ARIEL UNIVERSITY

MASTER THESIS

Secure Multiparty Computation with Fairness for n-Party Functionalities

Author:
Jonathan Bronshtein

Supervisors:
Dr. Eran Omri, Dr. Anat Paskin-Cherniavsky

Department of Computer Science

April 16, 2023

Declaration of Authorship

I, Jonathan Bronshtein, hereby declare that this thesis entitled, "Secure Multiparty Computation with Fairness for n-Party Functionalities" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

"If we knew what it was we were doing, it would not be called research, would it?"

Albert Einstein

ARIEL UNIVERSITY

Abstract

Faculty of Natural Sciences
Department of Computer Science

Master

Secure Multiparty Computation with Fairness for n-Party Functionalities

by Jonathan Bronshtein

The field of secure multiparty computation (MPC) explores the problem of several parties computing functionalities while guaranteeing certain security properties. The parties are not assumed to trust each other, and fundamental security properties include privacy of inputs and correctness of output. Additional properties are often desired, such as, fairness (either all parties receive an output or none do) or even guaranteed output delivery (every party is guaranteed to receive an output from the computation).

Richard Cleve's result [6] showed that when half the parties may be corrupted, computing a *fair* coin toss is impossible. Specifically, in this case corrupted parties can always bias the outcome by a factor that depends on the number of rounds. Cleve's impossibility result [6] caused a false belief that *embedded XOR functions*, i.e., functions that allow computing XOR for certain inputs, are impossible to compute with fairness without an honest majority. In fact, it was believed by many researchers that no "interesting" functions are computable with fairness.

The seminal work of Gordon, Hazay, Katz, and Lindell [10] disproved this by showing a two-party protocol for computing some functions with embedded XOR. The set of two-party functions that are computable with fairness were further explored by Asharov [2] and Makriyannis [13], and a characterization for *Boolean* such functions was given by Asharov, Beimel, Makriyannis, and Omri [3]. Additionally, [3] extended their result to the multiparty case, giving a complete characterization of n-party functions where up to half the parties are corrupted. Gordon and Katz [11] investigated fairness in the setting of more than two parties where *more than half* of them are corrupted. They showed that it is possible to compute the three-party majority function and n-party OR with complete fairness, by presenting a protocol, which they proved was optimal. Recently, Dachman-Soled [7] showed that n-party Majority (where $n > 3$) can be computed with fairness for up to $\frac{n}{2} + 1$ corruptions.

In this thesis, we aim to broaden our understanding of fair computation by further exploring the set of functions that can be computed with fairness in the setting of more than two parties and a dishonest majority. We show that fairness is possible for two non-Boolean, symmetric, n party functions for up to $n - 1$ corruptions. In addition, we investigate fair computation of a specific embedded-XOR function for three parties. We provide a partial result for an adversary controlling up to two parties, when given certain assumptions about its behavior.

Acknowledgements

I would like to thank Ariel University for the fellowship which supported me while conducting my research. In addition, I would like to express my special thanks of gratitude to both of my project advisors, Eran Omri and Anat Paskin-Cherniavsky for giving me the golden opportunity to do this fascinating research in the topic of Secure Multiparty Computation, and who helped me a lot throughout my research. Finally, I would like to thank my parents, my sister, and my close friends for supporting me throughout my research and for always motivating me towards pursuing what I love.

Contents

List of Abbreviations

MPC	(secure)**M**ulti **P**arty **C**omputation
TP	**T**rusted **P**arty

Chapter 1

Introduction

In the setting of Secure Multiparty Computation, abbreviated MPC, a group of mutually distrusting parties wish to compute a functionality while preserving security properties such as privacy, correctness, guaranteed output delivery and fairness. Ideally, to compute a functionality while preserving the above properties, parties could use a *trusted party* (TP) to which they would send their inputs to and form which they would receive the outputs. In this idealized scenario, security properties (e.g., input and output privacy, correctness, fairness) are preserved. In the real world, however, we cannot assume a trusted party and parties engage in a protocol, i.e., a (random) process executed by the parties, mapping their n inputs to their n respective outputs. The goal of the computation of the protocol is described by the functionality as it would be computed by the above mentioned trusted party.

We next give brief (intuitive) descriptions of the security requirements mentioned above. Privacy means that an attacker cannot learn anything that is not implied by its input and its output. Correctness means that the output of honest parties agrees with the inputs of honest parties and some set of inputs for the corrupted parties. The requirement of guaranteed output delivery means that parties who follow the rules of the protocol receive the output regardless of what the other parties do. A slightly weaker requirement is that of fairness, where a party receives the output if and only if all other parties receive it too. Usually, when we talk about fairness, we mean fairness together with guaranteed output delivery. The security requirements are formalized by the *real vs. ideal* paradigm, which is described in chapter 2.

The main challenge in designing a protocol is handling the behavior of parties who deviate from the protocol while maintaining the security requirements. In order to model the behavior of parties who deviate from the protocol, we use the notion of an external entity (called an *adversary*) controlling a set of *corrupted* parties. The adversary has full control over the inner state of all corrupted parties and instructs them what actions to perform during the computation of the protocol. A malicious adversary can instruct parties to deviate from the protocol arbitrarily, in contrast to a semi-honest adversary, which follows the prescribed protocol faithfully, but can try to learn additional information given the view of the corrupted parties. We consider an adversary that is static, namely, it selects the set of corrupted parties before the beginning of the execution of the protocol (and does not change it afterwards). When an adversary is said to be polynomial-time, it means that the total number of operations it performs during the execution of the protocol is bounded by some polynomial in the security parameter. In our work, we only deal with static, malicious, polynomial-time adversaries.

In his seminal work, Cleve [6] proved that a coin-flip cannot be computed with fairness without an honest majority. He showed this by proving the existence of an efficient adversary who can bias the output of a generic two-party protocol by a factor that depends on the number of rounds, and then using a simple reduction to the n-party case, where up to $n/2$ of the parties are corrupted. The result of [6] has significant implications in the MPC field. Specifically, it is very important to understanding whether functions can be computed with fairness. For example, the problem of coin-flipping can be reduced to the computation of the Boolean XOR function, meaning that the latter cannot be computed fairly without an honest majority. For a time, the results of Cleve were interpreted as to imply impossibility of fair computation for any non-trivial functionality and, specifically, for *embedded-XOR functions*. An *embedded-XOR function* is a function that allows computing XOR by using specific inputs.

This belief was proved wrong in the work "Complete Fairness in Secure Two-Party Computation" of Gordon, Hazay, Katz, and Lindell [10], who presented a protocol for computing a function that contains an embedded XOR, as well as a protocol for computing any function without an embedded XOR. Asharov [2] and Makriyannis [13] identified the set of two-party Boolean functions that are computable with the GHKL protocol. Asharov [2] also extended the discussion to non-Boolean and non-symmetric functions that can be computed with the GHKL protocol. Makriyannis [13] additionally presented a class of functions for which fair computation is impossible by showing that they imply *non-trivial sampling*, which is impossible to compute with fairness by [1]. Asharov, Beimel, Makriyannis, and Omri [3] completed the characterization of two-party Boolean functions that can be computed with fairness (proving the impossibility result of [13] to be tight). In addition, [3] gave a complete characterization of symmetric n player functions where at most half of the parties are corrupted by a static adversary, where n is logarithmic in the security parameter.

Gordon and Katz [11] showed that some functions can be computed with fairness in the setting of more than two parties with a corrupted majority, namely, the n-party OR function and three-party majority (if the majority of inputs are $b \in \{0,1\}$, output is b). To prove security for the Majority function, they constructed a secure protocol. They complemented their positive result with a lower bound on the number of required rounds, showing that $\omega(\log(k))$ rounds are necessary. Since two-party majority can be computed with fairness in $O(1)$ time, this shows that functionalities with more than two parties are, in a sense, harder to compute than two-party functionalities. More evidence to that effect was given by Beimel, Haitner, Makriyannis, and Omri [4], who showed that the bias that can be imposed by the adversary on a fair coin-flipping protocol grows with the number of parties. Recently, the open question of [11], which asked whether the n-party Majority function can be computed in the presence of an adversary corrupting more than half the parties, was solved by Dachman-Soled [7], who constructed a fair protocol tolerating up to $\frac{n}{2}+1$ corrupted parties. Dachman-Soled [7] further considered n-party Boolean functions, where each party holds a bit and the output depends only on the number of one-bits. For this class of functions, Dachman-Soled extended the positive result of [3] to a number of parties that is polynomial in the security parameter.

1.1 Full Security for the Case of More than Two Parties

Our aim in this research is to expand our knowledge on what functions with more than two parties are computable with fairness and guaranteed output delivery when more than half the parties are corrupted by static, polynomial-time adversaries. We show possibility for three symmetric functions (meaning all parties get the same output), where the first two functions are non-Boolean, and the third function is a three-party function that contains an embedded XOR. Unfortunately, the construction we provide for the latter function is only shown to be secure against a restricted subset of the set of static, polynomial-time adversaries that may corrupt two parties. In fact, as far as we know, three party functions with an embedded XOR may be impossible to compute with fairness.

In Chapter 4, we discuss the function $\mathsf{max} : [m]^n \to [m]$, which is defined as follows:

$$\mathsf{max}(x_1, ..., x_n) := x_i \text{ such that } x_i \geq x_j \text{ for all } j \in [n]$$

where m is polynomial in the size of the input domain. Note, that m is not polynomial in the security parameter n, since n is logarithmic in the size of the input domain.

Theorem 1.1.1 (informal). *Under suitable cryptographic assumptions, there exists a protocol that computes the function* max *with full security in the presence of a non-uniform, poly-time adversary corrupting up to $n-1$ parties.*

As we noted, the n-party OR function (which was considered by [11]) is a special case of the Maximum function, where the size of the input domain is 2.

We note that the above result is weaker than the one we describe next. However, we chose to include it in order to present a different approach towards designing a secure protocol (and proving security for it).

In Chapter 5, we discuss the function $\mathsf{maxi} : [m]^n \to [m] \times [n]$, defined as follows:

$$\mathsf{maxi}(x_1, ..., x_n) := (\mathsf{max}\{x_1, ..., x_n\}, \mathsf{min}\{j : x_j = \mathsf{max}\{x_1, ..., x_n\}\})$$

where min is the minimum function and $m = m(n)$ for a security parameter n and a polynomial function m. That is, this function returns the maximum and the index of the first party that holds the maximum value. We prove the following.

Theorem 1.1.2 (informal). *Under suitable cryptographic assumptions, there exists a protocol that computes the function* maxi *with full security in the presence of a non-uniform, poly-time adversary corrupting up to $n-1$ parties.*

We note that this result implies the possibility of computing an arbitrary function, where the output is the input of one of the parties and is chosen according to some a priory ordering on the domain of inputs.

Finally, in Chapter 6, we discuss the function $\mathsf{XOR}_{01} : \{0,1,2,3\} \times \{0,1\} \times \{0,1\} \to \{0,1\}$, defined as follows:

$$\mathsf{XOR}_{01}(x_1, x_2, x_3) = \begin{cases} 0, & \text{if } x_1 = 2 \\ 1, & \text{if } x_1 = 3 \\ x_1 \oplus x_2 \oplus x_3, & \text{otherwise} \end{cases}$$

The XOR_{01} function can be seen as an extension of the GHKL function to the three party case, with the addition that the "stronger" party can now also fix the output to 0. We prove the following (partial) result.

Theorem 1.1.3 (informal). *Under suitable cryptographic assumptions, there exists a protocol that computes the function* XOR_{01} *with full security in the presence of a non-uniform,*

poly-time adversary corrupting up to two parties, assuming the following restriction on the behavior of the adversary:
If the adversary ever aborts any party, then it either immediately aborts the other one too or never aborts it.

We stress that this is a weaker result that we would have liked to obtain, since we are assuming certain artificial limitations on the adversary. However, we believe that this result sheds some light on the challenges in proving security for such a function that may be useful to consider in further research.

Chapter 2

Definitions

2.1 Notation

We use either calligraphic or capital letters to denote sets, e.g., $\mathcal{S}$ or X, uppercase for random variables, lowercase for values, and bold characters for vectors, e.g. $\mathbf{v}$. For $n \in \mathbb{N}$, let $[n] = \{1, 2, ..., n\}$; given a random variable (or a distribution) X, we write $x \leftarrow X$ to indicate that x is selected according to X. When X is a set, $x \leftarrow_u X$ indicates that the value x is sampled uniformly at random form X.

2.2 Computational indistinguishability and Statistical Closeness

We borrow most definitions from [10] and [3]. A function $\mu(\cdot)$ is *negligible* if for every polynomial $p(\cdot)$ and all sufficiently large n it holds that $\mu(n) < 1/p(n)$. A *distribution ensemble* $X = \{X(a,n)\}_{\{a \in \mathcal{D}_n, n \in \mathbb{N}\}}$ is an infinite sequence of random variables indexed by $a \in \mathcal{D}_n$ and $n \in \mathbb{N}$, where $\mathcal{D}_n$ is a set that may depend on n. Looking ahead, n will be the security parameter, $\mathcal{D}_n$ will be the input domain, and the set of random variables will be the view of the adversary and the output distribution of the honest players. Two distribution ensembles $X = \{X(a,n)\}_{\{a \in \mathcal{D}_n, n \in \mathbb{N}\}}$ and $Y = \{Y(a,n)\}_{\{a \in \mathcal{D}_n, n \in \mathbb{N}\}}$ are said to be *computationally indistinguishable*, denoted $X \overset{\mathsf{c}}{\equiv} Y$, if for every non-uniform polynomial time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every n and every $a \in \mathcal{D}$

$$\left|\Pr[D(X(a,n)) = 1] - \Pr[D(Y(a,n)) = 1]\right| \leq \mu(n)$$

The statistical difference between the two distributions $X(a,n)$ and $Y(a,n)$ is defined as

$$\mathrm{SD}(X(a,n), Y(a,n)) = \frac{1}{2} \cdot \sum_{s} \left|Pr[D(X(a,n)) = s] - Pr[D(Y(a,n)) = s]\right|,$$

where the sum ranges over s in the support of either $X(a,n)$ or $Y(a,n)$. Two distribution ensembles $X = \{X(a,n)\}_{\{a \in \mathcal{D}_n, n \in \mathbb{N}\}}$ and $Y = \{Y(a,n)\}_{\{a \in \mathcal{D}_n, n \in \mathbb{N}\}}$ are *statistically close*, denoted by $X \overset{\mathsf{s}}{\equiv} Y$, if there is there is a negligible function $\mu(\cdot)$ and for every $a \in \mathcal{D}$ it holds that $\mathrm{SD}(X(a,n), Y(a,n)) < \mu(n)$.

2.3 Secure Multiparty Computation

Denote the parties by $\mathsf{P}_1, ..., \mathsf{P}_m$. A functionality $f = (f_1, ..., f_m)$ is a randomized process that maps m tuples of inputs (corresponding to inputs of parties) to m tuples of outputs (corresponding to outputs of parties). The domain of f is denoted by $X = X_1 \times \cdots \times X_m$. When the functionality is deterministic, we refer to it as a function. When $f_1 = \cdots = f_m$ we say that f is *symmetric* and denote by f the output of each party. Otherwise we say f is *asymmetric*. An *m-party protocol* Π for computing a functionality f is a polynomial-time protocol such that on a global input 1^n – the security parameter – and on private inputs $x_1, ..., x_m \in X_1 \times \cdots \times X_m$, the joint distribution of the outputs of the honest executions of $\{\Pi(1^n, x_1, ..., x_m)\}_{n \in \mathbb{N}}$ is statistically close to $f(x_1, ..., x_m) = (f_1(x_1, ..., x_m), ..., f_m(x_1, ..., x_m))$. The parties run the protocol in polynomial time in the security parameter n (we consider f as a fixed, finite function).

The Adversary. Following the usual convention, we introduce an adversary $\mathcal{A}$, which corrupts a subset of the parties. An adversary may be either *semi-honest* or *malicious*. Parties controlled by a *semi-honest* adversary follow the rules of the protocol, but try to learn additional information, whereas parties controlled by a *malicious* adversary may arbitrarily deviate from the instructions of the protocol. We say that an adversary is *static* when it chooses the subset of parties it corrupts prior to the execution of the protocol and cannot alter its choice during the execution, and we say that it is non-uniform when it is given an auxiliary input z at the beggining of the execution. Throughout our work, we will be handling static, poly-time, non-uniform adversaries. In addition, we may introduce *fail-stop* and *rushing* adversaries. We say that an adversary is *fail-stop* when it may deviate from the protocol only by aborting parties, and that it's *rushing* when it learns all information in a single round before the honest parties do. That is, the decision of the adversary in a round also depends on the messages of honest parties in this round.

For a protocol Π computing f, let

$$(\mathsf{Out}^{\mathsf{Real}}_{\mathcal{A}(z),\Pi}, \mathsf{View}^{\mathsf{Real}}_{\mathcal{A}(z),\Pi})(1^n, x_1, ..., x_m)$$

denote the joint distribution of the honest parties' outputs and the adversary's view during an execution of Π, where $x_1, ..., x_m$ are the specified inputs, 1^n is the security parameter (in unary representation), and the view of the adversary contains its auxiliary input, the inputs of the parties it controls, their random inputs, and the messages they get during the execution of the protocol.

2.4 Security Using the Real-vs.-Ideal Model Paradigm

We consider security in the standalone model with static adversaries using the standard real-vs-ideal paradigm.

2.4.1 Full security ideal-world (without abort)

Let $f = (f_1, ..., f_m)$ be an m-party functionality and let Π be a protocol for computing f. Further, assume that an adversary corrupts a fixed subset $B \subseteq \{P_1, ..., P_m\}$ of the parties. Security in multiparty computation is defined via an *ideal model*. Namely, we assume that parties have access to a trusted party $\mathcal{T}$ that performs the computation of the functionality for them and we attempt to show that a protocol Π *emulates* this ideal scenario. In Figure 2.1 we describe the *ideal model with full security*.

Inputs: Each party P_i holds 1^n and $x_i \in X_i$. The adversary is given an auxiliary input $z \in \{0,1\}^*$. The trusted party $\mathcal{T}$ has no input.

Parties send inputs: Each honest party sends its input to the trusted party $\mathcal{T}$, each corrupted party sends a value of the adversary's choice. Write $(x'_1, ..., x'_m)$ for the tuple of inputs received by $\mathcal{T}$.

The trusted party preforms computation: If any x'_i is not in the appropriate domain (or was not sent at all), then $\mathcal{T}$ reassigns the aberrant input to some default value. Write $(x'_1, ..., x'_m)$ for the tuple of inputs after (possible) reassignment. The trusted party then chooses a random string r and computes $f(x'_1, ..., x'_m; r)$.

The trusted party sends outputs: Each party P_i receives $f_i(x'_1, ..., x'_m; r)$.

Outputs: Each honest party outputs whatever $\mathcal{T}$ sent it, the corrupted parties output nothing, and the adversary outputs a probabilistic polynomial-time function of its view.

FIGURE 2.1: The Ideal model with full security.

Let $\mathcal{S}$ be an adversary in the ideal world, which we usually refer to as "the simulator". $\mathcal{S}$ is given the auxiliary input z and corrupts a subset B of the parties. Denote the honest parties' outputs and the adversary's view by

$$(\mathsf{Out}^{\mathsf{Real}}_{\mathcal{A}(z),\Pi}, \mathsf{View}^{\mathsf{Real}}_{\mathcal{A}(z),\Pi})(1^n, x_1, ..., x_m)$$

where $x_1, ..., x_m$ are the prescribed inputs and 1^n is the security parameter. The view of the adversary is an infinite sequence of random variables corresponding to the information that the adversary observes during an execution. We now define security with respect to the ideal model with full security.

Definition 2.4.1. *Let Π be a protocol for computing f. We say that Π computes f with full security tolerating coalitions of size at most t, if for every non-uniform polynomial time adversary $\mathcal{A}$ controlling a set B of at most t parties in the real model, there exists a non-uniform polynomial time adversary $\mathcal{S}$ (called the simulator) controlling B in the ideal model such that*

$$\left\{ \left(\mathsf{Out}^{\mathsf{Real}}_{\mathcal{A}(z),\Pi}, \mathsf{View}^{\mathsf{Real}}_{\mathcal{A}(z),\Pi}\right)(1^n, x_1, ..., x_m) \right\}_{(x_1,...,x_m)\in X, z\in\{0,1\}^*, n\in\mathbb{N}}$$
$$\overset{\mathsf{c}}{\equiv} \left\{ \left(\mathsf{Out}^{\mathsf{Ideal}}_{\mathcal{S}(z),f}, \mathsf{View}^{\mathsf{Ideal}}_{\mathcal{S}(z),f}\right)(1^n, x_1, ..., x_m) \right\}_{(x_1,...,x_m)\in X, z\in\{0,1\}^*, n\in\mathbb{N}}$$

2.4.2 Ideal model with identifiable abort

The difference between the ideal model with abort to the ideal model without abort is that in the latter, before sending outputs to the honest parties, the trusted party $\mathcal{T}$ sends the output to $\mathcal{A}$ and then it asks $\mathcal{A}$ whether to *continue* or to *abort*. If $\mathcal{A}$ chooses to continue, then $\mathcal{T}$ sends the output to the honest parties as before; otherwise, $\mathcal{T}$ sends $\perp$ to all honest parties and nothing else. When we say that we use the ideal model with *identifiable* abort, we mean that if the adversary decides to abort (i.e., sends an *abort* message to the trusted party), then it must identify at least one corrupted party to $\mathcal{T}$, and then $\mathcal{T}$ sends the names of the identified parties to the honest parties. The ideal model with identifiable abort is given in Figure 2.2.

Inputs: Each party $P_i \in \mathcal{P}$ holds 1^n and $x_i \in X_i$. The adversary $\mathcal{A}$ corrupting $\mathcal{D} \subset \mathcal{P}$ is given an auxiliary input $z \in \{0,1\}^*$.

Parties send inputs: Each honest party sends its input to $\mathcal{T}$, each corrupted party sends a value of the adversary's choice. Write $(x'_1, ..., x'_m)$ for the tuple of inputs received by $\mathcal{T}$.

The trusted party preforms computation: If any x'_i is not in the appropriate domain (or was not sent at all), then $\mathcal{T}$ reassigns the aberrant input to some default value. Write $(x'_1, ..., x'_m)$ for the tuple of inputs after (possible) reassignment. The trusted party then chooses a random string r and computes $f(x'_1, ..., x'_m; r)$.

The trusted party sends outputs to the adversary: The adversary $\mathcal{A}$ receives the output of the computation $f_i(x'_1, ..., x'_m; r)$ on behalf of the corrupted parties $P_i \in \mathcal{D}$.
The adversary instructs the trusted party whether to continue: If $\mathcal{A}$ sends continue to $\mathcal{T}$, then $\mathcal{T}$ sends $f_i(x'_1, ..., x'_m; r)$ to each party $P_i \in \mathcal{P} \setminus \mathcal{D}$. Otherwise, the $\mathcal{A}$ sends a message of the form (abort,$\mathcal{I}$), for some non-empty subset $\mathcal{I} \subseteq \mathcal{D}$ of corrupted parties. In this case, $\mathcal{T}$ sends $(\perp, \mathcal{I})$ to all honest parties.
Outputs: Each honest party outputs whatever $\mathcal{T}$ sent it, the corrupted parties output nothing, and the adversary outputs a probabilistic polynomial-time function of its view.

FIGURE 2.2: The Ideal model for security with identifiable abort.

2.5 The Hybrid Model

The hybrid model combines both the real and ideal models. Specifically, an execution of a protocol π in the $\mathcal{G}$-hybrid model, for some functionality $\mathcal{G}$, involves the parties executing the protocol as in the real model and, in addition, having access to a trusted party computing $\mathcal{G}$. Whether the functionality is given in the ideal model with or without abort must be specified. In our case, we will use hybrid functionalities in the ideal model with identifiable abort.

Let $\mathcal{G}$ be a functionality and π be a multi-party protocol for computing some functionality $\mathcal{F}$, where π includes real messages between the parties in addition to calls to $\mathcal{G}$. Let $\mathcal{A}$ be a non-uniform probabilistic polynomial-time adversary with auxiliary input z. We let $(\mathsf{Out}^{\mathsf{Hybrid}^{\mathcal{G}}}_{\mathcal{A}(z),\Pi}, \mathsf{View}^{\mathsf{Hybrid}^{\mathcal{G}}}_{\mathcal{A}(z),\Pi})(1^n, x_1, ..., x_m)$ be the random variable consisting of the view of the adversary and the output of the honest parties, following an execution of π (with ideal calls to $\mathcal{G}$) where $(1^n, x_1, ..., x_n)$ are the inputs of parties computing π. The definitions of security and security with abort can be easily adapted from the non-hybrid case.
Sequential composition. If $\mathcal{G}$ is called sequentially, meaning that calls are given one after another and not all at once, then, according to the sequential composition theorem [5] of Canetti, $\mathcal{G}$ can be replaced by a protocol implementing $\mathcal{G}$ with security

as in the specification of $\mathcal{G}$ while preserving security. Thus, we can prove security in the hybrid-vs-ideal model, then apply the sequential composition theorem and achieve security by the real-ideal definition.

2.6 Cryptographic Tools

Next, we informally describe a few cryptographic tools that we use in our work.
Signature Schemes. A signature on a message proves that the message was sent by its presumed sender, and that its content was not altered. It is a triple $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ containing a key generation algorithm Gen, which gets as input a security parameter 1^n and outputs a pair of keys: the signing key sk and the verification key pk, the signing algorithm Sign, and the verifying algorithm Verify. We assume that it is infeasible to produce signatures without holding the signing key.
Secret-Sharing Schemes. A $t+1$-out-of-n secret-sharing scheme is a mechanism for sharing data among a set of parties such that every set of parties of size $t+1$ can reconstruct the secret, while any smaller set knows nothing about it. Specifically, the shares of any t parties are uniformly distributed and independent of the secret. In our work, we use only n-out-of-n secret sharing.

An example of an n-out-of-n secret sharing scheme is additive secret sharing. Let $\mathsf{P}_1, ..., \mathsf{P}_n$ be be parties wishing to compute a secret sharing of a bit s, which is held by P_1. To n-out-of-n additively secret share s, party P_1 samples $n-1$ random bits $s_2, ..., s_n$, and computes $s_1 = s \oplus \left(\bigoplus_{i=2}^{n} s_i\right)$. Then, P_1 sends s_i to P_i for $i \in \{2, ..., n\}$. It holds that all shares s_i are uniform random bits. Any combination of $t \leq n-1$ of the shares equals to the XOR of s with the remaining $n-t$ shares, and is therefore also uniform random. In addition, it is easy to see that $\bigoplus_{i=1}^{n} s_i = s$.

In the simulation, we often provide the adversary with secret shares of the value 0 as shares of a value that is not yet known. Later, we may provide the adversary with additional shares that complete its shares of 0 to an arbitrary value of our choice. This is possible due to the fact that secret shares expose no information about the secret, and as a result, shares of the value 0 are indistinguishable from shares of any other value. In the case of additive secret sharing, it is easy to see how the secret shared values can be completed to an arbitrary value of our choice.
Commitment Schemes. We borrow the definition from [12]. Informally, a *commitment scheme* allows one party to "commit" to a value m by sending a *commitment* com, and then reveal m (by "opening" the commitment) at a later point in time. A party commits to a value by generating randomness r and then activating Com on m and r to get $\mathsf{com} = \mathsf{Com}(1^n, m, r)$, where n is the security parameter. We require com to be *hiding* and *binding*. Informally, hiding means that com reveals nothing about m, and binding means that it is infeasible for the committer to output a commitment com that it can later "open" as two different messages m, m'.
Zero-Knowledge for commitments. Allows proving a commitment com is valid without requiring the prover to expose the committed value m. We can view the language of valid commitments as an $\mathcal{NP}$ language where a committed message and the randomness r used by com are the witness. Thus, as for any other language in $\mathcal{NP}$, there exists a zero-knowledge proof for proving that a word belongs to it. We do not implicitly use zero-knowledge in our protocols, but we do present functionalities that would require zero knowledge proofs to work. The reader may find a complete definition of zero-knowledge proofs in [14].

The Functionality of Oblivious Transfer (OT). In the setting of 1-out-of-2 OT, we have a receiver holding a bit $b \in \{0,1\}$ and a sender holding two messages m_0 and m_1. At the end of the interaction, the receiver learns m_b and nothing else and the sender learns nothing. Protocols that securely compute OT under several hardness assumptions are known, e.g., [8].

2.7 Mathematical Background

The Geometric Distribution. Let $n \in \mathbb{N}$ and $\alpha \in (0,1)$, we define the geometric distribution over support μ with parameter α to be

$$\mu(k) = (1-\alpha)^k \alpha$$

for any $k \in \mathbb{N}$. It can be seen as flipping a biased coin until we get heads. When we sample x from $\mathsf{Geom}(\alpha)$ we mean that we flip a coin with probability α for tails until we get tails, and x is the number of times we flipped that coin.
Output distribution vector. Let $f : \{x_1,...,x_l\} \times \{y_1,...,y_m\} \to \{z_1,...,z_n\}$ be a symmetric function. Let P_1 and P_2 be two parties computing f in the ideal world and holding inputs $x \in \{x_i\}_{i\in[l]}$ and $y \in \{y_i\}_{i\in[m]}$ respectively. The *output distribution vector* of P_2 is defined to be the vector

$$\mathbf{q} = (q_1,...,q_n)$$

where

$$q_i = \Pr[f(x,y_i) = 1 \mid x = x_j \text{ with probability } p_j \geq 0, \text{ for } j \in [m], \text{ such that } \sum_j p_j = 1]$$

When there are more that two parties, we can define the output distribution vector in a similar way, except using tuples of inputs instead of single inputs.

Chapter 3

Previous work

In what follows, we attempt to give a rich background to the problem of fairness in Secure Multiplarty Computation (MPC) and to present the current state of affairs. MPC was formally introduced by Yao [15]. Andrew Yao presented a protocol for two parties with inputs x and y that allows them to compute $x \geq y$ while preserving privacy of inputs (even for malicious adversaries). Yao's protocol was generalized to the n party setting by Goldreich, Micali, and Wigderson [9] for semi-honest adversaries, and can be made secure for malicious adversaries by using a special compiler (we discuss the protocol of [9] is Section 3.1). The protocol of [9] is secure with abort, i.e., it does not guarantee output delivery and fairness.

Oftentimes, parties wish to guarantee stronger security requirements, such as guaranteed output delivery and fairness. Such requirements are much more challenging to fulfill, and often downright impossible. For example, the XOR function is impossible to compute while guaranteeing the above requirements, as implied by the work of Cleve [6]. We discuss the work of Cleve in Section 3.2. Many functions are, however, possible to compute with fairness (and guaranteed output delivery). In fact, it is known exactly which two-party Boolean functions are possible to compute with fairness, as shown by Asharov, Beimel, Makriyannis, and Omri [3]. We present the work of [3] in Section 3.5. Fairness in the case of more than two parties is more challenging and much less explored. Some works which discuss the case of more than two parties are that of Gordon and Katz [11] and Dachman-Soled [7]. We discuss [11] in Sections 3.6 and 3.7.

3.1 Computing a Secret Sharing of any Function with abort

The GMW protocol [9] is one of several protocols for computing a secret sharing of any function and are secure-with-identifiable-abort. We begin by describing the GMW protocol in the setting of semi-honest adversaries, corrupting up to $n-1$ parties. Then, we briefly explain how this protocol can be made secure-with-identifiable-abort against malicious adversaries. We assume that inputs are binary, where the following may be generalized to inputs of arbitrary length. Additionally, we assume that the function f that the parties wish to compute is represented as a circuit, made up of AND and XOR gates with a fan-in of size two. There is an input wire for the input of each party and a single output wire.

The n parties begin by computing an n-out-of-n secret sharing of their inputs, which correspond to the input wires of the circuit. Party P_i computes an n-out-of-n additive sharing of its input x_i as follows. Party P_i sets P_j's share of x_i to be a uniform random bit x_i^j, and sets its own share to be $x_i^i = \left(\bigoplus_{j\in[n]\setminus\{i\}} x_i^j\right) \oplus x_i$. Then, P_i sends to each party its designated share. Thus, parties attain secret shares corresponding to each input wire and proceed to compute the circuit sequentially, gate by gate, where each gate is computed as follows, depending on its type:

XOR gate: Let w be the output of a XOR gate with inputs u and v, where the parties are assumed to have shares $u_1, \dots, u_n$ and $v_1, \dots, v_n$ of u and v. To compute a sharing of w, each party P_i computes $w_i = u_i \oplus v_i$ for $i \in [n]$. It is easy to see that w_i expose no information about w. Thus, it remains to show that w_i constitute a valid sharing of w, which can be seen as follows

$$w_1 \oplus \cdots \oplus w_n = (u_1 \oplus v_1) \oplus \cdots \oplus (u_n \oplus v_n) = (u_1 \oplus \cdots \oplus u_n) \oplus (v_1 \oplus \cdots \oplus v_n) = u \oplus v = w$$

AND gate: Let u, v, and w be the input and output wires of the gate as before, with parties holding shares of the inputs. w can be expressed in terms of parties' shares of u and v as follows:

$$\begin{aligned} w = u \cdot v &= \left(\bigoplus_{i=1}^{n} u_i \right) \cdot \left(\bigoplus_{i=1}^{n} v_i \right) \\ &= \left(\bigoplus_{i=1}^{n} u_i v_i \right) \oplus \left(\bigoplus_{i<j} (u_i v_j \oplus u_j v_i) \right) \end{aligned}$$

Party P_i can locally compute $u_i v_i$. Thus, it remains to show how parties can compute a sharing of the term $\bigoplus_{i<j}(u_i v_j \oplus u_j v_i)$. This is reduced to allowing each pair of parties $\mathsf{P}_i, \mathsf{P}_j$ for $i < j$ to compute a sharing of $u_i v_j \oplus u_j v_i$, which can be done by using *Oblivious Transfer* (defined in Section 2.6).

In more detail, party P_j, acting as the sender in the oblivious transfer, starts by selecting random bit mask $c_j^{\{i,j\}}$ (which will also serve as P_j's share). Then, P_j computes the four possible values of $c_j^{\{i,j\}} \oplus u_i v_j \oplus u_j v_i$, given the values of v_j and u_j (which are known to P_j) and according to the four possible assignments to (u_i, v_i) (which are not known to P_j). Party P_j will use these four values as its inputs to the 1-out-of-4 OT interaction with P_i, in particular, these four values are:

$$c_j^{\{i,j\}}, \quad c_j^{\{i,j\}} \oplus u_j, \quad c_j^{\{i,j\}} \oplus v_j, \quad c_j^{\{i,j\}} \oplus u_j \oplus v_j.$$

Party P_i then acts as a receiver in 1-out-of-4 OT, with index determined by the actual values of its shares u_i, v_i, to obtain the appropriate value from P_j, which we denote by $c_i^{\{j,i\}}$. Note that $c_i^{\{i,j\}} \oplus c_j^{\{j,i\}} = u_i v_j \oplus u_j v_i$. Finally, each party P_i computes its share of w by setting

$$w_i = u_i v_i \oplus \left(\bigoplus_{j \neq i} c_i^{\{i,j\}} \right)$$

This process is repeated until the parties hold a sharing of the output. Finally, each party sends its share to all other parties and each party reconstructs the output.

A compiler to malicious security ([9]). The difference between a semi-honest adversary and a malicious adversary is that the latter can deviate from the protocol arbitrarily. To prevent this, the parties are required to commit to their inputs and to broadcast their commitments so that in every step, the honest parties can verify that other parties "play by the rules". That is, the i-th party P_i verifies that the the messages it received up until this moment were created by a correct execution of Π with the committed inputs. This is done by running a zero-knowledge proof of knowledge on the language of valid messages given the protocol and the committed inputs. If a party is unable to prove that the messages it sent were created by a correct execution of Π, then parties output $\perp$ and halt.

An important thing to consider is that since the protocol is randomized, there exists some randomness for which invalid messages appear valid. To prevent this, the "randomness" is determined at the start of the execution of the protocol as follows. In more detail, to generate the randomness that P_i will use throughout the execution of the protocol, parties may do the following. First, P_i generates r_i^i, commits to it and broadcasts the commitment. Then, in order to prevent such situation that the randomness that P_i uses is not actually random, each other party P_j generates randomness r_i^j for P_i, commits to it, and broadcasts the commitment. After all parties have received the commitment of r_i^j, party P_j broadcasts r_i^j and opens its commitment (note that P_i does not do this for its commitment of r_i^i). The randomness that P_i will use is $\bigoplus_{j\in[n]} r_i^j$.

3.2 Impossibility of Computing XOR with Complete Fairness

The seminal work by Cleve [6] shows that securely flipping a coin is impossible. More precisely, in the setting where two parties wish to flip an unbiased coin while maintaining *agreement* (i.e., parties agree on a common output) and *guaranteed output delivery*, (i.e., even when one party aborts, the other party still outputs a random bit). Since XOR implies a fair coin flip (if at least one party inputs a uniform random bit, then the output is uniformly random), Cleve's result implies that it is impossible to compute XOR with fairness.

Specifically, [6] shows that for any two-party r round protocol computing a coin flip with agreement and guaranteed output delivery, one of the parties can bias the output of the other by a factor of $1/r$. Agreement means that parties must output the same bit at the end of the computation. We proceed to give a more detailed description of the setting. We assume a general protocol that computes a secure coin flip with agreement and guaranteed output delivery. Let n be the security parameter and let P_1 and P_2 be two parties. The protocol proceeds in $r = r(n)$ rounds, where in round i, party P_1 sends a message to P_2 and then P_2 sends its message to P_1. If one of the parties fails to send message, then, due to the guaranteed output delivery requirement, the other party must still be instructed by the protocol to output some bit. If P_1 does not send its message in round i, let b_{i-1} be the bit P_2 outputs, and let a_i be the bit that P_1 outputs in the analogous case. In case no party aborts, P_1 and P_2 output a_{r+1} and b_r, respectively. Since a_1 and b_0 are computed before any interaction, it must hold that $\Pr[a_1 = 1] = \Pr[b_0 = 1] = 1/2$. Otherwise, one of the parties can bias the output of the other by not sending any messages at all. In addition, due to the agreement requirement, it must hold that $a_{r+1} = b_r$.

Next, we next describe the intuition behind the proof. $4r$ adversaries are constructed: $\{\mathcal{A}_i^0\}_{i\in[r]}$, $\{\mathcal{A}_i^1\}_{i\in[r]}$, $\{\mathcal{B}_i^0\}_{i\in[r]}$,$\{\mathcal{B}_i^1\}_{i\in[r]}$, which we proceed to define. Let

$\mathcal{A}_i^0$ to be an adversary controlling P_1, who tries to bias the output of P_2 to 0 by deciding whether to abort in round i as follows. If $a_i = 0$, $\mathcal{A}_i^0$ aborts P_1 after sending the i-th message, so that P_2 outputs b_i, and if $a_i = 1$, then $\mathcal{A}_i^0$ aborts P_1 immediately so that P_2 outputs b_{i-1}. We can define $\mathcal{A}_i^1$ similarly, except that it tries to bias to 1, and $\mathcal{B}_i^0$ and $\mathcal{B}_i^1$ similarly to $\mathcal{A}_i^0$ and $\mathcal{A}_i^1$, respectively, except that they control P_2.

Recall that P_2 should output 1 with probability $1/2$. Then, the bias caused by $\mathcal{A}_i^0$ towards 0 is

$$\Pr[a_i = 0 \wedge b_i = 0] + \Pr[a_i = 1 \wedge b_{i-1} = 0] - \frac{1}{2}$$

Biases for other adversaries may be defined similarly. It turns out that if we average over all biases, then the biases form a telescopic series. Then, by taking into consideration the requirements that $\Pr[b_0 = 1] = 1/2$ and $a_{r+1} = b_r$, the average of the biases that we get is

$$\frac{1}{4r}$$

which means that at least one of the adversaries we described succeeds with at least with such probability.

This result can be generalized into the n party setting by a reduction to the two-party case. Informally, if there exists a protocol Π that allows a secure coin flip for n parties out of which half are corrupt, then we can construct a two-party protocol for a secure coin flip by having each party simulate the actions of one half of the n parties running Π.

3.3 Computing a Specific Function with an Embedded XOR with Fairness

A function with an *embedded XOR* is a function such that if the parties could be restricted to using only certain inputs, then they would be computing the XOR function. Due to Cleve [6], it is tempting to think that functions with an embedded XOR cannot be computed. However, as [10] show, this is not true. The intuitive reason for this is that a function may "allow biasing". That is, by choosing certain inputs, one of the parties can make the output (even in the ideal world) closer to either 1 or to 0 independently of the other party's input. This property allows simulating the inevitable bias that may occur in an execution of the protocol.

In their work, Gordon, Hazay, Katz, and Lindell [10] show how to compute a specific function with an embedded XOR with full security. They present a protocol for generic two-party Boolean functions that works for *some* two party Boolean functions, and they prove feasibility for the function $f_{\mathsf{GHKL}} : X \times Y \to \{0,1\}$, where $X = \{x_1, x_2, x_2\}$,$Y = \{y_1, y_2\}$, defined in Figure 3.1. We later refer to f_{GHKL} as the *GHKL function*.

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

FIGURE 3.1: The function f_{GHKL} (the GHKL function).

We proceed to give an informal overview of the GHKL protocol, Π_{GHKL}, which is described in Figure 3.2. Π_{GHKL} is parameterized with a special parameter α and takes as common input a security parameter n. The parties begin by computing the functionality $\mathsf{ShareGen}_{\mathsf{GHKL}}$ (Figure 3.3) with their inputs x and y ($\mathsf{ShareGen}_{\mathsf{GHKL}}$ may be implementd by using the GMW protocol, described in Section 3.1). Parties receive as output from $\mathsf{ShareGen}_{\mathsf{GHKL}}$ shares of the backup values a_i and b_i for $i \in [r]$. Next, parties proceed to learn their respective backup values in rounds $i = 1, ..., r$, where party P_1 learns the backup value a_i, and after this P_2 learns the backup value b_i. If no party aborts, parties P_1 and P_2 output a_r and b_r, respectively. If one of the parties aborts in round i, then the remaining party outputs its backup value from round $i-1$ (note that parties learn a_0 and b_0 before the first round).

The values a_i and b_i are constructed in such way that both parties learn the output in a random round i^* that is unknown to both. Specifically, in rounds $i < i^*$, the values a_i and b_i expose no information about the output (i.e., a_i is independent of y and b_i is independent of x), and in rounds $i \geq i^*$, it holds that $a_i = b_i = f(x, y)$. The round number i^* is sampled from the geometric distribution with parameter α before the start of the computation. This may be viewed as if a coin with probability α for heads is flipped in each round (with the result being unknown to either party) until the result is heads. If the result is heads, then from this round onward the parties learn the real output. Otherwise, the result is tails, parties receive a random backup value (which is indistinguishable from a "real" output), and proceed to do the same thing in the next round.

Π_{GHKL}

Inputs: Party P_1 has input $x \in X$ and party P_2 has input $y \in Y$. The security parameter is n.
Computation:
Let $r := r(n)$ be the number of rounds.

1. Parties P_1 and P_2 invoke a secure-with-abort MPC protocol for the functionality $\mathsf{ShareGen}_{\mathsf{GHKL}}$ (Figure 3.3) with their inputs x and y.
2. If $\mathsf{ShareGen}_{\mathsf{GHKL}}$ returns $\perp$ to P_1, it outputs $f(x, \hat{y})$ for $\hat{y} \leftarrow_u Y$. In the same situation, P_2 outputs $f(\hat{x}, y)$ for $x \leftarrow_u X$.
3. Otherwise, each party receives a secret share[a] of each of the values a_i and b_i for $i \in [r]$ (together with signatures where applicable). In addition, parties receive a public key pk.
4. For rounds $i = 1, ..., r$:
   (a) P_2 sends its signed share of a_i to P_1, who uses pk to verify the signature and, if its valid, restores a_i. Otherwise, P_1 outputs a_{i-1}.
   (b) P_1 sends its signed share of b_i to P_2, who uses pk to verify the signature and, if its valid, restores b_i. Otherwise, P_2 outputs b_{i-1}.
5. If no party aborted, then P_1 outputs a_r and P_2 outputs b_r

[a] A single share of a value exposes no information about it. By obtaining both shares of a value, a party may learn the value itself.

FIGURE 3.2: Protocol for two parties for computing the GHKL function.

ShareGen$_{\mathsf{GHKL}}$

Inputs: Party P_1 has input $x \in X$ and party P_2 has input $y \in Y$.
computation:

1. Define values $a_1, ..., a_r$ and $b_1, ..., b_r$ in the following way:
   - Sample $i^* > 0$ from the geometric distribution with parameter α.
   - For $i = 0, ..., r$ compute:

   $$a_i = \begin{cases} f(x, \hat{y}), & \text{if } i < i^* \\ f(x, y), & \text{otherwise} \end{cases}$$

   and

   $$b_i = \begin{cases} f(\hat{x}, y), & \text{if } i < i^* \\ f(x, y), & \text{otherwise} \end{cases}$$

   where $\hat{x} \leftarrow_u X$ and $\hat{y} \leftarrow_u Y$ are sampled independently for each i.
2. For $1 \leq i \leq r$, compute a random secret sharing of a_i and b_i, generate signing and verification keys $(\mathsf{sk}, \mathsf{pk})$ and sign P_1's shares of b_i and P_2's shares of a_i with sk.

Output: Each party gets its shares of a_i and b_i (with signatures where applicable) for $i \in \{1, ..., r\}$ and the public key pk. In addition, P_1 receives a_0 and P_2 receives b_0.

FIGURE 3.3: The dealer functionality for the GHKL protocol.

If the adversary sends $x' = x_2$ to ShareGen$_{\mathsf{GHKL}}$ (which it does in both worlds since its view equals the empty set at this point of the execution), then all the backup values that the adversary observes in the real run are equal to 1. Thus, it never learns anything. Thus, we only discuss adversaries that send $x \in \{x_1, x_2\}$ ShareGen$_{\mathsf{GHKL}}$ (in both worlds).

When a party aborts in round $i < i^*$, both parties have learned nothing, and if it aborts in round $i > i^*$, then both output $f(x, y)$. The problematic case is when P_1, who learns information first in every round, aborts exactly in round i^*. A key observation is that, since i^* is sampled from the geometric distribution, the information observed so far by the adversary does not help it guess i^*. So, even if the adversary aborts in round i^*, it does not know that it has.

Let $\mathcal{A}$ be an adversary corrupting P_1 and aborting in round $i \leq i^*$ and $\mathcal{S}$ be the corresponding adversary in the ideal world. We proceed to discuss how the view of $\mathcal{A}$ and the output of the honest party can be made to distribute the same way in both worlds. Before round i^*, the simulator $\mathcal{S}$ can generate the view of $\mathcal{A}$ without any interaction with the TP. To simulate the view of $\mathcal{A}$ in round i^*, the simulator $\mathcal{S}$ sends x to learn $f(x, y)$ and sends it to $\mathcal{A}$. Thus, if $\mathcal{A}$ aborts in round i^*, the honest party outputs a random backup value in the hybrid world and the real output in the ideal world, and it holds that

$$\Pr[\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}} \mid i = i^*] \neq \Pr[\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}} \mid i = i^*] \tag{3.1}$$

Where $\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}, \mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}}$ are shortened notations for the view and output in the hybrid and ideal worlds, respectively.

However, since i^* is unknown, both $\mathcal{A}$ and P_2 cannot distinguish between the cases when $\mathcal{A}$ aborts in round i^* and in round $i < i^*$. Therefore, it is possible to compensate for Equation 3.1 by controlling z carefully so that the following holds

$$\Pr[\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}} \mid i \leq i^*] = \Pr[\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}} \mid i \leq i^*]$$

3.4 Intuition on Why the GHKL Function Can Be Computed with Fairness, while the XOR Function Cannot

We borrow the following from [2]. Remember that the XOR function is impossible to compute with fairness according to [6], whereas the GHKL function is possible to compute with fairness using the GHKL protocol. Denote the XOR function and the GHKL function by f_{XOR} and f_{GHKL} respectively. The matrix representations for both functions are given in Figure 3.4.

	y_1	y_2
x_1	0	1
x_2	1	0

	y_1	y_2
x_1	0	1
x_2	1	0
x_3	1	1

FIGURE 3.4: The XOR function (left) and the GHKL function (right).

We can view the "freedom" of the simulator by how much it can control the output distribution vector of the honest party. Informally, the output distribution vector $\mathbf{v}$ of a party P in the ideal world is a vector containing the probability that P outputs 1 for different inputs, when the other party sends inputs to the TP with certain probabilities (we refer the reader to Section 2.7 for a formal definition). The reason that the XOR function cannot be simulated whereas the GHKL function can is that with the latter, the simulator has more power in the ideal world than with the former and can therefore simulate the influence of the adversary on the output of the honest party.

Assume w.l.o.g. that the adversary always corrupts the first party. Any simulator for a protocol computing the XOR function sends $x' = x_1$ with some probability p and $x' = x_2$ with probability $1 - p$ to the TP. The output distribution of the honest party is

$$\mathbf{k}_{\mathsf{XOR}} = \begin{bmatrix} \Pr[f_{\mathsf{XOR}}(x', y_1) = 1] \\ \Pr[f_{\mathsf{XOR}}(x', y_2) = 1] \end{bmatrix} = \begin{bmatrix} 1 - p \\ p \end{bmatrix}$$

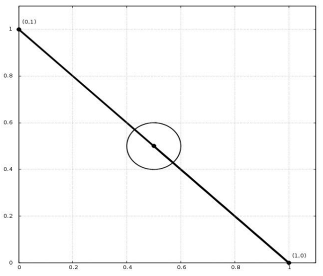
which can be seen geometrically as a line as the simulator has only one degree of freedom.

Any simulator for the GHKL protocol sends $x' = x_1$ to the TP with probability p and $x' = x_2$ with probability q, so that it sends $x' = x_3$ to the TP with probability $(1 - p - q)$. The output distribution of the honest party in such case is

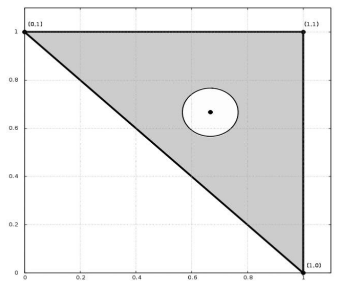
$$\mathbf{k}_{\mathsf{GHKL}} = \begin{bmatrix} \Pr[f_{\mathsf{GHKL}}(x', y_1) = 1] \\ \Pr[f_{\mathsf{GHKL}}(x', y_2) = 1] \end{bmatrix} = \begin{bmatrix} 1 - p \\ 1 - q \end{bmatrix}$$

which allows the simulator two degrees of freedom.

Thus, the potential output distribution vectors for the XOR function when viewed as a geometric object are a line between $(0, 1)$ and $(1, 0)$, whereas for the GHKL function, it can be viewed a triangle between $(0, 1)$, $(1, 0)$ and $(1, 1)$.



(A) The potential output distribution vectors of the XOR function: a line segment between $(0, 1)$ and $(1, 0)$.



(B) The potential output distribution vectors of the GHKL function: a triangle between $(0, 1)$, $(1, 0)$ and $(1, 1)$.

FIGURE 3.5: The geometric objects defined by the XOR function and the GHKL function

The GHKL protocol is constructed in such way that the output of the honest party is strictly inside the triangle when the adversary aborts. Thus, the simulator in this case can send corresponding inputs to the TP that make the output of the honest party distribute identically to its output in the real world for any given view of the adversary (even when compensating for the fact that in some cases the honest party receives the real output only in the ideal world). With the XOR function, on the other hand, if the output of the honest party in the real world is not on the line, then it is impossible to make its output distribute the same way in the ideal world.

3.5 Complete Characterization of Fairness in Secure Two-Party Computation of Boolean Functions

Asharov, Beimel, Makriyannis, and Omri [3] gave a complete characterization of Boolean functions that can be computed with full security (i.e., with fairness and guaranteed output delivery).

The characterization is the following. A function $f : X \times Y \to \{0, 1\}$, described by the matrix M_f (i.e., $M_f[x, y] = f(x, y)$ for $x \in X$, $y \in Y$) can be computed with full security if and only if the all one vector is in the *affine span* of either the rows or of the columns of M_f. Recall that the *affine span* of a set of vectors $\mathbf{v}_1, ..., \mathbf{v}_n$ is the set

$$\left\{ \mathbf{v} : \mathbf{v} = \alpha_1 \cdot \mathbf{v}_1 + \cdots + \alpha_n \cdot \mathbf{v}_n, \sum_{i \in [n]} \alpha_i = 1 \right\}$$

On the negative side, [3] show that a function which does not satisfy the above property is reducible to the *fair sampling* problem[1], which was shown by [1] to be impossible to compute with fairness.

On the positive side, Asharov et al. [3] construct a protocol, which they call $\mathsf{FairTwoParty}_\sigma$, which is secure with fairness and guaranteed output delivery for any

[1]A functionality is said to imply fair sampling when it allows parties to learn correlated bits for some non-trivial correlation (their output doesn't have to be the same)

function satisfying the property given in the characterization. $\mathsf{FairTwoParty}_\sigma$, which is parameterized with a bit σ, is identical to the GHKL protocol, except for a slight modification to the $\mathsf{ShareGen}_{\mathsf{GHKL}}$ functionality (we refer the reader to Section 3.3 for a full description of the GHKL protocol). Specifically, the backup value b_{i^*-1} of P_2 for round $i^* - 1$ is set to σ. We present the modified $\mathsf{ShareGen}_{\mathsf{GHKL}}$ functionality, which is called $\mathsf{ShareGen}_\sigma$, in Figure 3.6.

$\mathsf{ShareGen}_\sigma$

Inputs: Party P_1 has input $x \in X$ and party P_2 has input $y \in Y$.
computation:

1. Define values $a_1, ..., a_r$ and $b_1, ..., b_r$ in the following way:
   - Choose $i^* \leftarrow \mathsf{Geom}(\alpha)$.
   - For $i = 1, ..., r$ compute

$$a_i = \begin{cases} f(x, \hat{y}), & \text{if } i < i^* \\ f(x, y), & \text{otherwise} \end{cases}$$

   and

$$b_i = \begin{cases} f(\hat{x}, y), & \text{if } i < i^* - 1 \\ \sigma, & \text{if } i = i^* - 1 \\ f(x, y), & \text{otherwise} \end{cases}$$

   where $\hat{x} \leftarrow_u X$ and $\hat{y} \leftarrow_u Y$ are sampled independently for each i.

2. For $1 \le i \le r$, compute a random secret sharing of a_i and b_i. Then, generate signing and verification keys $(\mathsf{sk}, \mathsf{pk})$ and sign P_1's shares of b_i and P_2's shares of a_i with sk.

Output: Each party gets its shares of a_i and b_i (with signatures where applicable) for $i \in [r]$ and pk.

FIGURE 3.6: The dealer functionality for the $\mathsf{FairTwoParty}_\sigma$ protocol.

This change makes it so that in round i^*, the backup value b_{i^*-1}, which is the output of the "weaker" party P_2, is not correlated to its input y. Otherwise, supposing that $\mathcal{A}$ guesses i^*, it has the ability to bias the output of certain functions by guessing y from $a_i = f(x, y)$ and deciding whether to abort P_1 depending on its value. Such functions are impossible to compute by the GHKL protocol because the adversary has control over P_2's output after it has learned some information about P_2's input.

The proof of security for the $\mathsf{FairTwoParty}_\sigma$ protocol is similar to that of the GHKL protocol. That is, inputs to the TP are shown to exist in rounds $i < i^*$ such that

$$\Pr[\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}} \mid i \le i^*] = \Pr[\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}} \mid i \le i^*]$$

even though

$$\Pr[\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}} \mid i = i^*] \ne \Pr[\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}} \mid i = i^*]$$

3.6 Fair computation of Majority for three parties

The following result of Gordon and Katz [11] shows the possibility of fair computation in the setting of three parties and a dishonest majority. Namely, they show that the three party Majority function[2] can be computed with fairness and guaranteed output delivery.

They prove security for the following protocol Π_{maj} (Figure 3.7) that accepts a security parameter n and works roughly as follows. Parties P_1,P_2, and P_3, invoke a secure-with-designated-abort [3] protocol π implementing the functionality $\mathsf{ShareGen}_{\mathsf{maj}}$ (Figure 3.8) with their inputs x_1, x_2 and x_3 resp. Most notably, $\mathsf{ShareGen}_{\mathsf{maj}}$ selects a round number $i^* \leftarrow \mathsf{Geom}(\alpha)$ similarly to $\mathsf{ShareGen}_{\mathsf{GHKL}}$ of [10] and generates the backup values $b_j^{(i)}$ for every pair $\mathsf{P}_k, \mathsf{P}_l$, where $k, l \in \{1,2,3\} \setminus \{j\}$ as follows. In rounds $i < i^*$ the backup $b_j^{(i)}$ is independent of x_j, and in rounds $i \geq i^*$, $b_j^{(i)} = \mathsf{maj}(x_1, x_2, x_3)$. After receiving their shares, parties perform the following computation for $m = m(n)$ rounds. In round i, P_j sends its share of $b_j^{(i)}$ to one of the other parties, so that they learn a 2-out-of-2 sharing of $b_j^{(i)}$. In case P_j aborts in round i, the pair $\mathsf{P}_k, \mathsf{P}_l$, where $k, l \in \{1,2,3\} \setminus \{j\}$, restores $b_j^{(i-1)}$ and outputs it. If two parties abort (either one after the other or at the same time), the remaining party P_j outputs x_j. If no party aborts, all parties restore $b_1^{(m)}$ and output it.

[2]The three party Majority function accepts three inputs in $\{0,1\}$ and outputs the majority of inputs. e.g., $\mathsf{maj}(0,0,1) = 0$.

[3]Secure-with-designated-abort roughly means that privacy and correctness always hold and that the adversary can only abort and prevent fairness if P_1 is corrupted, and, if it is not, then guaranteed output delivery holds.

Π_{maj}

Inputs: Parties $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ hold inputs x_1, x_2, x_3, respectively.
Computation:

1. **Preliminary phase:**
   - Parties invoke a secure-with-designated-abort protocol π implementing $\mathsf{ShareGen}_{\mathsf{maj}}$ with their inputs and the security parameter.
   - If P_2 and P_3 receive $\perp$ form $\mathsf{ShareGen}_{\mathsf{maj}}$, they run a two-party protocol π_{OR} for computing the OR function (which we assume we have).
2. **For** $i = 1, ..., m-1$ **do:**
   (a) Each party P_j broadcasts its share of $b_j^{(i)}$.
   (b) If only P_j aborts[a], P_j and P_l, where $k, l \in \{1,2,3\} \setminus \{j\}$, restore $b_j^{(i)}$ and output it.
   (c) If two parties abort, the remaining party outputs its own input.
3. **In round** $i = m$ **do:**
   (a) Each party P_j broadcasts its share of $b_1^{(m)}$.
   (b) If one party aborts, the remaining parties act as in step 2b.
   (c) If two parties abort, the remaining party acts as in step 2c.

[a] A party is said to abort if it doesn't send a share with a valid signature.

FIGURE 3.7: A protocol for computing the Majority function.

$\mathsf{ShareGen}_{\mathsf{maj}}$

Inputs: Parties $\mathsf{P}_1, \mathsf{P}_2, \mathsf{P}_3$ hold inputs x_1, x_2, x_3, respectively.
Computation:

1. Choose $i \leftarrow \mathsf{Geom}(1/5)$.
2. Define the following values for $i \in [m], j \in \{1, 2, 3\}$:

$$b_j^{(i)} = \begin{cases} \mathsf{maj}(\hat{x}_1, \hat{x}_2, \hat{x}_3) & \text{, if } i < i^* \\ \mathsf{maj}(x_1, x_2, x_3), & \text{otherwise} \end{cases}$$

   where $\hat{x}_k \leftarrow \{0, 1\}$ if $k = j$ and $\hat{x}_k = x_k$ for $k \in \{1, 2, 3\} \setminus \{j\}$.

3. For every value $b_j^{(i)}$, compute a 3-out-of-3 secret sharing.
4. Generate a secret key sk and a public key pk and sign all shares with sk.

Output:
Send each party its share of each value together with signatures and pk.

FIGURE 3.8: Share generator for the Π_{maj} protocol.

In order to give some intuition on the proof of security for Π_{maj}, let $\mathcal{A}$ be a rushing adversary corrupting P_1 and P_2 and attacking Π_{maj} in the hybrid world, and let $\mathcal{S}$ be the the corresponding simulator for $\mathcal{A}$ in the ideal world. We denote by x_1,x_2 the inputs used by $\mathcal{A}$ as the effective inputs of P_2 and P_3. In case $x_1 = x_2$, the whole view of $\mathcal{A}$ before it aborts one of the parties is independent of x_3 in the hybrid world. On the other hand, in case $x_1 \neq x_2$ then $\mathcal{A}$ learns the output $x_3 = \mathsf{maj}(x_1, x_2, x_3)$ in round i^* in the hybrid world. In this case in the ideal world, $\mathcal{S}$ sends the inputs (x_1, x_2) to the TP, so P_3 outputs x_3. In the hybrid world, if $\mathcal{A}$ aborts only P_1 in round i^* (w.l.o.g. the same applies to P_2 by symmetricity), it can make P_3 output $\bar{x}_3$ with some probability (we denote $\bar{x} = 1 - x$ for any $x \in \{0, 1\}$). Thus, when $x_1 \neq x_2$, $\mathcal{S}$ has to compensate for the fact that in some cases

$$\Pr[\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}} \mid i = i^*] \neq \Pr[\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}} \mid i = i^*] \tag{3.2}$$

Another important consideration is that when $\mathcal{A}$ aborts P_1, it observes the backup value $b_1^{(i-1)} = \mathsf{maj}(\hat{x}_1, x_2, x_3)$. In this case, $\mathcal{A}$ can decide whether P_3 outputs $b_1^{(i-1)}$ or x_3 by controlling whether P_2 aborts or not, respectively. An important property of the protocol (that is possible thanks to the unique structure of the Majority function) is that when $\hat{x}_1 = x_2$, then $b_1^{(i-1)} = x_2$ and $b_1^{(i-1)}$ independent of x_3, and when $\hat{x}_1 \neq x_2$, then $b_1^{(i-1)} = x_3$. In the latter case (when $\hat{x}_1 \neq x_2$) since P_3 outputs x_3 when the second party aborts, whether the second party aborts does not affect the output of P_3. Had this not been the case, $\mathcal{A}$ would have been able to learn x_3 and, after this, decide the output of P_3, which would make the proof of security impossible. The complete proof of security is rather involved and requires careful control of the backup $b_1^{(i-1)}$ and the inputs to the TP in order to compensate for (3.2).

3.7 Fair computation of Boolean OR for *n* parties

The following result is also due to [11]. It shows that the n-party OR function can be computed with fairness and guaranteed output delivery in the face of up to $n-1$ corrupted parties. Specifically, [11] present a protocol Π_{OR} for the OR (Figure 3.9) and prove security for it.

The outline of Π_{OR} is as follows. Parties in $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ begin by committing to their inputs and broadcasting their commitment, where each party P_i saves a copy of all other parties' commitments $\mathbf{c}_i$. Then, the computation proceeds in iterations, where in each iteration, parties either compute the OR of their inputs or some parties are removed from the computation and the rest proceed to attempt again, where the inputs of the removed parties are set to 0. In more detail, parties invoke a protocol that computes $\mathsf{CommittedOR}_{\mathcal{P}}$ with designated abort, which means that only the party with the lowest index can initiate an abort after learning the output. The $\mathsf{CommittedOR}_{\mathcal{P}}$ functionality checks that the parties agree on the vector of commitments and that the commitments match the respective inputs of the parties. In case parties disagree on commitments or inputs don't match commitments, each party gets a list of names of those who disagreed with it, so that it can continue the computation with the ones that agree with it. If parties agree on the commitments, then $\mathsf{CommittedOR}_{\mathcal{P}}$ computes the OR of their committed inputs (where the lowest indexed party can still abort after learning the output).

Π_{OR}

Inputs: Each party P_i holds input in $x_i \in \{0, 1\}$. The security parameter is k.
Computation:

1. Let $\mathcal{P}$ be the set of all parties.
2. Each party P_i computes a commitment $c_i = \mathsf{Com}(x_i, r_i)$ and broadcasts its commitment. P_i saves the commitments it received as $\mathbf{c}_i = (c_1, ..., c_n)$. If some party does not broadcast a valid commitment, all other parties output 1.
3. Parties $\mathsf{P}_i \in \mathcal{P}$ run a secure with identifiable abort protocol $\pi_{\mathcal{P}}$ for computing the functionality $\mathsf{CommittedOR}_{\mathcal{P}}$ (figure 3.10), where P_i inputs x_i, r_i and $\mathbf{c}_i$.
4. If parties receive $\perp$ from $\pi_{\mathcal{P}}$, they set $\mathcal{P} = \mathcal{P} \setminus \mathsf{P}^*$ where P^* is the lowest-indexed party in $\mathcal{P}$ and return to step 3.
5. If parties receive a set $\mathcal{D}$ from $\pi_{\mathcal{P}}$, they set $\mathcal{P} = \mathcal{P} \setminus \mathcal{D}$ and return to step 3.
6. If parties receive $\mathsf{out} \in \{0, 1\}$from $\pi_{\mathcal{P}}$, they output out.

FIGURE 3.9: A protocol computing OR for n parties.

$\mathsf{CommittedOR}_{\mathcal{P}}$

Inputs: Let $\mathcal{P}$ be the set of invoking parties and the input of $\mathsf{P}_i \in \mathcal{P}$ be $(x_1, r_i, \mathbf{c}^i)$ where $\mathbf{c}^i = (c^i_j)_{j:P_j \in \mathcal{P}}$.
Determine the output for each party $\mathsf{P}_i \in \mathcal{P}$ **as follows:**

- Say P_j disagrees with P_i if either (1) $\mathbf{c}^j \neq \mathbf{c}^i$ or (2) $c^i_j \neq \mathsf{Com}(x_j, r_j)$.
- Let $\mathcal{D}_i$ be the set of parties who disagree with P_i.
- If there exists a set of parties $\mathcal{D}_i$ that disagree with P_i, return $\mathcal{D}_i$ as output to P_i. Otherwise, return $\bigvee_{j:P_j \in \mathcal{P}} x_j$ to all parties.

FIGURE 3.10: Functionality $\mathsf{CommittedOR}_{\mathcal{P}}$, parameterized by a set $\mathcal{P}$.

Intuitively, this protocol is secure with fairness because when the party with the lowest index P^* aborts $\mathsf{CommittedOR}_{\mathcal{P}}$, it has only learned something if its input is 0. In this case, by setting the input of P^* to 0 and attempting the computation again, the remaining parties learn the same thing.

The OR can be seen as a special case of the Maximum function, where the input domain is $\{0, 1\}$. In our work, we extend the above intuition to allow secure computation for the maxi function for n parties. This result is described in Section 5.

Chapter 4

Computing *n* Party Maximum with Complete Fairness

In this chapter, we show possibility for computing the Maximum function with fairness and guaranteed output delivery in the setting of l parties and up to $l-1$ corruptions. Define the maximum function $\mathsf{max} : \{0, ..., m\}^l \to \{0, ..., m\}$ as follows:

$$\mathsf{max}(x_1, ..., x_n) := x_i \text{ such that } x_i \geq x_j \text{ for all } j \in [n]$$

where $m = m(n)$ is polynomial in the security parameter n.

4.1 Our protocol

The main idea behind our protocol is that parties learn the maximum in rounds, starting from round 0, where in round i, they find out whether the maximum is i (implying that are at most m rounds). If they learn that the maximum is i, then the computation ends; otherwise, they continue to the next round. If some parties abort in round i, the remaining parties continue the computation from round i, this time computing the maximum without the inputs of the aborted parties. In case i is larger than the inputs of the remaining parties, then i is assumed to be the maximum. Intuitively, fairness holds because if some party finds out that i is the maximum and aborts, then the remaining parties already know that the maximum is at least i.

The complete description of the protocol is somewhat more involved, where, among other minor technical details, commitments are used to guarantee that parties cannot modify their inputs every time some parties abort, in a similar fashion to [11].

Protocol for computing the max function

Inputs: Let $\mathcal{P}$ be the set of invoking parties and n be the security parameter. Every party $\mathsf{P}_j \in \mathcal{P}$ holds an input $x_j \in \{0, ..., m\}$, where $m = m(n)$.

1. **Initialization:**
   (a) Each party $\mathsf{P}_i \in \mathcal{P}$ generates randomness r_i, computes a commitment $k_i = \mathsf{Com}(r_i, x_i)$[a] and broadcasts k_i. Additionally, each party $\mathsf{P}_i \in \mathcal{P}$ initializes a value $t_i = 0$ (which will hold the highest round reached in the computation).
   (b) In case some party aborts by not broadcasting a commitment, the other parties output m and halt.
2. **Preprocessing and input checks:**
   (a) Parties invoke $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$, where each $\mathsf{P}_i \in \mathcal{P}$ inputs x_i, r_i, t_i and the commitments of parties in $\mathcal{P}$ that were broadcast at the start of the execution, $\mathbf{k}_i = (k_j)_{j:\mathsf{P}_j \in \mathcal{P}}$, (note that $\mathbf{k}_i$ and t_i are the same for every honest party P_i).
   (b) If P_j receives a set of names $\mathcal{D}$ from $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$, then it sets $\mathcal{P} = \mathcal{P} \backslash \mathcal{D}$ (note that $\mathcal{P}$ is the same for all honest parties). If $|\mathcal{P}| > 1$, parties in $\mathcal{P}$ restart step 2, and, if only P_j remains, it outputs $\mathsf{max}\{t_j, x_j\}$.
   (c) Otherwise, parties get signed shares of c_i for $i \in \{t, ..., m\}$ and the public key pk, and continue to the next step.
3. **Parties perform computation in rounds $i = t, ..., m$:**
   (a) Every party $\mathsf{P}_j \in \mathcal{P}$, broadcasts its signed share of c_i.
   (b) If all parties broadcast shares with valid signatures, then every party P_j restores c_i and does the following:
      i. If $c_i = 0$, continue to the next round.
      ii. If $c_i = 1$, output i and halt.
   (c) If some parties abort by not broadcasting shares with valid signatures:
      i. If only P_j remains, it outputs $\mathsf{max}\{i, x_j\}$.
      ii. Otherwise, let $\mathcal{D}$ be the set of aborted parties. Remaining parties update $\mathcal{P} = \mathcal{P} \setminus \mathcal{D}$, and go to Step 2, where $\mathsf{P}_j \in \mathcal{P}$ sets $t_i = i$.

[a] Com is a computationally secure commitment scheme.

FIGURE 4.1: Protocol for n parties for computing the maximum over their inputs.

$\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$

Tools: A signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and a commitment scheme Com.
Inputs: Let $\mathcal{P}$ be the set of invoking parties, where $l = |\mathcal{P}|$. Party $\mathsf{P}_i \in \mathcal{P}$ holds input x_i, randomness r_i, t_i and a vector of commitments $\mathbf{k}_i$.
Computation:

1. Check agreement and inputs:
   (a) We say that party P_i disagrees with party P_j if one of the following occurs, (1) $\mathbf{k}_i \neq \mathbf{k}_j$, (2) $t_i \neq t_j$.
   (b) If some parties disagree with each other, let $\mathcal{D}_i$ be the set of parties who disagree with party P_i. Send $\mathcal{D}_i$ to P_i as its output and halt.
   (c) Let $\mathbf{k}$ be the commitment vector that parties agreed upon, and $k_i \in \mathbf{k}$ be P_i's commitment. Set $\mathcal{D} = \{\mathsf{P}_j : \mathsf{Com}(x_j, r_j) \neq k_j\}$. If $\mathcal{D} \neq \emptyset$, send $\mathcal{D}$ to parties in $\mathcal{P} \setminus \mathcal{D}$ and halt.
2. Let t be the value agreed upon. Compute the following values.
   (a) Compute $z = \mathsf{max}\{t, h\}$, where $h = \mathsf{max}\{x_j\}_{j:\mathsf{P}_j \in \mathcal{P}}$.
   (b) Compute the following values for $i \in \{t, ..., m\}$:

$$c_i = \begin{cases} 0, & \text{for } i < z \\ 1, & \text{otherwise} \end{cases}$$

3. Compute a l-out-of-l secret sharing of c_i for every $i \in \{t, ..., m\}$.
4. Generate secret and public keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}$. Sign every share using sk.

Output:
Party P_j receives its signed shares of c_i for $i \in \{t, ..., m\}$ and the public key pk (parties are assumed to know the algorithm Verify).

FIGURE 4.2: The functionality ShareGen which is used by the protocol for max function.

4.2 Proof of security

We will prove the following.

Theorem 4.2.1. *Assume that there exists a protocol π that securely computes $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$ (Figure 4.2) with identifiable abort. Then, the function* max *can be computed with fairness and guaranteed output delivery in the multiparty setting against a static, non-uniform, polynomial-time, malicious adversary corrupting up to $n-1$ parties.*

Proof. Towards proving Theorem 4.2.1, we prove security against fail-stop adversaries and in the hybrid world with access to an ideal functioanlity computing $\mathsf{ShareGen}_{\mathsf{max}}$ (with identifiable abort). The probability of a computationally bounded adversary forging a signature or a commitment is negligible, and, since every failed attempt at forging a signature or a commitment is interpreted as an abort, Claim 4.2.1 together with the composition theorem of Canetti [5] yield Theorem 4.2. In addition, we assume that $\mathcal{A}$ is rushing, which informally means that $\mathcal{A}$ learns all information in any single round before honest parties do.

Claim 4.2.1. *Consider executions of Protocol* Π *(Figure 4.1) in a hybrid model with access to the ideal functionality computing* $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$ *(shortened to* SG*) with identifiable abort. For every static, non-uniform, polynomial-time, fail-stop adversary* $\mathcal{A}$ *that corrupts a subset of the parties of size up to* $n-1$ *and attacks* Π*, there exists a poly-time adversary (simulator)* $\mathcal{S}$*, corrupting the same subset of parties and running in the ideal world with access to an ideal functionality computing* max *(with complete fairness), such that the following holds.*

$$\left\{ \left(\mathsf{Out}^{\mathsf{Hybrid}^{\mathsf{SG}}}_{\mathcal{A}(z),\Pi}, \mathsf{View}^{\mathsf{Hybrid}^{\mathsf{SG}}}_{\mathcal{A}(z),\Pi}\right)(1^n, x_1, ..., x_m)\right\}_{(x_1,...,x_m)\in X, z\in\{0,1\}^*, n\in\mathbb{N}}$$
$$\equiv \left\{ \left(\mathsf{Out}^{\mathsf{Ideal}}_{\mathcal{S}(z),f}, \mathsf{View}^{\mathsf{Ideal}}_{\mathcal{S}(z),f}\right)(1^n, x_1, ..., x_m)\right\}_{(x_1,...,x_m)\in X, z\in\{0,1\}^*, n\in\mathbb{N}}$$

where the notation is as defined in section 2.5.

Proof. We shorten the notation of the view and output and omit the parts that are clear from the context, which are the inputs, $\mathcal{S}(z)$, f, $\mathcal{A}(z)$, and Π. Thus, we will prove the following

$$(\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}) \equiv (\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}}) \tag{4.1}$$

We proceed to define a simulator $\mathcal{S}$, which uses an adversary $\mathcal{A}$ as a black box. Let y be the maximum of the inputs of the remaining corrupted parties (that is, the values they are committed to). Before reaching round y of the computation for the first time, all information observed by the adversary $\mathcal{A}$ is independent of the inputs of the honest parties (by the hiding property of commitments), and $\mathcal{S}$ can generate it without interaction with the TP. The first time round $i = y$ is reached, $\mathcal{A}$ learns whether y is the maximum or not in the hybrid run, so $\mathcal{S}$ invokes the TP to learn z and generates the rest of the view of $\mathcal{A}$ using z.

Simulator $\mathcal{S}$ for $\mathcal{A}$ attacking Π_{max}:
Let $\mathcal{P}$ be the initial set of parties, let $\mathcal{C}$ be the set of parties corrupted by $\mathcal{A}$, let $\mathcal{H}$ be the set of honest parties, and let $\mathcal{I} \subseteq \mathcal{C}$ be the set of corrupted parties that remain in the protocol. Initially, $\mathcal{C} = \mathcal{I}$.

1. $\mathcal{S}$ initializes $\mathcal{A}$ on randomness r, security parameter n, inputs $\{x_i\}_{i:\mathsf{P}_i\in\mathcal{C}}$ and auxiliary input z.
2. **Simulate initialization:**
   (a) Receive commitments of corrupted parties from $\mathcal{A}$ and send commitments of 0 to $\mathcal{A}$ behalf of honest parties. Save all commitments as $\mathbf{k} = (k_j)_{j:\mathsf{P}_j\in\mathcal{P}}$. If some party does not send a commitment, send m to the TP on behalf of all corrupted parties. Output the view of $\mathcal{A}$ and halt.
   (b) Initialize $t \leftarrow 0$.
3. **Simulate $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$:**
   (a) If $\mathcal{I} = \emptyset$, send t to the TP on behalf of all corrupted parties, output the view of $\mathcal{A}$ and halt.

(b) Otherwise, let $\mathcal{P} = \mathcal{I} \cup \mathcal{H}$ and $\mathbf{k} = (k_j)_{j:\mathsf{P}_j \in \mathcal{P}}$.

(c) Receive inputs to $\mathsf{ShareGen}_{\mathsf{max}}^{\mathcal{P}}$ from $\mathcal{A}$, which include x'_i, r_i $\mathbf{k}_i$ and t_i, where $i : \mathsf{P}_i \in \mathcal{I}$.

(d) We say a party $\mathsf{P}_i \in \mathcal{P}$ disagrees with party $\mathsf{P}_j \in \mathcal{P}$ if $\mathbf{k}_i \neq \mathbf{k}_j$ or $t_i \neq t_j$, where $\mathbf{k}_a = \mathbf{k}$ and $t_a = t$ for $\mathsf{P}_a \in \mathcal{H}$ (we note that there can only be disagreement between two parties when one of them is in $\mathcal{I}$).

(e) If there is some disagreement, let $\mathcal{D}_j := \{\mathsf{P}_i : \mathsf{P}_i \in \mathcal{P} \text{ disagrees with } \mathsf{P}_j \in \mathcal{P}\}$. Send $\mathcal{D}_j$ to $\mathcal{A}$ for all $j : \mathsf{P}_j \in \mathcal{I}$. Finally, update $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}_i$ for some $i : \mathsf{P}_i \in \mathcal{H}$ (note that $\mathcal{D}_i$ is the same for all $i : \mathsf{P}_i \in \mathcal{H}$), and go to Step 3.

(f) If there is no disagreement, let $\mathcal{D} = \{\mathsf{P}_i : \mathsf{Com}(x'_i, r_i) \neq k_i, \mathsf{P}_i \in \mathcal{I}, k_i \in \mathbf{k}\}$. If $\mathcal{D} \neq \emptyset$, update $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}$, send $\mathcal{D}$ to $\mathcal{A}$ if $\mathcal{I} \neq \emptyset$, and go to Step 3.

(g) Give $\mathcal{A}$ the public key pk and signed shares of the value 0 which correspond to shares of c_i for $i \in \{t, ..., m\}$.

(h) If $\mathcal{A}$ sends abort to $\mathsf{ShareGen}_{\mathsf{max}}^{\mathcal{P}}$ at some point, let $\mathcal{D}$ be the set of names that $\mathcal{A}$ specifies (remember that $\mathsf{ShareGen}_{\mathsf{max}}^{\mathcal{P}}$ is secure with identifiable abort). Update $\mathcal{I} \setminus \mathcal{D}$ and go to Step 3.

4. If no inputs were sent to the TP, compute $z \leftarrow \max\{h, t\}$, where $h \leftarrow \max\{x'_i\}_{i:\mathsf{P}_i \in \mathcal{I}}$.

5. **In rounds** $i = t, ..., m$ (remember that computation starts from round t):

(a) If $i = z$ and no inputs were sent to the TP, send i to the TP on behalf of all corrupted parties and save output as z.

(b) Send c_i[1] to $\mathcal{A}$, where

- If $i < z$, $c_i = 0$.
- If $i = z$, $c_i = 1$.

(c) If some (possibly all) parties abort[2], let $\mathcal{D}$ be the set of aborted parties. Set $t = i$, update $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}$, and then go to Step 3.

(d) If $i = z$ (note that this step is only reached after inputs have been sent to the TP) and no parties abort, output the view of $\mathcal{A}$ and halt.

In our analysis, we ignore random shares, signatures and commitments in the view of $\mathcal{A}$ and assume that its behavior is independent of them. It is easy to see that if $\mathcal{A}$ uses any of those to break the security of Π, then this contradicts the security of the signature and commitments schemes that we use, or the properties of secret sharing schemes. Thus, we will show that the view of $\mathcal{A}$, without the above, and the output of the honest parties are identical in both runs, which will imply that they distribute identically in total (remember that we are assuming that $\mathcal{A}$ is fail-stop).

In what follows, we observe two parallel executions, where $\mathcal{A}$ provides the same inputs, and show that the view of $\mathcal{A}$ remains the same throughout and that the honest parties learn the same output. Since $\mathcal{S}$ is constructed to closely imitate the protocol and $\mathcal{A}$ behaves the same way in both worlds, when conditioned on the same view, the value t (that is held by $\mathcal{S}$ in the ideal world and by the honest parties in the hybrid world) and the set of remaining corrupted parties are the same in both worlds throughout the execution, so we refer to them as t and $\mathcal{I}$.

[1]When we say that we send $c_i$to $\mathcal{A}$, we mean that we send the corresponding shares with signatures that complete the shares of corrupted parties to c_i

[2]A party is said to abort when $\mathcal{A}$ does not send the corresponding share with a valid signature on its behalf.

During the initialization phase, $\mathcal{A}$ only sees commitments. In addition, if some malicious party aborts, then the honest parties output m in both worlds and the claim holds. Otherwise, $\mathcal{A}$ commits to the same inputs in both worlds, and we proceed to analyze the view of $\mathcal{A}$ throughout the execution. In the preprocessing and input check phase, because $\mathcal{S}$ closely imitates $\mathsf{ShareGen}_{\mathsf{max}}$, it is easy to see that the disagreement lists generated by $\mathcal{S}$ are the same as those generated by $\mathsf{ShareGen}_{\mathsf{max}}$ in case $\mathcal{A}$ deviates from the protocol (ignoring the possibility that $\mathcal{A}$ forges a commitment, which is negligible), and, if $\mathcal{A}$ sends inputs as described by the protocol, then it receives secret shares and a public key in both worlds and proceeds to the computation phase.

Next, we analyze the view of $\mathcal{A}$ in the computation phase, which proceeds in rounds. In the ideal world, inputs are sent to the TP in round i such that no party in $\mathcal{I}$ has an input greater than i (inputs are sent to the TP also when all parties abort, but we ignore this case for now because $\mathcal{A}$ learns no additional information after aborting all parties). Thus, before inputs are sent to the TP in the ideal world, it must hold that a party in $\mathcal{I}$ has an input greater than i and that $c_i = 0$. It is easy to see that $c_i = 0$ in the hybrid world too in this case. The first time inputs are sent to the TP, as we said, no party in $\mathcal{I}$ has an input greater than i, and i is sent on behalf of all corrupted parties. Thus, $\mathcal{S}$ learns the maximum between the inputs of honest parties and i. From here on, it is easy to see that the value c_i is generated the same way in both worlds.

Finally, we analyze the output of the honest parties. In the ideal world, inputs are sent to the TP in one of two cases. The first case is that round i is reached and no party in $\mathcal{I}$ has an input greater than i, so i is sent to the TP on behalf of all corrupted parties, and the honest parties output the maximum between their inputs and i. In this case in the hybrid world, if honest parties have no input greater than i, then they output i by the specification of the protocol no matter what corrupted parties abort, and if some honest party has an input greater than i, then, again, what the adversary does has no effect the output of honest parties. The second case in which inputs are send to the TP in the ideal world is when all corrupted parties abort (and inputs were not sent before). In this case in the ideal world, $\mathcal{S}$ sends the highest round number reached t to the TP, while in the hybrid world the honest parties output the maximum between t and their inputs.

$\square$

$\square$

Chapter 5

Computing *n* Party Maximum Together with Argmax with Complete Fairness

In this chapter, we explore the possibility of computing the maximum of the inputs of l parties together with the index of the first party holding the maximum value. Define the function $\mathsf{maxi} : \{0, ..., m\}^l \to \{0, ..., m\} \times \{1, ..., l\}$ as follows:

$$\mathsf{maxi}(x_1, ..., x_l) := (\mathsf{max}\{x_1, ..., x_l\}, \mathsf{min}\{j : x_j = \mathsf{max}\{x_1, ..., x_l\}\})$$

where max and min are the maximum and minimum functions, respectively, and $m = m(n)$ for a security parameter n and a polynomial function m. An important property of this function is that the output is always an input of one of the parties, which is what makes it possible to compute with fairness, as we later show.

We note that this result can be easily modified to show possibility of fair computation of the max function; however, in the proof of the previous result (Chapter 4) we do not use rewinding, whereas here we do.

5.1 Our protocol

Our protocol for maxi is similar the the protocol for the OR function of [11]. Specifically, parties commit to their inputs and then try to compute the output, possibly multiple times, until succeeding. In case some parties abort, the rest attempt again with the inputs of the aborted parties set to 0. The intuition behind why this protocol works is that when the output is (x_j, j), such that P_j is honest, then whether corrupted parties abort or not does not affect the output of the honest parties. Thus, the adversary $\mathcal{A}$ can only affect the output while it is of the form (x'_j, j), where x'_j is the input $\mathcal{A}$ provides for a corrupted party P_j. In this case, the decision whether to abort depends only on information $\mathcal{A}$ already possesses, and can therefore be determined without knowledge of the inputs of honest parties ahead of the execution.

The Protocol Π_{maxi} for computing maxi

Inputs: Let $\mathcal{P}$ be the set of invoking parties. Every party $\mathsf{P}_j \in \mathcal{P}$ holds an input $x_j \in \{0, ..., m\}$.

1. **Initialization:**
   (a) Each party P_i generates randomness r_i, computes a commitment $k_i = \mathsf{Com}(r_i, x_i)$[a] and broadcasts k_i.
   (b) If some parties do not send commitments, let $\mathcal{D}$ be a set containing their names. Set $\mathcal{P} = \mathcal{P} \setminus \mathcal{D}$ and continue.
2. **Preprocessing and input checks:**
   (a) Parties $\mathcal{P}$ invoke $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$, where $\mathsf{P}_i \in \mathcal{P}$ sends x_i, r_i and $\mathbf{k}_i = (k_j)_{j:\mathsf{P}_j \in \mathcal{P}}$, (note that $\mathbf{k}_i$ and t_i are the same for all honest parties P_i).
   (b) If P_i receives a set of names $\mathcal{D}$ from $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{max}}$, then it sets $\mathcal{P} = \mathcal{P} \setminus \mathcal{D}$ (note that $\mathcal{P}$ is the same for all honest parties). If $|\mathcal{P}| > 1$, parties in $\mathcal{P}$ restart step 2. Otherwise, only P_j remains and it outputs (x_j, j) if $x_j > 0$ or $(0, 1)$ if $x_j = 0$.
   (c) Otherwise, the parties get signed shares of $\mathbf{c}$ and a public key pk.
3. **Parties attempt to learn $\mathbf{c}$:**
   (a) Party P_j broadcasts its signed share of $\mathbf{c}$.
   (b) Parties receive the signed shares and check the signatures using pk. If all shares are sent and signatures are valid the parties restore $\mathbf{c}$ and output it.
   (c) If some parties do not broadcast shares with valid signatures, let $\mathcal{D}$ be the set of their names, the remaining parties set $\mathcal{P} = \mathcal{P} \setminus \mathcal{D}$. If $|\mathcal{P}| > 1$, the remaining parties go to Step 2. If only one party P_j remains, it outputs (x_j, j) if $x_j > 0$ or $(0, 1)$ if $x_j = 0$.

[a]Com is a computationally secure commitment scheme.

FIGURE 5.1: Protocol for computing maximum together with index of party holding maximum (if several parties hold the maximum value, then the smallest of their indexes).

$\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ for Π_{maxi}

Tools: A signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and a commitment scheme Com.
Inputs: Let $\mathcal{P}$ be the set of invoking parties, where $l = |\mathcal{P}|$. Party $\mathsf{P}_i \in \mathcal{P}$ holds input x_i, randomness r_i and a vector of commitments $\mathbf{k}_i$.
Computation:

1. Check inputs:
   (a) We say that party P_i disagrees with party P_j if $\mathbf{k}_i \neq \mathbf{k}_j$.
   (b) If some parties disagree with each other, let $\mathcal{D}_i$ be the set of parties who disagree with party P_i. Send $\mathcal{D}_i$ to P_i as its output and halt.
   (c) Let $\mathbf{k}$ be the commitment vector parties agreed upon, and $k_i \in \mathbf{k}$ be P_i's commitment. Set $\mathcal{D} = \{\mathsf{P}_i : \mathsf{Com}(x_i, r_i) \neq k_i\}$, and if $\mathcal{D} \neq \emptyset$, send $\mathcal{D}$ to parties in $\mathcal{P} \setminus \mathcal{D}$ and halt.
2. If $x_i = 0$ for all $\mathsf{P}_i \in \mathcal{P}$, compute $\mathbf{c} = (0, 1)$. Otherwise, compute $\mathbf{c} = (z, j)$, where $z = \max\{x_i\}_{i:\mathsf{P}_i \in \mathcal{P}}$, and $j = \min\{i : x_i = z, \mathsf{P}_i \in \mathcal{P}\}$.
3. Compute an l-out-of-l secret sharing of $\mathbf{c}$.
4. Generate secret and public keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}$. Sign every share using sk.

Output:

1. Party P_j receives its share of $\mathbf{c}$ and the public key pk (parties are assumed to know the algorithm Verify).

FIGURE 5.2: The functionality $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ for Π_{maxi}

5.2 Proof of security

Theorem 5.2.1. *Assume that there exists a protocol π that securely computes $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ (figure 5.2) with identifiable abort. Then, the function maxi can be computed with fairness and guaranteed output delivery) in the multiparty setting against a static, non-uniform, polynomial-time, malicious adversary corrupting up to $n-1$ parties.*

Proof. In Claim 5.2.1, we prove security against fail-stop adversaries and in the hybrid world with access to an ideal functionality computing $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ (with identifiable abort). The probability of a computationally bounded adversary forging a signature or a commitment is negligible, and, since every failed attempt at forging a signature or a commitment is interpreted as an abort, Claim 5.2.1 together with the composition theorem of Canetti [5] yield Theorem 5.2. In addition, we assume that $\mathcal{A}$ is rushing.

Claim 5.2.1. *Consider executions of Protocol Π (Figure 5.1) in a hybrid model with access to the ideal functionality computing $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ with identifiable abort. For every static, non-uniform, polynomial-time, fail-stop adversary $\mathcal{A}$ that corrupts a subset of the parties of* *size up to $n-1$ and attacks Π, there exists a poly-time adversary (simulator) $\mathcal{S}$, corrupting the same subset of parties and running in the ideal world with access to an ideal functionality computing* maxi *(with complete fairness), such that the following holds.*

$$\left\{ \left(\mathsf{Out}^{\mathsf{Hybrid}^{\mathsf{SG}}}_{\mathcal{A}(z),\Pi}, \mathsf{View}^{\mathsf{Hybrid}^{\mathsf{SG}}}_{\mathcal{A}(z),\Pi}\right)(1^n, x_1, ..., x_m) \right\}_{(x_1,...,x_m)\in X, z\in\{0,1\}^*, n\in\mathbb{N}}$$
$$\overset{\mathsf{c}}{\equiv} \left\{ \left(\mathsf{Out}^{\mathsf{Ideal}}_{\mathcal{S}(z),f}, \mathsf{View}^{\mathsf{Ideal}}_{\mathcal{S}(z),f}\right)(1^n, x_1, ..., x_m) \right\}_{(x_1,...,x_m)\in X, z\in\{0,1\}^*, n\in\mathbb{N}}$$

where the notation is as defined in section 2.5.

Proof. As before, we shorten notation. Thus, we will prove the following

$$(\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}) \equiv (\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}}) \tag{5.1}$$

We proceed to define a simulator $\mathcal{S}$, which uses as a black box an adversary $\mathcal{A}$ who controls a subset of parties of size up to $l-1$. The simulator $\mathcal{S}$ begins by simulating an execution of the protocol as if all honest parties were holding the same input 0. This is done in order to learn the pattern in which $\mathcal{A}$ aborts corrupted parties during such an execution. When all corrupted parties are aborted or when $\mathcal{A}$ allows the honest parties to learn the output, the pattern is learned and $\mathcal{S}$ proceeds to invoke the trusted party. In doing so, $\mathcal{S}$ sends 0 to the TP on behalf of the corrupted parties that would abort during the above execution (i.e., if all honest parties were holding 0) and the "real" input (that they are committed to) for the remaining corrupted parties. The simulator $\mathcal{S}$ then stores the output pair from the TP as (a, b) and rewinds the adversary $\mathcal{A}$ to the moment after commitments were sent. Then, it moves on to simulate another execution that is consistent with the output (a, b).
Simulator $\mathcal{S}$ for $\mathcal{A}$ attacking Π_{maxi}:
Let $\mathcal{P}$ be the initial set of parties, $\mathcal{C}$ be the set of parties corrupted by $\mathcal{A}$, $\mathcal{H}$ be the set of honest parties, and $\mathcal{I} \subseteq \mathcal{C}$ be the set of corrupted parties who are remaining in the protocol. Initially, $\mathcal{C} = \mathcal{I}$.

1. $\mathcal{S}$ initializes $\mathcal{A}$ on randomness r, security parameter n, inputs $\{x_i\}_{i:\mathsf{P}_i\in\mathcal{C}}$ and auxiliary input z.

2. **Simulate initialization:**

   (a) Receive commitments of corrupted parties from $\mathcal{A}$ and send commitments of 0 to $\mathcal{A}$ behalf of honest parties. Save all commitments as $\mathbf{k} = (k_j)_{j:\mathsf{P}_j\in\mathcal{P}}$.

   (b) If some parties do not send commitments, let $\mathcal{D}$ be the set of their names. Set $\mathcal{I} = \mathcal{P} \setminus \mathcal{D}$. If $\mathcal{I} = \emptyset$, send 0 to the TP on behalf of all corrupted parties, output the view of $\mathcal{A}$ and halt.

3. **Simulate** $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$**:**

   (a) If $\mathcal{I} = \emptyset$ do the following.

      i. If no inputs were sent to the TP, send 0 to the TP on behalf of all corrupted parties. Save output as $\mathbf{z}$, rewind $\mathcal{A}$ to the first time Step 3 is reached and go to Step 3.

      ii. If inputs were already sent to the TP, output the view of $\mathcal{A}$ and halt.

   (b) Otherwise, let $\mathcal{P} = \mathcal{I} \cup \mathcal{H}$ and $\mathbf{k} = (k_j)_{j:\mathsf{P}_j\in\mathcal{P}}$.

(c) Receive inputs to $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ from $\mathcal{A}$, which include x'_i, r_i $\mathbf{k}_i$, where $i : \mathsf{P}_i \in \mathcal{I}$.

(d) We say a party $\mathsf{P}_i \in \mathcal{P}$ disagrees with party $\mathsf{P}_j \in \mathcal{P}$ if $\mathbf{k}_i \neq \mathbf{k}_j$ or $t_i \neq t_j$, where $\mathbf{k}_a = \mathbf{k}$ and $t_a = t$ for $\mathsf{P}_a \in \mathcal{H}$ (we note that there can only be disagreement between two parties when one of them is in $\mathcal{I}$).

(e) If there is some disagreement, let $\mathcal{D}_j := \{\mathsf{P}_i : \mathsf{P}_i \in \mathcal{P} \text{ disagrees with } \mathsf{P}_j \in \mathcal{P}\}$. Send $\mathcal{D}_j$ to $\mathcal{A}$ for all $j : \mathsf{P}_j \in \mathcal{I}$. Finally, update $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}_i$ for some $i : \mathsf{P}_i \in \mathcal{H}$ (note that $\mathcal{D}_i$ is the same for all $i : \mathsf{P}_i \in \mathcal{H}$), and go to Step 3.

(f) If there is no disagreement, let $\mathcal{D} = \{\mathsf{P}_i : \mathsf{Com}(x'_i, r_i) \neq k_i, \mathsf{P}_i \in \mathcal{I}, k_i \in \mathbf{k}\}$. If $\mathcal{D} \neq \emptyset$, update $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}$, send $\mathcal{D}$ to $\mathcal{A}$ if $\mathcal{I} \neq \emptyset$, and go to Step 3.

(g) Give $\mathcal{A}$ the public key pk and signed shares of the value 0 which correspond to shares of $\mathbf{c}$.

(h) If $\mathcal{A}$ sends abort to $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ at some point, let $\mathcal{D}$ be the set of names that $\mathcal{A}$ specifies (remember that $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ is secure with identifiable abort). Update $\mathcal{I} \setminus \mathcal{D}$ and go to Step 3.

4. **Simulate the share broadcast phase:**

(a) Compute $(u, v) = \mathsf{maxi}(\hat{x}_1, ..., \hat{x}_l)$, where $\hat{x}_i = x'_i$ for $i : \mathsf{P}_i \in \mathcal{I}$ and $\hat{x}_i = 0$ for $i : \mathsf{P}_i \in \mathcal{H}$ and $i : \mathsf{P}_i \in \mathcal{C} \setminus \mathcal{I}$. Set $\mathbf{c}$ as follows:

i. If no inputs were sent to the TP, set $\mathbf{c} = (u, v)$.

ii. Otherwise, let $\mathbf{z} = (a, b)$ be the output received from the TP. If $u > a$, set $\mathbf{c} = (u, v)$, if $u = a$, then set $\mathbf{c} = (u, \min\{v, b\})$, and if $u < a$, set $\mathbf{c} = (a, b)$.

(b) Send[1] $\mathbf{c}$ to $\mathcal{A}$

(c) If some corrupted parties abort[2], let $\mathcal{D}$ be the set of aborted parties. Set $\mathcal{I} = \mathcal{I} \setminus \mathcal{D}$ and go to Step 3.

(d) Otherwise, no parties aborted. Do the following:

i. If no inputs were sent to the TP, send $\hat{x}_i$ to the TP, where $\hat{x}_i = x'_i$ for $\mathsf{P}_i \in \mathcal{I}$ and $\hat{x}_i = 0$ for $\mathsf{P}_i \in \mathcal{C} \setminus \mathcal{I}$. Save output as $\mathbf{z}$, rewind $\mathcal{A}$ to the first time Step 3 is reached and go to Step 3.

ii. If inputs were already sent to the TP, output the view of $\mathcal{A}$ and halt.

As in the analysis of max, we ignore random shares, signatures and commitments in the view of $\mathcal{A}$ and assume that its behavior is independent of them. As we explained before, showing that the view of $\mathcal{A}$, without the above, and the output of the honest parties are identical in both runs, will imply that they distribute identically in total.

The outline of our proof is roughly the following. We first show that while $\mathcal{A}$ can affect the outcome in both worlds, its partial view is identical in both worlds, and as a result, the output of the honest parties is also the same. Then, we show that the view that $\mathcal{S}$ constructs after learning the output from the TP is the same as $\mathcal{A}$'s view in the hybrid world.

Since we ignore commitments, the view of $\mathcal{A}$ is the same during the initialization phase, and it aborts the same parties in both worlds (in case it aborts any parties).

[1]When we say that we send $\mathbf{c}$ to $\mathcal{A}$, we mean that we send the corresponding shares with signatures that complete the shares of corrupted parties to $\mathbf{c}$.

[2]We say that a party aborts when it does not broadcast a share of $\mathbf{c}$ with a valid signature.

If all parties are aborted, then $\mathsf{O}_{\mathsf{Hybrid}} = \mathsf{O}_{\mathsf{Ideal}} = \mathsf{maxi}(\hat{x}_1, ..., \hat{x}_l)$, where $\hat{x}_i = x_i$ for $i : \mathsf{P}_i \in \mathcal{H}$ and $\hat{x}_i = 0$ for $i : \mathsf{P}_i \in \mathcal{C}$ and the claim holds.

Let x_i be the inputs that $\mathcal{A}$ sent on behalf of $\mathsf{P}_i \in \mathcal{C}$ and $\mathcal{I}$ be the set of remaining corrupted parties in both worlds. When $\mathsf{maxi}(\hat{x}_1, ..., \hat{x}_l) = (x_i, i)$, where $i : \mathsf{P}_i \in \mathcal{I}$, $\hat{x}_j = x_j$ for $j : \mathsf{P}_j \in \mathcal{I} \cup \mathcal{H}$, and $\hat{x}_j = 0$ for $j : \mathsf{P}_j \in \mathcal{C} \setminus \mathcal{I}$, we say that $\mathcal{A}$ *controls the output*. We proceed to show that while $\mathcal{A}$ controls the output, its view is the same in both worlds. We can show by similar arguments to those we used in Claim 4.2.1 that if $\mathcal{A}$ modifies inputs or commitments, then it receives the same sets of names from the computation of $\mathsf{ShareGen}^{\mathcal{P}}_{\mathsf{maxi}}$ in the hybrid world and in the corresponding part of the simulation in the ideal world. If $\mathcal{A}$ does not modify inputs or commitments, it receives random secret shares and a public key and proceeds to the phase where parties restore $\mathbf{c}$. Since we assume that $\mathcal{A}$ controls the output, and because the difference in the way $\mathbf{c}$ is computed in the ideal world from the way it is computed in the hybrid world is only that the inputs of honest parties are set to 0, it is easy to see that $\mathbf{c}$ is the same in both worlds.

We proceed to analyze the output of the honest parties. If $\mathcal{A}$ allows the honest parties to learn $\mathbf{c}$ in both worlds while it controls the output, then, since its view is the same in both worlds, the set of remaining corrupted parties is the same in both, and $\mathcal{S}$ sends the same inputs to the TP on behalf of corrupted parties as those used to generate $\mathbf{c}$ in the hybrid world. Specifically, 0 is sent on behalf of aborted parties and x'_j on behalf of the remaining ones, and we have that $\mathsf{O}_{\mathsf{Hybrid}} = \mathsf{O}_{\mathsf{Ideal}}$.

When $\mathcal{A}$ does not control the output, it is either the case that it never controlled the output to begin with, or that at some point $\mathcal{A}$ aborted some parties and from that moment on it no longer controls the output. At this moment (either the start of the execution or the moment when $\mathcal{A}$ aborts the respective parties), the sets of remaining corrupted parties in both worlds are the same because the view of $\mathcal{A}$ has been the same so far. In addition, when $\mathcal{A}$ does not control the output, whether the honest parties compute maxi using the effective inputs of remaining corrupted parties or 0 in their stead does not affect the output that they receive. Since this is the only difference between the way that the output of the honest parties is computed in either world, it holds that $\mathsf{O}_{\mathsf{Hybrid}} = \mathsf{O}_{\mathsf{Ideal}}$.

Finally, it remains to argue that the view of $\mathcal{A}$ is the same in both worlds. However, this is easy to see because after rewinding $\mathcal{A}$, the simulator $\mathcal{S}$ uses the inputs of corrupted parties and the output it learned from the TP to send to $\mathcal{A}$ the appropriate value $\mathbf{c}$ depending on whether $\mathcal{A}$ controls the output or not.

□

□

Chapter 6

Securely Computing A Three-Party, Embedded XOR Function

In this chapter we show a partial result about computing a three-party function with an embedded XOR with fairness and guaranteed output delivery for up to two corrupted parties, where we only prove security against a restricted subset of the class of static, non-uniform, polynomial-time adversaries.

6.1 Problem overview

The function we consider here has a similar structure to the GHKL function [10]. The main difference (apart from it being a three-party function) is that the "dominating party", which has the power to determine the output of the other parties, has even more control (it can bias the output both towards zero and towards one). Define the function $\mathsf{XOR}_{01} : X_1 \times X_2 \times X_3 \to \{0,1\}$ for $X_1 := \{0,1,2,3\}$, $X_2 := \{0,1\}$ and $X_3 := \{0,1\}$ as follows:

$$\mathsf{XOR}_{01}(x_1,x_2,x_3) = \begin{cases} 0, & \text{if } x_1 = 2 \\ 1, & \text{if } x_1 = 3 \\ x_1 \oplus x_2 \oplus x_3, & \text{otherwise} \end{cases}$$

Throughout this chapter, we refer to XOR_{01} simply as f.

6.2 Our Protocol

We present a protocol for computing XOR_{01} with full security (against a restricted class of adversaries). We note that a simpler protocol might suffice than the one we intend to present. However, we chose to present a protocol and a proof of security that were designed for an unrestricted adversary in mind. By doing so, we hope to bring forth our considerations and the challenges in constructing fully-secure three-party protocols for computing functions with an embedded XOR in the face of a dishonest majority. We assume that the parties have access to a broadcast channel, and we let the security parameter be denoted n.

The protocol begins by generating 3-out-of-3 secret shares of certain values for the parties, which they then restore in m rounds, where $m = m(n)$. There are six values for every round – a backup value for every party an a backup value for every pair of parties. In each round, every party learns its personal backup value and every pair learns a 2-out-of-2 secret sharing of its pair backup. The reconstruction of these backup values proceeds in a special predetermined order. Specifically, first P_1 learns its personal backup and the backup value for each pair $(\mathsf{P}_1, \mathsf{P}_j)$ is shared in a 2-out-of-2 secret sharing between P_1 and P_j. Then, in a second step, all the others backup values are appropriately shared. We stress that the order of learning is important for the security of the protocol.

If no party aborts, each party outputs its m-th personal backup value, in case a single party aborts, the other two reconstruct their pair backup value in a single round (where P_1 learns it first where applicable) and output it. In case two parties abort (either together or one after the other), the remaining third party outputs its personal backup value.

The backup values are randomly generated similarly to the way its done in the GHKL protocol [10]. Specifically, a special round i^* is chosen from the geometric distribution with parameter α. Backup values in rounds $i < i^*$ depend only on the inputs of the parties for which they are intended, and in rounds $i \geq i^*$ they contain the output of the function. We note that when $x_1 \in \{0, 1\}$, the backup values for pairs of parties and for single parties are distributed identically in rounds $i < i^*$ and are independent of the respective parties' inputs. Thus, when a party aborts in a round $i < i^*$, then no information has been learned by any of the parties, and if a party aborts in a round $i > i^*$, then all backup values already contain the true output. Therefore, the only case where fairness is not obvious is when an adversary corrupts a pair which learns information first and aborts a party in round $i = i^*$. What allows proving security is that $\mathcal{A}$ does not know i^*, and as a result even when it guesses i^* correctly, it does not know for sure that it did.

Protocol for computing XOR_{01}

Inputs: Parties P_1, P_2 and P_3 hold inputs $x_1 \in X_1$,$x_2 \in X_2$ and $x_3 \in X_3$, resp. The security parameter is n.
Computation:

1. **Preliminary phase:**
   (a) Parties P_1, P_2 and P_3 invoke a secure-with-identifiable-abort protocol $\Pi_{\mathsf{XOR}_{01}}$ for computing $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ with their inputs x_1, x_2, and x_3.
   (b) If P_1 aborts, the remaining parties (or party) output 0.
   (c) If both P_2 and P_3 abort, P_1 outputs $f(x_1, \hat{x}_2, \hat{x}_3)$ for $\hat{x}_2 \leftarrow_u X_2, \hat{x}_3 \leftarrow_u X_3$.
   (d) If P_2 (resp., P_3) aborts, then P_1 and P_3 (resp., P_2) invoke a two-party protocol for computing XOR^2_{01}[a] with full security.
   (e) If no party aborts, then each party receives 3-out-of-3 shares[b] of the values $c_j^{(i)}$ and $d_{k,l}^{(i)}$ for $j,k,l \in \{1,2,3\}$, where $k < l$, and $i \in [m]$.
2. **For rounds** $i = 1, ..., m$ **do**:
   (a) Party P_2 broadcasts its shares of $c_1^{(i)}$, $c_3^{(i)}$ and $d_{1,3}^{(i)}$, and party P_3 broadcasts its shares of $c_1^{(i)}$, $c_2^{(i)}$ and $d_{1,2}^{(i)}$. Then P_1 reconstructs $c_1^{(i)}$.
   (b) P_1 broadcasts its shares of $c_2^{(i)}$, $c_3^{(i)}$ and $d_{2,3}^{(i)}$. P_2 and P_3 reconstruct $c_2^{(i)}$ and $c_3^{(i)}$.
   (c) If one party aborts[c], the remaining parties P_j and P_k, where $j < k \in \{1,2,3\}$, send each other their shares of $d_{j,k}^{(i-1)}$, where P_1 always sends its share second. Then they reconstruct $d_{j,k}^{(i-1)}$, output it, and halt.
   (d) If two parties abort and P_j remains, it outputs $c_j^{(i-1)}$.
3. If no party aborted throughout the execution, each party P_j outputs $c_j^{(m)}$.

[a]we denote by XOR^2_{01} the XOR_{01} function for two parties with the input of P_2 or P_3 set to 0 (note that XOR_{01} is symmetrical w.r.t. P_2 and P_3).
[b]Each share is signed and parties receive a public key pk. Whenever a party receives a share, it checks that the signature is valid. We leave this implicit in the description of the protocol for readability.
[c]A party is said to abort if it does not send shares with valid signatures.

FIGURE 6.1: Protocol for computing three party binary XOR where one party can fix the output to either 0 or 1.

$\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$

Tools: A signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$.
Inputs: Parties P_1, P_2 and P_3 hold inputs $x_1 \in X_1$,$x_2 \in X_2$ and $x_3 \in X_3$, resp. The security parameter is n.

Computation:

1. Choose $i^* \geq 2$ according to a geometric distribution with parameter α to be defined later.
2. Compute the following values for $i \in [m]$:
   (a) For $i = 0$ to $i^* - 1$:
   - For all $j \in \{1,2,3\}$, set $c_j^{(i)} = f(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3)$, where $\tilde{x}_j = x_j$ and $\tilde{x}_t \leftarrow_u X_t$ for $t \neq j$.
   - For $j, k \in \{1,2,3\}$ such that $j < k$, set $d_{j,k}^{(i)} = f(\tilde{x}_1, \tilde{x}_2, \tilde{x}_3)$, where $\tilde{x}_j = x_j, \tilde{x}_k = x_k$ and $\tilde{x}_t \leftarrow_u X_t$ for $t \notin \{j,k\}$.

   (b) For $i \geq i^*$:
   - $c_j^{(i)} = d_{k,l}^{(i)} = f(x_1, x_2, x_3)$ for $j, k, l \in \{1,2,3\}$, where $k < l$.
3. For $j, k, l \in \{1,2,3\}$, where $k < l$, compute a 3-out-of-3 secret sharing of the values $c_j^{(i)}$ and $d_{k,l}^{(i)}$, generate secret and public keys $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^n)$ and sign every share using $\mathsf{Sign}_{\mathsf{sk}}$.

Output:

1. For $j, k, l \in \{1,2,3\}$, where $k < l$, send to each party its shares of the values $c_j^{(i)}$ and $d_{k,l}^{(i)}$ with signatures, as well as the public key pk (parties are assumed to know the algorithm Verify).

FIGURE 6.2: The functionality $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$

Remark (On the round complexity of the protocol). *It was shown in [10] that any two-party protocol computing a function with an embedded XOR must run for at least $\omega(\log(n))$ rounds, where n is the security parameter. This result naturally extends to the three-party embedded XOR function XOR_{01}. (Otherwise, consider a simple reduction that allows computing the XOR_{01} for two parties with the input of P_2 set to 0 in less than $\omega(\log(n))$ rounds). Together with Claim 6.3.1, this implies that our protocol has an optimal round complexity.*

6.3 Proof of security

We proceed to prove the security of our protocol. Denote by XOR_{01}^2 the XOR_{01} function for two parties with the input of P_2 set to 0 (note that XOR_{01} is symmetrical for P_2 and P_3). We know that XOR_{01}^2 can be computed with full security by [3].

Theorem 6.3.1. *Assume that $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a secure signature scheme, that π computes the functionality $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ in figure 6.2 securely with identifiable abort, and π'*

computes XOR_{01}^2 *with full security, then the protocol in figure 6.1 computes the function* XOR_{01} *with full security in the presence of a non-uniform, poly-time adversary corrupting up to two parties, assuming the following restriction on the behavior of the adversary:*
If the adversary ever aborts any party, then it either immediately aborts the other one too or never aborts it (this restriction may be weakened to only apply for certain cases).

Proof. Towards proving Theorem 6.3.2, we prove security against fail-stop adversaries and in the hybrid world with access to ideal functionalities computing $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ (with identifiable abort) and XOR_{01}^2 (with full security). The probability of a computationally bounded adversary forging a signature is negligible, and, since every failed attempt at forging a signature is interpreted as an abort, Theorem 6.3.2 together with the composition theorem of Canetti [5] yield Theorem 6.3.1. In addition, we assume that $\mathcal{A}$ is rushing, which informally means that $\mathcal{A}$ learns all information in a single round first (where order is unspecified by the protocol). When simulating a fail-stop adversary, we need not mention any signatures or public keys (which we do not, for simplicity of presentation). We stress, however, that a simulator can sample the keys and generate signatures that are distributed in the same way as those generated by the functionality $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$.

We begin by showing correctness.

Claim 6.3.1. *The protocol computes* XOR_{01} *with correctness except with negligible probability for* $m = \omega(\log(n))$*, where* n *is the security parameter.*

Proof. If parties follow the steps of the protocol, every party P_j outputs $c_j^{(m)}$ for $j \in \{1,2,3\}$. Recall that $c_j^{(m)} = f(x_1, x_2, x_3)$ for $i \geq i^*$ and a random value which may be different than the output otherwise. Since i^* is sampled from the geometric distribution, it is possible that $i^* > m$, in which case parties never get the correct output. The probability that parties never get the correct output is $(1-\alpha)^m$, where α is a constant. So, for correctness to hold, we need that for any polynomial n^c (for a constant $c > 0$) and for all large enough n (security parameter), it holds that $(1-\alpha)^m \leq \frac{1}{n^c}$, or equivalently, $m \cdot \log(1-\alpha) \leq -c \cdot \log(n)$. In other words, since $\log(1-\alpha) < 0$, we need that

$$m \geq -\frac{c \cdot \log(n)}{\log(1-\alpha)} = c' \cdot \log(n) = O(\log(n)),$$

for some constant $c' > 0$. Indeed, this holds for large enough n, since we took $m = \omega(\log(n))$. □

We proceed by proving the following claim regarding the security of the protocol.

Claim 6.3.2. *Let* $\mathcal{A}$ *be a non-uniform, poly-time, fail-stop adversary corrupting up to two parties and running* $\Pi_{\mathsf{XOR}_{01}}$ *(Figure 6.1) in a hybrid model with access to ideal functionalities computing* $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ *(with identifiable abort) and* XOR_{01}^2 *(with full security). Assume that in case* $\mathcal{A}$ *aborts a party, then it aborts the second party independently of the value that it learns after aborting the first party. Then, there exists a non-uniform, poly-time adversary* $\mathcal{S}$ *corrupting the same parties and running in the ideal world with access to an* *ideal functionality computing* XOR_{01} *with full security such that*

$$\left\{ \left(\mathsf{View}_{\mathcal{A}(z),\Pi}^{\mathsf{Hybrid}^{\mathsf{SG},\mathsf{X2}}}, \mathsf{Out}_{\mathcal{A}(z),\Pi}^{\mathsf{Hybrid}^{\mathsf{SG},\mathsf{X2}}} \right) (1^n, x_1, x_2, x_3) \right\}_{(x_1,x_2,x_3)\in X, z\in\{0,1\}^*, n\in\mathbb{N}} \tag{6.1}$$

$$\equiv \left\{ \left(\mathsf{View}_{\mathcal{S}(z),f}^{\mathsf{Ideal}}, \mathsf{Out}_{\mathcal{S}(z),f}^{\mathsf{Ideal}} \right) (1^n, x_1, x_2, x_3) \right\}_{(x_1,x_2,x_3)\in X, z\in\{0,1\}^*, n\in\mathbb{N}} \tag{6.2}$$

where SG *and* $\mathsf{X2}$ *are shortened notations for* $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ *and* XOR_{01}^2*. The rest of the notation is as defined in Section 2.5.*

Proof. In what follows, we shorten the notation of the view and output and omit the parts that are clear from the context, such as the inputs, $\mathcal{S}(z)$, f, $\mathcal{A}(z)$, and $\Pi_{\mathsf{XOR}_{01}}$. Thus, our goal is to prove the following.

$$(\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}) \equiv (\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}}) \tag{6.3}$$

Since we are considering a fail-stop adversary, we do not consider the possibility of braking a signature; thus, we will be looking to show equivalence of the above distributions, rather than computational indistinguishability.

Claim 6.3.3. *Theorem 6.3.2 holds for any adversary* $\mathcal{A}$ *(as per the restrictions specified in Theorem 6.3.2) that corrupts* P_1 *and* P_2*.*

Proof. We first give an informal overview of the simulation. Let x_1, x_2 be the inputs that $\mathcal{A}$ sends to $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ in the simulation. When $x_1 \in \{2,3\}$, all information that $\mathcal{A}$ observes throughout the execution can be generated using only its inputs, which is what $\mathcal{S}$ does. In case $\mathcal{A}$ aborts one or two parties, or the end of round m is reached, $\mathcal{S}$ makes P_3 output the same value as in the corresponding case in the hybrid run.

When $x_1 \in \{0,1\}$, the main idea of our simulation is similar to that of the GHKL simulation [10]. In rounds $i < i^*$, the simulator $\mathcal{S}$ simply imitates the execution of the protocol by sending the appropriate values to $\mathcal{A}$ (by sending the corresponding shares) and checking that $\mathcal{A}$ sends back what it should. Note, that all information that $\mathcal{A}$ learns in rounds $i < i^*$ can be generated using only the inputs it provides. To generate the view of $\mathcal{A}$ in rounds $i \geq i^*$, the simulator sends to the TP the inputs x_1, x_2 that $\mathcal{A}$ used for the $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ functionality.

Thus, we have that P_3 outputs $f(x_1, x_2, x_3)$ in the ideal run even if $\mathcal{A}$ aborts in round i^*. On the other hand, when $\mathcal{A}$ aborts in round i^* in the hybrid execution, the honest party outputs either $c_3^{(i^*-1)}$, $d_{1,3}^{(i^*-1)}$ or $d_{2,3}^{(i^*-1)}$, depending on which party $\mathcal{A}$ aborts first and whether it aborts the second party as well, which may be different from the real output. To compensate for this difference, when $\mathcal{A}$ aborts parties in some (but not all) rounds $i < i^*$, the simulator $\mathcal{S}$ sends special inputs to the TP.

Unlike in [10], the output of the honest party may be observed by $\mathcal{A}$ before it receives it. Specifically, when P_1 is aborted in round i (all the following also applies to when P_2 is aborted first, except switching the roles of P_1 and P_2 and using $d_{1,3}^{(i-1)}$ instead of $d_{2,3}^{(i-1)}$), then P_2 and P_3 restore $d_{2,3}^{(i-1)}$, where P_2 learns it first. If P_2 stays, then P_3 learns and outputs $d_{2,3}^{(i-1)}$, and if P_2 aborts, then P_3 outputs $c_3^{(i-1)}$. In order to determine what inputs to send to the TP in this case, $\mathcal{S}$ determines the output of P_3 by using rewinding to see what $\mathcal{A}$ does when given different values of $d_{2,3}^{(i-1)}$. We note that when analyzing a restricted adversary, this is not actually required.

If $\mathcal{A}$ aborts P_2 independently of $d_{2,3}^{(i-1)}$, then $d_{2,3}^{(i-1)}$ is independent of the output of P_3 and can be sampled the same way as in the hybrid world. If $\mathcal{A}$ does not abort P_2 independently of $d_{2,3}^{(i-1)}$, then the corresponding inputs are sent to the TP (this requires knowing the distribution o $d_{2,3}^{(i-1)}$ in the hybrid world beforehand, which is possible because $d_{2,3}^{(i-1)}$ is independent of x_3). Then $\mathcal{A}$ receives $d_{2,3}^{(i-1)} = \mathsf{O}_{\mathsf{Hybrid}}$. A challenging case which $\mathcal{S}$ does not handle and which limits security to the restricted class of adversaries, is when the adversary aborts P_2 only when $d_{2,3}^{(i-1)}$ is equal to a certain value (either 0 or 1). The problem in this case is that, with our current approach, it is impossible to know whether the output of P_3 should be equal to $d_{2,3}^{(i-1)}$ or not, and to simulate $\mathcal{A}$'s view accordingly.

We assume that $\mathcal{A}$ is rushing, so whenever P_1 aborts, P_2 learns $d_{2,3}^{(i)}$ before P_3 does, and construct the simulator accordingly. We stress that simulating the adversary when P_2 learns $d_{2,3}^{(i)}$ second is an easier task.

The simulator for the case that $\mathsf{P}_1, \mathsf{P}_2$ are corrupted:
Simulate $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$:

1. Choose $i^* \geq 2$ from the geometric distribution with parameter α.
2. Receive inputs x_1, x_2 from $\mathcal{A}$ to $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$, and send $\mathcal{A}$ random shares corresponding to the values $c_j^{(i)}$ and $d_{k,l}^{(i)}$ for $j, k, l \in \{1, 2, 3\}$, $k < l$, and $i \in [m]$, that $\mathcal{A}$ would get from $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$.
3. If $\mathcal{A}$ aborts P_1[1] during the computation of $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$, send $(x_1', x_2') = (2, 0)$[2] to the TP.
4. If $\mathcal{A}$ aborts (only) P_2 during the computation of $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$, receive x_1' from $\mathcal{A}$ on the behalf of P_1 as input to the functionality XOR_{01}^2, send $(x_1', 0)$ to the TP and save the output as z. Send z to $\mathcal{A}$ as the output of XOR_{01}^2, output the view of $\mathcal{A}$ and halt.

Simulate rounds $i = 1, ..., m$ when $x_1 \in \{2, 3\}$:

In rounds $i \leq i^*$:

1. Send[3] to $\mathcal{A}$ the values $c_1^{(i)} = f(x_1, 0, 0)$, $c_2^{(i)} \leftarrow_u \{0, 1\}$ and $d_{1,2}^{(i)} = f(x_1, 0, 0)$. In addition, send $\mathcal{A}$ a random share of 0 as the share of $d_{1,3}^{(1)}$ that $\mathcal{A}$ receives from P_3 in the protocol.
2. If $\mathcal{A}$ aborts[4] P_1 (before any party is aborted):
   (a) Sample $b \leftarrow_u \{0, 1\}$ and send $d_{2,3}^{(i)} = b$ to $\mathcal{A}$.

[1]Since $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ is computed with identifiable abort, $\mathcal{A}$ supplies at least one name of the corrupted parties.

[2]$\mathcal{S}$ sends $x_1' \in \{2, 3\}$ to to TP to fix the output of P_3 to either 0 or to 1, and it sends $x_1' \leftarrow_u \{1, 0\}$ to the TP to make the output of P_3 be 1 with probability 1/2. In both cases, the choice of x_2' is arbitrary.

[3]When we say that $\mathcal{S}$ sends to $\mathcal{A}$ a value that $\mathcal{A}$ received shares of in step 2, we mean that $\mathcal{S}$ sends $\mathcal{A}$ shares of that value which complete the shares that $\mathcal{A}$ holds.

[4]We say that a party is aborted during the computation if $\mathcal{A}$ does not send the required shares on its behalf.

   (b) If $\mathcal{A}$ sends its share of $d_{2,3}^{(i-1)}$ on behalf of P_2, send the following inputs to the TP. If $b = 0$, send $(x_1', x_2') = (2, 0)$ to the TP, and if $b = 1$, send $(x_1', x_2') = (3, 0)$ to the TP.

3. If $\mathcal{A}$ aborts P_2 (before any party is aborted):
   (a) Send $d_{1,3}^{(i-1)} = f(x_1, 0, 0)$ to $\mathcal{A}$.
   (b) If $\mathcal{A}$ sends its share of $d_{1,3}^{(i-1)}$ on behalf of P_1, send (x_1, x_2') to the TP, where $x_2' \leftarrow_u X_2$.
4. If $\mathcal{A}$ aborts a second party (by not sending a valid share of $d_{2,3}^{(i-1)}$ or $d_{1,3}^{(i-1)}$ depending on whether P_1 or P_2 is aborted first), send (x_1', x_2') to the TP, where $x_1' \leftarrow_u \{0, 1\}$ and $x_2' = 0$.

In rounds $i > i^*$:

1. Send $f(x_1, 0, 0)$ to $\mathcal{A}$ as the values $c_1^{(i)}$, $c_2^{(i)}$ and $d_{1,2}^{(i)}$, as well as a random share of 0 as the share of $d_{1,3}^{(1)}$ that $\mathcal{A}$ receives from P_3 in the protocol.
2. If $\mathcal{A}$ aborts P_1 (resp. P_2), send $d_{1,3}^{(i)} = f(x_1, 0, 0)$ (resp. $d_{2,3}^{(i)} = f(x_1, 0, 0)$) to $\mathcal{A}$. Send $(x_1, 0)$ to the TP, output the view of $\mathcal{A}$ and halt.

If $\mathcal{A}$ never aborts, send (x_1, x_2) to the TP.

Simulate rounds $i = 1, ..., m$ **when** $x_1 \in \{0, 1\}$**:**
In rounds $i \leq i^*$:

1. If $i < i^*$, send to $\mathcal{A}$ the values $c_1^{(i)} \leftarrow_u \{0, 1\}$, $c_2^{(i)} \leftarrow_u \{0, 1\}$ and $d_{1,2}^{(i)} \leftarrow_u \{0, 1\}$. In addition, send $\mathcal{A}$ a random share of 0 as the share of $d_{1,3}^{(1)}$ that $\mathcal{A}$ receives from P_3 in the protocol.
2. If $i = i^*$, send (x_1, x_2) to the TP and save the output as z. Send z to $\mathcal{A}$ as backup values $c_1^{(i)}$, $c_2^{(i)}$ and $d_{1,2}^{(i)}$.
3. If P_1 aborts and it does **not** hold that $c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)}$ (w.l.o.g. do the same for P_2, except using $d_{1,3}^{(i-1)}$ in place of $d_{2,3}^{(i-1)}$):
   (a) Sample $b \leftarrow_u \{0, 1\}$ and send $d_{2,3}^{(i-1)} = b$ to $\mathcal{A}$.
   (b) If $\mathcal{A}$ sends its share of $d_{2,3}^{(i-1)}$, send $(2, 0)$ or $(3, 0)$ to the TP depending on whether $b = 0$ or $b = 1$, respectively.
   (c) If $\mathcal{A}$ does not send back its share of $d_{2,3}^{(i-1)}$, send $(x_1', 0)$ to the TP where $x_1' \leftarrow_u \{0, 1\}$.
   (d) Output the view of $\mathcal{A}$ and halt.
4. If P_1 aborts and $c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)}$ (w.l.o.g. do the same for P_2, except using $d_{1,3}^{(i-1)}$ in place of $d_{2,3}^{(i-1)}$):
   (a) Send $d_{2,3}^{(i-1)} = 0$ to $\mathcal{A}$ and see whether $\mathcal{A}$ sends its share of $d_{2,3}^{(i-1)}$ (i.e., aborts the second party), then rewind it. Do the same with $d_{2,3}^{(i-1)} = 1$. Set

Strat to abort, stay, bias_1 or bias_0 depending on whether the second party always aborts, always stays, aborts only when $d_{2,3}^{(i-1)} = 0$ or aborts only when $d_{2,3}^{(i-1)} = 1$, respectively (note that simulation may be done without rewinding for a restricted adversary).

(b) If Strat = abort, send $d_{2,3}^{(i-1)} \leftarrow_u \{0,1\}$ to $\mathcal{A}$. If $i < i^*$, then send $(x'_1, x'_2) \leftarrow \mathbf{x}^{\mathbf{w}}_{x_1,x_2}$ to the TP, where $\mathbf{w} = (c_1^{(i)}, \mathsf{abort})$ (we show later in the proof that exists such $\mathbf{x}^{\mathbf{w}}_{x_1,x_2}$ for which security holds).

(c) If Strat = stay, do the following. If $i < i^*$, send $(x'_1, x'_2) \leftarrow \mathbf{x}^{\mathbf{w}}_{x_1,x_2}$ to the TP, where $\mathbf{w} = (c_1^{(i)}, \mathsf{stay})$ (we show later in the proof that exists such $\mathbf{x}^{\mathbf{w}}_{x_1,x_2}$ for which security holds), and save the output as z. Send $d_{2,3}^{(i-1)} = z$ to $\mathcal{A}$ (also when $i = i^*$).

(d) The case Strat $\in \{\mathsf{bias}_1, \mathsf{bias}_0\}$ is not handled for the restricted adversary $\mathcal{A}$.

(e) Output the view of $\mathcal{A}$ and halt.

In rounds $i > i^*$:

1. Send z to $\mathcal{A}$ as backup values $c_1^{(i)}$, $c_2^{(i)}$ and $d_{1,2}^{(i)}$.
2. If P_1 aborts, send $d_{2,3}^{(i-1)} = z$ to P_2, output $\mathcal{A}$'s view and halt.
3. If P_2 aborts, send $d_{1,3}^{(i-1)} = z$ to P_1, output $\mathcal{A}$'s view and halt.

If $\mathcal{A}$ did not abort, output its view and halt.

In the case where $x_1 \in \{2,3\}$ it is easy to see by the construction of $\mathcal{S}$ that the view of $\mathcal{A}$ together with the output of the honest party are distributed in the same way as in the protocol. Thus, we focus on the case where $x_1 \in \{0,1\}$. In what follows, all probabilities are implicitly conditioned on $x_1 \in \{0,1\}$.

Let i^*_{Hybrid} and i^*_{Ideal} denote the value i^* in either world. We begin by observing that i^*_{Ideal} is sampled by $\mathcal{S}$ in the same way as i^*_{Hybrid} by $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$. Then, for any $t > 1$, we have that

$$\Pr[i^*_{\mathsf{Hybrid}} = t] = \Pr[i^*_{\mathsf{Ideal}} = t]$$

In what follows, we analyze such executions in which $i^*_{\mathsf{Hybrid}} = i^*_{\mathsf{Ideal}} = t$ and refer to i^*_{Hybrid} and i^*_{Ideal} as i^*.

Since the view of $\mathcal{A}$ is the same in both worlds when it sends its inputs to $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$ (or to $\mathcal{S}$), then the inputs it sends are the same in both worlds and we refer to them as x_1, x_2. We observe that $\mathcal{S}$ samples the backup values in the same way as they are sampled by $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$, except for the backup value $d_{2,3}^{(i-1)}$ (resp., $d_{1,3}^{(i-1)}$) for $i \le i^*$, which $\mathcal{A}$ learns when aborting P_1 (resp., P_2) in round $i \le i^*$. That is, when $x_1 \in \{0,1\}$ in both worlds, all backup values except for the latter are uniform random bits in rounds $i < i^*$ in both worlds, and in rounds $i \ge i^*$, they contain $f(x_1, x_2, x_3)$ in both worlds. It may also be observed that $\mathcal{S}$ sends shares to $\mathcal{A}$ in the same way as they are sent to it by $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$, which look random in both worlds by the properties of secret sharing schemes. Combining all the above, we have that

$$\mathsf{V}_{\mathsf{Hybrid}} \equiv \mathsf{V}_{\mathsf{Ideal}},$$

unless $\mathcal{A}$ aborts a party in round $i \le i^*$.

Let $\mathbf{v}$ be a view of $\mathcal{A}$, such that $\mathcal{A}$ either aborts in round $i > i^*$ or never aborts (note that $\mathcal{A}$ acts the same way when conditioned on the same view). Then

$$\Pr[\mathsf{V}_{\mathsf{Hybrid}} = \mathbf{v}] = \Pr[\mathsf{V}_{\mathsf{Ideal}} = \mathbf{v}]$$

We proceed to show for such $\mathbf{v}$ that when $\mathsf{V}_{\mathsf{Hybrid}} = \mathsf{V}_{\mathsf{Ideal}} = \mathbf{v}$, the output is the *same* in both worlds. We allow ourselves to abuse notation by referring to the backup values in both worlds by the same names, since they are the same when $\mathsf{V}_{\mathsf{Hybrid}} = \mathsf{V}_{\mathsf{Ideal}} = \mathbf{v}$. When $\mathbf{v}$ is such that $\mathcal{A}$ never aborts a party, then except with negligible probability that $i^* > n$ (as shown in Claim 6.3.1), we have that x_1, x_2 are sent to the TP in the ideal run and P_3 outputs $c_3^{(m)} = f(x_1, x_2, x_3)$ in the hybrid run. When $\mathbf{v}$ is such that $\mathcal{A}$ aborts a party in round $i > i^*$, in the hybrid world, P_3 may output either $c_3^{(i)}$, $d_{1,3}^{(i)}$ or $d_{2,3}^{(i)}$, depending on which party aborts first and whether the second party aborts as well, and we have that $c_3^{(i)} = d_{1,3}^{(i)} = d_{2,3}^{(i)} = f(x_1, x_2, x_3)$. In the ideal world, $\mathcal{S}$ already sent (x_1, x_2) to the TP in round $i = i^*$.

The case when $\mathcal{A}$ aborts a party in round $i \leq i^*$:

Let $\mathbf{v}_i$ be a view of $\mathcal{A}$ when it aborts some party in round i. Since up until this moment the view was distributed identically in both worlds (even when $i \leq i^*$), we have that

$$\Pr[\mathsf{V}^i_{\mathsf{Hybrid}} = \mathbf{v}_i] = \Pr[\mathsf{V}^i_{\mathsf{Ideal}} = \mathbf{v}_i]$$

Recall that after $\mathcal{A}$ aborts, the only value that it learns is either $d_{2,3}^{(i-1)}$ or $d_{1,3}^{(i-1)}$, depending on which party aborts first, which may be correlated to P_3's output in the hybrid world.

When $\mathcal{A}$ aborts one of the parties in round $i = i^*$, P_3 learns a random backup value in the hybrid world, whereas in the ideal run it learns the real output $f(x_1, x_2, x_3)$. Thus, for $\mathsf{out}_h \in \{0, 1\}$ we have that

$$\Pr[(\mathsf{V}^i_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}) = (\mathbf{v}_i, \mathsf{out}_h) | i = i^*] \neq \Pr[(\mathsf{V}_{\mathsf{Ideal}} i, \mathsf{O}_{\mathsf{Ideal}}) = (\mathbf{v}_i, \mathsf{out}_h) | i = i^*] \quad (6.4)$$

In addition, recall that when $\mathcal{A}$ aborts in round $i = i^*$, it learns $c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)} = f(x_1, x_2, x_3)$ in both worlds. However, since i^* is unknown, $\mathcal{A}$ cannot distinguish between the cases where $i = i^*$ and it got the real output and cases where $i < i^*$ and it just so happens that $c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)}$. In what follows, we use the above observation to show that for any $\mathbf{v}_i$, there exists a choice of inputs for $\mathsf{P}_1, \mathsf{P}_2$ that the simulator can send to the trusted party in rounds i, where $i < i^*$ and $c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)}$, such that for any view $\mathbf{v} = \mathbf{v}_i \cup \left\{ d_{2,3}^{(i-1)} \right\}$ or $\mathbf{v} = \mathbf{v}_i \cup \left\{ d_{1,3}^{(i-1)} \right\}$ and $\mathsf{out}_h \in \{0, 1\}$, we have that

$$\Pr[(\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}) = (\mathbf{v}, \mathsf{out}_h) | i \leq i^*] = \Pr[(\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}}) = (\mathbf{v}, \mathsf{out}_h) | i \leq i^*] \quad (6.5)$$

We note that if it does not hold that $c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)}$, then $\mathcal{A}$ knows that $i < i^*$ in both runs and we cannot use these cases to compensate for Equation 6.4. In these cases $\mathcal{S}$ samples $d_{2,3}^{(i-1)}$ or $d_{1,3}^{(i-1)}$ the same way as $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$. Also, $\mathcal{S}$ can send such inputs to the TP that make the output of P_3 in the ideal world be distributed the same way as its output in the hybrid world. Thus, it is easy to see that

$$(\mathsf{V}_{\mathsf{Hybrid}}, \mathsf{O}_{\mathsf{Hybrid}}) \equiv (\mathsf{V}_{\mathsf{Ideal}}, \mathsf{O}_{\mathsf{Ideal}}).$$

In what follows, we show that when $\mathsf{V}^i_{\mathsf{Hybrid}} = \mathsf{V}^i_{\mathsf{Ideal}} = \mathbf{v}_i$, there exist corresponding inputs to the TP such that Equation 6.5 holds. Since $\mathcal{A}$ performs the same actions when conditioned on the same view, we will be considering and adversary with a fixed behavior. Since we are considering an adversary with a fixed behavior, and because we condition on $i \leq i^*$, the variables in $\mathbf{v}_i$ corresponding to rounds smaller than i are independent of the output of P_3 and of i^*, and so it is easy to see that they cancel out from Equation 6.5. Thus, we will be looking to show that for every $\mathsf{out}_h \in \{0,1\}$ and $a, b \in \{0,1\}$ that

$$\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((a,a,a,b), \mathsf{out}_h) \mid i \leq i^*] = \tag{6.6}$$

$$\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Ideal}}) = ((a,a,a,b), \mathsf{out}_h) \mid i \leq i^*] \tag{6.7}$$

We distinguish between two cases depending on the value of Strat (which is the same in both worlds since the behavior of $\mathcal{A}$ is fixed). Recall that Strat is set to abort or stay depending on whether the second party aborts or stays independently of the value of $d_{2,3}^{(i-1)}$ (we assume that $\mathcal{A}$ acts independently of $d_{2,3}^{(i-1)}$, so the cases where Strat $\in \{\mathsf{bias}_1, \mathsf{bias}_0\}$ do not occur). The cases where P_1 is aborted first are symmetric to the cases where P_2 is aborted first. Therefore, we only consider the former cases.

Before we continue, we introduce some notation. Denote the output distribution vector of the honest party in rounds $i < i^*$ by

$$\mathbf{q}^{\mathbf{w}}_{x_1,x_2} := \begin{bmatrix} Pr[\mathsf{O}_{\mathsf{Ideal}} = 1 | x_3 = 0, (x_1', x_2') \leftarrow_u \mathbf{x}^{\mathbf{w}}_{x_1,x_2}] \\ Pr[\mathsf{O}_{\mathsf{Ideal}} = 1 | x_3 = 1, (x_1', x_2') \leftarrow_u \mathbf{x}^{\mathbf{w}}_{x_1,x_2}] \end{bmatrix}$$

where recall that $\mathbf{w}$ is $(c_1^{(i)}, \mathsf{Strat})$. We write $\mathbf{q}_{x_1,x_2}$ ($\mathbf{x}_{x_1,x_2}$ resp.) instead of $\mathbf{q}^{\mathbf{w}}_{x_1,x_2}$ ($\mathbf{x}^{\mathbf{w}}_{x_1,x_2}$ resp.) when $\mathbf{w}$ is clear from the context. Next, denote

$$\mathbf{f}_{x_1,x_2} = \begin{bmatrix} f(x_1,x_2,0) \\ f(x_1,x_2,1) \end{bmatrix}$$

In what follows, we treat the addition operation applied to a scalar with a vector, denoted $a + \mathbf{v}$, as $a \cdot \mathbf{1}^l + \mathbf{v}$, where l is the length of the vector $\mathbf{v}$.

The case when $\mathcal{A}$ aborts P_1 first and P_2 is aborted independently of $d_{2,3}^{(i-1)}$ (Strat $=$ abort).

Recall that when Strat $=$ abort, then $\mathcal{S}$ sends $d_{2,3}^{(i-1)} \leftarrow_u \{0,1\}$ to $\mathcal{A}$ in the ideal world, which is distributed identically to $d_{2,3}^{(i-1)}$ in the hybrid world. In addition, P_3 always outputs $c_3^{(i-1)}$, which is independent of $d_{2,3}^{(i-1)}$ in both worlds. Thus, $d_{2,3}^{(i-1)}$ cancels out from Equation 6.6 and we will be looking to show that for every $\mathsf{out}_h \in \{0,1\}$ and $a \in \{0,1\}$

$$\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((a,a,a), \mathsf{out}_h) \mid i \leq i^*] = \tag{6.8}$$

$$\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}), \mathsf{O}_{\mathsf{Ideal}}) = ((a,a,a), \mathsf{out}_h) \mid i \leq i^*] \tag{6.9}$$

Next, we consider the different possible cases for the values of a and out_h.
When $(a, \mathsf{out}_h) = (1,1)$:

In the hybrid world, we have that

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((1,1,1),1) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, c_3^{(i-1)}) = (1,1,1,1) \mid i < i^*] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot \Pr[c_3^{(i-1)} = 1 \mid i = i^*] \cdot f(x_1, x_2, x_3) \\
&= \frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2}
\end{aligned}$$

Where the last equality can be seen by observing that all the backup values $c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, c_3^{(i-1)}$ equal 1 with probability $1/2$ by the specification of the protocol, and that $\Pr[i = i^* | i \le i^*] = \alpha$ since i^* is sampled from the geometric distribution.

The difference in the ideal world, compared to the hybrid world, is that the output of the honest party in round $i = i^*$ is equal to the real output, and in rounds $i < i^*$ it depends on what $\mathcal{S}$ sends to the TP. Thus,

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}), \mathsf{O}_{\mathsf{Ideal}}) = ((1,1,1),1) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}) = (1,1,1) \mid i < i^*] \cdot \Pr[\mathsf{O}_{\mathsf{Ideal}} = 1 \mid (x_1', x_2') \leftarrow_u \mathbf{x}_{x_1,x_2}] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot f(x_1, x_2, x_3) \\
&= \frac{(1-\alpha) \cdot \mathbf{q}_{x_1,x_2}}{2^3} + \alpha \cdot \mathbf{f}_{x_1,x_2}
\end{aligned}$$

For security to hold, we need

$$\frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2} = \frac{(1-\alpha) \cdot \mathbf{q}_{x_1,x_2}}{2^3} + \alpha \cdot \mathbf{f}_{x_1,x_2}$$

Separating $\mathbf{q}_{x_1,x_2}$ from the equation, we get

$$\mathbf{q}_{x_1,x_2} = \frac{1}{2} - \frac{2^2 \alpha \mathbf{f}_{x_1,x_2}}{1-\alpha} \tag{6.10}$$

For small enough α, we get that $\mathbf{q}_{x_1,x_2}$ is a positive vector, which is easy to get by sending the correct combination of (x_1', x_2').

For example, when $(x_1, x_2) = (0,0)$ then $\mathbf{f}_{x_1,x_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, and for some $0 < \beta < 1/2$ (where $\beta = \beta(\alpha)$). Then, we have that

$$\mathbf{q}_{x_1,x_2} = \begin{bmatrix} 1/2 \\ 1/2 \end{bmatrix} - \begin{bmatrix} 0 \\ \beta \end{bmatrix} = \frac{1-2\beta}{2} \cdot \begin{bmatrix} 1 \\ 1 \end{bmatrix} + \frac{1+2\beta}{2} \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

which can be attained by sending $(3,0)$ to the TP with probability $1/2 - \beta(\alpha)$ (which give $\mathbf{f}_{3,0} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$) and $(1,0)$ with probability $1/2 + \beta(\alpha)$, (which give $\mathbf{f}_{1,0} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$).

When $(a, \mathsf{out}_h) = (1,0)$:

In the hybrid world, by the specification of the protocol

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((1,1,1),0) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, c_3^{(i-1)}) = (1,1,1,0) \mid i < i^*] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot \Pr[c_1^{(i-1)} = 0 \mid i = i^*] \cdot f(x_1, x_2, x_3) \\
&= \frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2}
\end{aligned}$$

just as in the previous case.

In the ideal world, in round i^*, it holds that $\mathsf{O}_{\mathsf{Ideal}} = c_1^{(i)} = c_2^{(i)} = d_{1,2}^{(i)}$, so $a \neq \mathsf{out}_h$ with probability 0 (recall that we are analyzing the case where $(a, \mathsf{out}_h) = (1,0)$). Thus, we have that

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}), \mathsf{O}_{\mathsf{Ideal}}) = ((1,1,1),0) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}) = (1,1,1) \mid i < i^*] \cdot \Pr[\mathsf{O}_{\mathsf{Ideal}} = 0 \mid (x_1', x_2') \leftarrow_u \mathbf{x}_{x_1,x_2}] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot 0 \\
&= \frac{(1-\alpha)\cdot(1-\mathbf{q}_{x_1,x_2})}{2^3}
\end{aligned}$$

For security to hold, we need

$$\frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2} = \frac{(1-\alpha)\cdot(1-\mathbf{q}_{x_1,x_2})}{2^3}$$

which holds for $\mathbf{q}_{x_1,x_2}$ that is given in Equation 6.10. The claim holds for the cases where $(a, \mathsf{out}_h) = (0,0)$ and $(a, \mathsf{out}_h) = (0,1)$ by symmetricity to the cases where $(a, \mathsf{out}_h) = (1,1)$ and $(a, \mathsf{out}_h) = (1,0)$, respectively.

The case where P_1 is aborted and P_2 stays independently of the value of $d_{2,3}^{(i-1)}$ ($\mathsf{Strat} = \mathsf{stay}$):

In this case, P_3 always outputs $d_{2,3}^{(i-1)}$, so $\mathsf{out}_h = b$ and it is enough to show for every and $a, b \in \{0,1\}$ that

$$\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((a,a,a,b),b) \mid i \le i^*] = \tag{6.11}$$

$$\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Ideal}}) = ((a,a,a,b),b) \mid i \le i^*] \tag{6.12}$$

When $(a,b) = (1,1)$:

In the hybrid world, we have that

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((1,1,1,1),1) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}) = (1,1,1,1) \mid i < i^*] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot \Pr[d_{2,3}^{(i-1)} = 1 \mid i = i^*] \cdot f(x_1, x_2, x_3) \\
&= \frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2}
\end{aligned}$$

Remember that in the ideal world, $\mathcal{A}$ receives the output from the TP as $d_{2,3}^{(i-1)}$. Thus,

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Ideal}}) = ((1,1,1,1),1) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}) = (1,1,1) \mid i < i^*] \cdot \Pr[\mathsf{O}_{\mathsf{Ideal}} = 1 \mid (x_1', x_2') \leftarrow_u \mathbf{x}_{x_1,x_2}] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot f(x_1,x_2,x_3) \\
&= \frac{(1-\alpha)\cdot \mathbf{q}_{x_1,x_2}}{2^3} + \alpha \cdot \mathbf{f}_{x_1,x_2}
\end{aligned}$$

For security to hold, we need

$$\frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2} = \frac{(1-\alpha)\cdot \mathbf{q}_{x_1,x_2}}{2^3} + \alpha \cdot \mathbf{f}_{x_1,x_2}$$

Separating $\mathbf{q}_{x_1,x_2}$ from the equation, we get

$$\mathbf{q}_{x_1,x_2} = \frac{1}{2} - \frac{2^2 \cdot \alpha \cdot \mathbf{f}_{x_1,x_2}}{1-\alpha} \tag{6.13}$$

As before, it is easy to see that a corresponding $\mathbf{x}_{x_1,x_2}$ exists for small enough α.
When $(a,b) = (1,0)$:

In the hybrid world, we have that

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Hybrid}}) = ((1,1,1,0),0) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}) = (1,1,1,0) \mid i < i^*] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot \Pr[d_{2,3}^{(i-1)} = 0 \mid i = i^*] \cdot f(x_1,x_2,x_3) \\
&= \frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2}
\end{aligned}$$

And, in the ideal world, we have that

$$\begin{aligned}
&\Pr[((c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}, d_{2,3}^{(i-1)}), \mathsf{O}_{\mathsf{Ideal}}) = ((1,1,1,0),0) \mid i \le i^*] = \\
&\quad \Pr[i < i^* \mid i \le i^*] \cdot \Pr[(c_1^{(i)}, c_2^{(i)}, d_{1,2}^{(i)}) = (1,1,1) \mid i < i^*] \cdot \Pr[\mathsf{O}_{\mathsf{Ideal}} = 0 \mid (x_1', x_2') \leftarrow_u \mathbf{x}_{x_1,x_2}] \\
&\quad + \Pr[i = i^* \mid i \le i^*] \cdot 0 \\
&= \frac{(1-\alpha)\cdot(1-\mathbf{q}_{x_1,x_2})}{2^3}
\end{aligned}$$

For security to hold, we need

$$\frac{1-\alpha}{2^4} + \frac{\alpha \cdot \mathbf{f}_{x_1,x_2}}{2} = \frac{(1-\alpha)\cdot(1-\mathbf{q}_{x_1,x_2})}{2^3}$$

, which is satisfied by $\mathbf{q}_{x_1,x_2}$ that we have in Equation 6.13.

Finally, the claim holds for the cases where $(a,b) = (0,0)$ and $(a,b) = (0,1)$ by symmetry to the cases where $(a,b) = (1,1)$ and $(a,b) = (1,0)$, respectively.

□

The case of corrupted P_1, P_3 is symmetrical to the case of corrupted P_1, P_2, which we just covered in Claim 6.3.3. Thus, the proof for the following claim is omitted.

Claim 6.3.4. *Theorem 6.3.2 holds for any adversary $\mathcal{A}$ that corrupts* P_1 *and* P_3.

The next claim covers the case where the "dominant" party P_1 is honest.

Claim 6.3.5. *Theorem 6.3.2 holds for any adversary $\mathcal{A}$ that corrupts P_2 and P_3.*

Proof. The proof of this claim is similar in places and, generally, easier than that of Claim 6.3.3, so we allow ourselves to be concise. Recall that when P_2 and P_3 are corrupted, $\mathcal{A}$ learns all information after the honest party P_1. Therefore, our simulator $\mathcal{S}$ is relatively straightforward in this case. $\mathcal{S}$ imitates an execution of the protocol without interaction with the trusted party until round i^*. In round i^*, when P_1 learns $c_1^{(i)}$, the simulator $\mathcal{S}$ sends (x_2, x_3) to the TP and saves the output as z. Then, $\mathcal{S}$ proceeds to imitate the execution using z.

In case one or two parties abort in round $i \leq i^*$, the simulator $\mathcal{S}$ sends inputs to the TP, such that the output of P_1 is distributed the same way as in the corresponding case in the execution of the protocol (note that whether the second party aborts or not does not affect the distribution of P_1's output). In case the second party does not abort, $\mathcal{S}$ sends it the output that it previously received from the TP.

Simulator for when P_2 and P_3 are corrupted:

1. Choose $i^* \geq 2$ from the geometric distribution with parameter α.
2. Receive inputs x_2, x_3 from $\mathcal{A}$ to $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$. If $\mathcal{A}$ does not send abort, send $\mathcal{A}$ random shares of 0 corresponding to the values $c_j^{(i)}$ and $d_{k,l}^{(i)}$ for $j, k, l \in \{1, 2, 3\}$, $k < l$, and $i \in [m]$, that $\mathcal{A}$ would get from $\mathsf{ShareGen}_{\mathsf{XOR}_{01}}$.
3. If $\mathcal{A}$ aborts (only) P_2, receive x_3' from $\mathcal{A}$ on the behalf of P_3 as input to the functionality XOR_{01}^2, send $(x_2', 0)$ to the TP and save the output as z. Send z to $\mathcal{A}$ as the output of XOR_{01}^2, output the view of $\mathcal{A}$ and halt. If P_3 aborts, act analogously.
4. For rounds $i = 1, ..., m$:
   (a) Accept shares from $\mathcal{A}$ where P_2 and P_3 would send their shares in $\Pi_{\mathsf{XOR}_{01}}$. If $\mathcal{A}$ does not send shares on behalf of a party, treat it as aborted.
   (b) If no party aborts:
      i. If $i = i^*$, send (x_1, x_2) to the TP and save output as z.
      ii. Send to $\mathcal{A}$ the values $c_2^{(i)}$, $c_3^{(i)}$ and $d_{2,3}^{(i)}$ as follows.
         A. If $i < i^*$, $c_2^{(i)} \leftarrow_u \{0, 1\}$, $c_3^{(i)} \leftarrow_u \{0, 1\}$ and $d_{2,3}^{(i)} \leftarrow_u \{0, 1\}$.
         B. If $i \geq i^*$, $c_2^{(i)} = c_3^{(i)} = d_{2,3}^{(i)} = z$.

         In addition, send to $\mathcal{A}$ a random share where P_1 would send it its share of $d_{2,3}^{(i)}$.
   (c) If $\mathcal{A}$ aborts one of the parties:
      i. If $i \leq i^*$, sample $x_2' \leftarrow_u \{0, 1\}$, send $(x_2', 0)$ to the TP and store the output as z.
      ii. If P_2 was aborted and $\mathcal{A}$ sends its share of $d_{1,3}^{(i-1)}$, send z to $\mathcal{A}$ as $d_{1,3}^{(i-1)}$.
      iii. If P_3 was aborted and $\mathcal{A}$ sends its share of $d_{1,2}^{(i-1)}$, send z to $\mathcal{A}$ as $d_{1,2}^{(i-1)}$.
      iv. Output the view of $\mathcal{A}$ and halt.
5. If $\mathcal{A}$ did not abort, output its view and halt.

We can show that

$$\mathsf{V}_{\mathsf{Hybrid}} \equiv \mathsf{V}_{\mathsf{Hybrid}}$$

similarly to the way we do it when P_1 and P_2 are corrupted. Thus, it remains to show that the output is distributed identically given that the view does. But this is easy to see because when $\mathcal{A}$ aborts P_2 in round $i \leq i^*$ (w.l.o.g. the same applies to when P_3 is aborted first), $\mathcal{S}$ sends $(x_2', 0)$ to the TP, where $x_2' \leftarrow_u \{0,1\}$, and $f(x_1, x_2', 0)$ are distributed the same way as either one of the possible outputs of P_1 in the hybrid world, $c_1^{(i-1)}$ or $d_{1,2}^{(i-1)}$ (depending on whether P_3 is aborted too). If $\mathcal{A}$ aborts in round $i > i^*$, then it is easy to see that P_1 outputs $f(x_1, x_2, x_3)$ in both worlds. This occludes the proof of the case of corrupted P_2,P_3. □

Finally, we discuss the case of a single corrupted party. We have that when a single party is corrupted, $\mathcal{S}$ has as much control in the ideal world over the output of the two honest parties, as it has over the output of a single honest party in the ideal world when two parties are corrupted. Specifically, when P_1 and P_2 are corrupted, for any choice of inputs x_1', x_2' to the TP, there is a choice of x_1' when only P_1 is corrupted such that the output of the honest parties is the same. The same can be said when comparing the case where P_2 and P_3 are corrupted to the cases where only P_2 or only P_3 is corrupted. In addition, when only a single party is corrupted, $\mathcal{A}$ is strictly more limited than in the case when two parties are corrupted. Thus, if some adversary $\mathcal{A}$ were able to break the security of the protocol in the case when a single party is corrupted, then there must exist an $\mathcal{A}'$ that breaks the security of the protocol when two parties are corrupted, in contradiction to the Claims 6.3.3 and 6.3.5. This implies the following claim and concludes the proof.

Claim 6.3.6. *Theorem 6.3.2 holds for any adversary $\mathcal{A}$ that corrupts a single party.*

□

□

Bibliography

[1] Shashank Agrawal and Manoj Prabhakaran. "On Fair Exchange, Fair Coins and Fair Sampling". In: *IACR Cryptol. ePrint Arch.* (2013), p. 442. URL: http://eprint.iacr.org/2013/442.

[2] Gilad Asharov. "Towards Characterizing Complete Fairness in Secure Two-Party Computation". In: *Theory of Cryptography - 11th Theory of Cryptography Conference, TCC 2014, San Diego, CA, USA, February 24-26, 2014. Proceedings*. Ed. by Yehuda Lindell. Vol. 8349. Lecture Notes in Computer Science. Springer, 2014, pp. 291–316. DOI: 10.1007/978-3-642-54242-8_13. URL: https://doi.org/10.1007/978-3-642-54242-8%5C_13.

[3] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. "Complete Characterization of Fairness in Secure Two-Party Computation of Boolean Functions". In: *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*. Ed. by Yevgeniy Dodis and Jesper Buus Nielsen. Vol. 9014. Lecture Notes in Computer Science. Springer, 2015, pp. 199–228. DOI: 10.1007/978-3-662-46494-6_10. URL: https://doi.org/10.1007/978-3-662-46494-6%5C_10.

[4] Amos Beimel, Iftach Haitner, Nikolaos Makriyannis, and Eran Omri. "Tighter Bounds on Multi-Party Coin Flipping via Augmented Weak Martingales and Differentially Private Sampling". In: *59th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2018, Paris, France, October 7-9, 2018*. Ed. by Mikkel Thorup. IEEE Computer Society, 2018, pp. 838–849. DOI: 10.1109/FOCS.2018.00084. URL: https://doi.org/10.1109/FOCS.2018.00084.

[5] Ran Canetti. "Security and Composition of Multi-party Cryptographic Protocols". In: *IACR Cryptol. ePrint Arch.* (1998), p. 18. URL: http://eprint.iacr.org/1998/018.

[6] Richard Cleve. "Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract)". In: *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*. Ed. by Juris Hartmanis. ACM, 1986, pp. 364–369. DOI: 10.1145/12130.12168. URL: https://doi.org/10.1145/12130.12168.

[7] Dana Dachman-Soled. "Revisiting Fairness in MPC: Polynomial Number of Parties and General Adversarial Structures". In: *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part II*. Ed. by Rafael Pass and Krzysztof Pietrzak. Vol. 12551. Lecture Notes in Computer Science. Springer, 2020, pp. 595–620. DOI: 10.1007/978-3-030-64378-2_21. URL: https://doi.org/10.1007/978-3-030-64378-2%5C_21.

[8] Shimon Even, Oded Goldreich, and Abraham Lempel. "A Randomized Protocol for Signing Contracts". In: *Advances in Cryptology: Proceedings of CRYPTO '82, Santa Barbara, California, USA, August 23-25, 1982*. Ed. by David Chaum, Ronald L. Rivest, and Alan T. Sherman. Plenum Press, New York, 1982, pp. 205–210. DOI: 10.1007/978-1-4757-0602-4_19. URL: https://doi.org/10.1007/978-1-4757-0602-4%5C_19.

[9] Oded Goldreich, Silvio Micali, and Avi Wigderson. "How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority". In: *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*. Ed. by Alfred V. Aho. ACM, 1987, pp. 218–229. DOI: 10.1145/28395.28420. URL: https://doi.org/10.1145/28395.28420.

[10] S. Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. "Complete Fairness in Secure Two-Party Computation". In: *J. ACM* 58.6 (2011), 24:1–24:37. DOI: 10.1145/2049697.2049698. URL: https://doi.org/10.1145/2049697.2049698.

[11] S. Dov Gordon and Jonathan Katz. "Complete Fairness in Multi-party Computation without an Honest Majority". In: *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*. Ed. by Omer Reingold. Vol. 5444. Lecture Notes in Computer Science. Springer, 2009, pp. 19–35. DOI: 10.1007/978-3-642-00457-5_2. URL: https://doi.org/10.1007/978-3-642-00457-5%5C_2.

[12] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. 2nd. 2014.

[13] Nikolaos Makriyannis. "On the Classification of Finite Boolean Functions up to Fairness". In: *IACR Cryptol. ePrint Arch.* (2014), p. 516. URL: http://eprint.iacr.org/2014/516.

[14] Goldreich Oded. *Foundations of Cryptography: Volume 1, Basic Tools*. 1st. USA: Cambridge University Press, 2009.

[15] Andrew Chi-Chih Yao. "Protocols for Secure Computations (Extended Abstract)". In: *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*. IEEE Computer Society, 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38. URL: https://doi.org/10.1109/SFCS.1982.38.