{i,j,y_i}, w_{i,j',y_i}, \gamma_{i,j,j',y_i}, w_{i',j,z_{i'}}, w_{i',j',z_{i'}}, \gamma_{i',j,j',z_{i'}} : i \in I_B, i' \in I'_B \rangle$ for $\ell_j = \ell_{j'} = 0, 1 \le j < j' \le m$.

$\langle \widetilde{GRC}_j : \text{for } j \text{ s.t. } \ell_j = 1 \rangle, \langle \widetilde{B}_{j,j',b} : \text{for } \ell_j = \ell_{j'} = 1, 1 \le j < j' \le m, b \in \{0,1\} \rangle,$
$\langle \widetilde{B}^{input}_{j,j'} : \text{for } j, j' \text{ s.t. } \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m \rangle$
$\longleftarrow$

Check GRC_j for j s.t. $\ell_j = 1$ using $\widetilde{GRC}_j$.
$A_{i,j,b} \stackrel{?}{=} \mathrm{commit}_B(w_{i,j,b}; \alpha_{i,j,b})$ for $i \in I_A$, j s.t. $\ell_j = 1, b \in \{0,1\}$ using $\widetilde{GRC}_j$.
$B_{i,j,j',b} \stackrel{?}{=} \mathrm{commit}_B(w_{i,j,b} \| w_{i,j',b}; \gamma_{i,j,j',b})$ for $i \in I_B \cup I'_B, j, j'$ s.t. $\ell_j = \ell_{j'} = 1, 1 \le j < j' \le m, b \in \{0,1\}$ using $\widetilde{B}_{j,j',b}$.
$B_{i,j,j',y_i} \stackrel{?}{=} \mathrm{commit}_B(w_{i,j,y_i} || w_{i,j',y_i}; \gamma_{i,j,j',y_i})$ for $i \in I_B, \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m$.
$B_{i',j,j',z_{i'}} \stackrel{?}{=} \mathrm{commit}_B(w_{i',j,z_{i'}} || w_{i',j',z_{i'}}; \gamma_{i',j,j',z_{i'}})$ for $i' \in I'_B, \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m$.

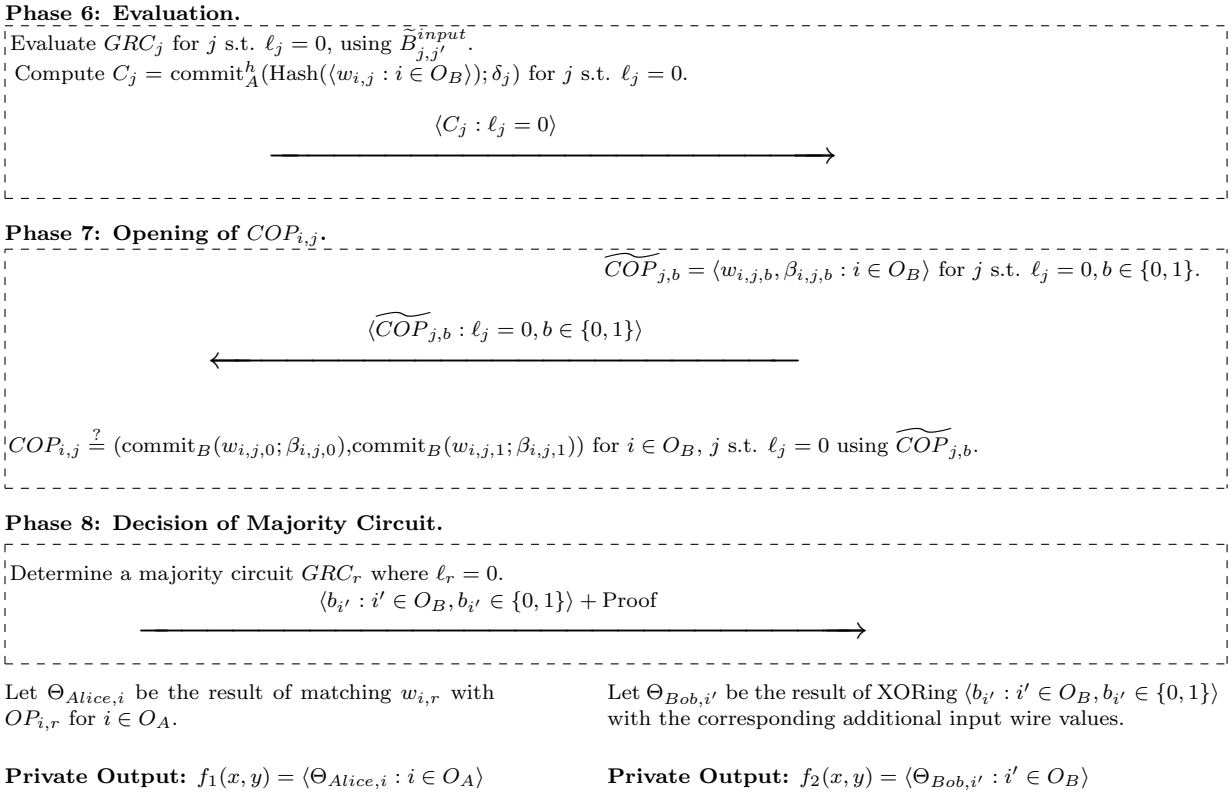

Figure 6.3: A two-party protocol in the malicious model

6.7 Security Analysis

The security analysis of our protocol is based on real/ideal simulation paradigm which is similar to the one in [LP07] (see Section 3.3.3 for a definition of security). Before we show a simulation we remark that the additional modifications of the circuit RC_f presented in Section 5.4 must be applied. We show later that by applying these modifications the failure probability of the simulator in [LP07] can be reduced.

We split the analysis into two cases where either Bob or Alice is corrupted, and we construct a simulator for each of these cases. Together, these two simulators analyze the security of our protocol.

Case 1 - Assume Bob is corrupted.

Let R_B be an adversary corrupting Bob; we construct a simulator S_B as follows. Since we assume that the committing OT protocol is secure, we analyze the security of the protocol in the hybrid model with a trusted party computing the committing OT function (see Section 3.3.4).

The simulator.

1. The simulator S_B chooses a fixed input $x' = 0$ for Alice and uses it only at the beginning of the protocol to be able to start the protocol (namely, to run OT phase) but not to be used later on.

2. S_B invokes R_B and extracts the garbled values that Bob assigns to Alice's inputs from the committing OT protocol. This is possible because we analyze the security of the protocol in the hybrid model with a trusted party computing the committing OT functionality. Namely, Bob just sends his garbled input values to the trusted party, and so the simulator S_B obtains them directly.

3. S_B receives all the garbled circuits and commitments from R_B.

4. S_B then runs the challenge phase to generate the random challenge strings.

5. Now the input of Bob will be extracted as follows. The simulator S_B receives all the required decommitments from R_B based on the challenge strings, including the garbled values that correspond to Bob's input. S_B verifies that all the commitments are correct as Alice would do. If any of the checks fail, S_B sends an abort message to R_B, sends $\perp$ to the trusted party and halts, outputting whatever R_B outputs. If none of the checks fail, S_B obtains $m/2$ input for Bob for $m/2$ circuits (because of the additional modifications described in Section 5.4). If no input value appears more than $m/4$ times, then S_B outputs fail.[2] We show below that fail does not occur with high probability. If the same value appears more than $m/4$ times, S_B sets y to be this value and sends it to the trusted party. The trusted party replies with $f_2(x, y)$ to S_B.

6. The simulator S_B evaluates the evaluation-circuits as in the real protocol and obtains garbled output values. It then computes the commitments $\text{commit}^h_A(\text{Hash}(\langle w_{i,j} : i \in O_B\rangle); \delta_j)$ for j such that $\ell_j = 0$ as Alice does in the real protocol in the evaluation phase and sends then to R_B. If the simulator cannot compute a circuit it will commit some garbage values not to reveal extra information to Bob that this circuit was not computed.

7. The commitments $\text{commit}_B(w_{i,j,b}; \beta_{i,j,b})$ for $b \in \{0, 1\}$ are opened by R_B for the evaluation-circuits as in the real protocol. If the committed values are not correct S_B sends an abort message to R_B, sends $\perp$ to the trusted party and halts, outputting whatever R_B outputs.

8. Since the simulator knows the actual private input of R_B (i.e. knows the additional input wire value) and knows $f_2(x, y)$ it can learn the randomized output bit by computing the exclusive-or for R_B's output wire for circuit RC_f. The simulator then determines a majority circuit GRC_r for some r whose output is $\langle b_{i'} : i' \in O_B, b_{i'} \in$

[2]The majority of inputs are taken to have a correct output based on cut-and-choose technique.

$\{0,1\}\rangle$. The simulator sends $\langle b_{i'} : i' \in O_B, b_{i'} \in \{0,1\}\rangle$ together with an OR-proof for the relation R_{output}.

Analysis. We show that the view of R_B in the simulation with S_B is statistically close to its view in a hybrid execution of the protocol with a trusted party computing the committing OT protocol. (Note that our protocol is not statistically secure since the simulation is in the hybrid model for committing OT functionality, and it depends on the implementation of OT subprotocol and the commitment schemes used.)

We must ensure that if at some point Alice aborts the protocol depending on a cheating behavior by Bob, then Bob does not get any information about Alice's input. This is, however, only possible either at Phase 5 (at the opening & checking stage) or at Phase 7 (at the opening of $COP_{i,j}$) while checking the correctness of the circuits and the commitments. In this case, the decision to abort is based on Bob's construction of the circuits as well as his commitments (including commitments from OT stage), and on the random inputs of the parties, and is independent of Alice's input. Thus, Bob does not get any information if Alice aborts the protocol. We therefore consider the case that Alice does not abort the protocol. We next show that S_B outputs fail with negligible probability. If Alice does not abort then with overwhelming probability the circuits (including the commitments $COP_{i,j}$ and the commitments from committing OT) are correct (by the m/2-out-of-m technique). Next, the equality-checker scheme assures that with high probability a majority of the evaluation-circuits obtain the same input bit, and fail does not occur. The simulator S_B can then decide on a majority circuit and sends its output $\langle b_{i'} : i' \in O_B, b_{i'} \in \{0,1\}\rangle$ together with proof. Since with very high probability there exists a majority circuit, the proof will pass successfully.

We next show that if S_B does not output any fail message, the simulated view of R_B is identical to his view of an execution of the protocol. Actually, they are identical since S_B just runs the same instructions as an honest Alice would do. Since S_B uses independent random coins in the challenge phase and follows Alice's instructions each time, the above process results in a distribution that is identical to the view of R_B in a real execution with Alice.

Case 2 - Assume Alice is corrupted.

The security analysis of the protocol in the case that Alice is corrupted is very similar to the one in [LP07]. In this case, an ideal-model simulator S_A will be constructed working with a real-adversary R_A that has corrupted Alice. Roughly speaking, the simulator S_A first extracts R_A's input bits from the OT protocol (since we analyze the security in the hybrid model), and then externally sends this input x to the trusted party and receives back $f_1(x,y)$. Given the output $f_1(x,y)$, S_A constructs m garbled circuits where some of them correctly compute $f(x,y)$ and the others are false ("fake" ones) that compute the constant function which always outputs $f_1(x,y)$. We note that the authors in [LP04] show that, given the output, S_A can generate the "fake" garbled circuits and R_A cannot distinguish between them and correctly generated circuits. Next, it is necessary to make sure that all

of the check-circuits compute $f(x, y)$ correctly, and all of the evaluation-circuits compute the constant function outputting $f_1(x, y)$ for Alice. In order to make sure that the it is not caught cheating, S_A runs the challenge phase so that all of the correctly generated garbled circuits are check-circuits, and all of the others are evaluation-circuits. The authors in [LP04] show that the challenge phase can be simulated with negligible failure probability. Intuitively, given that the simulation of the challenge phase simulation succeeds, it follows that all of the check-circuits are correctly constructed, as in the protocol. Namely, R_A evaluates only the circuits which always output $f_1(x, y)$. Thus, the view of R_A in the simulation with S_A with respect to these circuits is the same as its view in a real execution with an honest R_B.

6.8 Performance Analysis

We will now give a precise description of the complexity of our protocol and the protocol by Lindell and Pinkas [LP07] and compare them. In order to be able to make a comparison we assume that both protocols permit two private outputs. As said before, our construction is a little bit different from [LP07] in the case that both parties receive their respective private outputs. In our case, once Alice evaluates the circuits we know that she can compute only one garbled output value for every output wire, and Bob accepts it as output if and only if the value is the same as the circuit garbled value. However, in [LP07], the modification in Figure 6.1 is applied.

The protocol by Lindell and Pinkas [LP07].

The protocol of [LP07] needs 2 addition gates and 1 multiplication gate in order to have private outputs for both Alice and Bob (see Figure 6.1). Multiplication needs $O(\ell^2)$ and addition needs $O(\ell)$ number of Boolean gates. So, overall $O(\ell^2)$ additional Boolean gates are necessary in [LP07]. There are also m new input wires for the i-th input wire of Alice. Note that the number of Alice's input wires is now ℓm and the number of Bob's input wires is now 4ℓ. Therefore, the size of the new circuit is $O(|C_f| + \ell^2 + \ell m)$. The number of commitments for Bob's input wires is $O(\ell m^2)$. There are also $O(\ell m^2)$ commitments for Alice's input wires. Hence, the communication overhead is $O(m|C_f| + \ell^2 m + \ell m^2)$ times a security parameter.

For the number of exponentiations, in their protocol, each input bit of Alice is replaced by m new inputs and therefore ℓm OTs are required. The authors describes a randomized construction which reduces the number additional wires to $max(4\ell, 8m)$, and a deterministic construction which reduces to $m\ell/(\log(m) + 1)$.

Our protocol.

By the modification presented in Section 6.5 we need only one extra XOR gate for each of Bob's output wires. Therefore, the number of Bob's input wires is now 2ℓ. The size of

the new circuit is $O(|C_f| + \ell)$. There are $O(\ell m^2)$ commitments to Bob's input values and $O(\ell m)$ commitments to Bob's output values. There are also $O(\ell m)$ commitments to Bob's output values which are computed at the end of the evaluation. Therefore, there are in total $O(\ell m^2)$ commitments. Hence, the communication overhead is $O(m|C_f| + \ell m^2)$ times a security parameter.

For the number of exponentiations, the computational overhead is dominated by the committing OTs (public key operations) and OR-Proofs. For each input wire of Alice in all m circuits a single committing OT is run, and therefore exactly ℓ number of committing OTs are required. And, for each evaluation-circuit there is one OR-proof, therefore exactly $m/2$ number of OR-proofs are required.

	The Protocol in [LP07]	Our Protocol
Symmetric Encryptions	$O(m\|C_f\| + \ell m^2 + \ell^2 m)$	$O(m\|C_f\| + \ell m^2)$
# of OT	$m\ell/(\log(m) + 1)$	ℓ

Table 6.1: Complexity Analysis

6.9 Discussion on the Simulation

Note that in one part of the proof of [LP07], the failure probability of their simulator is quite large in the case that Bob is corrupted; namely $2^{1-m/17}$, where m is the number of circuits used, for a certain type of failure—in addition to other types of failures. However, this requires a large value for m, even for a low level of security. The reason is that the rewinding process is used to be able to extract the input from $m/4$ circuits (the circuits that are checked in the first execution of the rewinding and evaluated in the second execution, see [LP07] for details). We note that Modification 1 (in Figure 5.3) is sufficient to have a better bound of [LP07] (i.e., avoids the rewinding procedure). Namely, by applying Modification 1 we are able to reduce the failure probability to $2^{-m/4}$ which we believe is optimal for this type of cut-and-choose protocols relying on majorities.

Modification 2 (in Figure 5.4) is not necessary for [LP07] since the way our protocol permits two private outputs is different from theirs. In our case, once Alice evaluates the circuits we know that she can compute a single garbled value per output wire. And, Bob accepts it as output if and only if the value is the same as the circuit garbled value. In our security analysis, Modification 2 lets the simulator learn the corresponding garbled value and in this way, we avoid running the rewinding process in [LP07].

As we described before Alice cannot simply send garbled values of a majority circuit, since in this way Bob might learn extra information. The protocol in [LP07] does not have this problem. In our protocol, we added an OR-Proof in Phase 8 to show that the output bits she sent agrees with at least one circuit. In this way, Bob will not be able to learn which circuit has been evaluated.

7

Fair Secure Two-Party Computation

In this chapter, we revisit fairness while borrowing several improvements and ideas from the recent papers. So far the protocol by Pinkas (Eurocrypt 2003) is the only paper which deals with fairness for Yao's garbled circuit approach. We thus improve upon the protocol by presenting a more efficient variant, which includes several modifications including one that fixes a subtle security problem with the computation of the so-called majority circuit. We note that our protocol is a modular extension of the protocol presented in Chapter 6. We analyzed the security of our protocol according to the real/ideal simulation paradigm.

Parts of this chapter are based on [KS06a, KS06b, KS08] (joint work with Berry Schoenmakers).

7.1 Introduction

We have seen from Chapter 6 that there are several protocols in the literature based on garbled circuits covering malicious adversaries. Only a few papers, however, discuss the fundamental property of fairness for secure two-party computation. In fact, the protocol by Pinkas [Pin03] is the only one which deals with fairness for two-party protocols based on Yao's garbled circuit approach.

In general, fairness is achieved with special protocol techniques. To achieve fairness in the protocol by Pinkas [Pin03], he presents an intricate method which involves the modification on the truth tables for the garbled circuits. A crucial part of his protocol is that the evaluation of the garbled circuit, as performed by Alice, does not result in garbled values (one per output wire) but—roughly speaking—in *commitments* to these garbled values. Basically, Alice will hold commitments for Bob's output wires, and, vice versa, Bob will hold commitments for Alice's output wires. The important point here is to show the correctness of these commitments. Once the correctness of the commitments for the output values is guaranteed, both parties will gradually open their commitments.

Note that, if one would use the "gate-by-gate" approach instead of garbled circuits, fairness can be added easily in a modular way (see [GMPY06] for more details). The reason is that the protocols using the "gate-by-gate" approach [CDN01, ST04] already guarantee that the parties learn the correct encrypted (or committed, shared) values to

outputs. Hence, the parties only have to open them gradually instead of at once. On the other hand, suppose that we use a protocol using Yao's garbled circuit approach in the case of malicious adversaries. If we have a protocol without fairness, then we obtain only garbled output values rather than commitments to these values, because those commitments are not needed if we want to release the output values at once (instead of gradually). In the literature, one can see many such protocols that are secure in the case of malicious adversaries, but have no commitments to the garbled output values during the protocol execution (e.g. [MNPS04, FM06, Woo06, LP07]). Therefore, some significant changes must be made in order to let the parties learn the commitments to the correct outputs so that fairness is achieved. Namely, once the commitments are accepted by the parties then fairness follows by gradually opening the commitments.

Concretely, blind signatures are used in [Pin03] as a building block for the verification of these commitments. On a high level, we describe below how blind signatures are used in [Pin03] for a special gate which has two private input and two private output (for Alice and Bob resp.).

Phase 1. [**Generation of commitments**] In the beginning of the protocol, for output wire of Bob Alice generates k commitments to 0/1 bits and blinds them where k is some security parameter. She generates two sets, one consists of commitments to 0 and the other one consists of commitments to 1. She sends these two sets to Bob together with the proof of correctness (a further cut-and-choose subprotocol is used to ensure that Bob only signs correctly formed (blinded) commitments).

Phase 2. [**Blind signatures and construction of garbled circuits**] Bob signs to these blinded commitments and generates m garbled circuits in a usual way except that the garbled output values of Bob will be mapped to the signed commitments together (the number of commitments is $k/2$). Correctness of garbled circuits are proved with the cut-and-choose technique.

Phase 3. [**Evaluation**] Once Alice evaluates the circuits she obtains blind signatures to the commitments, and since she knows the blinding factor she can remove the blinding and obtain the signed commitments.

Phase 4. [**Gradual opening**] By the gradual release, the signed commitments will be opened and Bob will be sure if he sees his signatures to the opened commitments.

We note that in this protocol for a single output wire of Bob (for m garbled circuits) k blind signatures are used, which makes the protocol rather complex and inefficient. In our protocol, we do not use blind signatures, instead, we use the well-known OR-proofs [CDS94], to let Alice show that she correctly committed to the garbled output values that she obtained for each of Bob's output wires.

7.2 Main Protocol Ideas

In this section we briefly review some issues for Pinkas' protocol, and we present the main ideas behind our new protocol. As we said in Section 3.4.2, we will use the protocol for the "commit-prove-fair-open" functionality ("wrapped" version of it) [GMPY06] for gradual release phase in a black-box manner. The only thing we need to take care of before reaching the gradual release phase is that Alice and Bob hold commitments to the correct values for each other's output wires.

7.2.1 Problem with Pinkas' Computation of Majority Circuit for Bob

The computation of majority circuit for Bob can be avoided altogether for a protocol tolerating malicious adversaries but not achieving fairness (see Section 6.2.2). It is not clear, however, whether this protocol can be extended to cover fairness as well. Omitting details, the protocol by Pinkas reaches a state in which for each evaluated garbled circuit GC_j and for each output wire i of Bob, Bob knows a random bit k_{ij} and Alice knows the value $b_{ij} \oplus k_{ij}$, where b_{ij} is the actual output bit for wire i in circuit GC_j. Alice and Bob then use these values to determine a majority circuit GC_r for Bob. Pinkas proposes that Alice can be trusted to construct a garbled circuit for this task, as Alice needs this majority circuit to prevent Bob from cheating. But this way, nothing prevents Alice from constructing an arbitrary circuit which reveals some of Bob's input values and hence some of the $k_{i,j}$ values. Then Alice learns Bob's actual output bits, which should not be possible.

Of course, this problem can be solved by running any two-party protocol which is secure against malicious parties (e.g., [LP07]). However, in our protocol, we will not require any additional protocol for computing a majority circuit for Bob. We present a simple modification to the circuit and in this way we show that a majority circuit can be computed without considerable additional cost.

As we said in Chapter 6 the index of a majority circuit is revealed to Bob in [Pin03], which results in leaking some private information of Alice. Currently, we do not know whether there is a way to fix this issue for [Pin03].

7.2.2 Correctness of Garbled Input Values

Bob is not only providing Alice with m garbled circuits, but also with garbled values for each input wire for each circuit to be evaluated. It must be ensured that these garbled values are *correct* (i.e., correspond to the actual input value and fit with the garbled truth tables of the garbled circuits).

The correctness of Alice's garbled inputs will basically follow by definition of 1-out-of-2 oblivious transfer (OT) as in [Pin03]. However, as pointed out in Chapter 6, one cannot use plain OT for the malicious case; rather a stronger form of OT such as committed OT, or the potentially weaker form of committing OT should be used.

The correctness of Bob's garbled inputs is not as straightforward to handle. Pinkas [Pin03] originally used OR-proofs [CDS94], whereas later papers [FM06, Woo06, LP07]) aimed at using less expensive techniques relying on symmetric operations only (used in combination with cut-and-choose). In our protocol, we use the *equality-checker scheme* of Franklin and Mohassel [FM06] for proving correctness of Bob's inputs (see Section 6.2.3).

7.2.3 Correctness of Committed Outputs

In order that the two parties can safely enter the gradual release phase, one of the main problems that needs to be solved is that both parties are convinced of the correctness of the values contained in the commitments held by the other party. We treat this problem differently for Alice and Bob.

Bob's commitments to Alice's outputs will be guaranteed to be correct by cut-and-choose, exactly as in [Pin03]. For Alice's commitments to Bob's outputs, however, we will use a different approach than in [Pin03], which used blind signatures for this purpose. In our protocol, Alice will first obtain the garbled values for Bob's outputs for all evaluated circuits, and she commits to all of these values. At that point, Bob can safely reveal both garbled values for each output wire (of the randomized circuit, as described above). Additional commitments from Bob will ensure that these garbled values are correct. Finally, Alice proves that she committed to one of these garbled values, from which Bob deduces that Alice indeed committed to correct values for Bob's output wires.

Concretely, we let Alice produce an OR-proof as follows. Suppose Alice committed to a garbled value $w_{i,j}$ for output wire i of Bob in circuit GC_j, and that she received garbled values $w_{i,j,0}$ and $w_{i,j,1}$ from Bob. Using homomorphic commitments, such as Pedersen commitments [Ped91], Alice can prove that either $w_{i,j} = w_{i,j,0}$ or $w_{i,j} = w_{i,j,1}$ without revealing which is the case, applying [CDS94] to the Chaum-Pedersen protocol for proving equality of discrete logarithms [CP93]. We will use the Fiat-Shamir heuristic to make these proofs non-interactive (and provably secure in the random oracle model).

The above application of OR-proofs also critically relies on a slight modification of the circuit representing f, where a new input wire (for Bob) is introduced for every output wire of Bob (see Figure 6.2). Bob will use random values for these new input wires, to blind the values of his output wires. Nevertheless, Alice will still be able to determine a majority circuit.

7.3 A Fair and Secure Two-Party Protocol

Let RC_f denote the randomized boolean circuit for a function f which is a modified version of the circuit C_f, see Figure 6.2. We use I_A resp. O_A to denote the set of Alice's input resp. output wires, and I_B resp. O_B to denote the set of Bob's input resp. output wires. Also, we use I'_B to denote the additional input wires for Bob, used in the construction of RC_f from C_f. Note that $|I_A| = |I_B| = |I'_B| = |O_A| = |O_B| = \ell$. Let $x = \langle x_i \in \{0,1\}, i \in I_A \rangle$ denote Alice's input, $y = \langle y_i \in \{0,1\}, i \in I_B \rangle$ denotes Bob's in-

put, and $z = \langle z_i \in \{0,1\} : i \in I'_B \rangle$ denote Bob's random input to RC_f. Furthermore, let $|RC_f|$ denote the number of gates, and we $\mathcal{W}$ denote the set of all wires in the circuit RC_f.

In Phase 3 of the protocol, Bob will generate m garbled versions of the circuit RC_f, where m is a security parameter. We will denote these garbled circuits by GRC_j, for $j = 1, \ldots, m$. A garbled circuit GRC_j for RC_f is completely determined by the pair of garbled values $(w_{i,j,0}, w_{i,j,1})$ assigned by Bob to each wire $i \in \mathcal{W}$. Here, $w_{i,j,b} \in \{0,1\}^k$ corresponds to bit value $b \in 0, 1$, where k is another security parameter, denoting the length of the garbled values.

For our purposes it suffices to view a garbled circuit (as to be evaluated by Alice) as a concatenation of PEG-4-tuples, one for each (binary) gate in RC_f, and of permuted ordered pairs, one for each output wire of Alice:

$$GRC_j = \langle \langle \text{PEG-4-tuple}_{n,j} : 1 \leq n \leq |RC_f| \rangle, \langle POP_{i,j} : i \in O_A \rangle \rangle.$$

The permuted ordered pairs $POP_{i,j}$ are generated at random by Bob, using the garbled values $w_{i,j,0}$ and $w_{i,j,1}$ assigned to wire $i \in O_A$ in circuit GRC_j: $POP_{i,j} = (w_{i,j,\sigma_{i,j}}, w_{i,j,1-\sigma_{i,j}})$, where $\sigma_{i,j} \in_R \{0,1\}$.

As in the previous chapter, in our protocol we use two types of commitments, namely homomorphic ("asymmetric") commitments, e.g., Pedersen commitments [Ped91], and other ("symmetric") commitments, e.g., constructed from pseudorandom generators [Nao91]. We let $\text{commit}_P(m; r)$ denote a symmetric commitment to a message m using randomness r generated by party P, and we use $\text{commit}^h_P(m; r)$ to denote homomorphic commitments.

The protocol consists of 10 phases (see also Figure 7.1 for a protocol diagram).

Phase 1. [**Generation of garbled input values**] Bob generates garbled values $w_{i,j,b} \in_R \{0,1\}^k$, for $i \in I_A$, $j = 1, \ldots, m$, and $b \in \{0,1\}$.

Phase 2. [**Committing OT**] Committing OT is run in order for Alice to learn her garbled input values. Bob is the sender with private input $w_{i,j,0}, w_{i,j,1}$ for $i \in I_A$ and $j = 1, \ldots, m$ which were generated in Phase 1, and Alice is the receiver with private input $x_i \in \{0,1\}$. At the end of Committing OT Alice receives w_{i,j,x_i} and Bob gets no information about which of his inputs is chosen. Also, the common output is $A^{OT}_{i,j,b} = \langle\ \text{commit}_B(w_{i,j,b}; \alpha_{i,j,b})\ \rangle$ for $i \in I_A$, $j = 1, \ldots, m$, $b \in \{0,1\}$. These commitments will be checked by Alice later on, in order to prevent cheating by Bob; in particular to avoid the protocol issue addressed in Section 6.3.

Phase 3. [**Construction**] In this phase Bob does the following:

- He prepares the garbled circuits GRC_j for $j = 1, \ldots, m$ such that the garbled values $w_{i,j,0}, w_{i,j,1}$ for $i \in I_A$ and $j = 1, \ldots, m$ which are generated in Phase 1 are used for the corresponding wires.
- He also generates the commitments $B_{i,j,j',b}$=$\text{commit}_B(w_{i,j,b}, w_{i,j',b}; \beta_{i,j,j',b})$ for $i \in I_B \cup I'_B$ and j, j' such that $1 \leq j < j' \leq m, b \in \{0,1\}$ for the equality-checker scheme. $B_{i,j,j',b}$'s are committed to Bob's garbled input values and they

are generated to ensure that Bob's input is consistent for all the circuits (see Section 7.2.2).

- He also computes the commitments $C_{i,j}$=commit$_B(\sigma_{i,j};\gamma_{i,j})$ for $i \in O_A$ and $j = 1,\ldots,m$ where $\sigma_{i,j} \in_R \{0,1\}$. These committed values are used to permute Alice's output values and the correctness will be proved by the cut-and-choose technique, by opening half of them in the opening phase.
- Finally in this phase, the commitments $D_{i,j}$=commit$_B(w_{i,j,0}, w_{i,j,1};\delta_{i,j})$ for $i \in O_B$ and $j = 1,\ldots,m$ are computed. The $D_{i,j}$'s are committed to Bob's garbled output values and they are generated so that Alice can determine a correct majority circuit.

He sends the circuits and the commitments generated above to Alice. Each pair of commitments $(B_{i,j,j',0}, B_{i,j,j',1})$ is sent in random order, in order that Alice does not learn Bob's inputs when Bob opens one commitment for each of these pairs later on in the evaluation phase.

Phase 4. [**Challenge**] Alice and Bob run the challenge phase (a coin-tossing protocol) in order to choose a random bit string $\ell = \ell_1 \ldots \ell_m \in_R \{0,1\}^m$ that defines which garbled circuits and which commitments will be opened.

Phase 5. [**Opening & Checking**]Alice asks Bob to open the circuits GRC_j for j such that $\ell_j = 1$ which are called *check-circuits*. Similarly, the circuits GRC_j for j such that $\ell_j = 0$ will be called *evaluation-circuits*. She also asks Bob to open the corresponding commitments for j such that $\ell_j = 1$.
Bob sends the following for opening:

- Bob sends the opening set $\widetilde{GRC}_j = \langle\ w_{i,j,b}\colon\ i \in \mathcal{W}\ \rangle$, for j such that $\ell_j = 1$, $b \in \{0,1\}$ to open the check-circuits.
- He also sends $\widetilde{A}^{OT}_{j,b} = \langle\ \alpha_{i,j,b}\colon\ i \in I_A\ \rangle$ for j such that $\ell_j = 1$, $b \in \{0,1\}$ in order to open the corresponding commitments $A^{OT}_{i,j,b}$.
- He also sends the opening set $\widetilde{B}_{j,j',b} = \langle\ \beta_{i,j,j',b}\colon\ i \in I_B \cup I'_B\ \rangle$ for j, j' such that $\ell_j = \ell_{j'} = 1$, $1 \le j < j' \le m$, $b \in \{0,1\}$ to open the corresponding commitments $B_{i,j,j',b}$.
- The opening set $\widetilde{C}_j = \langle\ \sigma_{i,j},\gamma_{i,j}\colon\ i \in O_A\ \rangle$ for j such that $\ell_j = 1$ is also sent in this phase to open the corresponding commitments $C_{i,j}$.
- The opening set $\widetilde{D}_j = \langle\ \delta_{i,j}\colon\ i \in O_B\ \rangle$ for j such that $\ell_j = 1$ is also sent to open the corresponding commitments $D_{i,j}$.
- The opening set $\widetilde{B}^{input}_{j,j'} = \langle\ w_{i,j,y_i}, w_{i,j',y_i}, \beta_{i,j,j',y_i}, w_{i',j,z_{i'}}, w_{i',j',z_{i'}}, \beta_{i',j,j',z_{i'}}\colon\ i \in I_B, i' \in I'_B\ \rangle$ for j, j' such that $\ell_j = \ell_{j'} = 0$, $1 \le j < j' \le m$. This set contains the garbled values of Bob's input wires for the evaluation-circuits, and sent to Alice which is a part of the equality-checker scheme.

Alice verifies the circuits and the commitments. Note that the consistency check of Bob's input is done now by the equality-checker scheme with the commitment set $\widetilde{GRC}_j$ (contains all garbled values) for j such that $\ell_j = 1$ and $b \in 0, 1$ and the set $\widetilde{B}_{j,j',b}$ for j, j' such that $\ell_j = \ell_{j'} = 1, 1 \leq j < j' \leq m$ and $b \in \{0, 1\}$. Note that the opening sets $\widetilde{A}^{OT}_{j,b}$, $\widetilde{B}_{j,j',b}$ and $\widetilde{D}_j$ contain only randomness since the corresponding garbled values comes already from the set $\widetilde{GRC}_j$. If any of the verifications fail Alice aborts the protocol.

Phase 6. [**Evaluation**] Alice does the following in the evaluation phase:

- She first evaluates the circuits GRC_j for $\ell_j = 0$ and computes garbled output values.
- She then commits to Bob's output values as $E_{i,j} = \text{commit}^h_A(w_{i,j}; \zeta_{i,j})$ for $i \in O_B$ and j such that $\ell_j = 0$ and sends them to Bob. If Alice cannot compute a circuit she will commit some garbage values not to reveal extra information to Bob that this circuit was not computed.

  Note that the commitments $E_{i,j}$ are generated to assure Bob that the committed values in $E_{i,j}$ are circuit values. If, for example, Alice commits to values different from garbled output values then she will be detected in OR-proofs in Phase 9. The crucial property we need here is that these commitments are homomorphic in order to be able to use in OR-proofs.

  For example, if we use Pedersen commitments then $E_{i,j}$ will be equal to $g^{w_{i,j}} h^{\zeta_{i,j}}$ where g is a generator of group G and h is a random element of G assuming that nobody knows the discrete logarithm of h to the base g [Ped91].

Phase 7. [**Opening of Bob's ordered output**] After Bob receives the commitments $E_{i,j}$ for $i \in O_B$ and j such that $\ell_j = 0$ he opens the commitments $D_{i,j}$ by sending the opening set $\widetilde{D_j} = \langle\, w_{i,j,0}, w_{i,j,1}, \delta_{i,j} \colon\ i \in O_B \,\rangle$ for j such that $\ell_j = 0$. Note that the commitments $D_{i,j}$ can be opened since Bob's outputs are randomized (see Figure 6.2); hence Alice can only see which outputs match (and determine a majority circuit), but she does not learn Bob's output $f_2(x, y)$.

Phase 8. [**Decision of majority circuit**] Alice determines a majority circuit GRC_r for some r such that $\ell_r = 0$. Note that she can determine a correct majority circuit GRC_r without further interaction with Bob since the additional input values that were used to randomize Bob's output wires are all identical for the same wire in each circuit GRC_j for j such that $\ell_j = 0$. Let $\langle b_i : i \in O_B, b_i \in \{0, 1\}\rangle$ be the output of Bob for the circuit GRC_r. Alice commits to each b_i as $F_i = \text{commit}^h_A(b_i; \epsilon_i) = g^{b_i} h^{\epsilon_i}$ and sends $\langle F_i : i \in O_B\rangle$ to Bob.

Phase 9. [**Verification of Alice's commitments**] They run an OR-proof where Alice is the prover, Bob is the verifier. Alice proves that the committed value inside F_i is correct. More precisely, the OR-proof is generated as follows:

She will prove that the committed values inside F_i agree with at least the output of a circuit where the relation is

$$R_{\text{output}} = \{(g, h, \langle w_{i,j,0}, w_{i,j,1}, E_{i,j}, F_i : i \in O_B \text{ and } j \text{ s.t. } \ell_j = 0\rangle; \epsilon, \zeta) : \\ \exists_{b \in \{0,1\} \text{ and } j \text{ s.t. } \ell_j = 0} \text{ s.t. } \forall i \in O_B \;\; h^{\epsilon} = F_i / g^b, \; h^{\zeta} = E_{i,j} / g^{w_{i,j,b}}\}.$$

In this way, Alice will ensure that at least one of the circuit output agrees with the output of the majority circuit.

Phase 10. [**Gradual release**] They then run the protocol for the gradual release to open their respective commitments, namely, $C_{i,j}$'s and F_i's. At the end of the gradual release:

- Alice learns all $\sigma_{i,j}$ for $i \in O_A$ and $j \in U$ and applies it to $POP_{i,j}$ to learn her actual outputs for j circuits. She takes the majority of j circuits which will result in $f_1(x, y)$.
- Bob learns the bit values for randomized output wires. He finally computes his output $f_2(x, y)$ by XORing his randomized output wire values for the circuit GRC_r with the corresponding additional wire values.

A Fair and Secure Two-Party Protocol

Common Input: f
Compute: $f(x,y) = (f_1(x,y), f_2(x,y))$

Alice	**Bob**
Private Input: $x = \langle x_i \in \{0,1\}, i \in I_A \rangle$	**Private Input:** $y = \langle y_i \in \{0,1\}, i \in I_B \rangle$

Phase 1: Generation of garbled input values.

Generate $w_{i,j,b} \in_R \{0,1\}^k, i \in I_A, j = 1,\ldots,m, b \in \{0,1\}$.

Phase 2: Committing OT Run in parallel, for $i \in I_A$.

Receiver		**Sender**
Private Input: x_i		**Private Input:** $\langle w_{i,j,0}, w_{i,j,1} : j = 1,\ldots,m \rangle$
	Committing OT subprotocol $\longleftrightarrow$	
Private Output: $\langle w_{i,j,x_i} : j = 1,\ldots,m \rangle$		**Private Output:** $\langle \alpha_{i,j,0}, \alpha_{i,j,1} : j = 1,\ldots,m \rangle$

Common Output:
$A_{i,j,b} = \langle\ \mathrm{commit}_B(w_{i,j,b}; \alpha_{i,j,b})\ \rangle$ for $j = 1,\ldots,m, b \in \{0,1\}$

Phase 3: Construction.

Compute GRC_j for $j = 1,\ldots,m$ s.t. for all $i \in I_A$ $\langle w_{i,j,0}, w_{i,j,1} : j = 1,\ldots,m \rangle$ are used for the corresponding wires in GRC_j.

Compute $B_{i,j,j',b} = \mathrm{commit}_B(w_{i,j,b}, w_{i,j',b}; \beta_{i,j,j',b})$ for $i \in I_B \cup I'_B$ and j, j' s.t. $1 \le j < j' \le m, b \in \{0,1\}$.

Compute $C_{i,j} = \mathrm{commit}_B(\sigma_{i,j}; \gamma_{i,j})$ for $i \in O_A$ and $j = 1,\ldots,m$ where $\sigma_{i,j} \in_R \{0,1\}$.

Compute $D_{i,j} = \mathrm{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$ and $j = 1,\ldots,m$.

$\langle GRC_j : j = 1,\ldots,m \rangle, \langle C_{i,j} : i \in O_A, j = 1,\ldots,m \rangle, \langle D_{i,j} : i \in O_B, j = 1,\ldots,m \rangle,$
$\langle B_{i,j,j',b_{i,j,j'}}, B_{i,j,j',1-b_{i,j,j'}} : i \in I_B \cup I'_B, 1 \le j < j' \le m, b_{i,j,j'} \in_R \{0,1\} \rangle$

$\longleftarrow$

Phase 4: Challenge.

Secure coin-flipping subprotocol $\longleftrightarrow$

Common Output: $\ell = \ell_1 || \ell_2 || \ldots || \ell_m$ s.t. $\ell_i \in_R \{0,1\}$

Phase 5: Opening & Checking.

$\widetilde{GRC}_j = \langle w_{i,j,b} : i \in \mathcal{W}, b \in \{0,1\}\rangle$, for j s.t. $\ell_j = 1$,
$\widetilde{A}^{OT}_{j,b} = \langle \alpha_{i,j,b} : i \in I_A\rangle$, for j s.t. $\ell_j = 1, b \in \{0,1\}$
$\widetilde{B}_{j,j',b} = \langle \beta_{i,j,j',b} : i \in I_B \cup I'_B\rangle$ for j, j' s.t. $\ell_j = \ell_{j'} = 1$,
$1 \le j < j' \le m, b \in \{0,1\}$
$\widetilde{C}_j = \langle \sigma_{i,j}, \gamma_{i,j} : i \in O_A\rangle$ for j s.t. $\ell_j = 1$
$\widetilde{D}_j = \langle \delta_{i,j} : i \in O_B\rangle$ for j s.t. $\ell_j = 1$
$\widetilde{B}^{input}_{j,j'} = \langle w_{i,j,y_i}, w_{i,j',y_i}, \beta_{i,j,j',y_i}, w_{i',j,z_{i'}}, w_{i',j',z_{i'}},$
$\beta_{i',j,j',z_{i'}} : i \in I_B, i' \in I'_B\rangle$ for $\ell_j = \ell_{j'} = 0, 1 \le j < j' \le m$.

$\langle \widetilde{GRC}_j : \ell_j = 1, b \in \{0,1\}\rangle, \langle \widetilde{B}_{j,j',b} : \ell_j = \ell_{j'} = 1, 1 \le j < j' \le m, b \in \{0,1\}\rangle,$
$\langle \widetilde{C}_j : \ell_j = 1\rangle, \langle \widetilde{D}_j : \ell_j = 1\rangle, \langle \widetilde{B}^{input}_{j,j'} : \ell_j = \ell_{j'} = 0, 1 \le j < j' \le m\rangle$

$\longleftarrow$

Check GRC_j for j s.t. $\ell_j = 1$ using $\widetilde{GRC}_j$
$A^{OT}_{i,j,b} \stackrel{?}{=} \text{commit}_B(w_{i,j,b}; \alpha_{i,j,b})$ for $i \in I_A$, j s.t. $\ell_j = 1, b \in \{0,1\}$
$B_{i,j,j',b} \stackrel{?}{=} \text{commit}_B(w_{i,j,b}, w_{i,j',b}; \beta_{i,j,j',b})$ for $i \in I_B \cup I'_B$ s.t. $\ell_j = \ell_{j'} = 1, 1 \le j < j' \le m, b \in \{0,1\}$
$C_{i,j} \stackrel{?}{=} \text{commit}_B(\sigma_{i,j}; \gamma_{i,j})$ for $i \in O_A$, j s.t. $\ell_j = 1$
$D_{i,j} \stackrel{?}{=} \text{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$, j s.t. $\ell_j = 1$
$B_{i,j,j',y_i} \stackrel{?}{=} \text{commit}_B(w_{i,j,y_i}, w_{i,j',y_i}; \beta_{i,j,j',y_i})$ for $i \in I_B, j, j'$ s.t. $\ell_j = \ell_{j'} = 0, 1 \le j < j' \le m$
$B_{i',j,j',z_{i'}} \stackrel{?}{=} \text{commit}_B(w_{i',j,z_{i'}}, w_{i',j',z_{i'}}; \beta_{i',j,j',z_{i'}})$ for $i' \in I'_B, j, j'$ s.t. $\ell_j = \ell_{j'} = 0, 1 \le j < j' \le m$.

Phase 6: Evaluation.

Evaluate GRC_j for j s.t. $\ell_j = 0$, using $\widetilde{B}^{input}_{j,j'}$
Compute $E_{i,j} = \text{commit}^h_P(w_{i,j}; \zeta_{i,j})$ for $i \in O_B$ and j s.t. $\ell_j = 0$

$\langle E_{i,j} : i \in O_B, j \text{ s.t. } \ell_j = 0\rangle$

$\longrightarrow$

Phase 7: Opening of $D_{i,j}$.

$\widetilde{D}_j = \langle w_{i,j,0}, w_{i,j,1}, \delta_{i,j} : i \in O_B\rangle$ for j s.t. $\ell_j = 0$

$\langle \widetilde{D}_j : j \text{ s.t. } \ell_j = 0\rangle$

$\longleftarrow$

$D_{i,j} \stackrel{?}{=} \text{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$, j s.t. $\ell_j = 0$

Phase 8: Decision of majority circuit.

Determine a majority circuit GRC_r for some r s.t. $\ell_r = 0$.
Compute $F_i = \text{commit}^h_A(b_i; \epsilon_i)$ for $i \in O_B$ where
$b_i \in \{0,1\}$ is the bit value of the i-th output wire of GRC_r.

$\langle F_i : i \in O_B\rangle$

$\longrightarrow$

Phase 9: Verification of Alice's commitments. Run in parallel for $i \in O_B$.

Prover		**Verifier**
Private Input: $\epsilon_i, \zeta_{i,j}$	**Common Input:** $g, h, \langle w_{i,j,0}, w_{i,j,1}, E_{i,j}, F_i :$ $i \in O_B$ and j s.t. $\ell_j = 0\rangle$	**Private Input:** $\perp$
	OR-Proofs subprotocol $\longleftrightarrow$	
Private Output: $\perp$	**Common Output:** Proof of validity	**Private Output:** $\perp$

Phase 10: Gradual Release. Run in parallel, for $i \in O_A, i' \in O_B, \ell_j = 0$.

Private Input: $b_{i'}, \epsilon_{i'}$	**Common Input:** $C_{i,j}, F_{i'}$	**Private Input:** $\sigma_{i,j}, \gamma_{i,j}$
	Gradual Release subprotocol $\longleftrightarrow$	
Private Output: $\sigma_{i,j}$		**Private Output:** $b_{i'}$

Apply $\sigma_{i,j}$ to $POP_{i,j}$ for $i \in O_A$ and match $w_{i,r}$ with $(w_{i,j,0}, w_{i,j,1})$ for $i \in O_A$ and determine a majority circuit for Alice.

Private Output: $f_1(x, y)$

Compute XOR the randomized output bits with the corresponding additional input wire values.

Private Output: $f_2(x, y)$

Figure 7.1: A fair and secure two-party protocol

7.4 Security Analysis

In our security analysis we want to take advantage of the frameworks established by [LP07] and [GMPY06] for the real/ideal simulation paradigm, resp., for the malicious case in secure two-party computation (and Yao's protocol in particular) and for the case of fair protocols (see Section 3.4.2 for more discussion.). To do so we will actually focus on analyzing a variant of our protocol, in which Phase 10 is replaced by Phase 10′:
Phase 10′**. [Trivial opening]** Alice opens the commitments F_i for $i \in O_B$ and Bob opens the commitments $C_{i,j}$ for $i \in O_A$ and j such that $\ell_j = 0$.

This adapted protocol is not fair, but it withstand malicious adversaries. We will argue so by showing how to simulate it, following [LP07]. From this we conclude that the commitments upon entering Phase 10 in the protocol are correct, as a consequence of which the framework of [GMPY06] applies and the simulatability of our protocol follows. Before we analyze the security of our protocol we note that the additional modifications presented in Section 5.4 are applied to the circuit RC_f.

We are now ready to simulate the protocol (the one with the trivial opening) assuming that either Bob or Alice is corrupted.

Case 1- Assume Bob is corrupted.

Let R_B be an adversary corrupting Bob; we construct a simulator S_B as follows. Since we assume that the committing OT protocol is secure, we analyze the security of the protocol in the hybrid model with a trusted party computing the committing OT functionality (see Section 3.3.4).

The simulator.

1. The simulator S_B chooses a fixed input $x' = 0$ for Alice and uses it only in the beginning of the protocol (namely, to run the OT phase) but it is not used later on.

2. S_B invokes R_B and obtains the garbled input values $w_{i,j,0}$ and $w_{i,j,1}$ for $i \in I_A$ and $j \in \{1, \ldots, m\}$ which are R_B's inputs from the committing OT protocol (in the hybrid model).

3. S_B receives all of the garbled circuits and the commitments from R_B.

4. S_B then runs the challenge phase to generate the random challenge values.

5. Now the input of R_B will be extracted as follows. The simulator S_B receives all of the required decommitments from R_B based on the challenge values, including the garbled values that correspond to Bob's input. Let $w_{i,j}$ be Bob's garbled input value for $i \in I_B \cup I'_B$ and j such that $\ell_j = 0$. S_B verifies that all the commitments are correct as Alice would do in Phase 5. If any of the checks fail, S_B sends an abort message to R_B, sends $\perp$ to the trusted party and halts, outputting whatever R_B outputs. If none of the checks fail, S_B obtains $m/2$ input for Bob for $m/2$ circuits because of Modification 1. More precisely, the simulator knows $w_{i,j,0}$, $w_{i,j,1}$ for $i \in I_A$ and $w_{i,j}$ for $i \in I_B \cup I'_B$ for j such that $\ell_j = 0$, and by Modification 1 the simulator can learn the input bit of Bob for each $w_{i,j}$ for $i \in I_B \cup I'_B$ for j such that $\ell_j = 0$. (In the real case, this does not happen since Alice learns only one garbled input value from OT for each her input wire.) If no input value appears more than $m/4$ times, then S_B outputs fail [1]. We show below that fail also does not occur with high probability. Otherwise, S_B sets y to be the value that appears more than $m/4$ times and sends it to the trusted party. Trusted party replies with $f_2(x, y)$ to S_B.

6. Now the simulator knows $f_2(x, y)$ but it has to convert this value into the corresponding garbled values. The simulator S_B first computes the evaluation-circuits as in the real protocol and obtains one garbled output value per wire. The complementary values will appear as well which are in general not the correct ones since the simulator computes the garbled circuit in the case that $x' = 0$. However, Modification 2 has been applied in order to learn both garbled output values of Bob, and the corresponding bits. As we described above, the simulator learns the output bit of $w_{i,j}$ for $i \in I_B \cup I'_B$ and j such that $\ell_j = 0$ from the AND gate in Figure 5.4 (for the wire W_B). This bit value is the same as the bit value for the wire W'_B in Figure 5.4. Then, by decrypting the XOR gates the simulator learns both garbled values, and their corresponding bits. In the real case, this does not happen since Alice learns only one garbled input value from OT for each her input wire. Hence, since the simulator knows the private output of the corrupted party and corresponding garbled output values it then computes the commitments $\text{commit}_A(w_{i,j}; \zeta_{i,j})$ for $i \in O_B$ and j such that $\ell_j = 0$ as Alice does in the real protocol in Phase 6 and sends to R_B. If the simulator cannot compute a circuit it will commit some garbage values not to reveal extra information to Bob that this circuit was not computed.

7. The commitments $\text{commit}_B(w_{i,j,0}, w_{i,j,1}; \delta_{i,j})$ for $i \in O_B$ and j such that $\ell_j = 0$ are opened by R_B for the evaluation-circuits as in the real protocol.

8. The simulator then can determine the majority circuit whose output is $\langle b_{i'} : i' \in O_B, b_{i'} \in \{0, 1\}\rangle$ since it knows the garbled output values and the corresponding bits

[1] The majority of inputs are computed in order to have a correct output by the cut-and-choose technique.

as in the real protocol.

9. The simulator generates the commitments to $\langle b_i : i \in O_B, b_i \in \{0,1\}\rangle$ as $F_i = \text{commit}_A^h(b_i; \epsilon_i) = g^{b_i} h^{\epsilon_i}$ for $i \in O_B, b_i \in \{0,1\}$ and produces the proof for the relation R_{output}.

10′. Alice opens the commitments F_i for $i \in O_B$ and Bob opens the commitments $C_{i,j}$ for $i \in O_A$ and j such that $\ell_j = 0$.

Analysis. We claim that the view of R_B in the simulation with S_B is statistically close to its view in a hybrid execution of the protocol with a trusted party computing the committing OT protocol. (Note that our protocol is not statistically secure since the simulation is in the hybrid model for committing OT functionality, and it depends on the implementation of OT subprotocol, the commitment schemes and the OR-proofs used).

Initially, we show that if Alice aborts the protocol depending a cheating behavior by Bob, then Bob does not get any information about Alices input. This is only possible either at Phase 5 (at the opening & checking phase) or at Phase 7 (at the opening of $D_{i,j}$'s) while checking the correctness of the circuits and the commitments. In this case, the decision to abort is based on Bob's construction of the circuits as well as commitments (including commitments from the OT phase), and on the random inputs of the parties, and is independent of Alice's input. Thus, Bob does not get any information if Alice aborts the protocol. Thus, we know that the difference between Alice receives "abort" in an ideal execution with S_B and that Alice outputs "abort" in a real execution with R_B is negligible. From here on, we therefore consider the case that Alice does not abort the protocol.

We now prove that the circuits and the commitments are correct with overwhelming probability. We here note that the additional modifications does not compromise the security of the garbled circuit since, by definition of garbled circuit, having one garbled value for each input wire for a gate results in always one garbled output value, which ensures privacy. If Alice does not abort then with probability $2^{-m/4}$ at most $m/4$ of the circuits are bad (including the commitments). Also, we know that the equality-checker scheme [Woo06] assures with high probability a majority of the evaluation-circuits obtain the same input and OT assures with high probability that the values received from OT are garbled values, and therefore fail does not occur with negligibly probability. The simulator S_B can then decide on a majority circuit, prove that the commitments $E_{i,j}$'s are committed to the garbled values of Bob's output wires and open the commitments F_i's for his i-th output wires.

We next show that if S_B does not output any fail message, the simulated view of R_B is identically distributed to its view in an execution of the protocol. Actually, they are identical since S_B just runs the honest Alice's instructions when interacting with the corrupted Bob. Since S_B uses independent random coins in the challenge phase and follows Alice's instructions each time, the above process results in a distribution that is identical to the view of R_B in a real execution with Alice. As we mentioned before the protocol is not statistically secure since the simulation is considered in the hybrid model for committing

OT functionality, and it depends on the implementation of OT subprotocol and the OR-proofs.

Case 2- Assume Alice is corrupted.

The security analysis when Alice is corrupted is very similar to the proof of [LP07]. During the protocol Alice sees the circuits and the commitments and they run a secure committing OT where she gets only the garbled values corresponding to her input bits. On a high level, in the simulation, the simulator first extracts the input of Alice from OT functionality in the hybrid model and then sends the input x to the trusted party and learn the output value. Given the output, the simulator constructs the garbled circuits. The simulator constructs the garbled circuits where some of them correctly computes $f(x, y)$ and some of them compute a constant function which always outputs Alice's real output. Namely, the output of this garbled circuit is always equal to the value which is received from the trusted party. The simulator then chooses the challenge value in such a way that all the check-circuits correctly compute the function $f(x, y)$ while all the other circuits (representation of constant function) are going to be evaluation-circuits which compute the constant function. We refer to [LP07] for details.

7.5 Performance Analysis

We analyze the overall communication and computational complexity of our protocol, and compare with Pinkas' protocol by ignoring the constructions that are used in both protocols. We assume that Pinkas' protocol also uses the equality-checker for consistency of Bob's input. We also assume that Pinkas' protocol uses committing OT to fix the protocol issue described in Section 6.3. As we said before, [Pin03] also has problem by revealing the majority circuit to Bob and we do not know whether there is way to fix. Note that by the modification presented in Figure 6.2 we need ℓ additional XOR gate for each output wire of Bob which has only negligible affect to the overall complexity.

As we said before, the problem of Pinkas' protocol with majority circuit computation can be fixed by running any two-party protocol considering malicious adversaries. For example, if the protocol in [LP07] is used then the communication complexity of majority circuit computation is $O(\ell m^2 \log m)$. We note that there is no need to use such a protocol in our case.

We next consider the parts related to fairness. Note that the way we generate Bob's commitments to Alice's outputs is the same as in Pinkas' protocol (namely, there are $O(\ell m)$ commitments to permutations $\sigma_{i,j}$'s). However, for Alice's commitments to Bob's outputs is much different: Pinkas' protocol has $O(\ell m \kappa)$ commitments which are generated by Alice in order to be blindly signed by Bob, where κ is another security parameter (which are actually timed-commitments for the gradual opening). In our protocol, there are $O(\ell m)$ homomorphic commitments, and $O(\ell m)$ OR-proofs. In the gradual phase, we use the protocol of [GMPY06] in order to ensure fairness, namely new commitments (together

with the proof of correctness) will be generated before applying gradual opening.

The major difference between our approach and the construction by Pinkas [Pin03] is in the removal of the blind signatures and of the majority circuit computation. This leads to an improvement by a factor of κ for the computational complexity. The reason is that for every output wire of Bob 2κ blind signatures are needed in Pinkas' protocol while in ours only one proof of knowledge is needed together with a simple modification to the circuit.

8 Conclusions

In this thesis, we have presented several new protocols for general secure two-party computation. The first protocol, which is in Chapter 4, is designed to implement the Committed Oblivious Transfer (OT) functionality. This protocol has a major advantage over all existing protocols because of the following reasons: firstly, all existing protocols could transfer single bits only whereas our Committed OT protocol allows transferring arbitrary bit strings. Secondly, it is very efficient since it requires less back-and-forth communication among the parties. Also, it has (fixed) constant number of rounds. In fact, the efficiency of our protocol is comparable with the most efficient one in the literature that transfers only bits (even performing slightly better). We showed that the protocols implementing Committed OT functionality may solve subtle problems that occur in the malicious model within two-party protocols that use (plain) OT. Therefore, it gives a better result if Committed OT allows transferring bit strings. We also introduced a new variant called *Committing OT* which is a combination of OT and commitments. It is a weaker version of Committed OT but as we showed in Section 6.4.2, it can easily be exploited to yield more efficient constructions.

The protocols in Chapter 5, 6 and 7 are mainly based on garbled circuits and we consider both semi-honest and malicious adversaries. Before presenting these protocols we provided a clear insight into Yao's garbled circuit in Chapter 5. In Chapter 5, we presented a two-party protocol in the semi-honest model first which is a slightly modified version of Yao's original protocol. We then transformed this two-party protocol into another protocol in Chapter 6 to achieve security in the malicious model by forcing each party to ensure that he or she behaves as a semi-honest party. Before proposing our protocol for the malicious case, we have addressed a protocol issue in the malicious model with the use of OT. We showed that this issue arises the protocols of [Pin03, MNPS04, FM06], where it is possible for the malicious party (i.e., the constructor) to learn some information on the input of the honest party (i.e., the evaluator). We proposed our Committed OT protocol as a quick solution to solve this issue. We have also solved this issue using a Committing OT protocol that gives more efficient result.

We have examined the fairness property of secure two-party computation in Chapter 7, since we believe that fairness (apart from correctness and privacy) is an important security property for secure two-party computation problems. Therefore, we developed a new efficient and fair two-party protocol using garbled circuits. Only Pinkas [Pin03] has used

garbled circuits in the literature to deal with fairness. We pointed out a subtle problem regarding the fairness in his protocol (for the case where the evaluator is corrupted). Namely, we showed that a malicious evaluator may learn the constructor's private input. We have also repaired this problem in our protocol.

The security of our protocols based on garbled circuits was analyzed according to the real/ideal simulation paradigm, which is quite similar to [LP07]. We have also presented a simple modification to the circuit (computing the function f), which leads to an improvement in the failure probability of the simulator in [LP07].

Current focus in secure two-party computation is to implement the protocols described in the literature as well as to bring them into practice. As discussed in Chapter 3, there are two main approaches to solve the secure two-party computation problems. These are "garbled" circuit approach and the "gate-by-gate" approach. The computational cost of the former approach depends on the size of the boolean circuit (that computes f). On the other hand, the cost of the "gate-by-gate" approach is proportional to the depth of the arithmetic circuit (that implements f). In other words, Yao's approach is not expensive for large circuits (many gates with relatively few inputs) whereas the gate-by-gate approach is cheaper for small circuits (few gates with relatively many inputs). Therefore, there is a tradeoff between the two schemes.

We like to conclude by highlighting three topics for further research and then handle these one by one, where the last one is actually an open problem. To begin with, of course it remains challenging to improve the efficiency of the protocols presented here.

Firstly, Committed OT is a strong primitive that needs to be understood in detail. As we said before, we used Committed OT of bit strings in Chapter 6 to fix the protocol issue with the use of OT. Future research may be directed to finding applications that use Committed OT of bit strings. Another area to work on would be finding more protocols in which similar protocol issues occur and see whether our protocol for Committed OT is a good candidate to fix them.

Secondly, in the security proofs of existing protocols [LP04, FM06, LP07], and also in our security analyses, the execution is assumed to be performed in isolation (i.e., in the stand-alone model). Today's attempts consider security of execution in an arbitrary environment (i.e., within the UC framework of Canetti [Can01]). Therefore, it will be worthwhile to work on providing formal proofs of security in the UC framework.

Finally, we conclude with a technical aspect of the majority circuit computation. Initially, please note that the protocol in Chapter 7 deals with fairness (also the protocol by Pinkas [Pin03]) and evaluates different majority circuits for both parties Alice and Bob. The majority circuit for Bob is computed before the gradual release phase, and the majority circuit for Alice is computed after the gradual release phase. Therefore, during the gradual opening Alice opens commitments for only one circuit (i.e., majority circuit), whereas Bob opens commitments for *many* circuits. It would be more efficient if there is a way to determine a unique and correct majority circuit before the gradual phase. In that case, the cost of gradual opening would be much cheaper. In other words, Bob does not need to open commitments for many circuits, instead he would open the commitments for only that majority circuit.

Glossary

Notation	Meaning
$\mathbb{N}$	set of all integers
$\{0,1\}^*$	set of all finite length strings, including the empty string
$\{0,1\}^k$	set of all strings of length k
$s \in_R S$	element s is chosen uniformly at random from set S
$\Delta(X,Y)$	statistical distance between random variables X and Y
W_P	a wire of the party P
C_f	a circuit for a function f
GC_f	a garbled circuit for a function f
GRC_f	a garbled randomized circuit for a function f
$a\|\|b$	concatenation of a and b
$w_{i,b}$	a garbled value corresponds to the bit b for the i-th wire
$w_{P,i,b}$	a garbled value corresponds to the bit b for the i-th wire of the party P
$w_{P,i,j,b}$	a garbled value corresponds to the bit b for the i-th wire of the party P in the j-th garbled circuit
Π	a two-party protocol
Π_f	a two-party protocol for a functionality f
$\text{commit}_P(m;r)$	a commitment scheme to a message m using a random number r generated by the party P

Bibliography

[AL07] Y. Aumann and Y. Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *Theory of Cryptography Conference – TCC 2007*, volume 4392 of *Lecture Notes in Computer Science*, pages 137–156. Springer-Verlag, 2007.

[BMR90] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols. In *STOC '90: Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pages 503–513. ACM Press, 1990.

[BM90] M. Bellare and S. Micali. Non-interactive oblivious transfer and applications. In *Advances in Cryptology–Crypto 1989*, volume 435 of *Lecture Notes in Computer Science*, pages 547–557. Springer-Verlag, 1990.

[BN00] D. Boneh and M. Naor. Timed commitments. In *Advances in Cryptology–Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 236–254. Springer-Verlag, 2000.

[CC00] C. Cachin and J. Camenisch. Optimistic fair secure computation. In *Advances in Cryptology–Crypto 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 93–111. Springer-Verlag, 2000.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *IEEE Symposium on Foundations of Computer Science*, pages 136–145, 2001.

[CDG87] D. Chaum, I. Damgård, and J. van de Graaf. Multiparty computations ensuring the privacy of each partys input and correctness of the result. In *Advances in Cryptology–Crypto 1987*, Lecture Notes in Computer Science, pages 87–119. Springer-Verlag, 1987.

[CP93] D. Chaum and T. P. Pedersen. Wallet databases with observers. In *Advances in Cryptology–Crypto 1992*, volume 740 of *Lecture Notes in Computer Science*, pages 89–105. Springer-Verlag, 1993.

[Cle86] R. Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC '86: Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing*, pages 364–369. ACM Press, 1986.

[Cra97] R. Cramer. *Modular Design of Secure yet Practical Cryptographic Protocols*. PhD thesis, Universiteit van Amsterdam.

[CD97] R. Cramer and I. Damgård. Linear zero-knowledge – a note on efficient zero-knowledge proofs and arguments. In *ACM Symposium on Theory of Computing – STOC 1997*, pages 436–445. ACM Press, 1997.

[CDN01] R. Cramer, I. Damgård, and J. Nielsen. Multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology–Eurocrypt 2001*, volume 2045 of *Lecture Notes in Computer Science*, pages 280–300. Springer-Verlag, 2001.

[CDS94] R. Cramer, I. Damgård, and B. Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In *Advances in Cryptology–Crypto 1994*, volume 839 of *Lecture Notes in Computer Science*, pages 174–187. Springer-Verlag, 1994.

[Cre87] C. Crépeau. Equivalence between two flavours of oblivious transfers. In *Advances in Cryptology–Crypto 1987*, Lecture Notes in Computer Science, pages 350–354. Springer-Verlag, 1987.

[Cre90] C. Crépeau. Verifiable disclosure of secrets and applications. In *Advances in Cryptology–Eurocrypt 1990*, volume 434 of *Lecture Notes in Computer Science*, pages 181–191. Springer-Verlag, 1990.

[CGT95] C. Crépeau, J. van de Graaf, and A. Tapp. Committed oblivious transfer and private multi-party computation. In *Advances in Cryptology–Crypto 1995*, volume 963 of *Lecture Notes in Computer Science*, pages 110–123. Springer-Verlag, 1995.

[DI05] I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In *Advances in Cryptology–Crypto 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer-Verlag, 2005.

[DJ01] I. Damgård and M. Jurik. A generalization, a simplification and some applications of Paillier's probabilistic public-key system. In *Public Key Cryptography–PKC 2001*, volume 1992 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2001.

[DN03] I. Damgård and J. Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In *Advances in Cryptology–Crypto 2003*, volume 2779 of *Lecture Notes in Computer Science*, pages 247–264. Springer-Verlag, 2003.

[EGL85] S. Even, O. Goldreich, and A. Lempel. A randomized protocol for signing contracts. In *Communications of the ACM*, volume 28, pages 637–647. ACM Press, 1985.

[FS86] A. Fiat and A. Shamir. How to prove yourself: practical solutions to identification and signature problems. In *Advances in Cryptology–Crypto 1986*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.

[FM06] M. Franklin and P. Mohassel. Efficiency tradeoffs for malicious two-party computation. In *Public Key Cryptography–PKC 2006*, volume 3958 of *Lecture Notes in Computer Science*, pages 458–473, 2006.

[GMY04] J. Garay, P. MacKenzie, and K. Yang. Efficient and universally composable committed oblivious transfer and applications. In *Theory of Cryptography Conference – TCC 2004*, volume 2951 of *Lecture Notes in Computer Science*, pages 297–316. Springer-Verlag, 2004.

[GMY06] J. A. Garay, P. MacKenzie, and K. Yang. Strengthening zero-knowledge protocols using signatures. *Journal of Cryptology*, 19(2):169–209, 2006.

[GMPY06] J. A. Garay, P. D. MacKenzie, M. Prabhakaran, and K. Yang. Resource fairness and composability of cryptographic protocols. In *Theory of Cryptography Conference – TCC 2006*, volume 3876 of *Lecture Notes in Computer Science*, pages 404–428. Springer-Verlag, 2006. `http://eprint.iacr.org/2005/370`.

[Gol04] O. Goldreich. Foundations of cryptography. In *Volume 2- Basic Applications. Cambridge University Press*, 2004.

[GMR89] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on Computing*, 18(1):186–208, 1989.

[GM86] O. Goldreich and S. Micali. How to construct random functions. volume 33, pages 792–807. ACM Press, 1986.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. *In Proceedings of the 19th annual ACM conference on Theory of computing*, pages 218–229, ACM press, 1987.

[GMR88] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. volume 17, pages 281–308. Society for Industrial and Applied Mathematics, 1988.

[JJ00] M. Jakobsson and A. Juels. Mix and match: Secure function evaluation via ciphertexts. In *Advances in Cryptology–Asiacrypt 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 162–177. Springer-Verlag, 2000.

[JS07] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *Advances in Cryptology–Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 97–114. Springer-Verlag, 2007.

[Kil88] J. Kilian. Founding cryptography on oblivious transfer. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20–31. ACM Press, 1988.

[KS06a] 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*, pages 283–290, 2006.

[KS06b] M. S. Kiraz and B. Schoenmakers. Securing Yao's garbled circuit construction against active adversaries. In *Online Proceedings. 1st Benelux Workshop on Information and System Security (WISSEC 2006)*, 2006. `https://www.cosic.esat.kuleuven.be/wissec2006/papers/23.pdf`.

[KS08] M. S. Kiraz and B. Schoenmakers. An efficient protocol for fair secure two-party computation. In *CT-RSA conference (To Appear)*, Lecture Notes in Computer Science. Springer-Verlag, 2008.

[KSV07] M. S. Kiraz, B. Schoenmakers, and J. Villegas. Efficient committed oblivious transfer of bit strings. In *Information Security Conference–ISC 2007*, volume 4779 of *Lecture Notes in Computer Science*, pages 130–144. Springer-Verlag, 2007.

[Lin01] Y. Lindell. Parallel coin-tossing and constant-round secure two-party computation. *Journal of Cryptology*, 16(3):143–184, 2001.

[LP04] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. To appear in the Journal of Cryptology. `http://eprint.iacr.org/2004/175`.

[LP07] Y. Lindell and B. Pinkas. An efficient protocol for secure two-party computation in the presence of malicious adversaries. In *Advances in Cryptology–Eurocrypt 2007*, volume 4515 of *Lecture Notes in Computer Science*, pages 52–78. Springer-Verlag, 2007.

[Lip03] H. Lipmaa. Verifiable homomorphic oblivious transfer and private equality test. In *Advances in Cryptology–Asiacrypt 2003*, volume 2894 of *Lecture Notes in Computer Science*, pages 416–433. Springer-Verlag, 2003.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay – a secure two-party computation system. pages 287–302, 2004.

[Nao91] M. Naor. Bit commitment using pseudorandomness. In *Journal of Cryptology*, volume 4, pages 151–158, 1991.

[NP01] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 448–457. ACM Press, 2001.

[GMW86] S. Micali O. Goldreich and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design. *In Proceedings*

of the 27th IEEE Symposium on Foundations of Computer Science (FOCS), pages 174–187, 1986.

[GMW91] S. Micali O. Goldreich and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. volume 38, pages 690–728. ACM Press, 1991.

[Pai99] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology–Eurocrypt 1999*, volume 1592 of *Lecture Notes in Computer Science*, pages 223–238. Springer-Verlag, 1999.

[Ped91] T. Pedersen. A threshold cryptosystem without trusted party. In *Advances in Cryptology–Eurocrypt 1991*, volume 547 of *Lecture Notes in Computer Science*, pages 522–526. Springer-Verlag, 1991.

[Pin03] B. Pinkas. Fair secure two-party computation. In *Advances in Cryptology–Eurocrypt 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 87–105. Springer-Verlag, 2003.

[Pin05] B. Pinkas. Personal communication, 2005.

[Pin08] B. Pinkas. Personal communication, Thesis review, 2008.

[Rab78] M. Rabin. Digitalized signatures. In *Foundations of Secure Computation*, pages 155–168. Academic Press, 1978.

[Rab81] M. Rabin. How to exchange secrets by oblivious transfer. *Technical Report TR-81, Harvard Aiken Computation Laboratory*, 1981.

[Rog91] P. Rogaway. The round complexity of secure protocols. *Ph.D. Thesis, MIT*, june 1991.

[ST04] B. Schoenmakers and P. Tuyls. Practical two-party computation based on the conditional gate. In *Advances in Cryptology–Asiacrypt 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 119–136. Springer-Verlag, 2004.

[Tze02] W. Tzeng. Efficient 1-out-of-n oblivious transfer schemes. In *Public Key Cryptography–PKC 2002*, volume 2274 of *Lecture Notes in Computer Science*, pages 159–171. Springer-Verlag, 2002.

[Woo06] D. P. Woodruff. Revisiting the efficiency of malicious two-party computation. volume 4515 of *Lecture Notes in Computer Science*, pages 79–96. Springer-Verlag, 2007. `http://eprint.iacr.org/2006/397`.

[Yao82] A. Yao. Protocols for secure computation. In *23rd IEEE Symposium on Foundations of Computer Science*, pages 160–164, 1982.

[Yao86] A. Yao. How to generate and exchange secrets. In *27th IEEE Symposium on Foundations of Computer Science*, pages 162–168, 1986.

Index

Nederlandse samenvatting: Secure and Fair Two-Party Computation

We beschouwen enkele personen (of partijen) die elkaar niet vertrouwen, maar toch een gezamenlijke berekening willen doen. De berekening is afhankelijk van een geheime invoer van elke persoon waarbij de invoer geheim dient te blijven. Dit probleem staat bekend als "Secure Multi-Party Computation" (veilige berekening tussen meerdere personen) en is in 1982 door Andrew Yao geïntroduceerd. Deze berekeningen hebben praktische toepassingen zoals digitale veilingen en digitaal stemmen. In dit proefschrift concentreren we ons op het geval waarin slechts twee personen zijn betrokken. Dit staat bekend als "Secure Two-Party Computation".

Als er een derde persoon aanwezig is die door iedereen wordt vertrouwd (trusted third party) dan is er een simpele oplossing: beide deelnemers geven hun geheime invoer aan deze vertrouwde persoon, die op haar beurt de berekening uitvoert. Vervolgens geeft ze beide deelnemers het correcte antwoord terug. Doel is echter om (bijna) hetzelfde resultaat te bereiken zonder vertrouwde persoon.

Om dit soort problemen op te lossen worden cryptografische protocollen ontwikkeld. Deze protocollen worden vervolgens geanalyseerd in een model voor het gedrag van de deelnemers. Het meest simpele model is het zogeheten Semi-Honest (gedeeltelijk eerlijk) model. Hierbij wordt aangenomen dat iedereen zich aan het protocol houdt, maar probeert zoveel mogelijk informatie uit de ontvangen berichten te krijgen. Een meer realistisch scenario is het Malicious (kwaadaardig) model. De meest gebruikte aanpak is om een protocol eerst in het semi-honest model te analyseren en deze analyse daarna uit te breiden naar het malicious model.

Elk cryptografisch protocol voor secure two-party computation moet aan drie veiligheidseisen voldoen: correctheid, privacy en fairness (eerlijkheid). Het protocol moet de correctheid van het resultaat garanderen terwijl het de geheimhouding van alle invoer bewaakt, zelfs wanneer een van de deelnemers kwaadaardig is en willekeurig van het protocol af kan wijken. Het moet ook eerlijkheid garanderen. Dit betekent grofweg dat wanneer een deelnemer het protocol voortijdig afbreekt deze deelnemer het eindresultaat niet makkelijker te weten kan komen dan de andere spelers.

De belangrijkste onderzoeksvraag in het construeren van een nieuw protocol voor secure multi-party computation is of het aan deze eisen voldoet. Een protocol moet zowel efficiënt zijn als een goede veiligheid bieden. In 1986 presenteerde Yao het eerste algemene protocol voor secure two-party computation, dat toen alleen geanalyseerd is in het semi-honest model. Hij gebruikt een techniek die "Garbled Circuit" genoemd wordt en gebruikt ondermeer de onderliggende primitieven "Pseudorandom Generator" en "Oblivious Transfer" om tot efficiënte resultaten te komen. Na Yao zijn er vele varianten en verbeteringen voor het malicious model voorgesteld. In dit proefschrift ontwerpen we verschillende nieuwe protocollen voor secure two-party computation gebaseerd op Yao's garbled circuit. We tonen enkele zwakke punten en veiligheidsproblemen met bestaande protocollen. Vervolgens werken we details uit van nieuwe protocollen, eerst in het semi-honest model en vervolgens

in het malicious model. Tot slot voegen we fairness aan ons protocol toe.

Oblivious transfer (OT) is een fundamentele tool in moderne cryptografie en erg nuttig bij het implementeren van protocollen voor secure multi-party computation. In dit proefschrift bestuderen we verschillende varianten van oblivious transfer. We presenteren een nieuw protocol voor zogeheten “Committed OT”. Dit protocol is erg efficiënt, in de zin dat het goed presteert in vergelijking met de meest efficiënte bestaande protocollen in de literatuur. Een van de veiligheidsproblemen van eerdere protocollen die we beschrijven in dit proefschrift wordt verholpen door het gebruik van ons nieuwe protocol voor ”Committed OT”. Bovendien is het algemener dan bestaande protocollen, wat op zichzelf al een interessant resultaat is.

We behandelen ook fairness in dit proefschrift. Tot dusver heeft alleen Benny Pinkas een op garbled circuits gebaseerd protocol gepresenteerd dat fairness behaald. We tonen aan dat er een subtiel probleem met zijn protocol is waardoor de geheimhouding van de invoer van een van de deelnemers niet gegarandeerd kan worden. We beschrijven dit probleem in detail en geven een efficiënter protocol dat deze problemen oplost.

Summary: Secure and Fair Two-Party Computation

Consider several parties that do not trust each other, yet they wish to correctly compute some common function of their local inputs while keeping these inputs private. This problem is known as "Secure Multi-Party Computation", and was introduced by Andrew Yao in 1982. Secure multi-party computations have some real world examples like electronic auctions, electronic voting or fingerprinting. In this thesis we consider the case where there are only two parties involved. This is known as "Secure Two-Party Computation".

If there is a trusted third party called Carol, then the problem is pretty straightforward. The participating parties could hand their inputs in Carol who can compute the common function correctly and could return the outputs to the corresponding parties. The goal is to achieve (almost) the same result when there is no trusted third party.

Cryptographic protocols are designed in order to solve these kinds of problems. These protocols are analyzed within an appropriate model in which the behavior of parties is structured. The basic level is called the Semi-Honest Model where parties are assumed to follow the protocol specification, but later can derive additional information based on the messages which have been received so far. A more realistic model is the so-called Malicious Model. The common approach is to first analyze a protocol in the semi-honest model and then later extend it into the malicious model.

Any cryptographic protocol for secure two-party computation must satisfy the following security requirements: correctness, privacy and fairness. It must guarantee the correctness of the result while preserving the privacy of the parties' inputs, even if one of the parties is malicious and behaves arbitrarily throughout the protocol. It must also guarantee fairness. This roughly means that whenever a party aborts the protocol prematurely, he or she should not have any advantage over the other party in discovering the output.

The main question for researchers is to construct new protocols that achieve the above mentioned goals for secure multi-party computation. Of course, such protocols must be secure in a given model, as well as be as efficient as possible. In 1986, Yao presented the first general protocol for secure two-party computation which was applicable only to the semi-honest model. He uses a tool called "Garbled Circuit". Yao's protocol uses the underlying primitives ("Pseudorandom Generator" and "Oblivious Transfer") as black-boxes which lead to efficient results. After Yao's work many variants and improvements have been proposed for the malicious model. In this thesis, we design several new protocols for secure two-party computation based on Yao's garbled circuit. Before we present the details of our new designs, we first show several weaknesses, security flaws or problems with the existing protocols in the literature. We first work in the semi-honest model and then extend it into the malicious model by presenting new protocols. Finally we add fairness to our protocol.

Oblivious transfer (OT) is a fundamental primitive in modern cryptography which is useful for implementing protocols for secure multi-party computation. We study several variants of oblivious transfer in this thesis. We present a new protocol for the so-called

"Committed OT". This protocol is very efficient in the sense that it is quite good in comparison to the most efficient committed OT protocols in the literature. The above-mentioned flaw with the use of OT can be fixed with our committed oblivious transfer protocol. Furthermore, it is more general than all previous protocols, and, therefore, it is of independent interest.

We also deal with fairness in this thesis. For protocols based on garbled circuit, so far only Benny Pinkas has presented a protocol in the literature for achieving fairness. We show a subtle problem with this protocol where the privacy of the inputs of one party can be compromised. We also describe this problem in detail which is in fact related to the fairness, and finally propose a more efficient scheme that does achieve fairness.

Acknowledgments

First of all, I would like to thank to my supervisors Henk van Tilborg and Berry Schoenmakers. I appreciate Henk for giving me the opportunity to do Ph.D by inviting me to Eindhoven, without his support, it would not have been such an outcome. Thanks also to Berry for all the guidance and advice he has given me over the last four years. Without his input, I would not have been able to complete this thesis. I really appreciate him to be enthusiastic, giving continuous guidance, supervision, and encouragement.

Some results included in this thesis are obtained in collaboration with José Villegas. I thank him for his discussions, stimulating and successful observations on this thesis which helped me a lot.

I am grateful to my reading committee of Henk van Tilborg, Berry Schoenmakers, Benny Pinkas, Tanja Lange, Pim Tuyls, and Andries Brouwer. Their comments and suggestions helped to improve the thesis significantly. I thank Benny Pinkas for interesting discussions and insightful comments.

Ellen Jochemsz, Andrey Sidorenko, José Villegas, Peter van Liesdonk and Reza Rezaeian Farashahi gave me a lot of feedback on individual chapters. I appreciate it a lot. In particular, thanks to Peter van Liesdonk to help me for the diagrams in this thesis.

I am indebted to all members of the Coding Theory and Cryptography group as well as the members of the Discrete Algebra and Geometry group for the friendly atmosphere, which made my stay at TU Eindhoven enjoyable and unforgettable. Also special thanks to Ellen Jochemsz, who I had the pleasure to share the office for three years.

I thank my whole family, for their support, understanding, and encouragement. Last, but not least, I want to thank my wife Nurhayat for the continuous support.

Curriculum Vitae

Mehmet S. Kiraz was born on October 2, 1977 in Siirt, Turkey.

In 1996, he graduated from Mersin Anatolian Teacher's High School in Turkey, and started his university studies at the department of mathematics, Middle East Technical University (METU) in Ankara, Turkey. Following the graduation from METU, he worked at Pamukbank and Yapi Kredi Bank as software analyst and software programmer respectively, which are two major banks in Turkey. In 2002, he started his masters studies at the International Max-Planck Institute for Computer Science in Saarbrücken, Germany and finished at the end of 2003 with a specialization on "Security Protocols". His masters thesis was titled "Formalization and Verification of Informal Security Protocol Descriptions", and was supervised by dr. Bruno-Blanchet.

From 2004 to 2008, he was a Ph.D. student at the Coding & Crypto group of the Technische Universiteit Eindhoven, under the supervision of prof. Henk van Tilborg and Dr. Berry Schoenmakers. The present thesis is the result of his work during this period.

His research interests are Secure Computation, design of efficient Cryptographic Protocols.