Karlsruhe Institute of Technology

KASTEL

An Overview of Fairness Notions in Multi-Party Computation

Bachelor's Thesis of

Tim Strasser

at the Department of Informatics
KASTEL – Institute of Information Security and Dependability

Reviewer:	Prof. Dr. Jörn Müller-Quade
Second reviewer:	Prof. Dr. Thorsten Strufe
Advisor:	M.Sc. Saskia Bayreuther

15. May 2023 – 15. September 2023

Karlsruher Institut für Technologie
Fakultät für Informatik
Postfach 6980
76128 Karlsruhe

I declare that I have developed and written the enclosed thesis completely by myself, and have not used sources or means without declaration in the text.

PLACE, DATE

. . . . . .

(Tim Strasser)

Abstract

Multi-party Computation (MPC) is a cryptographic technique that enables multiple mutually distrusting parties to jointly compute a function over their private inputs. Fairness in MPC is defined as ensuring that if one party receives the output, all honest parties do. This thesis addresses the lack of comprehensive overviews on different fairness notions in MPC.

Complete fairness, often considered the ideal, guarantees that either all parties receive an outcome or none do. However, this ideal is not generally achievable due to theoretical and contextual constraints. As a result, alternative notions have emerged to address these limitations.

In this thesis, we review different notions of fairness in MPC, including complete fairness, partial fairness, Delta-fairness, gradual release, fairness with penalties, and probabilistic fairness. Each notion approaches different requirements and limitations to real-world scenarios. We find that complete fairness requires an honest majority to be achieved for general functions without stronger assumptions, such as access to public ledgers, while specific functions can be computed with complete fairness even without these assumptions. Other notions, such as Delta-fairness, require secure hardware components. We provide an overview of the notions, their interrelations, trade-offs, and practical implications of these notions. In addition, we summarize the findings in a comparative table that provides a compact overview of the protocols that achieve these notions of fairness, showing the trade-offs between security, efficiency, and applicability.

The thesis identifies assumptions and constraints associated with various notions of fairness, citing protocols from seminal works in the field. Several impossibility results are also presented, demonstrating the inherent challenges in achieving fairness in MPC. The practical implications of these fairness notions are explored, providing insights into their applicability and limitations in real-world scenarios.

Zusammenfassung

Die sichere Mehrparteienberechnung (“Multi-party Computation”, MPC) ist eine kryptografische Technik, die es mehreren Parteien, die sich gegenseitig misstrauen, ermöglicht, gemeinsam eine Funktion über ihre privaten Eingaben zu berechnen. Fairness in MPC ist definiert als die Eigenschaft, dass, wenn eine Partei die Ausgabe erhält, alle ehrlichen Parteien diese erhalten. Diese Arbeit befasst sich mit dem Defizit an umfassenden Übersichten über verschiedene Fairnessbegriffe in MPC.

Vollständige Fairness (“complete fairness”), die oft als Ideal angesehen wird, garantiert, dass entweder alle ehrlichen Parteien ein Ergebnis erhalten oder keine. Dieses Ideal ist jedoch aufgrund theoretischer und kontextbezogener Beschränkungen im Allgemeinen nicht zu erreichen. Infolgedessen haben sich alternative Begriffe herausgebildet, um diese Einschränkungen zu überwinden.

In dieser Arbeit werden verschiedene Fairnessbegriffe in MPC untersucht, darunter vollständige Fairness, partielle Fairness (“Partial Fairness”), Delta-Fairness, graduelle Freigabe, Fairness mit Strafen und probabilistische Fairness. Jedes Konzept stellt unterschiedliche Anforderungen und Einschränkungen für reale Szenarien dar. Wir stellen fest, dass vollständige Fairness eine ehrliche Mehrheit erfordert, um für allgemeine Funktionen ohne stärkere Annahmen, wie z. B. den Zugang zu öffentlichen Ledgern, erreicht zu werden, während bestimmte Funktionen auch ohne diese Annahmen mit vollständiger Fairness berechnet werden können. Andere Begriffe, wie Delta-Fairness, erfordern sichere Hardwarekomponenten. Wir geben einen Überblick über die Begriffe, ihre Zusammenhänge, Kompromisse und praktischen Implikationen dieser Begriffe. Darüber hinaus fassen wir die Ergebnisse in einer vergleichenden Tabelle zusammen, die einen kompakten Überblick über die Protokolle bietet, die diese Fairnessbegriffe erfüllen, und die Kompromisse zwischen Sicherheit, Effizienz und Anwendbarkeit aufzeigt.

In der Arbeit werden Annahmen und Einschränkungen im Zusammenhang mit verschiedenen Fairnessbegriffe aufgezeigt und Protokolle aus grundlegenden Arbeiten auf diesem Gebiet zitiert. Es werden auch mehrere Unmöglichkeitsergebnisse vorgestellt, die die inhärenten Herausforderungen beim Erreichen von Fairness im MPC aufzeigen. Die praktischen Implikationen dieser Fairnesskonzepte werden untersucht und geben Einblicke in ihre Anwendbarkeit und Grenzen in realen Szenarien.

Contents

1 Introduction

Let us imagine a scenario where multiple companies want to compute the average salary of their employees to assess industry standards. However, each company wants to keep the individual salaries of their employees private. This problem might seem impossible to solve initially because traditionally, to compute an average, one would need to know all the individual values. However, an area of cryptography called Multi-party Computation (MPC) makes this possible.

Multi-party computation (MPC) is a cryptographic approach that allows multiple parties to jointly compute a function over their private inputs. No information about these inputs is revealed, except what can be inferred from the function's output. This property is referred to as *input privacy*. Other desirable security properties of MPC are *correctness* and *fairness*. [25, 13, 30].

For example, in the scenario above, the companies could use MPC to jointly compute the average salary of their employees without revealing any individual salary information to each other. The only result they would learn is the average salary, which is the intended output of the function. This type of computation preserves the privacy of individual salary information.

MPC has found practical applications in various fields where sensitive data is involved, such as computing statistics on the gender wage gap, private DNA comparisons for medical purposes, gathering statistics without revealing anything but the aggregate results, and more [25]. Protecting the privacy of the inputs is crucial in these scenarios, as revealing the data may have legal implications. For example, in the case of the Boston wage gap study, companies were hesitant to provide their raw data due to privacy concerns; MPC allowed them to share information without revealing individual employees' compensation data [25].

In addition to privacy, fairness is an important property often desired in MPC protocols. This thesis explores fairness, which informally means that either all honest parties receive the output or no party does. Ensuring fairness in MPC protocols is necessary to maintain trust between participants, as well as to prevent any party from gaining an unfair advantage. An unfair advantage, in this context, means that a dishonest party obtains the output while honest parties do not, potentially allowing the dishonest party to exploit the results for their own benefit.

For example, in the context of an auction, multiple parties want to determine the highest bid without exposing individual bids. Each participant, or bidder, submits a bid, which is treated as their private input into the auction protocol. The protocol then computes the highest bid while ensuring that individual bids remain private. After evaluation, the outcome, is revealed to all participants.

However, in a system that lacks fairness, certain vulnerabilities can emerge. For instance, an adversary, might learn in advance that her bid wasn't the highest. With this information,

and before others receive the outcome, they could disrupt the auction process by simulating a network failure. If they get to submit their bid again, the adversary could strategically adjust their bid, placing it just above the highest bid. This provides the adversary with an advantage. [10]

Achieving fairness in MPC protocols is, however, not trivial. Certain notions of fairness may be unachievable under certain conditions. The feasibility of certain fairness notions can depend on many factors, including but not limited to the number of parties, the fraction of corrupted parties, or the acceptable round complexity.

The primary objective of this thesis is to deepen the understanding of various notions of fairness in Multi-Party Computation (MPC) protocols. We aim to identify their similarities, differences, and their role in different application scenarios. To achieve this, we address the following research questions:

1. What are different notions of fairness in multi-party computation (MPC) protocols?
2. How do these notions of fairness relate to each other, and what are the trade-offs between them in terms of security, efficiency, and applicability in different scenarios?
3. What assumptions are necessary to achieve certain levels of fairness?
4. What known impossibility results exist for achieving fairness in MPC protocols, and under what conditions or assumptions do these results hold?
5. What are the practical implications of the different notions of fairness in real-world application scenarios?

In order to achieve the objectives of this thesis and address the stated research questions, we undertook a literature review. Primarily, we used Google Scholar to identify and collect papers discussing protocols that achieve different notions of fairness in multi-party computation. We built a structural representation of the different notions of fairness, their interrelationships, and practical implications by methodically classifying, analysing and comparing the information extracted from these papers. We aim to analyse these notions, highlight their similarities and differences, and discuss the trade-offs between them. Furthermore, we will examine known impossibility results and study the conditions under which certain notions of fairness become unattainable.

This research contributes to the field by offering a comparison of different notions of fairness and by investigating known impossibility results and the preconditions under which they hold. More specifically, we explore the preconditions for various notions of fairness, including the number of participating parties, the threshold of corrupted parties, the underlying network model, and the acceptable round complexity. From these preconditions, we look at the resulting impossibilities and limitations in achieving fairness. The intended outcome is an understanding of the different notions of fairness in MPC protocols, including the conditions under which they can and cannot be achieved, and their practical implications in real-world scenarios.

2 Related Work

Li et al., 2023 [22] Li et al., 2023 [22] published a survey on Trusted Execution Environments (TEEs) (see Section 3.3.5) for secure computation. The survey systematically reviews TEE-based secure computation protocols and classifies them into three general categories: secure outsourced computation, secure distributed computation, and secure multiparty computation. Their comparison of different protocols serves as a useful resource for us, particularly as we dive into the specifics of various secure computation protocols.

While the [22] survey provides a solid understanding of the current state of TEE-based secure computation protocols, it does not go into the specifics of how these protocols address fairness, a core focus of our research. Their focus is primarily on privacy and integrity, while our work specifically addresses fairness in secure computation protocols.

Yin et al., 2021 [31] Federated learning (FL)[1] has seen significant development over the past four years, but this has also raised new privacy concerns, particularly during the aggregation[2] of distributed intermediate results [31]. To address these issues, privacy-preserving federated learning (PPFL) has been proposed as a solution to maintain privacy in machine learning. Regardless, the challenge of protecting privacy without compromising the utility of data through machine learning remains. A comprehensive survey on PPFL, as conducted by Yin et al., 2021 [31] proposes a taxonomy based on the "5Ws" (Who, What, When, Where, and Why) to categorize various scenarios in privacy-preserving federated learning. This classification provides an analysis of privacy leakage risks in FL from five different perspectives [31]. The review summarizes existing methodologies and also identifies potential future research directions in this area.

[1] FL refers to a machine learning approach where the training data remains on the client's device and only model updates are sent to the central server for aggregation. This helps in preserving data privacy.

[2] Aggregation in this context is the process of combining updates from multiple clients to improve the global model.

3 Preliminaries

This introduces the fundamental concepts of multi-party computation (MPC). We explore the basic components of MPC, from basic definitions to secure computation techniques, parameters, and models. This basic understanding will provide the necessary background for our examination of fairness in MPC protocols.

3.1 System Parameters

This section outlines the fundamental parameters in cryptographic protocols: the security parameter, the number of participants and the threshold for resilience of the protocol.

3.1.1 Security Parameter

The security parameter, referred to as λ, is a variable that determines the level of security in cryptographic protocols. For a security parameter λ, an adversary would need to perform approximately 2^{λ} operations to break the security, making the probability of a successful attack approximately $1/2^{\lambda}$. [14]

3.1.2 Number of Parties

The number of parties, represented as n, denotes the total number of participants involved in the protocol. Each party has its own private input and wishes to jointly compute a function over its collective inputs, without revealing individual inputs to others (unless the output of the function dictates otherwise) [30]. The value of n can vary and, depending on its relationship to the threshold t, affects the protocol's resilience to malicious parties.

3.1.3 Threshold

The threshold t in a system with n parties refers to the maximum number of parties that can be corrupted without violating the security guarantees of a protocol. In some protocols, an *honest majority* is assumed, where $t < \frac{n}{2}$, meaning that more than half of the total parties are honest and follow the protocol.

3.2 Basic Concepts in Multi-Party Computation

3.2.1 Multi-Party Computation (MPC)

In many real-world scenarios, a number of participants need to calculate a function over their collective data, for example, determining the average salary in a company, without revealing their individual inputs to each other. Multi-party computation (MPC) provides an approach to such computations that uses cryptographic methods to ensure that these security properties are met. More precisely, MPC is a cryptographic protocol in which n parties jointly compute a function $f : \mathbb{N} \times (\{0,1\}^*)^n \times \{0,1\}^* \to (\{0,1\}^*)^n$ over their private inputs while maintaining certain security properties. The function f represents a joint computation, where the first parameter represents the security parameter, the next n parameters represent the private inputs of the n parties, and the last parameter represents the randomness built into f.

The security properties guaranteed by MPC protocols are:

1. **Correctness:** The protocols should match the output of the function f when applied to the inputs, for all honest parties.

2. **Input privacy:** During the computation, no party should learn more information about the private inputs of the other parties than can be inferred from the output of the function and its own input

3. **Fairness:** The computation should adhere to a predefined fairness notion, the details of which are discussed in Chapter 4.

The protocol operates under the assumption of an adversarial model (see Section 3.4), where the adversary may control a subset of the participating parties and even potentially manipulate the communication network. However, the protocol is designed to maintain its security properties in the face of such adversarial behaviour. [20]

3.2.2 Parties

In the context of MPC, a party refers to an entity participating in a computation. Each party has a private input value that it wishes to keep confidential while participating in a joint computation with others. In some scenarios, the set of parties is static, that is, the set of participants remains the same throughout the entire computation. However, certain applications may allow for a dynamic set of parties, where participants can join or leave the computation process.

3.2.3 Protocol

A protocol in MPC is a predefined set of rules or instructions that define how the participating parties should interact and process their inputs to jointly compute a function. It specifies the sequence of computation and communication steps that each party should follow, and is designed to ensure certain security properties such as privacy, correctness, and fairness.

Protocols can be either synchronous or asynchronous. Synchronous protocols work based on defined time limits, where there is a known upper bound on both the processing of tasks and the delivery of messages. In these systems, the time taken by a process to perform a step, which may involve receiving or sending a message or performing local computation, never exceeds this known limit. Such protocols function in fixed time intervals called "rounds". Each round has specific start and end times, and during a round specific actions or tasks are performed. To maintain synchronization, all parties involved typically align their local clocks with a global clock. This global clock ensures that all processes remain synchronized, Asynchronous protocols do not assume a global clock or shared time reference. Instead, they're designed to deal with the uncertainty of message delivery times and potential network delays. [8]

Protocols, whether synchronous or asynchronous, rely on different communication models:

- **Authenticated Channels**: These channels assure the recipient that a message really comes from its declared sender. They prevent external entities from impersonating a party within the protocol, thereby mitigating man-in-the-middle attacks.
- **Broadcast Channels**: In scenarios where a message needs to be communicated to all participants simultaneously, broadcast channels come into play. They guarantee that each participant receives an identical copy of the message, thus ensuring uniformity of information.
- **Peer-to-Peer Communication**: This model assumes direct communication between two specific parties, without the interference or mediation of other entities.

This sequence of computation and communication is repeated, round after round, until the protocol is completed, and the final output is revealed.

3.3 Cryptographic Primitives and Assumptions

3.3.1 Encryption

Encryption is a cryptographic primitive that enables secure communication between parties over an insecure channel. It provides both confidentiality, by preventing unauthorized access to information, and integrity, by ensuring that the information is not modified during transit. There are two main types of encryption: symmetric and asymmetric. Symmetric encryption uses the same key for both encryption and decryption, with AES (Advanced Encryption Standard) being a commonly used, state-of-the-art algorithm. Asymmetric encryption, also known as public key encryption, uses two different keys: a public key for encryption and a private key for decryption, with RSA (Rivest-Shamir-Adleman) and ECC (Elliptic Curve Cryptography) being popular examples.

Definition 3.3.1 (Encryption Scheme)**.** Formally, an encryption scheme consists of the following three algorithms:

- **KeyGen**(1^λ): Given a security parameter λ, this probabilistic algorithm outputs a pair of bit strings (e, d), where e is the encryption key and d is the decryption key.
- **Encrypt**(e, α): Given an encryption key e and a plaintext α, this algorithm produces a ciphertext β.
- **Decrypt**(d, β): Given a decryption key d and a ciphertext β, this deterministic algorithm returns the original plaintext α if the decryption is successful.

It holds that for every pair (e, d) in the range of **KeyGen**(1^λ), and for every $\alpha \in \{0,1\}^*$, algorithms **Encrypt** and **Decrypt** satisfy **Decrypt**$(d, \textbf{Encrypt}(e, \alpha)) = \alpha$. [14]

3.3.2 Signatures

Signatures are cryptographic primitives that allow an entity to prove the authenticity and integrity of a message. They allow a sender to attach a unique signature to a message, which can then be verified by any receiver to confirm that the message originated from the claimed sender and has not been manipulated.

Definition 3.3.2 (Signature Scheme)**.** Formally, a signature scheme consists of the following three algorithms:

- **KeyGen**(1^λ): Given a security parameter λ, this probabilistic algorithm outputs a pair of bit strings (s, v), where s is the signing key and v is the verification key.
- **Sign**(s, α): Given a signing key s and a message α, this algorithm generates a signature σ.
- **Verify**(v, α, σ): Given a verification key v, a message α and a signature σ, this deterministic algorithm returns 1 if the signature is valid for the given message and 0 otherwise.

It holds that for every pair (s, v) in the range of **KeyGen**(1^λ), and for every $\alpha \in \{0,1\}^*$, the algorithms **Sign** and **Verify** satisfy **Verify**$(v, \alpha, \textbf{Sign}(s, \alpha)) = 1$. [14]

3.3.3 Message Authentication Codes (MAC)

A Message Authentication Code (MAC) is a cryptographic primitive that provides a means of verifying the authenticity and integrity (see Section 3.3.1) of a message.

Definition 3.3.3 (Message Authentication Code)**.** Formally, a MAC scheme consists of the following three algorithms:

- **KeyGen**(1^λ): Given a security parameter λ, this probabilistic algorithm generates a secret key k. Where probabilistic means, the algorithm can produce different outputs on different executions, even with the same input.
- **MAC**(k, m): Given a secret key k and a message m, this algorithm produces an authentication tag t.

- **Verify**(k, m, t): Given a secret key k, a message m, and an alleged authentication tag t, this deterministic algorithm returns *valid* if the tag is valid for the message under the key, and *invalid* otherwise.

[29]

3.3.4 Secret Sharing

A Secret Sharing scheme involves a trusted authority that distributes pieces of information, known as "shares", among participants. Each party receives a share of the secret, and when a predetermined number of shares are combined, the secret can be reconstructed. This ensures that even if some parties are corrupted, the secret remains secure, as long as a sufficient number of shares are held by honest parties. A prominent type of secret sharing is a threshold scheme.

A threshold scheme, specifically called a (k, n)-threshold scheme, involves generating n shares such that any combination of k shares allows the reconstruction of the secret. However, any subset of $k - 1$ or fewer shares reveals no information about the value of the secret.

Definition 3.3.4 (Threshold Scheme)**.** Let t, w be positive integers with $t \leq w$. A (t, w)-threshold scheme is a procedure to share a key K among a set of w participants, such that any group of t participants can deduce the value of K. However, no group of $t - 1$ participants can determine it. [29]

3.3.5 Trusted Execution Environments (TEEs) and Secure Processors

Trusted Execution Environments (TEEs) are secure areas of a main processor that ensure that the code and data loaded into these areas is protected in terms of privacy and integrity. TEEs provide an isolated environment where applications can run protected from external threats, even if the overall system is compromised. In essence, a TEE is a conceptual framework or environment that can be realized through various hardware or software mechanisms.

Secure processors, which are hardware components, allow the realization of such TEEs. They are specialized hardware designed to securely execute code, protecting both the execution process and the data involved.

A prominent example of a secure processor is Intel's Software Guard Extensions (SGX). Intel SGX allows applications to execute code in private areas of memory called enclaves, which are protected from processes running at higher privilege levels.

The term "attested execution processor" in this context refers to a processor that can provide cryptographic proof of its actions. It can attest or guarantee that it has executed a particular piece of code securely without manipulation. This attestation mechanism is important in scenarios where, in addition to ensuring the confidentiality and integrity of code and data, there is a need to validate that the code has been executed exactly as intended in a trusted environment. [22]

3.3.6 Quadratic Residucity Assumption

The Quadratic Residucity Assumption (QRA) is a computational hardness assumption in cryptography. It states that, given an integer n which is the product of two different odd prime numbers p and q, and an integer x such that the Jacobi symbol $(x/n) = +1$, it is computationally infeasible to determine whether x is a quadratic residue modulo n (i.e, there exists an integer y such that $y^2 \equiv x \mod n$) without knowing the factors p and q. This problem, known as the Quadratic Residucity Problem (QRP), is easy to solve when p and q are known, but is thought to be difficult when only x and n are known. [19]

3.4 Adversarial Models

An adversary in MPC is a malicious entity that tries to disrupt the computation, compromise security, or gain unauthorized knowledge of the honest parties inputs. The adversary may control one or more parties and can have different capabilities. The exact model of the adversary depends on the security assumptions of the protocol. Below are brief descriptions of common adversary models:

3.4.1 Semi-honest adversary:

Often referred to as passive adversaries, semi-honest adversaries follow the steps of the protocol but try to learn additional information from the messages they see. [14]

3.4.2 Malicious adversary:

Unlike semi-honest adversaries, malicious or active adversaries may not follow the rules of the protocol. They may insert messages in the name of the corrupted party and modify or withhold messages, that are being sent between the parties, with the intention of disrupting computation or cheating other participants. [14]

3.4.3 Static Adversary:

This adversary decides which parties to corrupt before the protocol starts, and cannot change his choice once the protocol is running. His corruption targets are predetermined and unchangeable. [1]

3.4.4 Adaptive Adversary:

These adversaries can adaptively choose which parties to corrupt based on observed protocol execution and the parties internal state. [1]

3.5 Proofs and Frameworks

3.5.1 Simulation Proofs

Simulation proofs are a key tool in cryptography for proving the security of cryptographic protocols. Instead of proving multiple safety properties individually, simulation proofs offer a more streamlined approach: they work by comparing an adversary's behaviour in a "real world" execution of a protocol with an "ideal world" scenario in which a trusted party securely computes the function in question.

In real execution, parties have their own private inputs and a shared initial input. They send and receive messages following a protocol. Honest parties follow the protocol to generate their messages, whereas corrupt parties may deviate under the attack of an adversary. The adversary can view the messages sent by the honest parties and decide when or if they are delivered, but can't alter or forge them. At the end of the process, the parties produce outputs based on the protocols, while the attacker's output is a function of what he has observed throughout.

On the other hand, the ideal execution is a simplified scenario where there's no protocol for message exchange between parties. Instead, all parties send their input to a trusted party called the "ideal functionality". This trusted party computes the desired function from the parties private inputs and then provides an output to each party. In this model, for every real-world attacker, there is a "ideal world" adversary called the "simulator". The goal is to ensure that any information an attacker can obtain in the real world can also be obtained by the simulator in the ideal world, thus proving the security of the protocol, as in this ideal world, the primitive in question is secure by definition. [20]

In the context of proving security within this framework, a simulator is an algorithm designed to mimic the interaction between the adversary and the honest parties in the real world, while having access only to the information available in the ideal world. The goal is to show that for every adversary in the real world, there exists a simulator in the ideal world such that no distinguisher can tell whether it is interacting with the real-world adversary or the ideal-world simulator.

In the context of protocol execution, let $EXEC_{\pi,A}(x, y, \lambda)$ be the random variable describing the output of party P_i in protocol π with adversary A. Similarly, in the ideal execution, $IDEAL_{F,Sim}(x, y, \lambda)$ represents the corresponding random variable for the ideal functionality F.

Definition 3.5.1 (Ideal Model: $\text{IDEAL}_F^{\text{MPC}}$)**.** The ideal model describes a protocol execution scenario in which two parties, each with their private inputs, interact with a trusted third party to compute the function F. This model guarantees complete fairness as both parties receive their respective outputs from the trusted third party simultaneously. Either of the parties can be corrupted by an adversary A, but there is also the possibility that both parties remain honest.

In the ideal model, the parties first send their inputs to the trusted party. If a party is honest, it sends its true input. However, a party under the adversary's control may choose to send any value, or even its true input. The inputs given to the trusted party are represented as (x', y'). If x' or y' do not belong to their respective input domain X_n or Y_n, the trusted party replaces them with a default element from the appropriate domain.

Upon receiving the inputs, the trusted party generates a uniform random variable r and computes the function outputs $f_n^1(x', y'; r)$ and $f_n^2(x', y'; r)$. These results are then sent simultaneously to P_1 and P_2, ensuring complete fairness.

The output of the adversary and the honest party after protocol execution are represented by the random variables $\text{OUT}^A_{F,A(\text{aux})}(x, y, \lambda)$ and $\text{OUT}^{\text{hon}}_{F,A(\text{aux})}(x, y, \lambda)$ respectively. The combined output of the model, represented by the random variable $\text{IDEAL}^{\text{MPC}}_{F,A(\text{aux})}(x, y, \lambda)$, is given by a tuple of the outputs of the adversary and the honest party. The formal definition is

$$\text{IDEAL}^{\text{MPC}}_{F,A(\text{aux})}(x, y, \lambda) \stackrel{\text{def}}{=} \left(\text{OUT}^A_{F,A(\text{aux})}(x, y, \lambda), \text{OUT}^{\text{hon}}_{F,A(\text{aux})}(x, y, \lambda)\right).$$

[15]

A protocol is said to be secure if, for every PPT attacker A in real execution, there exists a PPT simulator Sim in the ideal world such that

$$EXEC_{\pi,A} \approx IDEAL^{\text{MPC}}_{F,Sim}.$$

[20]

If such a simulator can be constructed, the protocol is considered secure, since the real-world adversary cannot learn more than the ideal-world simulator.

Secure with Abort In some cases, a protocol can be proven *secure with abort*, as defined by [15]. In this two party model, fairness is not always guaranteed. Here, the ideal functionality is altered, such that a malicious party P_1 has the option of sending either "continue" or "abort" to the trusted party. If the message is "abort", the computation is aborted and the other party, P_2, does not receive their output. In this specific definition, by [15] only a malicious P_1 is given the opportunity to abort the computation. The reason for this restriction is, that for the protocols proven to be secure-with-abort in, [15], a malicious P_2, by protocol design, can not gain any learn more than P_1 by aborting.

3.5.2 The Universal Composability (UC) Framework

The Universal Composability (UC) Framework as introduced by Canetti, 2000 [9] provides a model for proving the security of cryptographic protocols. It is designed to handle complex, concurrent protocol executions, providing a way to formulate and prove security properties that remain valid even when the protocol interacts with arbitrary other protocols. [9]

In [24] which uses the standalone security model, the focus is on understanding the security of an isolated protocol instance. The UC framework expands on this model. To achieve a security model that maintains security under arbitrary protocol composition, the stand-alone model is extended by allowing the distinguisher to interact with both the protocol and the adversary during protocol execution. In the UC context, the (interactive) distinguisher is called the environment Z.

Let $EXEC_{\pi,A,Z}(\lambda, r)$ be the output of the environment Z in the UC experiment with protocol π, adversary A, randomness r and security parameter λ. $EXEC_{\pi,A,Z}(\lambda)$ denotes the random variable associated with $EXEC_{\pi,A,Z}(\lambda, r)$, where r is chosen uniformly at random. $EXEC_{\pi,A,Z}$ represents the corresponding probability ensemble $\{EXEC_{\pi,A,Z}\}_{\lambda\in\mathbb{N}}$ [18].

Definition 3.5.2 (UC security)**.** A protocol π is said to UC-emulate a function Protocol ϕ, if for all Probabilistic Polynomial Time (PPT) adversaries A, there exists a PPT simulator S such that, for all PPT environments Z, the environment Z (acting as a distinguisher) cannot tell whether it is interacting with the real-world adversary or the ideal-world simulator. Formally, this can be denoted as:

$$EXEC_{\pi,A,Z} \approx EXEC_{\phi,S,Z}$$

Key to the UC framework is the concept of security preservation under composition: a protocol that has been proven secure in isolation remains secure when combined with other protocol instances. These instances may be of the same or different protocols. Composition can occur sequentially, where one protocol follows another; concurrently, where protocols overlap in time without executing simultaneously; or in parallel, where concurrent tasks are executed simultaneously on different processors. [9, 18]

4 Fairness Notions and Protocols

In this section, we look at the different notions of fairness that apply to secure multiparty computation (MPC). Complete fairness, which ensures that either all parties receive an output or none do, is an ideal goal. However, it is not always achievable in practice due to computational and contextual constraints. As a result, a variety of alternative notions of fairness have been developed. These different notions impose different requirements and constraints.

In the following subsections, we will explore these notions, by looking at how they can be defined and how they can be implemented: complete fairness, partial fairness, Δ-fairness, probabilistic fairness, fairness with penalties and gradual release along with protocols that achieve these notions.

4.1 Complete Fairness

Complete fairness in the context of MPC ensures that, by the end of the protocol, either all honest receive the computations output or none of the parties receive their output.

4.1.1 Definition and Context

For the definition of Complete Fairness, we use the concept of a simulation proof (see section 3.5.1). For this, we first define the real and ideal model similar to [15].

Definition 4.1.1 (Real Model)**.** In the real model, we describe the execution environment for a two-party protocol Π that takes place between two parties, P_1 and P_2. In this model, there is no trusted third party involved. One of the parties is controlled by an adversary A. The adversary A is a non-uniform probabilistic polynomial-time machine, which means that it is capable of performing probabilistic computations in polynomial time and its behaviour can change depending on certain circumstances. The adversary assumes the role of one party and issues all messages on behalf of the corrupted party using an arbitrary polynomial-time strategy. Whereas the other party, referred to as the honest party, strictly follows the instructions given by the protocol Π.

In such a setup, the "view" of the adversary after an execution of Π is represented by the random variable $\mathrm{VIEW}^{A}_{\Pi,A(\mathrm{aux})}(x, y, n)$. Here x and y are the inputs of the two parties, λ is the security parameter, and aux refers to some auxiliary input that might influence the adversary's behaviour. Similarly, the random variable $\mathrm{OUT}^{\mathrm{hon}}_{\Pi,A(\mathrm{aux})}(x, y, n)$ denotes the output that the honest party receives from the execution of the protocol.

The complete execution of the protocol in the real model is contained in the random variable $\mathrm{REAL}_{\Pi,A(\mathrm{aux})}(x, y, n)$, which is defined as a tuple combining the adversary's view and the honest party's output. The formal definition is

$$\mathrm{REAL}_{\Pi,A(\mathrm{aux})}(x,y,\lambda) \stackrel{\mathrm{def}}{=} \left(\mathrm{VIEW}^{A}_{\Pi,A(\mathrm{aux})}(x,y,\lambda), \mathrm{OUT}^{\mathrm{hon}}_{\Pi,A(\mathrm{aux})}(x,y,\lambda)\right).$$

[15]

Definition 4.1.2 (Complete Fairness)**.** A protocol Π is said to securely compute a functionality F with complete fairness if for every non-uniform probabilistic polynomial-time adversary A in the real model, there exists a non-uniform probabilistic polynomial-time adversary S in the ideal model such that

$$\left\{\mathrm{IDEAL}^{\mathrm{MPC}}_{F,S(\mathrm{aux})}(x,y,n)\right\}_{(x,y)\in X\times Y,\mathrm{aux}\in\{0,1\}^*} \stackrel{c}{\equiv} \left\{\mathrm{REAL}_{\Pi,A(\mathrm{aux})}(x,y,n)\right\}_{(x,y)\in X\times Y,\mathrm{aux}\in\{0,1\}^*}$$

In this definition, the notion of complete fairness is defined by the indistinguishability between the output of ideal model (see Definition 3.5.1) and the real model. Here, $\mathrm{IDEAL}_{F,S(\mathrm{aux})}(x,y,n)$ represents the output of the ideal model, where a trusted third party ensures that all parties receive their output simultaneously. This ensures fairness in the ideal model, as no party can gain an advantage by learning its output before others. $\mathrm{REAL}_{\pi,A(\mathrm{aux})}(x,y,n)$ represents the output of the real execution of the protocol π. The ideal model guarantees fairness, as all parties receive their output simultaneously from the trusted party. The notation $X \stackrel{c}{\equiv} Y$ denotes that the two distribution ensembles X, Y are computationally indistinguishable. [15, 16]

4.1.2 Protocols that achieve Complete Fairness

Choudhuri et al., 2017 [10] Achieving fairness in secure multiparty computation (MPC) is a not trivial, especially when faced with a dishonest majority. Choudhuri et al., 2017 [10] addresses this problem by proposing a model that achieves fairness in MPC through the use of public bulletin boards. These boards can be established using pre-existing infrastructure, such as blockchain technology or Google's certificate transparency logs.

A public bulletin board in this context is essentially a publicly accessible ledger where anyone can publish arbitrary strings. When a party publishes their data D on the bulletin board, they receive proof that D was indeed published. This proof is crucial, as it prevents unauthorized removal or alteration of the data once it's posted. Because of its public nature, anyone can view the contents of the bulletin board. This is critical to the security of the model, as it ensures that the content of the board cannot be deleted and the proof of publication cannot be forged, meaning that no one can create a counterfeit proof claiming that a particular data D was published when it was not. Both centralized systems, such as Google's certificate transparency project, and decentralized systems, such as blockchain-based ledgers like Ethereum with proofs of participation, can provide practical implementations of these bulletin boards.

Building on this, [10] offers both theoretical and practical constructions that use either witness encryption or trusted hardware, such as Intel SGX[1]. Their model achieves complete

[1]Trusted hardware refers to a secure and tamper-resistant computing environment provided by specialized hardware components. Intel SGX (Software Guard Extensions) is an example of such hardware, which creates secure enclaves within the CPU to execute code and store data in a protected manner.

fairness by leveraging the existing infrastructure and presents a new way to achieve fairness in MPC.

The protocol in [10] focuses on the two-party case, which can be extended to a multi-party case. At its core, the approach reduces the problem of fairness in MPC to the problem of fair decryption. The authors use a public bulletin board to implement a fair decryption protocol for a witness encryption scheme.

Witness encryption is a method of encryption (see Section 3.3.1) that allows a message to be encrypted with a problem statement, so that the message can only be decrypted by providing a solution to the problem. The authors use this to transform the problem of fair computation into a problem of fair decryption.

In this context, to securely compute a function f with complete fairness, the parties first run a standard (possibly unfair) MPC protocol to compute a randomized function. This function takes the parties' private inputs (y_1, y_2) and outputs a witness encrypted ciphertext CT of the desired output $f(y_1, y_2)$. The statement x associated with CT is set so that a valid witness for x is equivalent to proving the posting of a "release token" α (to be determined later) on the bulletin board.

The only way for any party to obtain such a witness is to post α on the bulletin board and obtain the corresponding proof of posting σ. However, this makes the pair (α, σ) public, so anyone can obtain it. This ensures that if a malicious adversary learns the witness for decrypting CT, so can the honest party, since they can simply read the public bulletin board.

This protocol works in a two-party setting, with a protocol that can be extended to a multi-party scenario. However, it does not require an honest majority, allowing complete fairness even in the presence of a dishonest majority. The communication model includes public bulletin boards for publishing arbitrary strings and achieving a fair decryption mechanism.

Gaddam et al., 2023 [12] Related to the work of [10], Gaddam et al., 2023 [12] extends the idea of fair computation protocols in a setting that uses multiple blockchains, as opposed to Choudhuri et al., 2017 [10]'s single bulletin board model. They explore a scenario where not all parties have equal access to read or write to a single blockchain, and not all parties are willing to post to a public blockchain. The authors propose protocols for fair secure computation in this multi-blockchain setting, where secure computation can be achieved as long as there is a subset of parties with access to a trusted hardware, and each of these parties shares some blockchain access with the others. Their work highlights the potential of using multiple blockchains, as well as existing trusted execution environments, to achieve complete fairness.

This protocol is designed for an n-party setting, where at most $t < n$ parties can be corrupt. For the protocol to achieve fair, secure computation, t parties must have access to a trusted execution environment, and each of these t parties shares blockchain access with all other parties, whereas only these t parties need write permissions on the blockchains. If all parties are honest, the protocol operates entirely outside the blockchain, which can significantly improve efficiency.

Gordon et al., 2008 [17] Gordon et al., 2008 [17] demonstrate that many interesting and non-trivial functions can be computed with complete fairness, even without an honest

majority, in the two-party setting. They propose two protocols, one for the fair computation of Yao's millionaires' problem[2] and one for fair computation of functions with an embedded XOR[3]. Both protocols are equally relevant, but for the scope of this thesis, we will focus on the first protocol.

In the first protocol, they achieve completely fair computation of the millionaires problem (and related functionalities).

The function f that both parties try to evaluate returns 1 if the first parties input is higher than the second parties input. The function is defined by

$$f_m(x_i, y_j) = \begin{cases} 1 & \text{if } i > j \\ 0 & \text{if } i \leq j, \end{cases}$$

where $X_m = \{x_1, \ldots, x_m\}$ denotes the valid inputs for the first party, $Y_m = \{y_1, ..., y_m\}$ denotes the valid inputs for the second party, ordered such that for $i < j, x_i < y_j$, and a polynomial $m = m(\lambda)$ denotes the size of the domain of each input, with security parameter λ.

The approach is based on a series of m iterations, where P_1 chooses an input x_i and learns the output in iteration i, and P_2 chooses an input y_j and learns the output in iteration j. Depending on the party's own input, they learn the output at different iterations. If one party aborts after receiving its message in round k and the second party hasn't received its output yet, the second party assumes that the first party learned its output in iteration k, and therefore had chosen x_k as its input. The second party then computes the output using input x_k for P_1.

The protocol does this in two phases. In the first phase, a secure with abort (see Section 3.5.1) protocol computes the output of a protocol referred to as "ShareGen" (defined in detail in Figure 8.1). ShareGen receives the parties inputs x_i and y_j and generates the messages $a_1, ..., a_m$ and $b_1, ..., b_m$, so it holds that $a_i = b_j = f(x_i, y_j)$ while all the other a_ℓ respectively b_ℓ are set to `NULL`. Then random shares of these a_ℓ respectively b_ℓ are generated (I.e., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.). Further, the shares, that are intended to be exchanged in the next phase, are authenticated with a MAC (see Section 3.3.3), denoted t_ℓ^a respectively t_ℓ^b. The MAC-keys generated by ShareGen will be sent to the receiving party respectively, so that MACs can be verified. Finally, for every a_ℓ respectively b_ℓ each party receives a share. P_1 receives the values $a_1^{(1)}, ..., a_m^{(1)}$ and $(b_1^{(1)}, t_1^b), ..., (b_m^{(1)}, t_m^b)$, and the MAC-key k_a and P_2 receives the values $(a_1^{(2)}, t_1^a), ..., (a_m^{(2)}, t_m^a)$ and $b_1^{(2)}, ..., b_m^{(2)}$, and the MAC-key k_b.

In the second phase, the shares $a_i^{(2)}$ and $b_i^{(1)}$, which P_2 respectively P_1 are holding, are exchanged, so that the opposing party can reconstruct the a_ℓ respectively b_ℓ. In each of the m rounds, P_2 begins sending $(a_i^{(2)}, t_i^a)$ to P_1, which verifies the MAC and then sets its output to the reconstructed a_i, if a value different to `NULL` is reconstructed. Then P_1 sends

[2] A classic problem in secure multi-party computation, where two millionaires want to determine who is richer without revealing their actual wealth to each other. This problem illustrates the need for MPC protocols that allow parties to jointly compute a function over their private inputs while preserving the privacy of those inputs. [30, 25]

[3] An "embedded XOR" refers to a function that includes an XOR operation as a component within a more complex function.

$(b_i^{(1)}, t_i^b)$ to P_2. P_2 also verifies the MAC and sets its output accordingly. If, during the second phase, one party, here P_1, receives a message with an invalid MAC or no message (because the other party aborted), P_1 outputs $f(x, y_1)$ and halts. Likewise, if P_1 aborts, P_2 outputs $f(x_1, y)$ and halts. The protocol is explained in detail in Figure 8.2.

If P_1 aborts in iteration $k < i$, it never learns the correct output, ensuring fairness. If P_1 aborts in iteration $k \geq i$, it obtains the output in iteration i and aborts in some iteration $k \geq i$. In this case, there are two subcases: If $j < k$, P_2 has already received its output in a previous iteration, ensuring fairness. If $j \geq k$, P_2 has not yet received its output. Since P_1 aborts in iteration k, the protocol directs P_2 to output $f(x_k, y) = f(x_k, y_j)$. Since $j \geq k \geq i$, we have $f(x_k, y_j) = 0 = f(x_i, y_j)$, ensuring fairness.

This protocol works in a two-party system and demonstrates that complete fairness is possible even without an honest majority for certain functions.

In summary, achieving complete fairness in MPC protocols depends on certain assumptions. As demonstrated by Gordon et al., 2015 [16], the assumption of an honest majority is crucial for ensuring complete fairness. Additionally, Gordon et al., 2008 [17] shows that complete fairness can be achieved without an honest majority in the two-party setting, using specific protocols for the computation of certain non-trivial functions.

4.1.3 Limitations and Impossibility Results

Achieving complete fairness in secure multiparty computation protocols is far from trivial. It is often subject to limitations. It has also been shown to be impossible in certain settings. The limitations and impossibilities associated with complete fairness have been studied in detail in various contexts.

General impossibility One of the earliest impossibility results was shown by Cleve, 1986 [11]. They showed that in a setting without an honest majority, complete fairness is generally impossible to achieve. The idea behind this is based on the idea that in a protocol, communication takes place in rounds. During these rounds, there exists a critical moment where a participant receive enough information to be able to compute their own output for the first time. If a dishonest party can predict this round, they can strategically abort the communication. By doing so, the dishonest party can obtain their output without revealing their information, leaving the honest party without knowledge of their output.

Public ledgers Despite this initially strict constraint, researchers have found scenarios where complete fairness can be realized. Choudhuri et al., 2017 [10] showed that complete fairness for general functions can indeed be achieved when parties have access to a public ledger or bulletin board.

Lower Bounds for Round Complexity Achieving complete fairness becomes much more complex when the round complexity of a protocol is taken into account. A notable result by Gordon et al., 2015 [16] showed that 2-round fair multiparty computation (MPC) for general functions is impossible, even with an honest majority. However, a 3-round MPC

protocol with guaranteed output delivery is feasible for general functions in the Common Random String (CRS) model[4].

Work by Asharov et al., 2011 [2] demonstrated a 5-round protocol that provides security with guaranteed output delivery, thus ensuring fairness with the best-known round complexity, assuming the existence of an honest majority.

Exceptions to [11]'s result In addition, the question of round complexity in achieving complete fairness was addressed by Gordon et al. [17]. They showed the feasibility of complete fairness without honest majority for certain functions over polynomial-size domains that do not contain an "embedded XOR". They also showed certain functions that contain an embedded XOR that also allow complete fairness. However, they proved a lower bound, showing that any protocol that ensures complete fairness for such functions must have a round complexity that is super-logarithmic in the security parameter.

4.2 Partial Fairness

Partial fairness is a relaxation of complete fairness where the security requirement is satisfied within a certain probability bound. This notion is particularly relevant in scenarios where complete fairness may be infeasible or too expensive in terms of computational resources or round complexity. Here, the goal is still that either all parties learn the correct output or none of them do. However, this can only be guaranteed with a certain probability.

4.2.1 Definition and Context

In the case of the protocols proposed by Gordon and Katz, 2008 [15] and Bailey et al., 2022 [3], the terms $\frac{1}{p}$-security and $\frac{1}{R}$-fairness are used to describe the level of fairness a protocol achieves. While both describe similar concepts, they differ in their underlying parameters.

The term $\frac{1}{p}$-security, as used by [15], allows for a relaxation of the simulation requirements between the real and ideal world, while reusing the same ideal world (see Definition 3.5.1) that was used in Definition 4.1.2. , so that the real and ideal worlds may be distinguishable with probability at most $\frac{1}{p}$. Here, p is some specified polynomial, and the number of rounds is dependent on this p.

On the other hand, $\frac{1}{R}$ fairness, introduced by [3], is directly related to the total number of communication rounds R in the protocol. If the adversary cannot predict the round in which the true output is revealed, and the best strategy is to abort a round at random, there is a $\frac{1}{R}$ probability that he will abort in the critical round, such that he gets the output while the honest parties do not. Hence, if the adversary aborts at round r, he has a probability of $\frac{r+1}{R}$ of getting the correct output, while the honest party has a probability of $\frac{r}{R}$. For the scope of this thesis, we use the definition proposed by [15].

Definition 4.2.1 (Partial Fairness)**.** Formally, a protocol Π is said to $1/p$-securely compute a functionality F if for every non-uniform probabilistic polynomial-time (PPT) adversary

[4]The Common Random String (CRS) model is a setup model in cryptography. In this model, all parties have access to a common, randomly chosen string. The string is assumed to be generated honestly. It is used to simulate shared randomness among the parties, which can be leveraged for protocol design.

A in the real model, there exists a non-uniform PPT adversary S in the ideal model such that:

$$\left\{\text{IDEAL}^{\text{MPC}}_{F,S(\text{aux})}(x,y,n)\right\}_{(x,y)\in X\times Y,\text{aux}\in\{0,1\}^*,n\in\mathbb{N}} \overset{1/p}{\approx} \left\{\text{REAL}_{\Pi,A(\text{aux})}(x,y,n)\right\}_{(x,y)\in X\times Y,\text{aux}\in\{0,1\}^*,n\in\mathbb{N}}$$

Here, the notation $\overset{1/p}{\approx}$ is defined as follows: A function $\mu(\cdot)$ is negligible if for every positive polynomial $p(\cdot)$ and all sufficiently large n, it holds that $\mu(n) < \frac{1}{p(n)}$. For a fixed function p, the ensembles $X = \{X(a,n)\}_{a\in D_n,n\in\mathbb{N}}$ and $Y = \{Y(a,n)\}_{a\in D_n,n\in\mathbb{N}}$ are computationally $1/p$-indistinguishable, denoted $X \overset{1/p}{\approx} Y$, if for every non-uniform polynomial-time algorithm D there exists a negligible function $\mu(\cdot)$ such that for every n and every $a \in D_n$, we have

$$|\Pr[D(X(a,n)) = 1] - \Pr[D(Y(a,n)) = 1]| \leq \frac{1}{p(n)} + \mu(n).$$

We two distribution ensembles are computationally indistinguishable, denoted $X \overset{c}{\equiv} Y$, if for every $c \in \mathbb{N}$ they are computationally $1/n^c$-indistinguishable. The real model and ideal model are defined as in Definition 4.1.1 respectively Definition 3.5.1. [15]

4.2.2 Protocols that achieve Partial Fairness

Gordon and Katz, 2008 [15] Building on the work of Gordon et al., 2008 [17], which shows that complete fairness can only be achieved for certain specific functions, Gordon and Katz, 2008 [15] introduce the concept of partial fairness. In their work, they describe a $\frac{1}{p}$-secure protocol that can compute any function with polynomial inputs and outputs. In their approach, the authors propose a two-stage protocol.

Like the protocol by [17] described in Section 4.1.2, this protocol works in two stages. The stages themselves however differ from the earlier protocol. The first stage, referred to as the pre-processing stage, chooses a "critical round" at random in which both parties discover the "true output". This is in contrast to [17], where this round was chosen dependent on the parties private inputs. This stage involves the generation of random shares containing the parties outputs and distributing these shares between the parties. The second stage involves several iterations in which the parties exchange and reconstruct the shared values from the first stage. At the end of these rounds, each party will have a final output value. If a party leaves the protocol early, the remaining party uses the last successfully reconstructed value as its output. This protocol is designed to maintain fairness, by ensuring that no party can reliably predict the round in which the "true output" will be learned.

The protocol starts with the first stage, where a trusted party, ShareGen, receives the parties inputs and determines a critical round $i^* \in 1,\ldots,r$ and generates corresponding sets of values $a_1,\ldots,a_r$ and $b_1,\ldots,b_r$. For each round i less than the critical round, a_i and b_i are derived from random function evaluations. Conversely, for rounds $i \geq i^*$, the values a_i and b_i correspond directly to the true function output $f(x,y)$. ShareGen then uses a secure-with-abort protocol to distribute these values between the parties. Each value is also authenticated with a MAC t_i^a respectively t_i^b, ensuring the integrity of the shared data.

In the second phase, the parties carry out r rounds. During each iteration, they exchange shares and reconstruct the values a_i and b_i, with P_2 initiating the exchange. If one party aborts during an iteration, the other party outputs the last successfully reconstructed value. This is repeated for r rounds.

After r rounds, P_1 outputs a_r and P_2 outputs b_r. If a party aborts in some iteration, the other party outputs the value reconstructed in the previous iteration. The protocol is explained in detail in Figure 8.4.

The protocol achieves partial fairness because fairness is violated only if P_1 aborts exactly in iteration i. If P_1 aborts before iteration i, neither party learns the correct value $f(x, y)$. If it aborts after iteration i, both parties learn the correct value. An abort by P_2 in iteration i^* does not violate fairness, as P_1 has already learned the output. The authors show that even if a party knows the value of the output, they cannot determine with certainty when iteration i, in which the "true output" is learned, occurs, thus maintaining partial fairness.

This protocol, as described above, is not secure-with-abort, however. To achieve this, ShareGen has to be modified, so the i^* is chosen uniformly from $\{2, \ldots, r+1\}$ and set $b_{i^*-1} = \perp$, where $\perp$ is a distinguished value outside the range of f. With this modification, [15] proves the protocol to be secure-with abort (see **??**). This security definition only gives P_1 the opportunity to abort. As described above, an abort by P_2 would not violate fairness.

In [17] this protocol (referred to as Protocol 2) is also used to illustrate that functions with an embedded XOR can even be computed with complete fairness. While [15] achieves Partial Fairness as long as the size of both input and output is polynomial, [17] shows that only for certain functions Complete Fairness can be achieved.

Bailey et al., 2022 [3] Bailey et al., 2022 [3] get around the limitations of [15], which presumed the size of both input and output to be polynomial. [3] present a protocol which allows for any MPC to be carried out with $\approx 1/R$ partial fairness, where R is the number of communication rounds.

This protocol, similar to [15], achieves partial fairness by obfuscating the critical round where the true output is revealed. However, this protocol is extended to support general multi-party functionality.

In the protocol every participant first generates a key pair and a random value, then uses a Verifiable Delay Function (VDF)[5] and the private key to create share randomness and round randomness. These are then committed and broadcasted along with the public keys and random values. Following this, all parties enter their inputs, share randomness, and round randomness into a secure-with-abort MPC protocol.

[5] A Verifiable Delay Function (VDF) is a function that takes a fixed amount of time to compute, so the computation cannot be accelerated by parallel processing or faster hardware. Once the computation is complete, the output can be verified quickly and efficiently, despite the time taken for the initial computation. The function `VDF.KeyGen` generates a public and a secret key. The functions `VDF.Trapdoor` and `VDF.Eval` can be modelled as random oracles that compute y as a function of an input x. While `VDF.Eval` is designed to take the adversary time Δ to compute, even on a parallel computer, `VDF.Trapdoor` computes the same y using the secret key sk and may take less time. Both functions also produce π, a quickly verifiable "proof". The function `VDF.Verify` can then take such a proof along with x, y and will return 1 if this is the result of a valid call to `VDF.Trapdoor` or `VDF.Eval`, otherwise it will return 0. [7, 3]

This protocol calculates a critical round and an encryption pad, with commitments for messages being exchanged between every pair of parties for each round. These messages either contain a random function evaluation or the true output, depending on the round. All parties then receive commitments for messages intended for them, completing the pre-processing phase.

In the reveal phase, commitments are revealed and verified. After this, parties have the option to either compute the VDF evaluation to obtain the critical round and encryption pad, which takes a certain amount of time, or they can open their commitments to the share and round randomness. Upon receiving all messages, each party can then reconstruct the true output.

In detail, the protocol works as follows. In the initial phase of the protocol, all parties generate a key pair and broadcast the public key. Each party then chooses a random value v_i which, together with the private key, is used with the VDF trapdoor to generate a pair, $((s_i, r_i), \pi_i) := \text{VDF.Trapdoor}(sk, v_i, \Delta)$, representing share randomness and round randomness. A VDF proof π is also generated.

In a next step, The parties then provide their data x_i, their share randomness s_i and their round randomness r_i to a secure-with-abort MPC protocol. The protocol computes the critical round $r^* := \sum_{i \in N} r_i \mod R$ and $s := \oplus_{i \in N} s_i$, which will be used as an encryption pad. For the upcoming reveal phase, the MPC then sends a message $m_{i,j,r}$ to party i for each pair of parties i, j and every round r. The message is intended to be sent from party i to party j in the rth round. For $r \neq r^*$ the value for $m_{i,j,r}$ is chosen uniformly at random. For $r = r^*$ the messages $(m_{i,j,r^*})_{j \in N}$ are a random secret shares OTP encrypted with s with $\bigoplus_{j \in N} m_{i,j,r^*} = s \oplus f_j(\overline{x})$. Additionally, every party j receives commitments to all the messages $m_{i,j,r}$ intended for them and party i receives the corresponding opening for that commitment. Lastly, all parties receive commitments to s_i, r_i that can be opened by party i. This concludes the pre-processing step.

For the reveal phase, every party starts the VDF timer and the parties open the commitments to their v_i to each other. The protocol continues by the every party i opening their commitments to each $m_{i,j,r}$ and party j verifying the opening.

With the v_i revealed, every party could compute VDF.Eval in order to calculate r^* and s, which requires Δ time to do. Instead, parties can also open their commitments to the s_i, r_i. If a party has received all messages m_{i,j,r^*} it can output $s \oplus (\bigoplus_{j \in N} m_{i,j,r^*})$. The protocol is explained in detail in Figure 8.5.

This protocol archives $\frac{1}{R}$-fairness, as for the adversary, the share randomness and round randomness is unknown during the reveal phase and all shares are randomly distributed. This effectively means that the adversary's optimal strategy is to stop at a particular round r_A during the reveal phase.

Depending on the value of r_A relative to the critical round r^*, fairness is ensured. If $r_A = r^*$, the opponent learns his output while the honest parties do not. If $r_A < r^*$, no party learns its output. If $r_A > r^*$, all parties learn their output. The protocol is thus called 1/R-fair, since the probability of the first case occurring is at most 1/R.

The VDF plays a critical role in fairness by obscuring the critical round and preventing an adversary from making decisions based on instant calculations of the expected output. It is important to hide both the critical round r^* and the encryption pad s in the VDF output. Obfuscating r^* prevents the adversary from knowing the exact round to stop.

Similarly, hiding s prevents parties from computing $s \oplus (\bigoplus_{j\in[N]} m_{i,j,r^*})$ and comparing it the expected output, ensuring that no party can abort based on the output of the previous round.

The VDF within the protocol effectively hides r^* and s. Parties can find these values in two ways. They can wait until the reveal phase ends and then use the opened commitments to s_i, r_i to compute $r^* := \sum_{i\in N} r_i \mod R$ and $s := \oplus_{i\in N} s_i$. Alternatively, they can calculate VDF.Eval using v_i unveiled at the start of the reveal phase. Because computing VDF.Eval uses all remaining time in the reveal phase, it stops any adversary from revealing r^* and s during the reveal phase.

4.2.3 Limitations and Impossibility Results

Despite the trade-off between fairness and feasibility that partial fairness provides, there are still limitations and infeasibility results associated with it. The protocol proposed by Gordon and Katz, 2008 [15] is limited to only two parties and has shown specific limitations when dealing with super-polynomially large input domains.

In their work, they show that no protocol computing a certain deterministic Boolean function can simultaneously achieve both security-with abort and $\frac{1}{p}$-security for $p > 4$ when the input domains are of super-polynomial size. They also present a deterministic function with super-polynomial sized input and output domains that cannot be computed $\frac{1}{p}$-safely for $p > 2$.

Furthermore, they prove that for a given functionality there is no protocol that can compute it even under the notion of partial fairness (Theorem 11 in [15]).

4.3 Δ-Fairness

Δ-Fairness, introduced by Pass et al., 2016 [27], is a relaxed form of complete fairness. In this variant, a certain amount of time discrepancy between when the adversary and the honest party learn the result of the computation is acceptable.

4.3.1 Definition and Context

Definition 4.3.1 (Δ-Fairness)**.** In Δ-fairness, if an adversary decides to abort the protocol and learns the output in round r, the honest party should be able to determine the output no later than in round $\Delta \cdot r$. Here Δ is a polynomial that, given the round the adversary learns the output, determines the latest round in which the honest party leans the output.

Δ-Fair Ideal Model The Δ-Fair Ideal Model, as proposed in [27], works within the clock model of execution[6] and introduces a fairness parameter Δ, which is a fixed polynomial function. This model ensures that if an adversary receives the output by round r, the honest parties are guaranteed to receive their outputs by round $\Delta(r)$.

[6]The clock model of execution refers to a computational setting in which each party has access to a secure hardware clock.

If all parties behave honestly, the protocol will terminate in $g(\lambda)$ rounds, where g is a fixed polynomial. All parties will receive the correct output with negligible probability of error.

However, if at least one party is corrupt, the running time of the protocol may depend on the actions of the adversary and is not necessarily bounded by a fixed polynomial. In that case, the ideal fair two-party computation functionality can delay the honest parties output as follows:

When a message (`"compute"`, inp_i) is received from party P_i, where $i \in \{0, 1\}$, the ideal functionality checks whether the other party P_{1-i} has also sent the message (`"compute"`, inp_{1-i}). If both parties have sent their inputs, the model computes the function outputs $(\text{outp}_0, \text{outp}_1) := f(\text{inp}_0, \text{inp}_1)$.

The adversary A can send a message (`"output"`, δ^*) to the ideal functionality. Assuming, both parties had previously sent their inputs, the delay δ, by which the honest parties output delivery will be delayed, is then calculated as the minimum of δ^* and $\Delta(r)$, where r is the current round counter.

After verifying that the outputs $(\text{outp}_0, \text{outp}_1)$ have been stored, the model immediately sends outp_b to A, where b is the corrupt party. However, the output outp_{1-b} is sent to the honest party after a delay of δ rounds.

The output of the adversary and the honest party after protocol execution are represented by the random variables $\text{OUT}^{\Delta,A}_{F,A(\text{aux})}(x, y, n)$ and $\text{OUT}^{\Delta,\text{hon}}_{F,A(\text{aux})}(x, y, n)$ respectively. The combined output of the model, represented by the random variable $\text{IDEAL}^{\Delta}_{F,A(\text{aux})}(x, y, n)$, is given by a tuple of the outputs of the adversary and the honest party. For the scope of this thesis, we define the ideal model for Δ-Fairness as follows:

$$\text{IDEAL}^{\Delta}_{F,A(\text{aux})}(x, y, n) \stackrel{\text{def}}{=} \left(\text{OUT}^{\Delta,A}_{F,A(\text{aux})}(x, y, n), \text{OUT}^{\Delta,\text{hon}}_{F,A(\text{aux})}(x, y, n)\right).$$

[27]

4.3.2 Protocols that achieve Δ-Fairness

Pass et al., 2016 [27] In their work, Pass et al., 2016 [27] introduced the concept of Δ-fairness.

The authors propose a two-party computation protocol that achieves Δ-fairness (with $\Delta(r) = 2r$) by relying on all participants having secure processors equipped with tamper-resistant clocks. A secure processor is a piece of hardware that realizes a trusted execution environments. It provides secure computation that often includes mechanisms such as encryption, secure storage, and other features to prevent data leakage and resist tampering. Tamper-resistant clocks are crucial for time-dependent security measures, as they are designed to resist any malicious attempts to manipulate their timekeeping. They model the core functionality of secure attested execution processors[7] in an ideal functionality, they refer to as $\mathcal{G}_{att}$.

[7] "Secure attested execution processors" refers to specialized hardware components that, in addition to providing a trusted execution environment, also provide an attestation mechanism. Attestation allows the processor to provide cryptographically secure proof that a particular piece of code has been executed correctly

The protocol is structured so that the secure processors of both parties negotiate the timing of the release of the computation output. Both processors exchange encrypted inputs and are able to compute the output, but they deliberately delay revealing the results to their owner for a predefined period of time.

The protocol initiates with a timeout value (δ) which is exponentially large, specifically set as $\delta := 2^{\lambda}$, with a security parameter λ. The processors then engage in a series of rounds, during which they exchange acknowledgments via a secure channel.

Upon receipt of an acknowledgment, a processor commits to a reduction in the timeout period by a factor of 2, before it discloses the output to its associated party. Formally, the new timeout value is recalculated as $\delta := \frac{\delta}{2}$.

This process ensures a Δ-fair release of the output, accommodating the possibility of adversarial behaviour from one party. In the scenario where a dishonest party manages to acquire the output during round r before aborting, the mechanism guarantees that the honest party receives the output by round $2r$ This is ensured, as the delay δ after which the secure processor releases the output, is, for each party, halved in each round and thus by aborting the adversary can only obtain a value of δ that is half the value for δ the secure processor of an honest party holds. This leads to the establishment of Δ-fairness, with Δ being a function defined as $\Delta(r) = 2r$.

4.3.3 Limitations and Impossibility Results

The Δ-fairness protocol proposed by Pass et al., 2016 [27] represents a new approach to addressing fairness in secure two-party computation. However, in order to understand the practical implications of this method, some limitations, and constraints need to be acknowledged.

An essential factor that limits the application of this protocol is the requirement that all parties need to be equipped with a clock-aware secure processor. Fairness becomes impossible if only one of the two parties is equipped with a secure processor. This is because the party without a secure processor can complete the remaining computation immediately, thereby breaking fairness. This limitation, again, remains even if the adversary is only fail-stop[8], limiting the usefulness of the protocol in scenarios where not all participants have the necessary hardware.

However, it is notable that while general functions cannot be Δ-fairly computed when only one party has an attested execution processor, the protocol can Δ-fairly compute a broader range of functions than is achievable in a setting without any secure processors. [27] illustrates this by proposing how a Δ-fair 2-party coin toss protocol.

Finally, a significant limitation is the adversary's ability to choose the round r in which to abort. Although the protocol ensures that, if an adversary leans the output in round r^*, an honest party learns the output in twice the time ($\Delta(r^*) = 2r^*$) it takes the adversary, this time difference can still be significant.

[8] In the fail-stop model, an adversary can only prematurely abort the protocol, but cannot perform any other malicious actions such as modifying messages

4.4 Gradual Release

The concept of "Gradual Release", is a different fairness notion. When employing Gradual Release, secret information are exchanged gradually between two mutually distrusting parties. This notion of fairness ensures that neither party can cheat the other by releasing secrets in small chunks, each verifiable before the next.

4.4.1 Definition and Context

In this setup, we consider two parties: Party A, which can be either honest or potentially corrupted by an adversary, and Party B, which always acts honestly. Party B always initiates the process by sending a piece of its secret to party A. Since party B always starts the exchange, there's no advantage for it to abort prematurely; it would simply withhold its own secret without gaining any of party A's information. Therefore, only party A, if corrupted, has the decision to continue or abort the protocol after each round.

Definition 4.4.1 (Gradual Release)**.** With gradual release, if an adversary decides to abort the protocol after releasing part of its secret, then at most the adversary would have one more chunk (e.g. bit) of the honest party's secret than the honest party has of the adversary's secret.

4.4.1.1 Gradual Release Ideal Model

The gradual release ideal model involves the gradual, bit-by-bit release of secrets. In this model, a trusted third party mediates the exchange between a party (party A) that may be under the control of an adversary and an honest party (party B). The adversary can potentially decide to abort the computation after revealing part of its secret, but before Party B receives the corresponding part of its secret.

This trusted party, possessing the secrets of both party A and party B, first divides each secret into k equally sized chunks, possibly single bits. The secrets are represented as $s_1^A, s_2^A, \ldots, s_k^A$ and $s_1^B, s_2^B, \ldots, s_k^B$ for Party A and Party B respectively.

In the first round, the trusted party sends s_1^A to party A. If party A is honest, it will always send a *continue* message to the trusted party. If party A is adversarial, it can choose to send either a *continue* or a *abort* message. On receiving a "continue" message, the trusted party sends s_1^B to party B and the process moves on to the next chunk.

If Party A (the adversary) sends an abort message after the trusted party has sent s_i^A to Party A, where i is the current round, the trusted party then terminates the protocol. Thus, when the protocol is terminated, party A has exactly one more chunk than party B.

The outputs from this model are denoted as $\mathrm{OUT}^{\mathrm{grad},A}_{F,A(\mathrm{aux})}(x,y,n)$ for Party A and $\mathrm{OUT}^{\mathrm{grad},B}_{F,A(\mathrm{aux})}(x,y,n)$ for Party B. The combined output of the model, represented by $\mathrm{IDEAL}^{\mathrm{grad}}_{F,A(\mathrm{aux})}(x,y,n)$, includes the outputs of both parties after execution in this Gradual Release Ideal Model. The formal definition of this output is

$$\mathrm{IDEAL}^{\mathrm{grad}}_{F,A(\mathrm{aux})}(x,y,n) \stackrel{\mathrm{def}}{=} \left(\mathrm{OUT}^{\mathrm{grad},A}_{F,A(\mathrm{aux})}(x,y,n), \mathrm{OUT}^{\mathrm{grad},B}_{F,A(\mathrm{aux})}(x,y,n)\right)$$

4.4.2 Protocols that achieve Fairness using Gradual Release

Blum, 1983 [5] In their paper, Blum, 1983 [5] proposed a protocol that embodies the concept of gradual release and allows two adversaries to exchange secrets fairly. Specifically, the protocol addresses a scenario where two parties, Alice and Bob, wish to securely exchange secret prime factors of their respective public composite numbers.

Each public composite number, n_A and n_B, is a product of two primes: $n_A = p_{1A} \times p_{2A}$ and $n_B = p_{1B} \times p_{2B}$. The prime factors p_{1A} and p_{1B} are the secrets that Alice and Bob want to exchange.

While both Alice and Bob know their own factors and the composite number of the other party (n_B and n_A respectively), neither has knowledge of the prime factors of the other's composite number.

The protocol begins by constructing the numbers n_A and n_B, each of which is the product of two randomly generated 60-digit prime numbers. The numbers are then swapped and checked for validity: they must not be a unit, an even number, a prime, a non-trivial power of an integer, or anything other than a positive integer of 120 digits (or less). If any of these checks fail, the other party has violated the protocol and the process is terminated.

Next, Alice and Bob randomly choose 100 numbers a_i and b_i respectively from the set $Z_{n_B/2} = (1, 2, ..., [n_B/2])$ and $Z_{n_A/2} = (1, 2, ..., [n_A/2])$. They compute the greatest common divisor $\gcd(a_i, n_B)$ and $\gcd(b_i, n_A)$ respectively. If one of the numbers a_i or b_i splits n_B or n_A, the process is aborted, as this indicates either that n_B or n_A has more than two prime factors or, under certain assumptions, that the splitting party was lucky enough to pick a factor of the other number. If none of the numbers split n_B or n_A, Alice and Bob compute and exchange the quadratic residues $a_i^2 \bmod n_B$ and $b_i^2 \bmod n_A$ respectively.

In number theory, a number a is said to be relatively prime to another number n if their greatest common divisor (GCD) is 1. If n is a composite number, that is, the product of two different odd prime numbers, a has an interesting property. The number a, which is relatively prime to n, will have exactly four square roots modulo n. In other words, there will be four different numbers x_1, x_2, x_3, x_4 such that $x_i^2 \equiv a \mod n$ for $i = 1, 2, 3, 4$. This property is discussed in detail in [21].

Alice and Bob then calculate the four square roots $\bmod n_A$ respectively $\bmod n_B$ of each number received from the other party. Since the prime factors of n_A and n_B are known by Alice respectively Bob, this can be done efficiently. If one of the numbers is not a square root $\bmod n_A$ or $\bmod n_B$, the process is aborted, as this indicates cheating. For each number, the four square roots are ordered by size and the two largest roots are deleted. The remaining two roots are denoted as $\text{sqrt1}(b_j \bmod n_a), \text{sqrt2}(b_j \bmod n_a)$ respectively $\text{sqrt1}(a_j \bmod n_b), \text{sqrt2}(a_j \bmod n_b)$.

Finally, Alice and Bob exchange 100 pairs of square roots, one bit at a time, or more precisely, one hundred pairs of bits (i.e. 200 bits) at a time, with the most significant bits first. It should be noted that the number 100 is chosen arbitrarily by [5] to suggest a reasonable value for the exchange Throughout the exchange, they ensure that the other party is not cheating by checking the most significant bits of the square roots received from the other party to make sure that one of these strings matches the most significant bits of the number they initially selected. If either of these checks fails, the protocol is aborted.

This protocol provides a method of exchanging secrets bit by bit, with the revealing party providing a proof of validity for each bit revealed. This mitigates the risks associated with traditional bit-by-bit exchange methods, such as the other party aborting the exchange after receiving but not sending a bit, or the other party sending a "junk" bit. The protocol achieves fairness by ensuring that the level of unfairness, measured as the difference between the number of bits revealed by each party, remains small at all times.

Pinkas, 2003 [28] In their work, Pinkas, 2003 [28] discuss Yao's protocol [30] for secure two-party computation and its limitations. Yao's protocol allows two parties to compute a function, ensuring privacy. However, one of the main limitations of Yao's protocol is the lack of fairness. One party can learn its output first and terminate the protocol before the other party has learned its output.

To overcome this limitation, [28] propose a modification of Yao's protocol. They suggest using commitments on the outputs of both parties. After exchanging the commitments, the parties gradually reveal the openings for their commitments bit by bit, to prevent one party from gaining an advantage over the other.

However, this solution introduces a new problem. A more powerful party could use parallel brute-force attacks to discover the rest of the opening faster. To solve this problem, [28] introduce a new concept called "gradual release timed commitments", which combines timed commitments with a gradual release of the commitments opening.

Timed commitments, as introduced by Boneh and Naor, 2000 [6] represent an extension of standard commitments that includes a forced opening phase that allows the receiver to recover the committed value without the help of the sender. This ensures that the committed value remains hidden from the receiver only for a specified time and is resistant to parallel attacks. Thus, even if the receiver has significantly more computational resources, it cannot discover the committed value much earlier than a single-processor receiver.

By combining these timed commitments with the concept of gradual release, which refers to the slow release of information over a period of time, the modified protocol guarantees that the commitments opening is released in a timed manner.

4.4.3 Limitations and Impossibility Results

One of the major limitations of the gradual release protocols described above is that they can be computationally intensive and time-consuming, as secrets are exchanged bit by bit. This can be impractical for larger secrets or in time-sensitive applications.

The gradual release protocols described above are designed for two-party settings.

The Blum, 1983 [5] protocol assumes that the parties have equal computing power and knowledge of algorithms, and that factoring a number that is a product of two large "randomly chosen" primes is hard.

Protocols using timed commitments, such as the one proposed by Pinkas, 2003 [28], rely on the assumption that a particular computational task (e.g. factoring a large number) cannot be performed by the receiver within a given time. This assumption may not always hold, especially as computing power advances. Timed commitments, are designed to be resistant to parallel attacks. However, this assumes that the party does not have access to significantly more computational resources than expected.

Gradual release does not completely eliminate the risk of a party aborting the protocol prematurely. Although the concept mitigates this risk by reducing the advantage a party can gain from aborting, a party can still learn some parts of the secret before aborting.

4.5 Fairness with Penalties

Fairness with penalties is a concept in secure computation where an adversarial party is forced to pay a predetermined monetary penalty if it chooses to abort the protocol after receiving its output. This incentivizes parties to behave honestly and complete the protocol.

4.5.1 Protocols that achieve Fairness with Penalties

Bentov and Kumaresan, 2014 [4] A study by Bentov and Kumaresan, 2014 [4] proposed a new approach to fairness in secure computation by adding a monetary penalty for adversarial parties that abort after receiving their output. To implement this, they utilized the bitcoin network and extended its application to a wide range of scenarios, including those with a dishonest majority.

Central to their protocol is an ideal functionality called $\mathcal{F}_{CR}^{\star}$ (claim or refund), which is built around specific properties of bitcoin that are critical to its penalty-enforced fairness mechanism. In essence, $\mathcal{F}_{CR}^{\star}$ allows a party P_s to deposit an amount x of bitcoin while specifying a certain condition Φ_r. If party P_r can provide a witness w_r satisfying $\Phi_r(w_r) = 1$ within a certain time, it can claim the deposited amount. Otherwise, P_s can reclaim the amount after the timeframe. The ideal functionality $\mathcal{F}_{CR}^{\star}$ defined as follows:

Definition 4.5.1 ($\mathcal{F}_{CR}^{\star}$)**.** $\mathcal{F}_{CR}^{\star}$ with session identifier sid, running with parties $P_1, \ldots, P_n$, a parameter 1^{λ}, and an ideal adversary S proceeds as follows:

- **Deposit phase.** Upon receiving the tuple $(\text{deposit}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, \text{coins}(x))$ from P_s, record the message $(\text{deposit}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x)$ and send it to all parties. Ignore any future deposit messages with the same ssid from P_s to P_r.
- **Claim phase.** In round τ, upon receiving $(\text{claim}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x, w)$ from P_r, check if (1) a tuple $(\text{deposit}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x)$ was recorded, and (2) if $\phi_{s,r}(w) = 1$. If both checks pass, send $(\text{claim}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x, w)$ to all parties, send $(\text{claim}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, \text{coins}(x))$ to P_r, and delete the record $(\text{deposit}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x)$.
- **Refund phase:** In round $\tau + 1$, if the record $(\text{deposit}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x)$ was not deleted, then send $(\text{refund}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, \text{coins}(x))$ to P_s, and delete the record $(\text{deposit}, \text{sid}, \text{ssid}, s, r, \phi_{s,r}, \tau, x)$.

Definition 4.5.2 (Fairness with Penalties)**.** A protocol is said to be *fair with penalties* if it achieves secure computation by ensuring that

- An honest party never suffers a penalty.

- If an adversary S learns the output but aborts without delivering the output to honest parties, each honest party is compensated.

This is achieved by the ideal functionality $F_f^\star$, which works as follows:

Definition 4.5.3 ($\mathcal{F}_f^\star$)**.** Functionality $F_f^\star$ with session identifier sid[9] running with parties $P_1, \ldots, P_n$, a security parameter λ, and simulator S that corrupts parties $\{P_s\}_{s \in C}$ proceeds as follows. Let $H = [n] \setminus C$ and $h = |H|$. Let d be a parameter representing the safety deposit, and let q denote the penalty amount. The functionality is divided into the following phases:

- **Input phase:** The functionality receives inputs for the function f from all parties. Honest parties deposit a fixed amount of $coins(d)$. S may deposit coins, which may be used to compensate honest parties if S aborts after receiving outputs.
- **Output phase:** Honest parties will be refunded their initial deposits. Depending on the coins deposited by S:
  - If insufficient coins are deposited, S does not receive the output and may pay a penalty to some subset H' of honest parties.
  - If enough coins are deposited, S looks at the output and may decide:
    * Deliver the output to all parties, and possibly pay an additional penalty to some subset H''.
    * To abort. In this case, all honest parties are compensated with the penalty deposited by S.

The specification of sets H' and H'' in the definition may seem somewhat unnatural. However, these sets are required to prove the security properties of the functionality. H' represents the honest parties that have actively participated in a specific phase of the protocol by revealing their tokens, and (H'' deals with cases where the last party is corrupt and makes selective deposit claims. For a detailed understanding, we refer to the original work [4].

The functionality $\mathcal{F}_{CR}^\star$ can be implemented efficiently in the Bitcoin network using the protocol proposed in [4]. The depositing party, P_s, first creates a transaction that deposits the coins and sets the redemption condition (either the signatures of both parties, or the valid witness and the signature of P_r). P_s then prepares a refund transaction that issues this deposit back to itself, set to become valid only after the timeframe. After P_r has signed this refund transaction, P_s broadcasts the deposit transaction to the Bitcoin network, effectively locking the coins in the deposit. P_r can claim the coins by providing the required witness and signature, while P_s can reclaim the coins by broadcasting the signed refund transaction after the timeframe.

The concept of fair reconstruction proposed by Bentov and Kumaresan, 2014 [4] aims to ensure that an honest party never has to pay a penalty. Also, if the adversary gets to

[9] The session identifier *sid* uniquely identifies this specific execution of the functionality, ensuring isolation from other concurrent executions.

reconstruct the secret, but an honest party does not, then the honest party is compensated. This is made possible by using the $\mathcal{F}_{CR}^{\star}$ functionality.

For this, the authors use a public Non-Malleable Secret Sharing (see Section 3.3.4) scheme. In this scheme, the secret is split into "tag-token" pairs using the *Share* algorithm. The secret is then shared between the parties so that each party P_i has all tags (AllTags) and its own token (Token_i). The secret can only be reconstructed if all tokens are known.

These tags and tokens are used to specify transactions in the context of the $\mathcal{F}_{CR}^{\star}$ functionality. A deposit can only be claimed by a receiving party if it reveals the corresponding tokens associated with the tags used in the transaction.

In the two-party case, let's denote the parties as P_1 and P_2, and their respective tokens as T_1 and T_2. The protocol is executed as follows

1. P_1 creates a $\mathcal{F}_{CR}^{\star}$ deposit which can only be claimed by P_2 if P_2 presents both tokens T_1 and T_2.

2. P_2 makes a similar deposit that can only be claimed by P_1 if P_1 presents the token T_1.

The deposits are then claimed in reverse order. First, P_1 claims the deposit made by P_2 by revealing the token T_1. This revealed token, T_1, can then be used by P_2 (along with its own token, T_2) to claim the deposit made by P_1.

This protocol ensures that if both parties are honest, neither pays a penalty and both receive the tokens. It guarantees fair reconstruction and protects against scenarios where a malicious party might try to abort the process after the other party has made its deposit.

In particular, if P_2 aborts after P_1 has made its deposit, P_2 cannot claim P_1's deposit, since P_1 will not reveal its token T_1 without P_2 making its deposit. Similarly, if P_1 was malicious and aborted without making its deposit, an honest P_2 will not make a deposit, ensuring that neither party learns the secret nor loses its deposit.

Lindell, 2008 [23] In contrast to the Bentov and Kumaresan, 2014 [4] study, which uses a bitcoin-based penalty enforced fairness mechanism, Lindell, 2008 [23] introduces a legal approach to enforcing fairness in two-party computation. Their approach relies on the established legal status of digital signatures to ensure that if fairness is violated, the disadvantaged party holds a digitally signed cheque from the other participant. This allows fairness to be legally enforced: any violation will result in financial loss to the opposing party.

Their protocol works in such a way that either both parties receive output, maintaining fairness, or one party receives output while the other receives a digitally signed cheque that can be presented to a court or a bank. The only way one party can avoid paying the amount on the cheque is to reveal the other party's output, thereby restoring fairness.

4.5.2 Limitations and Impossibility Results

Although the fairness with penalties approach, as proposed by Bentov and Kumaresan, 2014 [4], provides an interesting approach to the fairness problem in secure computations, there are certain limitations and impossibility results to consider.

Number of parties: Both two-party and multi-party settings are considered. In the multi-party case, the protocols are designed to work with a dishonest majority.

Adversary Model: The proposed protocols consider a malicious adversary that behaves strategically to potentially abort the protocol after learning the output, but before delivering it to honest parties.

Dependence on monetary penalties: Fairness in these protocols is enforced by monetary penalties. This is on the assumption that the parties involved are economically rational and that the threat of financial loss will deter them from dishonest behaviour. However, this may not always be the case. In scenarios where the value of the secret information is much higher than the penalty, a malicious party might still choose to act dishonestly.

Dependence on Bitcoin: The protocol, as proposed by Bentov and Kumaresan, 2014 [4], relies on the characteristics of bitcoin (or similar cryptocurrencies) and the underlying blockchain technology. It is therefore susceptible to the limitations and uncertainties of these technologies. A significant drop in the value of bitcoin could have an impact on the effectiveness of this protocol. In addition, transactions on the Bitcoin network take time to validate and confirm, which could slow down the protocols and make them unsuitable for time-sensitive applications.

Further, the protocols require parties to pay deposits. However, this requirement may discourage parties with limited resources from participating, especially if the deposit is substantial.

4.6 Probabilistic Fairness

Probabilistic fairness is a form of fairness in secure computation that aims to ensure that both parties receive their outputs with a high probability.

Probabilistic fairness can be achieved through a sequence of stages, where the likelihood of both parties receiving their output increases with each stage.

4.6.1 Protocols that achieve Probabilistic Fairness

Luby et al., 1983 [26] Luby et al., 1983 [26] introduced a cryptographic protocol that allows two mutually distrustful parties, denoted A and B, to exchange secret bits "simultaneously". In this setting, both parties start with correctly encrypted versions of their secret bits. The fairness and security of the protocol rests on a fundamental cryptographic assumption called the Quadratic Residucity Assumption (see Section 3.3.6).

In the context of this protocol, there are a number of integers denoted by λ_i. These are derived from a security parameter λ that both machines receive in unary. Each λ_i is defined as $T_i(\lambda)$, where $T_i(\lambda)$ is a specific polynomial in λ, depending on the design parameters of the protocol. However, we won't go into the specifics of $T_i(\lambda)$ here.

The protocol aims to achieve the following definition of fairness:

Definition 4.6.1 (Probabilistic fairness)**.** Probabilistic fairness in secure computing is a model of fairness that refers to the probability of both parties receiving their intended outcomes.

For a cryptographic protocol to achieve probabilistic fairness, it should have certain properties:

Output Delivery: If all the parties follow the protocol, they should receive their output with a high probability, close to one, specifically $1 - \frac{1}{2^{\lambda_1}}$, where λ_1 is a security parameter [26].

Progressiveness: The protocol proceeds in rounds, and each round increases the probability that each party will receive its output.

Equal Opportunity: The distribution of outputs to all participating parties should not advantage any particular party. Each party should have an equal chance of receiving its output at the end of the process.

Minimal Advantage upon Abort: If a participating party aborts the computation prematurely, it should gain only a minimal advantage, if any. This is formalized as $P_A \leq P_B + \frac{1}{\lambda_2}$, where P_A and P_B are the probabilities that parties A and B receive their outputs, respectively, and λ_2 is a security parameter.

The protocol introduces a new form of coin toss, which can be better understood by visualizing a biased wheel. This wheel can be imagined as a flat, two-sided disc with an axis through the centre that allows it to rotate. Two points are chosen on the circumference of the wheel which divide it into a large arc and a small arc. The length of the larger arc is a fraction $\frac{1}{2} + \frac{1}{\lambda}$ of the total circumference.

Two colours are chosen with equal probability to be the secret bits of A and B. The parties encode their chosen colour on the wheel by painting the large arc with it. The small arc is painted with the other colour. The wheel is positioned in such a way that each party can see only one side of the wheel.

The wheel is now covered so that neither party can see it, then the wheel is turned so that, if uncovered, each party can see the side that the other party has painted. Then the wheel is spun by both parties having an equal opportunity to spin and stop the wheel randomly once. Finally, both parties (first A, then B) open a small slit at the top and observe the colour. At each turn, the probability that they see the other's chosen colour is $\frac{1}{2} + \frac{1}{\lambda}$.

The protocol is based on the simulated construction of a symmetrically biased wheel. Initially, both parties A and B independently generate large composite numbers n_A and n_B without revealing their factorization, the encryption of their secrets a and b, and the non-residues y_A and y_B mod n_A and n_B, respectively. The secrets a and b, which are assumed to be bits, are encrypted as such. If the secret bit is 1, the encrypted secret is a quadratic residue mod n_A respectively n_B. Otherwise, the encrypted secret is a quadratic non-residue mod n_A respectively n_B.

Next, both parties independently construct lists L_A and L_B of λ_4 elements in Z_{n_A} and, Z_{n_B} respectively. The first $(\frac{1}{2} + \frac{1}{\lambda_4}) \cdot \lambda_4$ elements are randomly chosen quadratic residues, while the other half are quadratic non-residues. These lists are encrypted using a quadratic residue scheme (see [5]).

Then each party randomly chooses a cyclic permutation over λ elements, Π_A and Π_B, and sends the encrypted version to the other party.

Then A applies Π_A to its ordered list L_A and sends $\Pi_A(L_A)$ to B. The same is done by B, which applies Π_B to its ordered list L_B and sends $\Pi_B(L_B)$ to A. This step mimics the spinning of the wheel. To ensure that the total spin on the wheel is the same for both parties, A applies Π_A to $\Pi_B(L_B)$, the list it received from B. Similarly, B applies Π_B to $\Pi_A(L_A)$.

At this point, each party asks the other to reveal the first element, t_B from the list received $\Pi_A(\Pi_B(L_B))$ respectively t_A from the list received $\Pi_B(\Pi_A(L_B))$, without disclosing its quadratic residucity. For this, A randomly chooses $r_B \in Z^\star_{n_B}$ and sends B the message m_B, where mB is selected with probability $\frac{1}{2}$ to be one of the following two numbers: $t_B \cdot b \cdot r_B \cdot r_B \pmod{n_B}$ or $t_B \cdot b \cdot r'_B \cdot r_B \cdot y_B \pmod{n_B}$. B does the same by sending m_A to A.

After receiving this message, the other party determines if it is a quadratic residue or non-residue mod n_B respectively mod n_A, and communicates this to the other party.

With this information, the parties can now compute whether the first element of the list (t_B respectively t_A) have the same residucity as the other parties secret bit b respectively a. As the lists L_A and L_B contain more elements with quadratic residues, the first element being a quadratic residue is more likely, which the probability being $\frac{1}{2} + \frac{1}{\lambda_4}$. With this in mind, a secret and the first element of the list sharing the same residucity increased the probability of the secret also being a quadratic residue. Contrary, opposing residucity would increase the probability of the secret also not being a quadratic residue.

This process is repeated many times. After λ_3 turns, both A and B will know each other's bit with probability greater than $1 - \frac{1}{2}^{\lambda_1}$.

4.6.2 Limitations and Impossibility Results

The probabilistic fairness protocol described above, however, also has its own limitations and problems.

Communication Model: The protocol assumes a synchronous communication model where messages are exchanged in a series of rounds.

Number of parties: The protocol described is designed for two-party computation. Extending it to multi-party computation may require significant changes and introduce new complexities.

The implementation of probabilistic fairness protocols often involves high computational and communication costs. For example, the protocol described by Luby et al., 1983 [26] requires numerous rounds of communication between the parties, with each round involving the encryption and exchange of quadratic residues and non-residues.

The probabilistic nature of fairness in the protocol can also lead to a degree of asymmetry, where one party has a slightly higher chance of learning the other party's bit before its own is revealed. This can potentially allow an aborting party an advantage. It is therefore possible for one party to learn the output and then abort the protocol, denying the other party its output. This possibility, however small, introduces an element of risk into the protocol that may be unacceptable for some applications.

5 Interrelations Between Fairness Notions

The concept of fairness in multi-party computation (MPC) spans a wide range of applications, protocols, and theoretical boundaries. This chapter explores the interrelations between the different notions of fairness explored in this thesis.

5.1 Complete Fairness and Partial Fairness

Complete Fairness and Partial Fairness share the common goal of ensuring that all parties are treated fairly in terms of receiving the output. In both cases, the goal is that either all parties receive the output or no party does.

However, while the principles may appear to be the equivalent, their practical implementations and guarantees differ significantly.

Complete Fairness, as its name suggests, provides an absolute guarantee. When a protocol is said to achieve complete fairness, it means that under no circumstances can an adversary receive the output without the honest parties also receiving the output.

Partial fairness provides a more flexible approach. While still aiming for a fair outcome, it recognizes that in certain scenarios complete fairness may not be achievable as the environment does not provide the necessary preconditions (e.g. an honest majority). Here, if an adversary is able to accurately predict the critical round and abort, he could receive the output while denying it to the honest parties. However, it is important to note that the probability of an adversary accurately predicting the critical round is only $\frac{1}{R}$, where R is the number of rounds.

This inherent vulnerability of partial fairness is a trade-off, allowing for weaker assumptions at the cost of a weakened fairness guarantee. Compared to the protocols described that achieved Complete Fairness, for partial fairness, no assumptions for an honest majority, public bulletin boards, blockchains or a trusted execution environment is required.

5.2 Probabilistic Fairness and Gradual Release

Probabilistic Fairness and Gradual Release are other approaches to ensuring fairness in multiparty computations. Contrary to Complete and Partial Fairness, these approaches do not try to stop an adversary from gaining advantage. Rather, their common basis is that, while an adversary may gain some advantage by aborting, the amount of this advantage is bounded by the security parameter.

The Gradual Release method, discussed in section 4.4, works by breaking secret information into bits and sharing them one at a time. This way, if one party decides to abort, they

only gain an advantage of one bit of the secret. Essentially, it's an incremental approach, minimizing risk at each step.

On the other hand, probabilistic fairness, described in section 4.6, takes a different approach. It uses a series of stages, and with each stage the chances of both parties getting their outputs increase. So if an adversary tries to gain an unfair advantage by aborting, their probability of computing the current output is only negligible (in the security parameter) higher than the other parties probability of computing their output.

5.3 Δ-Fairness and Fairness with Penalties

Both Δ-Fairness and Fairness with Penalties do not attempt to prevent a malicious party from gaining an advantage by aborting the protocol after receiving its output. However, they differ in the solutions they provide for dealing with the disadvantaged party.

Δ-fairness ensures that the honest party eventually receives the output, even if they have to wait. It provides honest parties with a guarantee that, when a malicious party aborts in a round r, the honest party will learn their output in round $\Delta(r)$.

In contrast, Fairness with Penalties introduces a monetary penalty as a means of discouragement. If an adversary aborts after receiving his outcome, he is penalized a predetermined monetary penalty. This approach doesn't try to ensure that the honest party gets the result, but rather tries to discourage malicious actions.

5.4 Complete Fairness and Δ-Fairness

Complete Fairness and Δ-Fairness both guarantee that either all participants receive the outcome or none do.

The difference is in the timing of when the parties receive their output. While Complete Fairness ensures that all parties receive their output at the same time, Δ-Fairness allows that honest parties may receive their output after a delay.

5.5 Overview

Table 5.1: Protocols that achieve complete, partial, Δ-fairness, gradual release, fairness with penalties, and probabilistic fairness

Name/Citation	Fairness Notion	Requirements	Efficiency
Choudhuri et al., 2017 [10]	Complete Fairness	Public bulletin board	runtime $O(1)$, but size of release token grows with number of parties
Gaddam et al., 2023 [12]	Complete Fairness	Blockchains, TEE, at most $t < n$ parties corrupt	runtime $O(1)$
Gordon et al., 2008 [17]	Complete Fairness	Two-party setting, Hybrid model	Boolean AND in $\Theta(m)$ rounds, where $m = \text{poly}(n)$
Gordon and Katz, 2008 [15]	Partial Fairness	Two-party setting	runtime $O(r)$ rounds, $r = p \cdot \lvert Y \rvert$, where p is a polynomial and $\lvert Y \rvert$ is the size of the domain Y
Bailey et al., 2022 [3]	Partial Fairness	Verifiable Delay Functions (VDFs)	Honest work to recover output is $O(R)$
Pass et al., 2016 [27]	Δ-Fairness	Two-party setting, Secure processors with tamper-resistant clocks	runtime bounded by $\Delta(g(\lambda))$, where $g(\lambda))$ is a fixed polynomial
Blum, 1983 [5]	Gradual Release	Two-party setting	runtime $O(1)$ for a single bit
Pinkas, 2003 [28]	Gradual Release	Two-party setting, Timed commitments	runtime $O(\lambda)$, with security parameter λ
Bentov and Kumaresan, 2014 [4]	Fairness with Penalties	Reliance on the Bitcoin network, Monetary penalty	runtime $O(1)$, however bitcoin transaction delays occur
Lindell, 2008 [23]	Fairness with Penalties	Two-party setting, external authority	runtime $O(1)$
Luby et al., 1983 [26]	Probabilistic Fairness	Quadratic Residucity Assumption, Synchronous communication, Two-party setting	runtime $O(\lambda)$

6 Practical Implications of Fairness Notions

6.1 Complete Fairness

The general functions to achieve Complete Fairness can only be used in scenarios where the majority of participants behave honestly. This means it is ideal for environments where there's an inherent trust among the majority. However, even without an honest majority, Complete Fairness is still achievable for specific functions.

Application Complete Fairness is ideal for situations where guaranteed fairness, in the sense that no adversary can gain any advantage by aborting, is critical, and where an honest majority can be guaranteed. A possible scenario could be a democratic election in a small, closed organization or community where all participants know each other and can vouch for the honesty of the majority.

Implications The main practical challenge with Complete Fairness is its limitations in general settings: it can't be achieved for general functions if there is no honest majority. This limits its applicability to scenarios where trust can be largely assured, or only specific functions (see [17]) need to be evaluated.

6.2 Partial Fairness

Partial fairness is a compromise approach that seeks to achieve the highest level of fairness when the ideal conditions for full fairness, namely an honest majority, aren't present.

Application Partial Fairness is particularly useful in settings where complete fairness is desirable but not guaranteed due to the lack of an honest majority. If the probability of an adversary gaining an advantage (e.g. 1/p, where p is a polynomial) is low, partial fairness becomes an attractive approach for many practical applications.

Implications This flexibility is both a strength and a weakness. Although it enables the function of systems with no honest majority, there is the risk of potential exploitation, however limited. Therefore, when using partial fairness, the risks must be evaluated.

6.3 Δ-Fairness

Δ-Fairness introduces a concept where an acceptable time difference is allowed between the adversary and the honest party in obtaining the outcome of the computation. During this period, the honest party is assured of learning their output, even if the malicious party decides to abort the procedure.

Application Δ-Fairness is very efficient, since it requires only $O(1)$ cryptographic computations. This level of efficiency, similar to many Trusted Execution Environment (TEE)-based schemes, makes Δ-Fairness ideal for scenarios where fast computation is a priority. In particular, in non-time-sensitive use cases where the value of the computation result does not degrade rapidly over time, Δ-Fairness can be almost as powerful as Complete Fairness. It can be used in various domains, such as privacy-preserving data mining, where computation speed is critical, or in the structured exchange of statistical information between entities such as companies, or in key exchange. Another potential application is elections, where the process is generally not time-sensitive, and the focus is on the integrity and fairness of the result rather than the speed of computation.

Implications Δ-Fairness might not be ideal for time-sensitive applications where the value output decreases over time. If a malicious party delays the delivery of the honest parties outputs, in time-sensitive applications, the malicious party would be advantaged, even though the protocol limits the delay.

For example, consider a secure multi-party computation where several companies compute joint statistics on business data for which it is advantageous to know the result on the stock market. Corrupt participants could gain an unfair advantage if they can abort the computation and thus gain access to the output, while the honest participants receive their output after a possibly significant delay. This illustrates a scenario where Δ fairness may not be useful because the information is time-sensitive.

6.4 Gradual Release

Gradual release builds on the idea of sharing information in bits, releasing them gradually to minimize the advantage any party could gain by aborting early. By releasing secrets bit by bit, this method ensures that each party only risks a small portion of the secret at any given time.

Application Key exchange is an example of a gradual release application, especially since keys are often distributed randomly. In a key exchange protocol, parties attempt to share a secret key. The gradual release method ensures that the key is exchanged bit by bit, and because the bits of the key are randomly distributed, there's no specific number of bits after which an adversary can gain a significant advantage by aborting.

Implications However, there are inherent limitations to this concept. Firstly, for small secrets, especially those of a few bits, gradual release isn't as effective. This is because an adversary's advantage in obtaining even one bit may be too great. The approach is therefore better suited to applications involving larger secrets.

In addition, the effectiveness of the gradual release method depends on how the information is distributed within the data chunks. If there are clusters of critical data, an adversary aware of this distribution could strategically abort the process after obtaining these bits. It's therefore important that the important data is randomly distributed throughout the secret to ensure the effectiveness of the approach. Randomly generated keys are a good example.

6.4.1 Fairness with Penalties

Fairness with Penalties is based on the idea of discouraging malicious behaviour by introducing monetary penalties. Unlike traditional cryptographic approaches that focus on the computational aspect of security, this notion combines the computational world with economic principles.

Application An interesting application of fairness with penalties can be seen in secure lotteries. In a multi-party setting, a protocol for a secure lottery with penalties ensures that if an adversary aborts after learning the outcome of the lottery but before revealing the outcome to honest parties, then each honest party is compensated with a pre-specified amount equal to the lottery prize. This provides an economic disincentive for malicious actors to act dishonestly and is consistent with the concept of introducing penalties to enforce fairness. Bentov and Kumaresan, 2014 [4] provides a concrete example by introducing a protocol for a "secure lottery with penalties".

Implications Using the bitcoin network to achieve fairness presents several challenges. First, the time it takes to confirm bitcoin transactions introduces delays into the protocol. For applications where immediacy is critical, such delays can be problematic. Second, there's the issue of transaction fees. Although these fees may be negligible when weighed against significant penalties, in scenarios where a protocol is executed many times on a large scale, these accumulated fees could become significant.

When, instead of using bitcoin, a legal approach is used to enforcing fairness with penalties, a protocols efficiency depends heavily on the legal infrastructure and the willingness and ability of the parties to enforce these digital agreements in court. On a large scale with many malicious actors, the process could be cumbersome and potentially expensive, depending on the jurisdiction.

6.4.2 Probabilistic Fairness

Probabilistic Fairness tries to maintain fairness in secure computations by ensuring that the probability that each participant receives its output increases uniformly with successive stages, thus limiting the gain that an adversary can achieve by aborting before the end of the protocol.

Application A simple example of where probabilistic fairness could be applied is in "coin flipping by phone". In situations where two parties need a fair way to generate a random outcome but cannot meet physically, this protocol could serve as a digital alternative to coin flipping.

Implications The main implication of probabilistic fairness, especially when considered looking at Luby et al., 1983 [26], is its inherent limitation to dealing with single-bit secrets. This can be inefficient for multi-bit or longer secrets, as the protocol would have to be run multiple times for each bit, increasing the runtime.

7 Conclusion

This thesis provides an overview of fairness notions in the area of secure multi-party protocols. In exploring the different notions of fairness in MPC protocols, various concepts such as complete fairness, partial fairness, Δ-fairness, gradual release, fairness with penalties and probabilistic fairness have been studied. These notions provide a range of options, each with its own advantages and limitations depending on the context.

The different notions of fairness are interrelated, with trade-offs in terms of certainty, efficiency, and applicability. Complete fairness, often considered the ideal, ensures that either all parties receive an output or none do. However, this ideal is not universally achievable due to theoretical limitations, such as the absence of an honest majority. As a result, alternative notions such as Partial Fairness, which provides fairness within certain probability bounds, and Δ-Fairness, which is efficient but may not be suitable for time-sensitive applications, have been developed. Probabilistic Fairness introduces fairness by targeting an equal probability of output delivery among the participants. This often comes at the cost of higher computational and communication overheads. Gradual Release allows incremental output delivery. Here, the output is delivered on a bit-by-bit basis, limiting the advantage an adversary can achieve. Fairness with penalties introduces monetary penalties for malicious behaviour, thus discouraging early aborts.

There are numerous similarities but also differences in these fairness notions. For example, complete fairness and partial fairness share the common goal of ensuring that all parties are receiving their output or none. However, they differ in the level of guarantee they offer. While complete fairness provides a guarantee that either all parties receive the output or none do, partial fairness allows for a small probability that an adversary could receive the output while the honest parties do not. However, their practical implementation and guarantees differ significantly. Both Probabilistic Fairness and Gradual Release limit the advantage an adversary can gain. Gradual Release limits the adversary's advantage to learning only one bit of information at a time. Probabilistic Fairness takes this a step further by also reducing the likelihood that an adversary can gain even this minimal advantage by exchanging a single bit over multiple rounds. Both Δ-Fairness and Fairness with Penalties do not prevent a malicious party from gaining an advantage, but differ in their solutions for dealing with the disadvantaged party. Δ-Fairness ensures that the honest party eventually receives the output, even if he has to wait, while Fairness with Penalties introduces a monetary penalty as a means of discouraging malicious actions.

Achieving fairness in secure multi-party computation (MPC) protocols depends on a number of assumptions, which often act as constraints. For example, to achieve complete fairness, the protocol of [10] requires access to a public ledger or bulletin board. The protocol by [17] is limited to a two-party setting and is only applicable to certain functions, not general ones. Another protocol by [12] assumes the use of Trusted Execution Environments (TEEs) and blockchains.

In the area of partial fairness, the protocol by [15] is also limited to two-party settings. Further, it has limitations when the input and output domains are super-polynomial in size. In particular, for domains of such size, the protocol can no longer achieve $1/p$ fairness for $p > 2$. The [3] protocol, however, overcomes these limitations and is applicable to multi-party settings and general functions.

For Δ-Fairness, the protocols require the use of secure processors (see Section 3.3.5) equipped with tamper-resistant clocks, as highlighted in the work of [27].

In the case of Gradual Release, both the [5] and [28] protocols are designed for two-party settings and can be computationally intensive and time-consuming.

Fairness with penalties introduces its own set of assumptions. The protocol by [4] assumes access to the Bitcoin network, along with the delays and fees that come with it. In contrast, the protocol of [23] requires an external authority such as a bank or a court to act as an intermediary. While [4] proposes an n-party protocol, [23] is designed for two-party settings.

Finally, probabilistic fairness, as described in the [26] protocol, assumes a two-party scenario where only a single bit is exchanged.

Several impossibility results have been identified that highlight the inherent challenges and limitations of fairness in MPC. For complete fairness, one of the earliest impossibility results was shown by [11]. They showed that in the absence of an honest majority, complete fairness is generally impossible. This is because there is always some communication round in which one party gets knowledge of its output for the first time. If an adversary knows which round this is, he can stop communicating in that round, thereby obtaining his output, but without the honest party having learned his output. However, there are exceptions to this result. For example, complete fairness can be achieved if the parties have access to a public ledger or bulletin board [10]. Further, a 2-round fair MPC for general functions is impossible, even with an honest majority. [16]. A more recent impossibility result in the context of Δ fairness states that if at least one party is not equipped with an attested execution processor, it is impossible to implement UC-secure multiparty computation without additional setup assumptions, even if all other parties are equipped with an attested execution processor (see Section 3.3.5) [27].

The practical implications of the notions of fairness vary and depend on the context. Complete fairness is best suited to environments where an honest majority can be assured, such as in small, closed organizations. Its applicability is limited to certain functions in the absence of an honest majority. Partial fairness offers a compromise that is useful in situations where complete fairness is desirable, but an honest majority is not guaranteed. Δ-Fairness is efficient and ideal for non-time-sensitive applications such as elections. However, it is not suitable for time-sensitive scenarios, such as stock market calculations, where a malicious party could gain an unfair advantage by delaying the output of honest parties. Gradual release is effective for larger secrets and is particularly useful in key exchange protocols. However, its effectiveness decreases for small secrets or when critical data is locally concentrated in the secret. Fairness with penalties can be used in secure lotteries to introduce a monetary penalty against malicious behaviour. However, its efficiency may be compromised by transaction delays or, in the case of legally enforceable fairness, by the complexity of legal infrastructures. Probabilistic fairness is limited to single-bit secrets, making it inefficient for multi-bit or longer secrets.

8 Appendix

8.1 Protocols

ShareGen as proposed by [17] used in Figure 8.2

Inputs: Let the inputs to ShareGen be x_i and y_j with $1 \leq i, j \leq m$. (If one of the received inputs is not in the correct domain, then both parties are given output $\perp$.) The security parameter is n.

Computation:

1. Define values $a_1, ..., a_m$ and $b_1, ..., b_m$ in the following way:
   - Set $a_i = b_j = f(x_i, y_j)$.
   - For $\ell \in \{1, ..., m\}$, $\ell \neq i$, set $a_\ell = \text{NULL}$.
   - For $\ell \in \{1, ..., m\}$, $\ell \neq j$, set $b_\ell = \text{NULL}$.
2. For $1 \leq \ell \leq m$, choose $(a_\ell^{(1)}, a_\ell^{(2)})$ and $(b_\ell^{(1)}, b_\ell^{(2)})$ as random secret sharings of a_ℓ and b_ℓ, respectively. (I.e., $a_\ell^{(1)}$ is random and $a_\ell^{(1)} \oplus a_\ell^{(2)} = a_\ell$.)
3. Compute $k_a, k_b \leftarrow \text{Gen}(1^n)$. For $1 \leq \ell \leq m$, let $t_\ell^a = \text{Mac}_{k_a}(\ell \| a_\ell^{(2)})$ and $t_\ell^b = \text{Mac}_{k_b}(\ell \| b_\ell^{(1)})$.

Output:

1. P_1 receives the values $a_1^{(1)}, ..., a_m^{(1)}$ and $(b_1^{(1)}, t_1^b), ..., (b_m^{(1)}, t_m^b)$, and the MAC-key k_a.
2. P_2 receives the values $(a_1^{(2)}, t_1^a), ..., (a_m^{(2)}, t_m^a)$ and $b_1^{(2)}, ..., b_m^{(2)}$, and the MAC-key k_b.

Figure 8.1: Functionality ShareGen [17]

Protocol 1 as proposed by [17]

Inputs: Party P_1 has input x and party P_2 has input y. The security parameter is n.
Protocol:

1. **Preliminary phase:**
   a) Parties P_1 and P_2 run protocol π for computing ShareGen, using their respective inputs x, y, and security parameter n.
   b) If P_1 receives $\perp$ from the above computation (because P_2 aborts the computation or uses an invalid input in π) it outputs $f(x, y_1)$ and halts. Likewise, if P_2 receives $\perp$, it outputs $f(x_1, y)$ and halts. Otherwise, the parties proceed.
   c) Denote the output of P_1 from π by $a_1^{(1)}, ..., a_m^{(1)}, (b_1^{(1)}, t_1^b), ..., (b_m^{(1)}, t_m^b)$, and k_a.
   d) Denote the output of P2 from π by $(a_1^{(2)}, t_1^a), ..., (a_m^{(2)}, t_m^a), b_1^{(2)}, ..., b_m^{(2)}$, and k_b.
2. For $\ell = 1, ..., m$ do:
   a) P_2 sends the next share to P_1:
      i. P2 sends $(a_\ell^{(2)}, t_\ell^a)$ to P_1.
      ii. P1 receives $(a_\ell^{(2)}, t_\ell^a)$ from P2. If $\text{Vrfy}_{k_a}(\ell \| a_\ell^{(2)}, t_\ell^a) = 0$ (or if P_1 received an invalid message, or no message), then P_1 halts. If P_1 has already determined its output in some earlier iteration, then it outputs that value. Otherwise, it outputs $f(x, y_{\ell-1})$ (if $\ell = 1$, then P_1 outputs $f(x, y_1)$).
      iii. If $\text{Vrfy}_{k_a}(\ell \| a_\ell^{(2)}, t_\ell^a) = 1$ and $a_\ell^{(1)} \oplus a_\ell^{(2)} \neq \text{NULL}$ (i.e., $x = x_\ell$), then P_1 sets its output to be $a_\ell^{(1)} \oplus a_\ell^{(2)}$ (and continues running the protocol).
   b) P_1 sends the next share to P_2:
      i. P_1 sends $(b_\ell^{(1)}, t_\ell^b)$ to P_2.
      ii. P_2 receives $(b_\ell^{(1)}, t_\ell^b)$ from P1. If $\text{Vrfy}_{k_b}(\ell \| b_\ell^{(1)}, t_\ell^b) = 0$ (or if P_2 received an invalid message, or no message), then P_2 halts. If P_2 has already determined its output in some earlier iteration, then it outputs that value. Otherwise, it outputs $f(x_\ell, y)$.
      iii. If $\text{Vrfy}_{k_b}(\ell \| b_\ell^{(1)}, t_\ell^b) = 1$ and $b_\ell^{(1)} \oplus b_\ell^{(2)} \neq \text{NULL}$ (i.e., $y = y_\ell$), then P_2 sets its output to be $b_\ell^{(1)} \oplus b_\ell^{(2)}$ (and continues running the protocol).

Figure 8.2: Protocol 1, to compute the Millionaire's Problem [17]

ShareGen$_r$ as proposed by [15] used in Figure 8.4

Inputs: The security parameter is n. Let the inputs to ShareGen$_r$ be $x \in X^n$ and $y \in Y^n$. (If one of the received inputs is not in the correct domain, a default input is substituted.)
Computation:

1. Define values $a_1, ..., a_r$ and $b_1, ..., b_r$ in the following way:
   - Choose i^* uniformly at random from $\{1, ..., r\}$.
   - For $i = 1$ to $i^* - 1$ do:
     - Choose $\hat{y} \leftarrow Y^n$ and set $a_i = f_n(x, \hat{y})$.
     - Choose $\hat{x} \leftarrow X^n$ and set $b_i = f_n(\hat{x}, y)$.
   - Compute $z = f_n(x, y)$. For $i = i^*$ to r, set $a_i = b_i = z$.
2. For $1 \leq i \leq r$, choose $(a_i^{(1)}, a_i^{(2)})$ and $(b_i^{(1)}, b_i^{(2)})$ as random secret sharings of a_i and b_i, respectively. (I.e., $a_i^{(1)}$ is random and $a_i^{(1)} \oplus a_i^{(2)} = a_i$.)
3. Compute $k_a, k_b \leftarrow \text{Gen}(1^n)$. For $1 \leq i \leq r$, let $t_i^a = \text{Mac}_{k_a}(i \| a_i^{(2)})$ and $t_i^b = \text{Mac}_{k_b}(i \| b_i^{(1)})$.

Output:

1. Send to P1 the values $a_1^{(1)}, ..., a_r^{(1)}$ and $(b_1^{(1)}, t_1^b), ..., (b_r^{(1)}, t_r^b)$, and the MAC-key k_a.
2. Send to P2 the values $(a_1^{(2)}, t_1^a), ..., (a_r^{(2)}, t_r^a)$ and $b_1^{(2)}, ..., b_r^{(2)}$, and the MAC-key k_b.

Figure 8.3: Functionality ShareGen$_r$ [15]

Protocol 1 by [15]

Inputs: Party P_1 has input x and party P_2 has input y. The security parameter is n. Let $r = p \cdot |Y^n|$.
The protocol:

1. Preliminary phase:
   (a) P_1 chooses $\hat{y} \in Y_n$ uniformly at random, and sets $a_0 = f_n(x, \hat{y})$. Similarly, P_2 chooses $\hat{x} \in X_n$ uniformly at random, and sets $b_0 = f_n(\hat{x}, y)$.
   (b) Parties P_1 and P_2 run a protocol π to compute ShareGen_r, using their inputs x and y.
   (c) If P_2 receives $\perp$ from the above computation, it outputs b_0 and halts. Otherwise, the parties proceed to the next step.
   (d) Denote the output of P_1 from π by $a_1^{(1)}, ..., a_r^{(1)}$, $(b_1^{(1)}, t_1^b), ..., (b_r^{(1)}, t_r^b)$, and k_a.
   (e) Denote the output of P_2 from π by $(a_1^{(2)}, t_1^a), ..., (a_r^{(2)}, t_r^a)$, $b_1^{(2)}, ..., b_r^{(2)}$, and k_b.
2. For $i = 1, ..., r$ do:
   - P_2 sends the next share to P_1:
     (a) P_2 sends $(a_i^{(2)}, t_{ia})$ to P_1.
     (b) P_1 receives $(a_i^{(2)}, t_{ia})$ from P_2. If $\text{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 0$ (or if P_1 received an invalid message, or no message), then P_1 outputs a_{i-1} and halts.
     (c) If $\text{Vrfy}_{k_a}(i \| a_i^{(2)}, t_i^a) = 1$, then P_1 sets $a_i = a_i^{(1)} \oplus a_i^{(2)}$ (and continues running the protocol).
   - P_1 sends the next share to P_2:
     (a) P_1 sends $(b_i^{(1)}, t_i^b)$ to P_2.
     (b) P_2 receives $(b_i^{(1)}, t_i^b)$ from P_1. If $\text{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 0$ (or if P_2 received an invalid message, or no message), then P_2 outputs b_{i-1} and halts.
     (c) If $\text{Vrfy}_{k_b}(i \| b_i^{(1)}, t_i^b) = 1$, then P_2 sets $b_i = b_i^{(1)} \oplus b_i^{(2)}$ (and continues running the protocol).
3. If all r iterations have been run, party P_1 outputs a_r and party P_2 outputs b_r.

Figure 8.4: Generic protocol for computing a functionality f_n [15]

Protocol for computing functionality with VDF parameters as proposed by [3]

1. Each party sets up VDF parameters and gets the keys $pk_i, sk_i \leftarrow$ VDF.KeyGen() and broadcasts the public key to all other parties. The parties individually choose random values v_i to input to their VDF. They compute the VDF on these values with $\Delta = \delta R$, where δ is the period of time taken up by one round of communication. They get an output in the form of a uniformly random pair of a share randomness and a round randomness $s_i, r_i \in \{0,1\}^n \times [R]$, as well as a VDF proof π_i output $((s_i, r_i), \pi_i) :=$ VDF.Trapdoor(sk, v_i, Δ). They broadcast commitments to the v_i values to the other parties.
2. The parties supply their inputs x_i, along with s_i, r_i, to a secure-with-abort MPC protocol which computes the following:

(a) The MPC computes the critical round as a mixture of the individual round randomnesses $r^* := (\sum_{i \in N} r_i) \mod R$.

(b) The MPC computes an encryption pad s as a mixture of the individual share randomnesses $s := \bigoplus_{i \in N} s_i$.

(c) For each pair of parties i, j, and each round r, the MPC sends a message $m_{i,j,r}$ to party i. This message is intended to be sent from party i to party j in the rth round of the reveal phase below:

- For $r = r^*$, the messages $(m_{i,j,r^*})_{j \in [N]}$ form a random secret share of party j's output OTP encrypted with the VDF pad s with the $\bigoplus_{j \in [N]} m_{i,j,r^*} = s \oplus f_j(\bar{x})$.
- If $r \neq r^*$, the value is chosen uniformly at random $m_{i,j,r} \leftarrow \{0,1\}^n$.

(d) Each party j additionally receives from the functionality a vector of commitments to all the entries of all messages $m_{i,j,r}$ intended for them, and party i receives an opening for that commitment. Parties additionally receive commitments to s_i, r_i that party i can open.

3. The parties, using the v_i, s_i, r_i, π_i values they hold, complete zero-knowledge proofs for each other that VDF.Verify yields a pass on these values and that the commitments to these values match the commitments issued by the MPC.
4. They then start the VDF timer and open their commitments to v_i to each other, and the reveal phase begins.

(a) In round r of the reveal phase, if all previous rounds have been successful, and the VDF timer has not elapsed, each party i opens the commitment to each $m_{i,j,r}$ they hold to the corresponding j. Party j verifies the opening, and halts if the verification fails.

5. After the reveal phase ends, all parties compute VDF.Eval on the v_i values to obtain s_i, r_i, and from this, s, and r^*. (Optionally, instead, parties can open their commitments to the s_i, r_i to save each other the trouble of recomputing the VDF).
6. If a party j received all messages m_{i,j,r^*} in the r^*th round of the reveal phase, then that party outputs $f_j(\bar{x}) = s \oplus \bigoplus_{j \in [N]} m_{i,j,r^*}$.

Figure 8.5: Generic protocol for computing a functionality with VDF parameters [3]

Bibliography

[1] Gilad Asharov, Ran Cohen, and Oren Shochat. *Static vs. Adaptive Security in Perfect MPC: A Separation and the Adaptive Security of BGW*. Report Number: 758. 2022. URL: https://eprint.iacr.org/2022/758 (visited on 07/25/2023).

[2] Gilad Asharov, Abhishek Jain, and Daniel Wichs. *Multiparty Computation with Low Communication, Computation and Interaction via Threshold FHE*. Report Number: 613. 2011. URL: https://eprint.iacr.org/2011/613 (visited on 07/05/2023).

[3] Bolton Bailey, Andrew Miller, and Or Sattath. *General Partially Fair Multi-Party Computation with VDFs*. 2022. URL: https://eprint.iacr.org/2022/1318.

[4] Iddo Bentov and Ranjit Kumaresan. *How to Use Bitcoin to Design Fair Protocols*. Report Number: 129. 2014. URL: https://eprint.iacr.org/2014/129.

[5] Manuel Blum. "How to exchange (secret) keys". In: *ACM Transactions on Computer Systems* 1.2 (May 1983), pp. 175–193. ISSN: 0734-2071. DOI: 10.1145/357360.357368. URL: https://dl.acm.org/doi/10.1145/357360.357368.

[6] Dan Boneh and Moni Naor. "Timed Commitments". en. In: *Advances in Cryptology — CRYPTO 2000*. Ed. by Mihir Bellare. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 236–254. ISBN: 978-3-540-44598-2. DOI: 10.1007/3-540-44598-6_15.

[7] Dan Boneh et al. *Verifiable Delay Functions*. Report Number: 601. 2018. URL: https://eprint.iacr.org/2018/601.

[8] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. en. Berlin, Heidelberg: Springer, 2011. ISBN: 978-3-642-15259-7 978-3-642-15260-3. DOI: 10.1007/978-3-642-15260-3. URL: http://link.springer.com/10.1007/978-3-642-15260-3 (visited on 07/28/2023).

[9] Ran Canetti. *Universally Composable Security: A New Paradigm for Cryptographic Protocols*. Report Number: 067. 2000. URL: https://eprint.iacr.org/2000/067 (visited on 06/22/2023).

[10] Arka Rai Choudhuri et al. *Fairness in an Unfair World: Fair Multiparty Computation from public Bulletin Boards*. Report Number: 1091. 2017. URL: https://eprint.iacr.org/2017/1091.

[11] R Cleve. "Limits on the security of coin flips when half the processors are faulty". In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. STOC '86. New York, NY, USA: Association for Computing Machinery, Nov. 1986, pp. 364–369. ISBN: 978-0-89791-193-1. DOI: 10.1145/12130.12168. URL: https://dl.acm.org/doi/10.1145/12130.12168.

[12] Sivanarayana Gaddam et al. *How to Design Fair Protocols in the Multi-Blockchain Setting*. Report Number: 762. 2023. URL: https://eprint.iacr.org/2023/762.

[13] O. Goldreich, S. Micali, and A. Wigderson. "How to play ANY mental game". In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. STOC '87. New York, NY, USA: Association for Computing Machinery, Jan. 1987, pp. 218–229. ISBN: 978-0-89791-221-1. DOI: 10.1145/28395.28420. URL: https://dl.acm.org/doi/10.1145/28395.28420.

[14] Oded Goldreich. *Foundations of cryptography*. en. Cambridge, UK ; New York: Cambridge University Press, 2003. ISBN: 978-0-521-79172-4 978-0-521-83084-3.

[15] Dov Gordon and Jonathan Katz. *Partial Fairness in Secure Two-Party Computation*. Report Number: 206. 2008. URL: https://eprint.iacr.org/2008/206.

[16] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. *Constant-Round MPC with Fairness and Guarantee of Output Delivery*. 2015. URL: https://eprint.iacr.org/2015/371.

[17] S. Dov Gordon et al. *Complete Fairness in Secure Two-Party Computation*. Report Number: 303. 2008. URL: https://eprint.iacr.org/2008/303.

[18] Jeremias Mechler and Timo Kuch. *Universelle Komponierbarkeit in der Kryptographie*. de. KASTEL – Institut für Informationssicherheit und Verlässlichkeit, KIT-Fakultät für Informatik. 2023.

[19] Burt Kaliski. "Quadratic Residuosity Problem". en. In: *Encyclopedia of Cryptography and Security*. Ed. by Henk C. A. van Tilborg and Sushil Jajodia. Boston, MA: Springer US, 2011, pp. 1003–1003. ISBN: 978-1-4419-5906-5. DOI: 10.1007/978-1-4419-5906-5_429. URL: https://doi.org/10.1007/978-1-4419-5906-5_429 (visited on 06/27/2023).

[20] Alexander Koch, Sebastian Faller, and Robin Berger. *Skript zur Vorlesung Kryptographische Protokolle*. KASTEL – Institut für Informationssicherheit und Verlässlichkeit, KIT-Fakultät für Informatik. Jan. 2023.

[21] William Judson LeVeque. *Fundamentals of Number Theory*. en. Google-Books-ID: sWWKAAAAQBAJ. Courier Corporation, Jan. 1996. ISBN: 978-0-486-68906-7.

[22] Xiaoguo Li et al. *A Survey of Secure Computation Using Trusted Execution Environments*. arXiv:2302.12150 [cs]. Feb. 2023. URL: http://arxiv.org/abs/2302.12150.

[23] Andrew Y. Lindell. "Legally-Enforceable Fairness in Secure Two-Party Computation". en. In: *Topics in Cryptology – CT-RSA 2008*. Ed. by Tal Malkin. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2008, pp. 121–137. ISBN: 978-3-540-79263-5. DOI: 10.1007/978-3-540-79263-5_8.

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

[25] Yehuda Lindell. *Secure Multiparty Computation (MPC)*. Report Number: 300. 2020. URL: https://eprint.iacr.org/2020/300.

[26] Michael Luby, Silvio Micali, and Charles Rackoff. “How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin”. In: *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. ISSN: 0272-5428. Nov. 1983, pp. 11–22. DOI: 10.1109/SFCS.1983.25.

[27] Rafael Pass, Elaine Shi, and Florian Tramer. *Formal Abstractions for Attested Execution Secure Processors*. Report Number: 1027. 2016. URL: https://eprint.iacr.org/2016/1027.

[28] Benny Pinkas. “Fair Secure Two-Party Computation”. en. In: *Advances in Cryptology — EUROCRYPT 2003*. Ed. by Eli Biham. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2003, pp. 87–105. ISBN: 978-3-540-39200-2. DOI: 10.1007/3-540-39200-9_6.

[29] Douglas R. Stinson and Maura B. Paterson. *Cryptography: theory and practice*. en. Fourth edition. Boca Raton: CRC Press, Taylor & Francis Group, 2019. ISBN: 978-1-138-19701-5.

[30] Andrew C. Yao. “Protocols for secure computations”. In: *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*. ISSN: 0272-5428. Nov. 1982, pp. 160–164. DOI: 10.1109/SFCS.1982.38.

[31] Xuefei Yin, Yanming Zhu, and Jiankun Hu. “A Comprehensive Survey of Privacy-preserving Federated Learning: A Taxonomy, Review, and Future Directions”. In: *ACM Computing Surveys* 54.6 (July 2021), 131:1–131:36. ISSN: 0360-0300. DOI: 10.1145/3460427. URL: https://dl.acm.org/doi/10.1145/3460427.