Diss. ETH No. 27673

Multi-Party Computation: Definitions, Enhanced Security Guarantees and Efficiency

A thesis submitted to attain the degree of

Doctor of Sciences of ETH Zurich
(Dr. sc. ETH Zurich)

presented by

Chen-Da Liu Zhang
MSc ETH in Computer Science, ETH Zurich

born on December 17, 1991
citizen of Spain

accepted on the recommendation of

Prof. Dr. Ueli Maurer, examiner
Prof. Dr. Jesper Buus Nielsen, co-examiner
Dr. Martin Hirt, co-examiner

2021

Acknowledgements

First of all, I would like to thank my advisor Ueli Maurer for his invaluable encouragement and support. His door was always open to discuss any topic, scientific or not. His unique way of thinking and his ability to ask the right questions deeply shaped the way I think about research, and life in general. Even after all these years, I am still amazed by his simple and natural approach to research.

Sincere thanks go to Jesper Buus Nielsen and Martin Hirt for co-refereeing this thesis.

During my Ph.D. studies I had the chance to interact with many brilliant minds. For the exciting former, on-going and hopefully future collaborations, and for the many great research discussions, I would like to thank Christian Badertscher, Fabio Banfi, Amey Bhangale, Erica Blum, Annick Chopard, Ran Cohen, Sandro Coretti, Giovanni Deligios, Gioia Ehrensperger, Matthias Fitzi, Diana Ghinea, Martin Hirt, Daniel Jost, Ard Kastrati, Jonathan Katz, David Lanzenberger, Rio LaVigne, Siqi Liu, Julian Loss, Christian Matt, Varun Maram, Ueli Maurer, Tal Moran, Marta Mularczyk, Kartik Nayak, Christopher Portmann, Martin Raszyk, Guilherme Rito, Daniel Tschudi, and Jiamin Zhu.

Special thanks go to Martin Hirt for the many research and non-research discussions, being an amazing lecturer and all the students we supervised together. I not only learned a lot on MPC from him, but also numerous tips on didactics and supervision of students.

Further thanks to all former and current members from the Cryptography and the Foundations of Cryptography groups, with whom I could enjoy all these passionate game nights, hikes, tennis, table-tennis and table-soccer games, and to Claudia Günthart and Denise Spicher for their administrative support.

Last but not least, this thesis would not have been possible without the invaluable encouragement of my parents Haoqing and Xina, my brothers Ming-Da and Xiao-Da, my partner Anna, and the support of my dear friends. Thanks for all these amazing years!

Abstract

Secure multi-party computation (MPC) allows a set of parties to jointly perform a distributed computation task in a secure manner. This very general problem is fundamental in cryptography and distributed protocols, and can be studied according to countless parameters: the underlying communication network, the task to compute, and so on.

Over the past decades, the problem of multi-party computation has been the subject of an enormous body of research. The number of applications for MPC is potentially unlimited, and protocols for MPC transitioned from being purely theoretical, to actually being deployed in practice. Continuing deep theoretical work to ensure that MPC stands on strong scientific foundations is crucial to achieve further progress.

This thesis makes contributions in various topics of multi-party computation. One can roughly differentiate three parts.

Multi-Party Constructive Cryptography. In the first part, we give contributions to the definitional part of MPC. We model the setting of MPC in the constructive cryptography framework (CC), by providing two instantiations of its abstract layer. In the first instantiation, we give a general framework that models the setting of MPC with so-called *adaptive security*, and provide a clean-slate treatment of adaptive security guarantees in MPC, exploiting the specification concept of CC. Our new security notion is technically weaker than standard adaptive security, but nevertheless captures security against a fully adaptive adversary. Moreover, the notion avoids the commitment problem and therefore the need to use non-committing encryption tools. In the second instantiation, we provide a very simple framework tailored to the meaningful setting of synchronous protocols and static corruption. The simplicity of the

framework makes it suitable for teaching purposes.

MPC with Enhanced Security Guarantees. In the second part, we explore MPC protocols with enhanced security guarantees. First, we explore protocols that achieve simultaneously the best of both synchronous and asynchronous protocols. Two main differences arise when comparing synchronous and asynchronous protocols. Synchronous protocols achieve a high corruption tolerance, and allow every party to give input to the computation. However, they lose all their security guarantees when synchrony assumptions are violated. Moreover, they proceed in 'rounds', so their speed is proportional to the publicly known worst-case delay upper bound. Asynchronous protocols, on the other hand, do not rely on any synchrony assumptions and work under arbitrary network conditions. Moreover, they are *responsive*: if the network is fast, parties output fast. However, they have a low corruption resilience, and parties with slow connections unavoidably cannot give input. We investigate if one can obtain advantages of both worlds: a high corruption tolerance and input completeness, with no synchrony assumptions and with responsiveness.

Second, we consider the setting of *topology-hiding computation* (THC), where parties have access to an incomplete network of synchronous channels. Here, protocols require the additional security goal that the topology of the network must remain private. We show a THC protocol under fail-stop corruption and standard assumptions, improving over previous results which were only semi-honest or required trusted hardware.

Efficient Byzantine Agreement. In the third part, we focus on the efficiency of Byzantine agreement (BA) protocols, a fundamental primitive in the design of MPC protocols. First, we generalize the Feldman-Micali paradigm and give a new elegant way to design round-efficient BA protocols. Our results show that the round complexity of state of the art BA protocols can be considerably improved with this approach. In particular, we give a BA protocol for the $t < n/3$ setting that improves upon the best known protocol by a factor of $1/2$, and uses a single coin tossing. Second, we show an asynchronous BA protocol with subquadratic communication assuming an initial trusted setup. At the core of this construction, lies an MPC protocol with subquadratic communication when computing no-input functionalities with short output (e.g., coin tossing), which is of independent interest.

Zusammenfassung

Sichere Mehrparteienberechnung (MPC von Englisch *Multi-Party Computation*) erlauben einer Menge von Spielern eine verteilte Berechnung auf sichere Weise gemeinsam auszuführen. Dieses sehr allgemeine Problem ist in der Kryptographie und in verteilten Protokollen von grundlegender Bedeutung und kann anhand zahlreicher Parameter untersucht werden: anhand des zugrunde liegenden Kommunikationsnetzwerks, der zu berechnenden Aufgabe und so weiter.

In den letzten Jahrzehnten war das Problem der Mehrparteienberechnung Gegenstand einer enormen Forschung. Die Anzahl der Anwendungen für MPC ist möglicherweise unbegrenzt, und die Protokolle für MPC wurden von rein theoretischen Anwendungen auf die tatsächliche Bereitstellung in der Praxis umgestellt. Die Fortsetzung tiefgreifender theoretischer Arbeiten, um sicherzustellen, dass MPC auf soliden wissenschaftlichen Grundlagen steht, ist entscheidend, um weitere Fortschritte zu erzielen.

Diese Arbeit leistet Beiträge zu verschiedenen Themen der Mehrparteienberechnung. Man kann diese grob in drei Bereiche unterteilen.

Multi-Party Konstruktive Kryptographie. Im ersten Teil leisten wir Beiträge zum Definitionsteil von MPC. Wir modellieren MPC in dem Konstruktiven Kryptographie Framework (CC von Englisch *Constructive Cryptography*), indem wir zwei Instanziierungen seiner abstrakten Schicht bereitstellen. In der ersten Instanziierung geben wir einen allgemeinen Rahmen an, der MPC mit der sogenannten *adaptiven Sicherheit* modelliert und eine neuere Behandlung der adaptiven Sicherheitsgarantien in MPC unter Ausnutzung des Spezifikationskonzepts von CC bietet. Unser neuer Sicherheitsbegriff ist technisch schwächer als die stan-

dardmässige adaptive Sicherheit, erfasst jedoch die Sicherheit gegen einen vollständig adaptiven Gegner. Darüber hinaus vermeidet der Begriff das Commitment-Problem und daher die Notwendigkeit, non-committing-Verschlüsselungstools zu verwenden. In der zweiten Instanziierung bieten wir ein sehr einfaches Framework, das auf den relevanten Bereich synchroner Protokolle und statischer Korruption zugeschnitten ist. Die Einfachheit des Frameworks macht es für Unterrichtszwecke geeignet.

MPC mit erweiterten Sicherheitsgarantien. Im zweiten Teil untersuchen wir MPC-Protokolle mit erweiterten Sicherheitsgarantien. Zunächst untersuchen wir Protokolle, die gleichzeitig das Beste aus synchronen und asynchronen Protokollen erzielen. Beim Vergleich von synchronen und asynchronen Protokollen ergeben sich zwei Hauptunterschiede. Synchrone Protokolle erreichen eine hohe Korruptionstoleranz und ermöglichen es jedem Spieler, Eingaben für die Berechnung zu machen. Sie verlieren jedoch alle ihre Sicherheitsgarantien, wenn Synchronisationsannahmen verletzt werden. Darüber hinaus operieren sie in "Runden", sodass ihre Geschwindigkeit proportional zur öffentlich bekannten Obergrenze für die Verzögerung im ungünstigsten Fall ist. Asynchrone Protokolle basieren hingegen nicht auf Synchronisationsannahmen und arbeiten unter beliebigen Netzwerkbedingungen. Darüber hinaus sind sie *responsiv*: Wenn das Netzwerk schnell ist, erhalten die Parteien ihre Ausgabe schnell. Sie weisen jedoch eine geringe Korruptionsresistenz auf, und Parteien mit langsamen Verbindungen können unvermeidlich keine Beiträge leisten. Wir untersuchen, ob man die Vorteile beider Welten kombinieren kann: eine hohe Korruptionstoleranz und Vollständigkeit der Eingabe, ohne Synchronisationsannahmen und responsivem Verhalten.

Zweitens betrachten wir *Topologie-Versteckende-Berechnung* (THC von Englisch *Topology-Hiding Computation*), bei der Parteien Zugriff auf ein unvollständiges Netzwerk synchroner Kanäle haben. Hier erfordern Protokolle das zusätzliche Sicherheitsziel, dass die Topologie des Netzwerks privat bleiben muss. Wir zeigen ein THC-Protokoll unter Fail-Stop-Korruption und Standardannahmen, das frühere Ergebnissen verbessert, die nur halb-ehrliche Korruption tolerieren konnten oder vertrauenswürdige Hardware erforderten.

Effiziente Byzantinische Vereinbarung. Im dritten Teil konzentrieren wir uns auf die Effizienz der Protokolle der byzantinischen Vereinbarung (BA von Englisch *Byzantine Agreement*), ein grundlegendes Ement

beim Entwurf von MPC-Protokollen. Zunächst verallgemeinern wir das Feldman-Micali-Paradigma und geben eine neue elegante Möglichkeit, runden-effiziente BA-Protokolle zu entwerfen. Unsere Ergebnisse zeigen, dass die Runden-Komplexität von BA-Protokollen nach dem Stand der Technik mit diesem Ansatz erheblich verbessert werden kann. Insbesondere geben wir ein BA-Protokoll für den Fall $t < n/3$ an, das das bekannteste Protokoll um den Faktor $1/2$ verbessert und einen einzelnen Münzwurf verwendet. Zweitens zeigen wir ein asynchrones BA-Protokoll mit subquadratischer Kommunikation unter der Annahme eines anfänglichen vertrauenswürdigen Setups. Im Kern dieser Konstruktion liegt ein MPC-Protokoll mit subquadratischer Kommunikation bei der Berechnung von Funktionen ohne Eingabe mit kurzer Ausgabe (z. B. Münzwurf), das von unabhängigem Interesse ist.

Contents

Chapter 1

Introduction

1.1 Motivation

Secure multi-party computation (MPC) is one of the most fundamental problems in cryptography. It considers the scenario where a set of parties wish to carry out a computation in a *secure* manner. Security in this context informally means guaranteeing the correctness of the computation as well as the privacy of parties' inputs, even when some of the parties might misbehave. There is a huge number of applications for MPC, including voting, consensus, privacy-preserving auctions, privacy-preserving machine learning, and many more. Since the invention of its first theoretical results a few decades ago, a huge amount of research has been done.

Security Definitions. A crucial step towards meaningfully designing and analyzing cryptographic protocols is to come up with appropriate definitions of security. Formulating good definitions is highly non-trivial: the definition should closely capture the aspects that we care about, while at the same time being simple and usable, even minimal, avoiding as much as possible unnecessary artifacts.

A very natural approach to understand the security of distributed protocols (and in particular MPC protocols), is by means of a construction. There, one needs to specify the assumed systems or modules available to the protocol, and the ideal system that the distributed protocol aims to

achieve. For example, one can consider a key agreement protocol, where parties have access to authenticated communication, and the goal is to construct an ideal uniform random shared secret key. Another example can be the classical BGW model, where parties have access to a network of synchronous pair-wise secure channels, and the ideal system allows parties to evaluate a function. Importantly, the construction statements should then be *composable*: in any larger application that assumes the availability of an ideal system, it is safe to replace it by the corresponding protocol and assumed modules that achieve it.

A long-term goal is therefore to build a solid library of constructions that are useful for large applications in a modular way.

Constructions in MPC. One can roughly classify MPC constructions according to two aspects about the real-world systems that parties have available when executing a protocol.

The first aspect is the reliability of computers that parties use to execute the protocol. Typical settings include parties that run the protocol on computers that can be hacked and arbitrarily taken over (often denoted the active and adaptive corruption model), or one may consider computers that execute correctly the protocol specification, but that can leak the internal state (the passive corruption model). Note that the space of computers that one can consider is potentially unlimited: one can consider randomness corruption, memory corruption, etc.[1]

The second aspect is the communication network that parties have available. The most common network considered in the literature of MPC is the so-called synchronous network, where an upper bound on the network delay is assumed. When considering synchronous networks, one typically also assumes that parties have access to synchronized clocks as well. Another popular communication network is the asynchronous network. Here, protocols do not rely on any timing assumptions and the network delay is arbitrary, but at the same time it is more challenging to achieve.

In this thesis, we explore such MPC constructions and the concrete case of Byzantine agreement according to both aspects, with different

[1] In the literature, one often considers an adversary entity explicitly, and specifies what the adversarial entity may or may not do. One then usually says that a party gets corrupted, meaning that its computer was hacked. Note that instead of defining different types of adversary entity, one can make constructive statements by simply specifying the guarantees of the computers available in the protocol.

security guarantees and different levels of efficiency.

1.2 Overview and Contributions

This thesis is based on some of the research done during my Ph.D. studies at ETH Zurich. Parts of the material were published in [LLM+18, BLL20, LM20b, BKLL20, LLM+20b, FLL21, HLM21], together with many of the collaborators I was lucky to interact with, and some parts are still unpublished. Some of the research done during my Ph.D. did not make it to the thesis, because it is still on-going, did not fit in the topic or various other reasons. These include the published papers [BCLM17, LMRT17, LLM+20a, GHL20, LMM20, HKL20, CHL21, DHL21].

The thesis makes contributions in various topics of multi-party computation, and is divided into three parts. In the first part, we give contributions to the definitional part of multi-party computation. The second part explores MPC constructions with enhanced security guarantees. The third part focuses on the efficiency of Byzantine agreement protocols, a fundamental primitive used to design MPC protocols. In the following, we summarize the main contributions and give pointers to the relevant chapters and the papers they are based on.

1.2.1 Multi-Party Constructive Cryptography

In this part, we give definitional contributions to the area of multi-party computation. The part contains Chapters 3 to 5.

Constructive Cryptography Framework: The Basics

We take the constructive cryptography framework (CC) developed by Maurer and Renner [MR11, Mau11, MR16] as the starting point. Chapter 3 is devoted to summarize the essential concepts that will be used in later chapters.

The constructive cryptography framework follows the very natural constructive paradigm described in the introduction. The central concept in constructive cryptography is that of a *resource*. Resources model both the systems or modules available to a protocol (e.g., a communication network, computers, clocks, etc.), and the ideal system that the

protocol is supposed to construct (e.g., a broadcast channel, a secure function evaluation system, etc.). The framework then specifies a theory of resources with an algebra, where protocols can transform resources to create further resources, following well-defined algebraic axioms.

Although Parts II and III of the thesis are not written in the CC framework, mostly because the development of the framework took place in parallel with the results of other parts (and some aspects are still under development), all results can be phrased within the CC framework. Moreover, the general paradigm and philosophy remains the basis of this thesis.

Adaptive Security in MPC

Chapter 4 contains an instantiation of the constructive cryptography framework to the setting of multi-party computation with adaptive security, where the adversary is allowed to choose adaptively which parties to corrupt during the protocol execution. A main technical obstacle in this context is the so-called "commitment problem", where the simulator is unable to consistently explain the internal state of a party with respect to its pre-corruption outputs. As a result, protocols typically resort to the use of cryptographic primitives like non-committing encryption, incurring an important efficiency loss.

We provide a new, clean-slate treatment of adaptive security in MPC, exploiting the specification concept of constructive cryptography. A new natural security notion, called CC-adaptive security, is proposed, which is technically weaker than standard adaptive security but nevertheless captures security against a fully adaptive adversary. Known protocol examples separating between adaptive and static security are also insecure in our notion. Moreover, our notion avoids the commitment problem and thereby the need to use non-committing tools. We exemplify this by showing that the protocols by Cramer, Damgard and Nielsen (EUROCRYPT'01) for the honest majority setting, and (the variant without non-committing encryption) by Canetti, Lindell, Ostrovsky and Sahai (STOC'02) for the dishonest majority setting, achieve CC-adaptive security. The latter example is of special interest since all UC-adaptive protocols in the dishonest majority setting require some form of non-committing or equivocal encryption. The material of this chapter is based on the published work in [HLM21].

Synchronous Constructive Cryptography

Chapter 5 contains a simple synchronous composable security framework as an instantiation of the constructive cryptography framework, aiming to capture minimally, without unnecessary artefacts, exactly what is needed to state synchronous security guarantees. The results in this chapter were previously published in [LM20b].

The objects of study are specifications (i.e., sets) of systems, and traditional security properties like consistency and validity can naturally be understood as specifications, thus unifying composable and property-based definitions. The framework's simplicity is in contrast to current composable frameworks for synchronous computation which are built on top of an asynchronous framework (e.g. the UC framework), thus not only inheriting artefacts and complex features used to handle asynchronous communication, but adding additional overhead to capture synchronous communication. We then demonstrate how secure (synchronous) multi-party computation protocols can be understood as constructing a computer that allows a set of parties to perform an arbitrary, on-going computation, in line with the arithmetic black-box functionality proposed by Damgard and Nielsen [DN03]. An interesting aspect is that the instructions of the computation need not be fixed before the protocol starts but can also be determined during an on-going computation, possibly depending on previous outputs.

1.2.2 Multi-Party Computation with Enhanced Security Guarantees

In this part, we explore multi-party computation constructions with enhanced security guarantees. By enhanced security guarantees we mean security guarantees that are typically not achievable by usual MPC protocols, as we summarize below. The part contains Chapters 6 to 9.

Universally Composable Framework: The Basics

The results in this part are described using the universally composable framework (UC) introduced by Canetti [Can01]. Chapter 6 summarizes the main concepts and notation of the framework, and an instantiation of UC functionalities for synchronized clocks and different types of networks.

Best of Synchronous and Asynchronous Protocols

The first two results in this part are devoted to the design of protocols achieving (simultaneously) guarantees that synchronous and asynchronous protocols achieve. The results are in Chapters 7 and Chapter 8, and are based on the previously published works in [BLL20, LLM+20b].

Two main differences arise when comparing synchronous and asynchronous protocols. Synchronous MPC protocols can achieve the optimal corruption threshold $n/2$ for full security assuming setup, and even arbitrary number of corruptions for the case of security with abort. Moreover, they allow every party to give input to the computation. However, they work under the assumption that the network is synchronous, and become completely insecure when synchrony assumptions are violated. Moreover, synchronous protocols proceed in rounds, so their speed is proportional to the publicly known worst-case delay upper bound.

Asynchronous MPC protocols do not rely on any synchrony timing assumptions and work under arbitrary network conditions. These protocols are message-driven and have the feature that the speed at which the parties can obtain output depends on the actual network delay. They are *responsive*: if the network is fast, parties output fast. However, they tolerate less than $n/3$ corruptions, and parties with slow connections unavoidably cannot give input.

Synchronous MPC with Asynchronous Fallback. Chapter 7 investigates whether there exists a protocol for MPC that can achieve security guarantees when executed in either network type. More concretely, we investigate if there is a protocol tolerating up to $t_s < n/2$ corruptions under a synchronous network and $t_a < n/3$ corruptions even when the network is asynchronous. We show that such a protocol exists if and only if $t_a + 2t_s < n$ and the number of inputs taken into account under an asynchronous network is at most $n - t_s$.

Synchronous MPC with Asynchronous Responsiveness. Chapter 8 investigates whether it is possible to leverage MPC protocols that run over a synchronous network to achieve responsiveness: full security with responsiveness up to t corruptions, and *extended* security (full security or security with unanimous abort) with no responsiveness up to $T \geq t$ corruptions. We show that:

- For the case of unanimous abort as extended security, there is an

MPC protocol if and only if $T + 2t < n$.

- For the case of full security as extended security, there is an MPC protocol if and only if $T < \frac{n}{2}$ and $T + 2t < n$. In particular, setting $t = \frac{n}{4}$ allows to achieve a fully secure MPC for honest majority, which in addition benefits from having substantial responsiveness.

Topology-Hiding Computation

The third result considers the setting of topology-hiding communication and computation, and can be found in Chapters 9. The material is based on the previously published work [LLM+18].

Topology-hiding communication protocols allow a set of parties, connected by an incomplete network with unknown communication graph, where each party only knows its neighbors, to construct a complete communication network such that the network topology remains hidden even from a powerful adversary who can corrupt parties. This communication network can then be used to perform arbitrary tasks, for example secure multi-party computation, in a topology-hiding manner.

Previously proposed protocols under standard assumptions could only tolerate passive corruption. Chapter 9 proposes protocols that can also tolerate fail-corruption (i.e., the adversary can crash any party at any point in time) and so-called semi-malicious corruption (i.e., the adversary can control a corrupted party's randomness), without leaking more than an arbitrarily small fraction of a bit of information about the topology. Since leaking a small amount of information is unavoidable, as is the need to abort the protocol in case of failures, our protocols seem to achieve the best possible goal in a model with fail-corruption. Further contributions include applications of the protocol to obtain secure MPC protocols, which requires a way to bound the aggregated leakage when multiple small-leakage protocols are executed in parallel or sequentially. Moreover, a protocol using fully-homomorphic encryption with better round complexity is proposed.

1.2.3 Efficient Byzantine Agreement Protocols

This part of the thesis is devoted to explore efficient protocols for Byzantine agreement (BA), a fundamental building block for MPC protocols.

Two parameters are of vital importance: the round and communication complexity. The part contains Chapters 10 and 11.

Expand-and-Extract: A New Way to Design Round-Efficient Byzantine Agreement

In Chapter 10, we address the round complexity of Byzantine agreement protocols. The material of this chapter is based on previously published work [FLL21].

Minimizing the round complexity of Byzantine Agreement protocols is a fundamental problem in distributed computing. The typical approach to achieve round efficient (randomized) BA is to have a weak form of BA, called graded consensus (GC), followed by a distributed coin, and to repeat this process until some termination condition is met—as introduced by Feldman and Micali (STOC '88).

In Chapter 10, we revisit the question of building BA from GC, or, more precisely, from generalizations of GC. Concretely, we demonstrate that the round complexity of *fixed-round* BA (with a given target error probability) can be considerably reduced based on this generalization. In particular, assuming a setup for threshold signatures among the parties and corruption threshold $t < n/3$, we improve over the round complexity of the best known protocol by a factor of $1/2$, asymptotically; this is achieved by applying *one single* Feldman-Micali iteration consisting of one (generalized) GC instance and one round of coin tossing. Our technique also applies to the dishonest-minority case ($t < n/2$), yielding an improvement by a factor of $1/4$ (asymptotically) over the round complexity of the best known fixed-round protocol.

Asynchronous Byzantine Agreement with Subquadratic Communication

In Chapter 11, we address the communication complexity of Byzantine agreement protocols. The material of this chapter is based on the previously published work [BKLL20].

Understanding the communication complexity of Byzantine agreement is a fundamental problem in distributed computing. In particular, for protocols involving a large number of parties (as in, e.g., the context of blockchain protocols), it is important to understand the dependence

of the communication on the number of parties n. In Chapter 11, we show how to use the so-called *player-replaceability* paradigm to achieve asynchronous BA protocols with subquadratic communication complexity. The results hold against an adaptive adversary who can corrupt $f < (1 - \epsilon)n/3$ of the parties (for any constant $\epsilon > 0$), in the *atomic-send* model, where parties can atomically send messages to all parties, and messages sent by honest parties cannot be retrieved back (even if they become corrupted). One protocol assumes initial setup done by a trusted dealer, after which an unbounded number of BA executions can be run; alternately, we can achieve subquadratic *amortized* communication with no prior setup. At the heart of the protocol is an MPC protocol in the same threat model that has $o(n^2)$ communication when computing no-input functionalities with short output (e.g., coin tossing).

1.3 Related Work

The thesis covers different topics within multi-party computation. We therefore provide the relevant related work for each topic in the respective chapters. The bibliography is found at the end of the thesis.

Chapter 2

Preliminaries

2.1 Notation

We denote $\mathcal{P}$ the set of parties participating in the distributed protocol. Moreover, we denote by $\mathcal{H}$ the set of honest parties. We denote the set of integers $\{1, ..., n\}$ as $[n]$. Throughout the thesis, whenever we use a security parameter, we denote it by κ.

We denote random variables by capital letters. Prefixes of sequences of random variables are denoted by a superscript, e.g. X^i denotes the finite sequence $X_1, \ldots, X_i$. For random variables X and Y, we denote by $\mathrm{p}_{X|Y}$ the corresponding conditional probability distribution. Given a tuple t, we write the projection to the j-th component of the tuple as $[t]_j$. Given a sequence t^i of tuples $t_1, \ldots, t_i$, we write $[t^i]_j$ as the sequence $[t_1]_j, \ldots, [t_i]_j$. For a finite set X, we will use $x \leftarrow_{\$} X$ and $x \leftarrow X$ similarly, to denote sampling x uniform randomly from X.

2.2 Cryptographic Primitives

In this section, we define the basic cryptographic primitives that are used in several chapters of the thesis. Other primitives that are only used in specific chapters are presented in the respective chapters. The reader familiar with these tools can skip this section.

2.2.1 Public-Key Encryption

We recall the definition of a public-key encryption scheme, firstly introduced by Goldwasser and Micali [GM84].

Definition 2.2.1. A public-key encryption scheme consists of three algorithms $\mathcal{E} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$:

- (Key generation) The key generation algorithm takes as input the security parameter κ, and outputs $(\mathsf{ek}, \mathsf{dk}) = \mathsf{Keygen}(1^\kappa)$, where ek is the public key, and dk is the secret key.
- (Encryption) There is an algorithm Enc, which on input public key ek and plaintext m, it outputs an encryption $c = \mathsf{Enc}_{\mathsf{ek}}(m; r)$ of m, with random input r.
- (Decryption) There is an algorithm that, given as input a decryption key dk and a ciphertext, it outputs $m = \mathsf{Dec}(c)$.

We require perfect correctness, in the sense that for any keys $(\mathsf{ek}, \mathsf{dk})$ in the support of Keygen and any plaintext m, it holds that if $c = \mathsf{Enc}_{\mathsf{ek}}(m; r)$ for some r, then $m = \mathsf{Dec}(c)$.

Semantic security of the encryption scheme is defined via the IND-CPA game.

Game $\mathsf{IND\text{-}CPA}^b_{\mathcal{E},\mathcal{A}}(\kappa)$

1: Let $(\mathsf{ek}, \mathsf{dk}) = \mathsf{Keygen}(1^\kappa)$. Output (κ, ek) to $\mathcal{A}$.
2: On input (m_0, m_1) from $\mathcal{A}$, output $c = \mathsf{Enc}_{\mathsf{ek}}(m_b)$ to $\mathcal{A}$.
3: On input b' from $\mathcal{A}$, output b'.

Definition 2.2.2. For a public-key encryption scheme $\mathcal{E}$, we define the CPA-advantage of an adversary $\mathcal{A}$ as the difference in the probability that it outputs 1 when interacting in the game $\mathsf{IND\text{-}CPA}^0_{\mathcal{E},\mathcal{A}}$ or $\mathsf{IND\text{-}CPA}^1_{\mathcal{E},\mathcal{A}}$. We say that $\mathcal{E}$ is IND-CPA secure, if the advantage of any efficient adversary $\mathcal{A}$ is negligible in κ.

2.2.2 Threshold Homomorphic Encryption

We recall the definition of a threshold encryption scheme. This is similar to a public-key encryption scheme, except that it has two additional

properties. First, it is homomorphic, meaning that one can compute linear operations on ciphertexts without the need to decrypt them. And second, it supports threshold decryption. This means that the decryption key is actually distributed among the set of n parties, and 1) given partial decryptions of $t+1$ parties, one can recover the original message, and 2) any adversary corrupting up to t parties has no information on the encrypted message.

Definition 2.2.3. A threshold homomorphic encryption scheme consists of five algorithms:

- (Key generation) The key generation algorithm is parameterized by (t, n) and outputs $(\mathsf{ek}, \mathsf{dk}) = \mathsf{Keygen}_{(t,n)}(1^\kappa)$, where ek is the public key, and $\mathsf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ is the list of secret keys.

- (Encryption) There is an algorithm Enc, which on input public key ek and plaintext m, it outputs an encryption $\overline{m} = \mathsf{Enc}_{\mathsf{ek}}(m; r)$ of m, with random input r.

- (Partial decryption) There is an algorithm that, given as input a decryption key dk_i and a ciphertext, it outputs $d_i = \mathsf{DecShare}_{\mathsf{dk}_i}(c)$, a decryption share.

- (Reconstruction) Given $t+1$ decryption shares $\{d_i\}$, one can reconstruct the plaintext $m = \mathsf{Rec}(\{d_i\})$.

- (Additively Homomorphic) There is an algorithm which, given public key ek and encryptions $\overline{a}$ and $\overline{b}$, it outputs a uniquely-determined encryption $\overline{a+b}$. We write $\overline{a+b} = \overline{a} \boxplus \overline{b}$. Likewise, there is an algorithm performing substraction: $\overline{a-b} = \overline{a} \boxminus \overline{b}$.

- (Multiplication by constant) There is an algorithm, which, given public key ek, a plaintext a and a ciphertext $\overline{b}$, it outputs a uniquely-determined encryption $\overline{a \cdot b}$. We write $\overline{a \cdot b} = a \boxdot \overline{b}$.

A threshold encryption scheme with a statically-secure threshold decryption protocol can be found for example in [CDN01] based on the Paillier's cryptosystem [Pai99], and based on the QR and DDH assumption. Such a scheme with an adaptively secure threshold decryption protocol can be found in [LP01], based on the Paillier cryptosystem.

2.2.3 Threshold Fully Homomorphic Encryption

A stronger primitive than threshold homomorphic encryption, is that of threshold *fully*-homomorphic encryption [Gen09, vGHV10, BV11, AJL+12, BGG+18]. The difference is that instead of being able to perform only linear operations on ciphertexts, one can perform arbitrary computations.

Definition 2.2.4. A threshold fully-homomorphic encryption (TFHE) scheme consists of the following algorithms:

- (Key generation) The key generation algorithm is parameterized by (t, n) and outputs $(\mathsf{ek}, \mathsf{dk}) = \mathsf{Keygen}_{(t,n)}(1^\kappa)$, where ek is the public key, and $\mathsf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ is the list of secret keys.
- (Encryption) There is an algorithm Enc, which on input public key ek and plaintext m, it outputs an encryption $\overline{m} = \mathsf{Enc}_{\mathsf{ek}}(m; r)$ of m, with random input r.
- (Partial decryption) There is an algorithm that, given as input a decryption key dk_i and a ciphertext, it outputs $d_i = \mathsf{DecShare}_{\mathsf{dk}_i}(c)$, a decryption share.
- (Reconstruction) Given $t+1$ decryption shares $\{d_i\}$, one can reconstruct the plaintext $m = \mathsf{Rec}(\{d_i\})$.
- (Evaluation) The homomorphic evaluation algorithm Eval takes as input the encryption key ek, an s-input circuit C, and s ciphertexts $c_1, \ldots, c_s$; it outputs a ciphertext c.

We require:

Correctness: For any integers s, t, n, messages $\{m_i\}_{i\in[s]}$, s-input circuit C, and set $I \subseteq [n]$ with $|I| = t$, if we run $(\mathsf{ek}, \{\mathsf{dk}_i\}_{i\in[n]}) \leftarrow \mathsf{Keygen}(1^\kappa, 1^t, 1^n)$ followed by

$$c := \mathsf{Eval}_{\mathsf{ek}}(C, \mathsf{Enc}_{\mathsf{ek}}(m_1), \ldots, \mathsf{Enc}_{\mathsf{ek}}(m_s)),$$

then $\mathsf{Rec}(\{\mathsf{Dec}_{\mathsf{dk}_i}(c)\}_{i\in I}) = C(m_1, \ldots, m_s)$.

Compactness: There is a polynomial p such that for all $(\mathsf{ek}, \mathsf{dk})$ output by $\mathsf{Keygen}(1^\kappa, 1^t, 1^n)$ and all $\{m_i\}$, the length of

$$\mathsf{Eval}_{\mathsf{ek}}(C, \mathsf{Enc}_{\mathsf{ek}}(m_1), \ldots, \mathsf{Enc}_{\mathsf{ek}}(m_s))$$

is at most $p(|C(m_1, \ldots, m_s)|, \kappa)$.

2.2.4 Digital Signatures

We review the syntax and definition of a signature scheme.

Definition 2.2.5. A digital signature scheme is a triple of algorithms $\mathcal{DS} = (\mathsf{Keygen}, \mathsf{Sign}, \mathsf{Ver})$:

- (Key generation) The key generation algorithm Keygen takes as input the security parameter κ, and outputs a key pair $(\mathsf{pk}, \mathsf{sk})$, where pk is the public key or verification key, and sk is the secret key or signing key.
- (Signing) There is an algorithm Sign, which on input the signing key sk and message m, it outputs a signature $\sigma = \mathsf{Sign}_{\mathsf{sk}}(m)$.
- (Verification) There is an algorithm Ver that, given as input a verification key pk, a message m and a signature σ, outputs a bit $b = \mathsf{Ver}_{\mathsf{pk}}(m, \sigma)$.

We require perfect correctness, in the sense that for any keys $(\mathsf{pk}, \mathsf{sk})$ in the support of Keygen, message m, and signature $\sigma = \mathsf{Sign}_{\mathsf{pk}}(m)$, it holds that $\mathsf{Ver}_{\mathsf{pk}}(m, \sigma) = 1$.

Security of the signature scheme is defined via the EU-CMA unforgeability game.

Game $\mathsf{EU\text{-}CMA}_{\mathcal{DS},\mathcal{A}}(\kappa)$

1: Let $(\mathsf{pk}, \mathsf{sk}) = \mathsf{Keygen}(1^\kappa)$. Output (κ, pk) to $\mathcal{A}$.
2: On input m from $\mathcal{A}$, output $\sigma = \mathsf{Sign}_{\mathsf{sk}}(m)$ to $\mathcal{A}$.
3: On input (m, σ) from $\mathcal{A}$, output a bit b, where $b = 1$ if and only if m was not queried before and $\mathsf{Ver}_{\mathsf{pk}}(m, \sigma) = 1$.

Definition 2.2.6. For a digital signature scheme $\mathcal{DS}$, we define the EU-CMA advantage of an adversary $\mathcal{A}$ as the probability that $\mathcal{A}$ wins the unforgeability game. We say that $\mathcal{DS}$ is EU-CMA secure, if the advantage of any efficient adversary $\mathcal{A}$ is negligible in κ.

Part I

Multi-Party Constructive Cryptography

Chapter 3

Constructive Cryptography Framework: The Basics

In this chapter, we summarize the basic concepts of the Constructive Cryptography framework by Maurer and Renner [MR11, Mau11, MR16] needed for this thesis. The general theory goes beyond the field of Cryptography, but in most parts we describe the theory with the goal of phrasing cryptographic statements in mind.

3.1 Specifications

A basic idea, which one finds in many disciplines, is that one considers a set Φ of objects and *specifications* of such objects. A specification $\mathcal{U} \subseteq \Phi$ is a subset of Φ and can equivalently be understood as a predicate on Φ defining the set of objects satisfying the specification, i.e., being in $\mathcal{U}$. Examples of this general paradigm are the specification of mechanical parts in terms of certain tolerances (e.g. the thickness of a bolt is between 1.33 and 1.34 millimeters), the specification of the property of a program (e.g. the set of programs that terminate, or the set of programs that compute a certain function within a given accuracy and time limit), or

in a cryptographic context the specification of a close-to-uniform n-bit key as the set of probability distributions over $\{0,1\}^n$ with statistical distance at most ϵ from the uniform distribution.

A specification corresponds to a guarantee, and smaller specifications hence correspond to stronger guarantees. An important principle is to *abstract* a specification $\mathcal{U}$ by a larger specification $\mathcal{V}$ (i.e., $\mathcal{U} \subseteq \mathcal{V}$) which is simpler to understand and work with. One could call $\mathcal{V}$ an ideal specification to hint at a certain resemblance with terminology often used in the cryptographic literature. If a construction (see below) requires an object satisfying specification $\mathcal{V}$, then it also works if the given object actually satisfies the stronger specification $\mathcal{U}$.

3.2 Constructions

A *construction* is a function $\gamma : \Phi \to \Phi$ transforming objects into (usually in some sense more useful) objects. A well-known example of a construction useful in cryptography, achieved by a so-called extractor, is the transformation of a pair of independent random variables (say a short uniform random bit-string, called seed, and a long bit-string for which only a bound on the min-entropy is known) into a close-to-uniform string.

A construction statement of specification $\mathcal{S}$ from specification $\mathcal{R}$ using construction γ, denoted $\mathcal{R} \xrightarrow{\gamma} \mathcal{S}$, is of the form

$$\mathcal{R} \xrightarrow{\gamma} \mathcal{S} \quad :\Longleftrightarrow \quad \gamma(\mathcal{R}) \subseteq \mathcal{S}.$$

It states that if construction γ is applied to any object satisfying specification $\mathcal{R}$, then the resulting object is guaranteed to satisfy (at least) specification $\mathcal{S}$.

The composability of this construction notion follows immediately from the transitivity of the subset relation:

$$\mathcal{R} \xrightarrow{\gamma} \mathcal{S} \ \wedge \ \mathcal{S} \xrightarrow{\gamma'} \mathcal{T} \ \implies \ \mathcal{R} \xrightarrow{\gamma' \circ \gamma} \mathcal{T}.$$

3.3 Systems Theory

The above natural and very general viewpoint is also taken in Constructive Cryptography, where the objects in Φ are systems, called *resources*,

with interfaces to the parties considered in the given setting. The framework is a resource theory with an algebra of systems, where converters and resources can be combined to create further resources, according to a series of natural algebraic axioms. In this thesis we model the basic objects as discrete reactive systems, formally modeled as random systems [Mau02, MPR07], characterized by their input and output behavior.

3.3.1 Random Systems

Definition 3.3.1. An $(\mathcal{X}, \mathcal{Y})$-random system $\mathbf{R}$ is a sequence of conditional probability distributions $\mathrm{p}^{\mathbf{R}}_{Y_i|X^iY^{i-1}}$, for $i \geq 1$. Equivalently, the random system can be characterized by the sequence of distributions $\mathrm{p}^{\mathbf{R}}_{Y^i|X^i} = \prod_{k=1}^{i} \mathrm{p}^{\mathbf{R}}_{Y_k|X^kY^{k-1}}$, for $i \geq 1$.

A random system is the mathematical object corresponding to the *behavior* of a discrete system. A deterministic system is a special type of function (or sequence of functions), and the composition of systems is defined via function composition. Probabilistic systems are often thought about (and described) at a more concrete level, where the randomness is made explicit (e.g. as the randomness of an algorithm or the random tape of a Turing machine). Hence a probabilistic discrete system (PDS) corresponds to a probability distribution over deterministic systems, and the definition of the composition of probabilistic systems is induced by the definition of composition of deterministic systems (analogously to the fact that the definition of the sum of real-valued random variables is naturally induced by the definition of the sum of real numbers, which are not probabilistic objects). Different PDS can have the same behavior, which means that the behavior, i.e., a random system, corresponds to an equivalence class of PDS (with the same behavior).

We often describe a random system in many different ways, e.g. by several variants of pseudo-code, and as is common in the literature we also use such an ad-hoc description language.

3.3.2 Resources and Converters

Resources. A resource $\mathbf{R}$ is a random system with interfaces, where the interface address and the actual input value are encoded as part of the input. Then, the system answers with an output value at the

same interface. One can take several independent resources $\mathbf{R}_1, \ldots, \mathbf{R}_k$ with disjoint interfaces, and form a new resource $[\mathbf{R}_1, \ldots, \mathbf{R}_k]$, with the interface set being the union. This resource is denoted as the parallel composition. For each party, all its interfaces are merged into a single interface, where the original interfaces can be thought of as sub-interfaces.

Converters. A converter models the local actions executed by a party at its interface, which can be thought of as a system or protocol engine. Formally, converters are modeled as random systems with two interfaces, an outside interface and an inside interface. At its inside, the converter gives input to the party's interface of the resource and at the outside it emulates an interface (of the transformed resource). Upon an input at an outside interface, the converter is allowed to make a bounded number of queries to the inside interfaces, before returning a value at the queried interface. Applying a converter induces a mapping $\Phi \to \Phi$. We denote the set of converters as Σ.

For a converter α and a resource $\mathbf{R}$, we denote by $\alpha^i\mathbf{R}$ the resource obtained from applying the converter to the resource at interface i. One can then see that converter attachment satisfies *composition order invariance*, meaning that applying converters at distinct interfaces commutes. That is, for any converters α and β, any resource $\mathbf{R}$ and any disjoint interfaces j, k, we have that $\alpha^j\beta^k\mathbf{R} = \beta^k\alpha^j\mathbf{R}$.[1]

Figure 3.1 shows a resource with four interfaces where converters are applied at two of the interfaces. The resource obtained by applying a converter α at interface j of resource $\mathbf{R}$ is denoted as $\alpha^j\mathbf{R}$. The resource shown in Figure 3.1 can hence be written

$$\alpha^2\beta^4\mathbf{R},$$

which is equal to $\beta^4\alpha^2\mathbf{R}$.

3.3.3 Modeling Aspects: Resources vs Converters

The general guiding principle in Constructive Cryptography is that everything that is considered relevant is modeled as part of the resource, and aspects within the converters are considered irrelevant. For example, in

[1] This is an abstract requirement, in the sense of an axiom, which for an instantiation of the theory, for example to the special case of discrete systems, must be proven to hold.

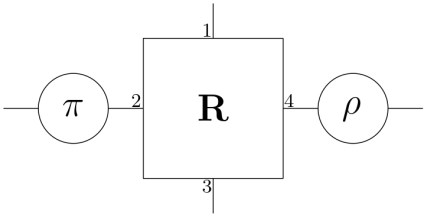

Figure 3.1: Example of a resource with 4 interfaces, where converters α and β are attached to interfaces 2 and 4.

the case where one only wants to make statements about an unbounded adversary, i.e. *information-theoretic security*, the converter set contains all systems, regardless of the actual computational complexity. This is the approach taken in Chapter 5.

On the other hand, in the more general case where aspects about the protocol converters are relevant, one can consider the set of converters that perform no computation, and only connect systems themselves. The protocol converters are then modeled as a parallel resource. In this case, all aspects of the protocol converters can be made explicit: its computing power, the memory requirements, the way they become adaptively corrupted, and so on. This is the approach taken in Chapter 4.

3.3.4 Basic Construction Notion and Aspects

The general construction notion from Section 3.2 leads to the basic construction notion in Constructive Cryptography, where objects in Φ are resources and constructions correspond to converter attachment. The construction notion satisfies the usual composition properties that one expects: sequential and parallel composition.

Theorem 3.3.2. *Let $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{T}$ be specifications, and let α and β be converters attached at the same interface i of specifications $\mathcal{R}$ and $\mathcal{S}$, respectively. Then,*

1. $\alpha^i\mathcal{R} \subseteq \mathcal{S} \ \wedge \ \beta^i\mathcal{S} \subseteq \mathcal{T} \Longrightarrow (\beta\alpha)^i\mathcal{R} \subseteq \mathcal{T}$.
2. $\alpha^i\mathcal{R} \subseteq \mathcal{S} \Longrightarrow \alpha^i[\mathcal{R}, \mathcal{T}] \subseteq [\mathcal{S}, \mathcal{T}]$.

Proof. The first property trivially follows from the transitivity of the subset relation, while the second property follows from the fact that $\alpha^i[\mathcal{R}, \mathcal{T}] = [\alpha^i\mathcal{R}, \mathcal{T}] \subseteq [\mathcal{S}, \mathcal{T}]$. □

An important aspect about the construction notion is its generality and simplicity. While the template of the contruction notion, the subset relation, is very simple and rigid, the definition of what the specifications are is completely flexible. This will allow for a more flexible way of defining ideal guarantees, as we show in Chapters 4 and 5.

Further note that the construction notion does not fix any computational model, asymptotic efficiency notion, or even the existence of a simulator. This is in contrast to the typical construction notion in the literature, which hard-codes all these aspects. First, it phrases the ideal specification $\mathcal{S}$ with the so-called simulation paradigm, e.g. by showing the existence of a monolithic simulator σ attached to a fixed ideal resource $\mathbf{S}$. Second, it also fixes a particular computational model, typically a model where the security statements hold except with negligible distinguishing advantage with respect to any polynomial-time distinguisher. The basic construction notion in Constructive Cryptography allows to consider different security notions in the literature as special cases of this, and further enable more flexible security statements.

3.4 Relaxations

The construction notion stated above does not incur any type of statistical error or computational assumption. However, often a construction statement does not achieve a desired specification $\mathcal{S}$, but only a *relaxed or approximated* version of it. We capture this via so-called relaxations [MR16], which map specifications to weaker, or relaxed, specifications. A relaxation formalizes the idea that we are often happy with resources being almost as good as a target resource specification. For example, one could consider the relaxation that maps a resource $\mathbf{S}$ to the set of resources that are indistinguishable from $\mathbf{S}$.

Definition 3.4.1. Let Φ denote the set of all resources. A relaxation $\phi : \Phi \to 2^\Phi$ is a function such that $\mathbf{R} \in \phi(\mathbf{R})$, for all $\mathbf{R} \in \Phi$. In addition, for a specification $\mathcal{R}$, we define $\mathcal{R}^\phi \coloneqq \bigcup_{\mathbf{R}\in\mathcal{R}} \phi(\mathbf{R})$.

Relaxations satisfy two important properties. The first is that $\mathcal{S} \subseteq \mathcal{S}^{\phi}$. And the second is that if $\mathcal{R} \subseteq \mathcal{S}$ then $\mathcal{R}^{\phi} \subseteq \mathcal{S}^{\phi}$. This simplifies the modular analysis, as it means that one can typically consider assumed resources that are completely ideal, or not relaxed. More concretely, from the statements $\mathcal{R} \subseteq \mathcal{S}^{\phi}$ and $\mathcal{S} \subseteq \mathcal{T}^{\phi'}$, one can conclude that $\mathcal{R} \subseteq \mathcal{T}^{\phi \circ \phi'}$.

In the following, we give a few generic types of relaxations [MR16, JM20].

3.4.1 ϵ-Relaxation

An important relaxation is the relaxation that captures resources that are close to the desired resource $\mathbf{R}$. This *closeness* can be captured in different ways. If a pseudo-metric d on Φ is defined[2], one can consider the specification of resources that are ϵ-close around $\mathbf{R}$.

Definition 3.4.2. Let d be a pseudo-metric on Φ. We define the ϵ-relaxation of a resource $\mathbf{R}$ with pseudo-metric d as:

$$\mathbf{R}^{\epsilon,d} \coloneqq \{\mathbf{S} \in \Phi \mid d(\mathbf{R}, \mathbf{S}) \leq \epsilon\}.$$

A typical pseudo-metric considered is the one obtained from the best distinguishing advantage of a distinguisher. For that, one defines a notion of a distinguisher, and the pseudo-metric corresponds to the usual distinguishing advantage $d(\mathbf{R}, \mathbf{S}) = \sup_D \Delta^D(\mathbf{R}, \mathbf{S})$.

A distinguisher D is a reactive system that interacts with a resource by making queries at its interfaces, and outputs a bit. The advantage of D in distinguishing two resources $\mathbf{R}$ and $\mathbf{T}$ is defined as

$$\Delta^D(\mathbf{R}, \mathbf{S}) \coloneqq \Pr[D(\mathbf{S}) = 1] - \Pr[D(\mathbf{R}) = 1].$$

This notion of ϵ-relaxation is natural and sufficient to capture indistinguishability in the information-theoretic sense[3].

[2] That is, a function $d : \Phi \times \Phi \to \mathbb{R}_{\geq 0}$ satisfying: for any elements $\mathbf{R}, \mathbf{S}, \mathbf{T} \in \Phi$: $d(\mathbf{R}, \mathbf{R}) = 0$, $d(\mathbf{R}, \mathbf{S}) = d(\mathbf{S}, \mathbf{R})$ and $d(\mathbf{R}, \mathbf{T}) \leq d(\mathbf{R}, \mathbf{S}) + d(\mathbf{S}, \mathbf{T})$

[3] An alternative and more basic way to capture indistinguishability of random systems *without* the need to formalize a distinguisher is introduced in [LM20a], where random systems are interpreted as the equivalence class of distributions over deterministic systems, and the information-theoretic indistinguishability is captured via the statistical distance of some particular distributions.

Computational Security with Explicit Reductions

In order to capture computational indistinguishability, a commonly adopted way is to formalize a class of *efficient* distinguishers and take the viewpoint that distinguishers are polynomial-time implementable systems.

An alternative way to capture computational security is based on explicit reductions. For that, ϵ is not a number, but a function ϵ that maps distinguishers to their respective advantage in $[0, 1]$. The usual interpretation is that $\epsilon(D)$ is the advantage in the underlying computational problem of the distinguisher which is modified by the reduction. This is the approach taken in Chapter 4.

Definition 3.4.3. Let ϵ be a function that maps distinguishers to real values in $[0, 1]$. We define the ϵ-relaxation of a resource $\mathbf{R}$ as:

$$\mathbf{R}^{\epsilon} \coloneqq \{\mathbf{S} \in \Phi \mid \forall D : \Delta^{D}(\mathbf{R}, \mathbf{S}) \leq \epsilon(D)\}.$$

3.4.2 Until-Relaxation

Sometimes we want to consider guarantees that hold up to the point where a certain event happens. This is formally modeled by considering an additional so-called monotone binary output (MBO) [MPR07], which is a binary value that can switch from 0 to 1, but not back. Such an MBO can for example model that all inputs to the system are distinct (no collisions).

Definition 3.4.4. Let $\mathbf{R}$ be a resource, and let $\mathcal{E}$ be an MBO for the resource. We denote by $\mathsf{until}_{\mathcal{E}}(\mathbf{R})$ the resource that behaves like $\mathbf{R}$, but halts when $\mathcal{E} = 1$. That is, for any inputs from the point when $\mathcal{E} = 1$ (and including the input that triggered the condition), the output is $\perp$.

The until-relaxation of a system $\mathbf{R}$ consists of the set of all systems that behave equivalently up to the point where the MBO switches to 1.

Definition 3.4.5. Let $\mathbf{R}$ be a resource, and let $\mathcal{E}$ be an MBO for the resource. The $\mathcal{E}$-until-relaxation of $\mathbf{R}$, denoted $\mathbf{R}^{\mathcal{E}]}$, is the set of all systems that have the same behavior as $\mathbf{R}$ until $\mathcal{E} = 1$. That is,

$$\mathbf{R}^{\mathcal{E}]} \coloneqq \{\mathbf{S} \in \Phi \mid \mathsf{until}_{\mathcal{E}}(\mathbf{R}) = \mathsf{until}_{\mathcal{E}}(\mathbf{S})\}.$$

3.4.3 $*$-Relaxation

In many cases we want to consider the case where there are no guarantees at a specific interface j, or that a dishonest party can do something arbitrary, i.e., apply an arbitrary converter.

Definition 3.4.6. Let $\mathbf{R}$ be a resource, and let j be an interface of $\mathbf{R}$. The $*$-relaxation of $\mathbf{R}$, denoted $\mathbf{R}^{*j}$, is the set of all resources that have an arbitrary converter attached at interface j. That is,

$$\mathbf{R}^{*j} \coloneqq \{\alpha^j \mathbf{R} \mid \alpha \in \Sigma\}.$$

One can also consider a set Z of interfaces being merged to a single interface with several sub-interfaces, and applying the above relaxation to this interface. The resulting specification is denoted $\mathbf{R}^{*Z}$. This will be useful later for example to model that a set of dishonest parties collude (or, as sometimes stated in the literature, are under control of a central adversary). It is easy to see that the described $*$-relaxation is idempotent: For any resource $\mathbf{R}$ and any set of interfaces Z, we have $(\mathbf{R}^{*Z})^{*Z} = \mathbf{R}^{*Z}$.

Chapter 4

Adaptive Security in MPC

4.1 Introduction

4.1.1 Multi-Party Computation

Secure multi-party computation (MPC) allows a set of parties to securely carry out a computation. Here security informally means that parties obtain the correct output of the computation, while at the same time keeping their local inputs as private as possible. In order to meaningfully design and analyze cryptographic protocols, one must come up with appropriate definitions of security. However, this is highly non-trivial: the definition should closely capture the aspects that we care about, while at the same time being simple and usable, even minimal, avoiding as much as possible unnecessary artifacts.

There is a vast literature on security definitions in the field of MPC. Initial works [GL91, MR92, Bea91, Can00, DM00] considered the *stand-alone* setting, which examines only the protocol at hand and does not capture what it means to use the protocol in a larger context, for the task of secure function evaluation [Yao82, Yao86, GMW87]. It was not until several years later, that definitions in so-called composable frameworks for general reactive tasks were introduced [PW00, Can01, DKMR05, MR11,

MT13, KTR20, HUM13]. Such definitions aim to capture all aspects of a protocol that can be relevant, with respect to any possible application, hence the term *universal composability* [Can01].

An important aspect of security definitions for secure computation is the way in which the corrupted parties are chosen. Here, two models are commonly considered. The static security model assumes that the set of corrupted parties is fixed before the computation starts and does not change. In the more general adaptive security model, the adversary may corrupt parties during the protocol execution, based on information that has been gathered so far. Indeed, adaptive security captures important concerns regarding cryptographic protocols that static security does not capture. These include scenarios where attackers, viruses, or other adversarial entities can take advantage of the communication to decide which parties to corrupt.

The currently considered standard MPC definition for adaptive security is the one introduced by Canetti [Can01] in the UC framework. The UC-adaptive security definition follows the well-known simulation paradigm, and is formalized by comparing the execution of the protocol in the real world, to an ideal world that has the desired security properties by design. Intuitively, it guarantees that for any attack in the real world performed by an adversary that can adaptively corrupt parties, the attack can be equivalently performed in the ideal world, achieving a similar effect. This is formalized by the existence of a simulator that has to simulate the entire protocol execution, with respect to any environment where the protocol is being executed.

4.1.2 The Commitment Problem in Adaptive Security

Despite the fact that the current standard notion has been the cornerstone of adaptive security in MPC and has lead to the development of many beautiful cryptographic protocols and primitives, the definition seems to be too strong.

To show this, consider the following example: Let $\boldsymbol{\pi}$ be any protocol for secure function evaluation that is adaptively secure. Now, consider a modified protocol $\tilde{\boldsymbol{\pi}}$, where each party i first commits to its input using for example any (non-equivocable) perfectly hiding and computationally binding commitment scheme and publishes the commitment. Then, all

parties execute the protocol $\boldsymbol{\pi}$. The commitments are never again used, and in particular they are never opened. Intuitively, protocol $\tilde{\boldsymbol{\pi}}$ *should* be adaptively secure, since the commitments do not reveal any secret information (the commitments are even statistically independent of the inputs!). However, protocol $\tilde{\boldsymbol{\pi}}$ is no longer adaptively secure: we run into the so-called *commitment problem*, where the simulator is unable to consistently explain the internal state of the parties that are adaptively corrupted. This is because the simulator first has to publish a commitment on behalf of each honest party without knowing its input, and later, upon corruption, output an internal state on behalf of each party that is consistent with its input and the previously published commitment.

Common ways to address this issue include the use of non-committing encryption (see e.g. [CFGN96, CLOS02]), or the availability of secure erasable memory (see e.g. [BH93]), therefore incurring to an important efficiency loss or an extra assumption.

However, at a more general level, this raises the question of whether one could have an alternative security definition that is not subject to this issue, but still captures natural security guarantees under adaptive corruption:

Is there a natural MPC security definition that captures security guarantees under adaptive corruption and is not subject to the commitment problem?

There have been a number of works that aimed to solve this issue. A line of work [Pas03, PS04, BDH$^+$17] considers simulators that have super-polynomial running time. Such approaches come at the price of being technical or sacrificing composition guarantees. Another approach [BDHK06] disallows certain activation sequences by the environment that cannot be simulated, avoiding some of the complications of the other approaches, but sacrificing some guarantees by excluding certain attacks. A recent work [JM20] addressed this issue by proposing a notion that formalizes guarantees that hold within a certain interval, between two events, and requiring the simulation to work within each interval, without forcing the simulation to be consistent between the intervals. Although this approach seems promising, the guarantees that are given turn out to be too weak for MPC applications. In particular, the corruptions can only depend on "external" events, and not on the outputs from the given

resources.

4.1.3 Contributions

CC-Adaptive Security. Intuitively, an MPC protocol should provide, at any point during the protocol execution, security guarantees to the set of honest parties at that point. That is, for every set of parties, there is a guarantee as long as these parties are honest. This is exactly what CC-adaptive security captures: we phrase the guarantees naturally as the intersection (i.e. conjunction) of the guarantees for every set of so-far honest parties. More concretely, we require for each subset X of parties, the following: as long as parties in X are honest, the protocol does not leak anything beyond what can be inferred from parties in $\overline{X}$ (irrespective of their corruption status). Technically, there must exist a simulator with access to inputs from $\overline{X}$ that correctly simulates the protocol execution until any party in X gets corrupted. As soon as a party in X gets corrupted, the guarantee for this set is dropped. (However, guarantees for other so-far honest sets still remain.)

The corruptions are completely adaptive in the strong and usual sense, where the selection of which parties become corrupted can be done based on any information gathered during the protocol execution. The more parties are corrupted, the less guarantees remain.

Intuitively, the commitment problem does not arise because the guarantees are dropped (i.e. the simulation stops) at the point where a party in X gets corrupted. Therefore, the simulator does not need to explain the secret state of a party in X. However, note that upon corruption of any party in $\overline{X}$, the simulator must explain its internal state, and we limit this information requiring that this must be explained from the inputs of parties in $\overline{X}$. Concretely, for X being the so-far honest set, the state does not reveal anything beyond what can be inferred from the current corrupted set.

This approach is in contrast to previous adaptive security definitions, which require the existence of a single simulator that explains all possible cases.

The described guarantees are naturally phrased within the constructive cryptography (CC) [Mau11, MR11, MR16] composable framework, where each guarantee corresponds to a set specification of systems, and

the conjunction of guarantees is simply the intersection of specifications.

Comparison with Standard Static and Adaptive Security. At a technical level, we show that our new definition lies in-between the current standard UC-security definitions for static and adaptive security, respectively. Interestingly, popular examples that separate the standard static and adaptive security notions and do not exploit the commitment problem, also separate static from CC-adaptive security, therefore showing that CC-adaptive security gives very strong adaptive security guarantees. More concretely, we show the following.

Static vs CC-Adaptive Security. We first show that CC-adaptive security implies static security in all settings. Moreover, we also show that CC-adaptive security is strictly stronger than static security: for the case of passive corruption and a large number of parties, the protocol shown in [CFGN96] separates the notions of static and CC-adaptive security, and in the case of active corruption and at least three parties, the protocol shown in [CDD$^+$01] makes the separation.

Adaptive vs CC-Adaptive Security. We show that UC-adaptive security is strictly stronger than CC-adaptive security in all settings, by showing a protocol example based on the commitment problem.

Applications. We demonstrate the usefulness of our notion with two examples, showing that known protocols achieve strong adaptive security guarantees without the use of non-committing encryption.

CDN Protocol. First, we show that the protocol by Cramer, Damgard and Nielsen [CDN01] (CDN) based on threshold (additively) homomorphic encryption (THE) achieves CC-adaptive security in the honest majority setting. In the passive corruption setting, the protocol is described assuming solely the key setup for the THE scheme, while in the active corruption setting, the protocol is described assuming in addition a multiparty zero-knowledge functionality. This shows that the CDN protocol approach achieves strong adaptive security guarantees as-is, even when using an encryption scheme that commits to the plaintext.

CLOS Protocol. Second, we show that the variant of the protocol by Canetti, Lindell, Ostrovsky and Sahai [CLOS02] (CLOS) that does not use non-committing encryption, previously only proven statically secure, actually achieves CC-adaptive security in the dishonest majority setting. This is achieved by showing that the oblivious transfer from [GMW87] achieves CC-adaptivity, and the CLOS compiler transforming passive to

active protocols preserves CC-adaptivity. Note that, to the best of our knowledge, all previous UC-adaptive protocols in the dishonest majority setting required some form of non-committing or equivocal encryption.

4.1.4 Further Related Work

The problem of MPC with adaptive security was first studied by Canetti, Feige, Goldreich and Naor [CFGN96], and there is a large literature on MPC protocols with adaptive security. In the case of honest majority, it was shown that classical MPC protocols are adaptively secure [BGW88, CCD88, RB89]. Using the results in [KLR06, KMTZ13], it was shown that these protocols achieve UC adaptive security with abort in the plain model, or guaranteed output delivery in the synchronous model. A more efficient protocol was shown in [DN03], following the CDN-approach based on threshold homomorphic encryption and assuming a CRS. In the case of dishonest majority, the protocols achieve security with abort, and all known protocols assume some form of non-committing encryption or equivocation. The first work achieving adaptive security for dishonest majority was the protocol by Canetti, Lindell, Ostrovsky and Sahai [CLOS02], assuming a CRS setup. Since then, several subsequent works have improved its round and communication complexity (e.g. [DKR15, GP15, CsW19, BLPV18, CPV17]). The work by Garg and Sahai [GS12] considered adaptive security in the stand-alone model without trusted setup.

The work by Garay, Wichs and Zhou [GWZ09] considers the notion of *semi-adaptive* security for two parties, which considers guarantees for the case where one party is corrupted, and the other party is honest and can be adaptively corrupted. In contrast, our security notion imposes guarantees also when both parties are honest and can be adaptively corrupted.

4.2 Preliminaries: A Combined Relaxation

The basic concepts for the CC framework are presented in Chapter 3.

In this chapter we are interested in the relaxation that corresponds to the intuitive interpretation of "the set of all systems that behave equally until $\mathcal{E} = 1$ given that the assumption of ϵ is valid". However, it was

proven in [JM20] that the ϵ-relaxation and the until-relaxation do not generally commute, i.e., $(\mathcal{R}^{\mathcal{E}]})^{\epsilon} \not\subseteq (\mathcal{R}^{\epsilon})^{\mathcal{E}]}$ and $(\mathcal{R}^{\mathcal{E}]})^{\epsilon} \not\supseteq (\mathcal{R}^{\epsilon})^{\mathcal{E}]}$, and therefore it is not clear whether any of the two corresponds to the intuitive interpretation. Moreover, choosing one of these would partially limit the composability of such statements. That is, if one construction assumes $\mathcal{S}^{\mathcal{E}]}$ to construct $\mathcal{T}$, and another one constructs $\mathcal{S}^{\epsilon}$, then adjusting the first construction to use $\mathcal{S}^{\epsilon}$ is not trivial. Following the solution in [JM20], we consider the next combined relaxation.

Definition 4.2.1. Let $\mathbf{R}$ be a resource, $\mathcal{E}$ be an MBO, and ϵ be a function mapping distinguishers to a real value in $[0, 1]$. The $(\mathcal{E}, \epsilon)$-until-relaxation of $\mathbf{R}$, denoted $\mathbf{R}^{\mathcal{E}:\epsilon}$, is defined as follows:

$$\mathbf{R}^{\mathcal{E}:\epsilon} \coloneqq \left(\left(\mathbf{R}^{\mathcal{E}]}\right)^{\epsilon}\right)^{\mathcal{E}]}.$$

The combined relaxation benefits from the following desired properties, as shown in [JM20].

Lemma 4.2.2. *Let $\mathcal{R}$ be a specification, $\mathcal{E}_1$, $\mathcal{E}_2$ be MBOs for the resource, and ϵ_1, ϵ_2 be functions mapping distinguishers to a real value in $[0, 1]$. Then,*

$$(\mathcal{R}^{\mathcal{E}:\epsilon})^{\mathcal{E}':\epsilon'} \subseteq \mathcal{R}^{\mathcal{E}\vee\mathcal{E}':\epsilon_{\mathcal{E}\vee\mathcal{E}'}+\epsilon'_{\mathcal{E}\vee\mathcal{E}'}},$$

where $\epsilon_{\mathcal{E}\vee\mathcal{E}'}(D) = \epsilon(D \circ \mathsf{until}_{\mathcal{E}\vee\mathcal{E}'})$ is the advantage of the distinguisher interacting with the projected (by the function $\mathsf{until}_{\mathcal{E}\vee\mathcal{E}'}(\cdot)$) resource, and analogously for $\epsilon'_{\mathcal{E}\vee\mathcal{E}'}$.

Lemma 4.2.3. *Let $\mathcal{R}$ and $\mathcal{S}$ be specifications, $\mathcal{E}$ be an MBO for the resource, and ϵ be a function mapping distinguishers to a real value in $[0, 1]$. Further let α be a converter, and let i be an interface of $\mathcal{R}$. The $(\mathcal{E}, \epsilon)$-until-relaxation is compatible with converter application and with parallel composition. That is,*

1. *$\alpha^i\left(\mathcal{R}^{\mathcal{E}:\epsilon}\right) \subseteq \left(\alpha^i\mathcal{R}\right)^{\mathcal{E}:\epsilon_\alpha}$, for $\epsilon_\alpha(D) \coloneqq \epsilon(D\alpha^i)$, where $D\alpha^i$ denotes the distinguisher that first attaches α at interface i of the given resource, and then executes D.*
2. *$[\mathcal{R}^{\mathcal{E}:\epsilon}, \mathcal{S}] \subseteq [\mathcal{R}, \mathcal{S}]^{\mathcal{E}:\epsilon_{\mathcal{S}}}$, for $\epsilon_{\mathcal{S}}(D) \coloneqq \sup_{\boldsymbol{S}\in\mathcal{S}} \epsilon(D[\cdot, \boldsymbol{S}])$, where $D[\cdot, \boldsymbol{S}]$ denotes the distinguisher that emulates $\boldsymbol{S}$ in parallel to the given resource, and then executes D.*

4.3 Multi-Party Constructive Cryptography with Adaptive Corruption

We consider the setting of multi-party computation with adaptive corruption. In contrast to the static corruption model, which models a setting where parties are either honest or dishonest, and where one does not necessarily need to consider an explicit adversary, in the adaptive setting we consider scenarios where an adversary may "hack" into the parties' systems.

Multi-Party Resources. We consider resources with $n + 2$ interfaces: a set $\mathcal{P} = \{1, \ldots, n\}$ of n party interfaces, an adversary interface A and a free interface W [BMT18]. The party interfaces allow each party to have access to the resource. The adversary interface models adversarial access to the resource. The free interface allows direct access by the environment to the resource[1], and is used to model aspects that are not used by the parties, but neither controlled by the adversary.

Protocols and Basic Construction Notion. A *protocol* consists of a tuple of converters $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$, one for each party. We denote by $\boldsymbol{\pi}\mathbf{R}$ the resource where each converter π_j is attached to party interface j.

We say that a protocol $\boldsymbol{\pi}$ constructs specification $\mathcal{S}$ from specification $\mathcal{R}$, if and only if $\boldsymbol{\pi}\mathcal{R} \coloneqq \{\boldsymbol{\pi}\mathbf{R} \mid \mathbf{R} \in \mathcal{R}\}$ satisfies specification $\mathcal{S}$, i.e., $\boldsymbol{\pi}\mathcal{R} \subseteq \mathcal{S}$.

Definition 4.3.1. Let $\mathcal{R}$ and $\mathcal{S}$ be specifications, and let $\boldsymbol{\pi}$ be a protocol. We say that $\boldsymbol{\pi}$ constructs $\mathcal{S}$ from $\mathcal{R}$, if and only if $\boldsymbol{\pi}\mathcal{R} \subseteq \mathcal{S}$.

The specifications $\boldsymbol{\pi}\mathcal{R}$ and $\mathcal{S}$ are usually called the real world and the ideal world, respectively. Typical constructions in the literature describe the ideal specification $\mathcal{S}$ with the so-called simulation-paradigm. That is, by showing the existence of a *monolithic* simulator σ attached to the adversary interface of a fixed ideal resource $\mathbf{S}$. Note that this is just a particular way of defining the security guarantees in our framework. As we show in our examples, our view provides a more flexible way of defining ideal guarantees.

Protocol Converters as Resources. In order to model adaptive corruptions in Constructive Cryptography, we take the viewpoint where only

[1] This is reminiscent of the environment access to the global setup in UC [CDPW07].

trivial converters are considered [MR16]. More concretely, we consider the class Σ of trivial converters which only define a wiring between resource interfaces, and the protocol engines are then interpreted as resources. In more detail, when writing a resource $\alpha^i\mathbf{R}$ consisting of a converter α attached to interface i of resource $\mathbf{R}$, we understand the converter α as a resource, for example denoted $\tilde{\alpha}$, in parallel with $\mathbf{R}$. And we consider a trivial converter β for interface i that simply connects $\tilde{\alpha}$ and $\mathbf{R}$, i.e., we have $\alpha^i\mathbf{R} = \pi_i^i[\mathbf{R}, \tilde{\alpha}]$. We depict in Figure 4.1 this interpretation.

Figure 4.1: On the left, the converter α is connected to a resource $\mathbf{R}$. On the right, the interpretation where the converter α is interpreted as a resource $\tilde{\alpha}$ in parallel, with a trivial converter that connects interfaces.

Modeling Corruptions. Protocol converters as resources have, like any resource, an adversary interface A and a free interface W. Corruption is modeled explicitly as an input to the resource via the free interface W. Upon input corrupt at interface W, the resource adds additional capabilities at the adversary interface A.[2]

One can then model different types of corruption. In order to model passive corruption, we require that upon input corrupt at interface W, the resource makes accessible the entire local state at interface A. One can then access the local state of the resource via interface A with an input `leak`. If active corruption is considered, the adversary can in addition take over the inside and outside interfaces of the protocol engine via the adversarial interface A. That is, any inputs given at the inside or outside interface are first made available to A, who then decides what the values

[2]One could alternatively model that the input corrupt is given at the adversary interface A, with an additional mechanism to ensure that the real and ideal world corruptions are the same; for example making available the set of currently corrupted parties via the free interface W.

are.

4.4 CC-Adaptive Security

A natural way to understand the guarantees obtained from an MPC protocol with adaptive corruption, is to understand the guarantees as an intersection of separate guarantees for every set of so-far honest parties. The corruptions are completely adaptive as usual, and the identity of the chosen parties to become corrupted can be made based on information gathered during the protocol execution. The more parties are corrupted, the less sets are so-far honest, and therefore less guarantees remain.

The described guarantees can naturally be captured within the constructive cryptography framework, where each guarantee corresponds to a resource specification, and the conjunction of guarantees is simply the intersection of specifications. This is in contrast to previous security definitions, which require the existence of a single simulator that explains all possible cases.

As we will show in Section 4.4.2, our notion of CC-adaptive security lies strictly in-between the standard UC-security notions of static and adaptive security. Popular examples that separate static and adaptive security and are not based on the commitment problem, also separate static and CC-adaptive security, and examples based on the commitment problem separate our notion from UC-adaptive security; therefore showing that CC-adaptive security achieves a strong resilience against adaptive corruption, while at the same time overcoming the commitment problem.

4.4.1 Definition of the Security Notion

As described above, our security notion gives a guarantee for every set of so-far honest parties. That is, we give a guarantee for each subset $X \subseteq \mathcal{P}$ of parties, as long as the subset X is honest, irrespective of whether the other parties are honest or not. Intuitively, the guarantee provides privacy to the set of parties in X, and is described as usual, by requiring the existence of a simulator (for this set X) that correctly simulates the protocol execution. The simulator has to simulate without knowing the secret inputs and outputs of parties in X, but since this guarantee doesn't state privacy for parties not in X (recall the guarantee holds irrespective

of the honesty of other parties), we allow the simulator to know the inputs of parties that are in $\overline{X} = \mathcal{P} \setminus X$. As soon as a party in X gets corrupted, the guarantee for this set is lost (and therefore the simulation stops at this point). However, guarantees for other so-far honest sets still remain.

Moreover, we state the guarantees with respect to a (monotone[3]) *adversary structure* $\mathcal{Z} \subseteq 2^{\mathcal{P}}$, meaning that if too many corruptions happen, i.e., the set of corrupted parties exceeds the adversary structure, all guarantees are lost.

Intuitively, the commitment problem does not arise, because the guarantees are lost (i.e. the simulation stops) at the point where a party in X gets corrupted. Therefore, the simulator does not need to explain the secret state of a party in X. And for parties that are in $\overline{X}$, the simulator can consistently explain the secret state because it has access to the inputs of these parties.

Let $\mathcal{E}_X$ be the MBO indicating whether any party in X is corrupted. Moreover, let $\sigma_{\overline{X}}$ be a simulator that has access to the inputs of all parties from the set $\overline{X}$. Formally, any inputs given at interfaces from parties in $\overline{X}$, are forwarded to the adversary interface. However, we only allow the simulator to modify the inputs of actively corrupted parties.[4]

Further let $\mathcal{E}_{\mathcal{Z}}$ be the MBO that is set to 1 when the set of corrupted parties does not lie in $\mathcal{Z}$. For the common case of threshold corruption where the adversary structure contains all sets of up to t parties, we denote $\mathcal{E}_t$ the MBO that is set to 1 when more than t parties are corrupted.

We require that for each set of parties X, there must be a simulator $\sigma_{\overline{X}}$ that simulates the protocol execution until any party in X is corrupted, or the adversary structure is no longer respected, i.e., until $\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}} = 1$.

Definition 4.4.1. Protocol $\boldsymbol{\pi}$ CC-adaptively constructs specification $\mathcal{S}$ from $\mathcal{R}$ with error ϵ and adversary structure $\mathcal{Z}$, if for each set $X \subseteq \mathcal{P}$, there exists (an efficient) simulator $\sigma_{\overline{X}}$, such that $\boldsymbol{\pi}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}} : \epsilon}$. In short, $\boldsymbol{\pi}\mathcal{R}$ satisfies the following intersection of specifications:

$$\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X \subseteq \mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}} : \epsilon}$$

[3] If $Z \in \mathcal{Z}$ and $Z' \subseteq Z$, then $Z' \in \mathcal{Z}$.

[4] Allowing the simulator to modify the inputs of honest parties in $\overline{X}$ results in an unnecessarily weak notion.

Moreover, we say that $\boldsymbol{\pi}$ CC-adaptively constructs $\mathcal{S}$ from $\mathcal{R}$ with error ϵ up to t corruptions if $\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{t:\epsilon}}$.

The following lemma shows that this type of construction statement benefits from desirable composition guarantees.

Lemma 4.4.2. *Let $\mathcal{R}$, $\mathcal{S}$, $\mathcal{T}$ be specifications, and let $\boldsymbol{\pi}$, $\boldsymbol{\pi}'$ be protocols. Further let $\mathcal{Z} \subseteq 2^{\mathcal{P}}$ be a monotone set. Then, we have the following composition guarantees:*

$$\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}:\epsilon}} \;\wedge\; \boldsymbol{\pi}'\mathcal{S} \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma'_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}:\epsilon'}} \implies \boldsymbol{\pi}'\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma'_{\overline{X}}\sigma_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}:\tilde{\epsilon}}},$$

for $\tilde{\epsilon} \coloneqq \sup_{X\subseteq\mathcal{P}}\{(\epsilon_{\pi'})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}} + (\epsilon'_{\sigma_{\overline{X}}})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}}\}$, where $(\epsilon_{\pi'})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}}$ is the advantage of the distinguisher that first attaches π' to the given resource, and then interacts with the projected resource, and same for $(\epsilon'_{\sigma_{\overline{X}}})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}}$.

Furthermore, we have

$$\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}:\epsilon}} \implies \boldsymbol{\pi}[\mathcal{R},\mathcal{T}] \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma_{\overline{X}}[\mathcal{S},\mathcal{T}])^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}:\epsilon_{\mathcal{T}}}},$$

for $\epsilon_{\mathcal{T}}(D) \coloneqq \sup_{\boldsymbol{T}\in\mathcal{T}} \epsilon(D[\cdot,\boldsymbol{T}])$, where $D[\cdot,\boldsymbol{T}]$ is the distinguisher that emulates $\boldsymbol{T}$ in parallel to the given resource, and then executes D.

Proof. The proof can be found in Section A.1. □

4.4.2 Comparison to Traditional Notions of Security

In this section we show how to phrase the standard notions of static and adaptive security within our framework, and further show that our new definition lies in-between the two standard notions of static and adaptive security.

Static Security. In the standard notion of static security, the set of protocol engines that are corrupted is fixed before the computation starts and does not change. The possible corruption sets are modelled by a given

adversary structure $\mathcal{Z} \subseteq 2^{\mathcal{P}}$. Given a set $Z \in \mathcal{Z}$, we denote by $\boldsymbol{\pi}_{\overline{Z}}\mathcal{R}$ the real-world resource, where the set of protocol engines π_i, $i \in Z$, are corrupted. The security definition requires the existence of a simulator σ_Z that simulates the protocol execution and has control over the inputs and outputs from corrupted parties. As usual, in the passive case, the simulator can read these values, while in the active case, it can also change them.

Definition 4.4.3. Protocol $\boldsymbol{\pi}$ statically constructs specification $\mathcal{S}$ from $\mathcal{R}$ with error ϵ and adversary structure $\mathcal{Z}$, if for each possible set of corrupted parties $Z \in \mathcal{Z}$, there exists a simulator σ_Z such that $\boldsymbol{\pi}_{\overline{Z}}\mathcal{R} \subseteq (\sigma_Z\mathcal{S})^{\epsilon}$, where $\boldsymbol{\pi}_{\overline{Z}}$ indicates that protocol converters π_i, $i \in Z$, are corrupted, and σ_Z indicates that the simulator has control over the inputs and outputs of parties in Z.

Lemma 4.4.4. *CC-adaptive security implies static security.*

Proof. Let $\boldsymbol{\pi}$ be a protocol that constructs $\mathcal{S}$ from specification $\mathcal{R}$ with error ϵ and adversary structure $\mathcal{Z}$, with CC-adaptive security. We prove that $\boldsymbol{\pi}$ also satisfies static security with the same parameters. Fix a set $Z \in \mathcal{Z}$. Consider the particular corruption strategy, where parties in Z are corrupted at the start of the protocol execution, and no more corruptions happen.

In this case, $\mathcal{E}_{\overline{Z}} \vee \mathcal{E}_{\mathcal{Z}} = 0$, because no party in $\overline{Z}$ is corrupted, and the set of corrupted parties lies within $\mathcal{Z}$. Therefore, for the case where $X = \overline{Z}$, there must exist a simulator σ_Z (with access to the inputs and outputs of parties in Z, which are corrupted) that satisfies $\boldsymbol{\pi}_{\overline{Z}}\mathcal{R} \subseteq (\sigma_Z\mathcal{S})^{\mathcal{E}_{\overline{Z}} \vee \mathcal{E}_{\mathcal{Z}} : \epsilon} = (\sigma_Z\mathcal{S})^{\epsilon}$.

□

In the following, we show that known examples of protocols that separate the standard notions of static and adaptive security [CFGN96, CDD+01], also separate static and CC-adaptive security, both in the case of passive as well as active corruption.

Lemma 4.4.5. *For passive corruption and a large number of parties, CC-adaptive security is strictly stronger than static security.*

Proof. We consider the classical example from Canetti et al. [CFGN96]. Consider a secure function evaluation protocol with guaranteed output

delivery where parties evaluate the function that outputs $\perp$. The adversary structure contains sets of up to $t = O(n)$ parties.

The protocol $\boldsymbol{\pi}$ proceeds as follows: A designated party D secret shares its input to a randomly selected set of parties U (out of all parties except D) of small size κ parties using a κ-out-of-κ sharing scheme, where κ is the security parameter. Then, D makes the set U public (e.g. by sending the set to all parties). Subsequently, all parties output $\perp$.

It is known that $\boldsymbol{\pi}$ achieves static security. This is because an adversary not corrupting D only learns D's secret if U happens to be the predefined set of corrupted parties, which occurs with probability exponentially small in κ. More concretely, for each $Z \in \mathcal{Z}$ not containing D, the probability that $U = Z$ is $\binom{n-1}{\kappa}^{-1} = \mathtt{neg}(\kappa)$. (Note that in the case where $U \neq Z$, the simulator trivially succeeds simply by emulating the shares as random values.)

Now we show that $\boldsymbol{\pi}$ does not achieve CC-adaptive security. Consider the singleton set $X = \{D\}$, containing only the designated party. Note that U does not contain D, since D chooses a set of κ parties randomly from the set of parties without D. The adversary can then corrupt the set of parties in U to find out D's secret without corrupting D. Note that the simulator has access to all inputs and outputs from parties in $\overline{X}$, but has no access to D's input. Formally, the simulator $\sigma_{\overline{X}}$ has to output shares for parties in U that add up to D's input, without knowing the input, which is impossible. □

Lemma 4.4.6. *For active corruption, CC-adaptive security is strictly stronger than static security, as long as there are at least three parties.*

Proof. We consider the example from Canetti et al. [CDD+01] with three parties D, R_1 and R_2. D has as input two bits $b_1, b_2 \in \{0, 1\}$, and R_1, R_2 have no input. The ideal resource evaluates the function f that, on input (b_1, b_2), it outputs b_1 to R_1, b_2 to R_2 and $\perp$ to D. The adversary structure contains $\{D, R_1\}$.

The protocol $\boldsymbol{\pi}$ proceeds as follows: at step 1 D sends b_1 to R_1. After that, at step 2 D sends b_2 to R_2. Finally, at step 3 each R_i outputs the bit that they received from D and terminates, and D outputs $\perp$ and terminates.

It was proven that $\boldsymbol{\pi}$ achieves static security: for the set $Z = \{D\}$, the simulator gets the values s'_1 and s'_2 from the adversary interface, and

it sends (b'_1, b'_2) to the ideal resource, who forwards each b'_i to R_i. It is easy to see that this simulator perfectly simulates the protocol execution. The case where $Z = \{D, R_1\}$ is similar.

For the set $Z = \{R_1\}$, the simulator obtains the output b_1 from the ideal resource, so it can simply forward this bit to the adversary interface. Again, it is easy to see that the simulation is successful.

Now let us argue why the above protocol does not satisfy CC-adaptive security. To show that, consider the singleton set $X = \{R_2\}$, containing only party R_2. We will show that the adversary can break correctness of the protocol, by 1) learning the value s_1 that is sent to R_1, and 2) depending on the value received after step 1 from the so-far honest D, possibly corrupt D and modify the value that is sent to R_2 at step 2. More concretely, the adversary strategy is as follows: Initially corrupt R_1, and learn the value s_1 from D. If $s_1 = 1$, then corrupt D and choose the value $s'_2 = 0$ as the value that is sent to R_2 at step 2. With this strategy, in the real-world protocol, whenever $s_1 = 1$, R_2 never outputs 1.

Consider the case where the input to D is $(s_1, s_2) = (1, 1)$. As argued above, in the real-world R_2 outputs 0. However, since D is honest at step 1, the simulator $\sigma_{\overline{X}}$ (even with knowledge of the input of D) has no power to change the input of D. Therefore, D inputs $(1, 1)$ to the ideal resource, and therefore the output of party R_2 is 1. □

Adaptive Security. In the standard notion of UC-adaptive security, the ideal-world is described as a single specification that consists of a simulator –with passive or active capabilities (i.e. can read, or also change the inputs and outputs of corrupted parties)– attached to the adversary interface of a fixed ideal resource. Moreover, guarantees are given as long as the corrupted parties respect the adversary structure.

Definition 4.4.7. Protocol $\boldsymbol{\pi}$ UC-adaptively constructs specification $\mathcal{S}$ from $\mathcal{R}$ with error ϵ and adversary structure $\mathcal{Z}$, if there is a simulator σ, such that $\boldsymbol{\pi}\mathcal{R} \subseteq (\sigma\mathcal{S})^{\mathcal{E}_{\mathcal{Z}:\epsilon}}$.

Lemma 4.4.8. *UC-adaptive security implies CC-adaptive security.*

Proof. Let $\boldsymbol{\pi}$ be a protocol that constructs $\mathcal{S}$ from specification $\mathcal{R}$ with error ϵ and adversary structure $\mathcal{Z}$, with standard adaptive security. We prove that $\boldsymbol{\pi}$ also satisfies CC-adaptive security. For each set $X \subseteq \mathcal{P}$, we have that there exists a simulator $\sigma_{\overline{X}}$ such that $\boldsymbol{\pi}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_{\mathcal{Z}:\epsilon}}$. This

is because one can consider the simulator $\sigma_{\overline{X}}$ that ignores the inputs and outputs from parties in $\overline{X}$ and simulates according to the UC-adaptive simulator σ. Moreover, we have that

$$(\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_{\mathcal{Z}}:\epsilon} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon},$$

because $\mathcal{E}_{\mathcal{Z}} = 1$ implies $\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}} = 1$. Therefore, we have the following:

$$\boldsymbol{\pi}\mathcal{R} \subseteq (\sigma\mathcal{S})^{\mathcal{E}_{\mathcal{Z}}:\epsilon} \subseteq \bigcap_{X \subseteq \mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon}.$$

□

Lemma 4.4.9. *For passive corruption and any number of parties, CC-adaptive security does not imply UC-adaptive security.*

Proof. Consider a secure function evaluation protocol where parties evaluate the function that outputs $\perp$. The adversary structure contains sets of up to $t = 2$ parties. The protocol $\boldsymbol{\pi}$ proceeds as follows: A designated party D computes a commitment of its private input, using a (non-equivocable) perfectly hiding and computationally binding commitment scheme and makes this commitment public. Then, all parties output $\perp$.

The protocol does not achieve standard adaptive security. Consider the corruption strategy where D is corrupted "after" he sent his commitment. The simulator then first has to come up with a commitment without knowing D's input, and then, upon corruption, learns D's input and has to output randomness consistent with D's input. Since the commitment is non-equivocable, this is not possible. That is, the simulation strategy runs into the commitment problem.

It is easy to see that $\boldsymbol{\pi}$ satisfies CC-adaptive security. This is because for any set X not containing D, the simulator can read D's input, so the simulation is straightforward. On the other hand, for any set X containing D, the simulation is only required until the point in time where D becomes corrupted (without including the answer to the corruption query, i.e., there is no need to output D's private state). □

4.5 Some Ideal Resource Specifications

In this section we introduce ideal specifications for the communication network model, broadcast and MPC. We consider the setting with open

authenticated and asynchronous channels, where the adversary can choose to drop messages. As a consequence, the ideal building blocks for broadcast and MPC that we consider achieve so-called security with abort.

4.5.1 Network Model

We consider a multi-party communication network with point-to-point asynchronous authenticated channels among any pair of parties, in line with the $\mathcal{F}_{\mathsf{auth}}$ functionality in [Can01]. Asynchronous means that the channels do not guarantee delivery of the messages, and the messages are not necessarily delivered in the order which they are sent. Authenticity ensures that a recipient will only receive a message from a sender if the sender actually sent the message. In the case of adaptive corruptions, this authenticity requirement holds as long as both sender and recipient are honest. In particular, a dishonest sender is allowed to modify the messages that have been sent, as long as they have not been delivered to the recipient yet.

We first describe a single-message authenticated channel $\mathsf{AUTH}_{i,j}$ from party i to party j. A multi-message authenticated channel is then accordingly obtained via parallel composition of single-message resources. The resource has $n+2$ interfaces, n party interfaces, an adversary interface A and a free interface W.

The channel expects an input message m at interface i, which is stored upon receipt. The adversary can learn the message that is input, and can choose to deliver the message by making it available at interface j. Moreover, if party i is corrupted, it can inject a new message, as long as the message has not been delivered yet.

Resource $\mathsf{AUTH}_{i,j}$

Initialization

1: $m_i, m_j \leftarrow \perp$
2: $\texttt{CorruptSender} = 0$

Party Interfaces

1: On input $(\texttt{send}, m)$ at interface i, if $m_i = \perp$, set $m_i = m$. Output $\perp$ at interface i.
2: On input `receive` at interface j, output m_j at interface j.
3: On any input at interface $k \in [n] \setminus \{i, j\}$, output $\perp$ at the same interface.

Adversary Interface

1: On input `leak` at interface A, output m_i at interface A.
2: On input `deliver` at interface A, set $m_j = m_i$. Output $\perp$ at interface A.
3: On input $(\texttt{inject}, m)$ at interface A, if $\texttt{CorruptSender} = 1$ and $m_j \neq \perp$, set $m_i = m$.

Free Interface

1: On input $(\texttt{corrupt}, i)$ at interface W, set $\texttt{CorruptSender} = 1$. Output $\perp$ at interface W.

Let $\mathcal{N}$ be the complete network of pairwise authenticated channels, i.e., the parallel composition of $\mathsf{AUTH}_{i,j}$, for $i, j \in [n]$.

4.5.2 Broadcast with Abort

In our protocols, we will assume that parties have access to an authenticated broadcast channel. The resource has $n + 2$ interfaces, n party interfaces, an adversary interface A and a free interface W. As a first step, we model a broadcast resource BC_i where party i is the sender. We then define the broadcast specification $\mathcal{BC}$ that allows any party to broadcast as the specification containing the parallel composition of broadcast resources BC_i, for each $i \in [n]$.

Since we operate with $\mathcal{N}$ as the communication network, the broadcast specification $\mathcal{BC}$ we consider corresponds to that of *broadcast with abort* [GL02], and does not guarantee delivery of messages. It only guarantees that no two uncorrupted parties will receive two different messages.

As pointed out by Hirt and Zikas [HZ10], traditional broadcast protocols do not construct the stronger broadcast functionality which simply forwards the sender's message to all parties, since they are subject to adaptive attacks, where the adversary can corrupt an initially honest sender depending on the broadcasted message, and change the broadcasted message. Therefore, we consider the "relaxed" version, which al-

lows an adaptively corrupted sender to change the message sent, as long as the message was not delivered to any party.

In the setting with $\mathcal{N}$, such a broadcast resource can be constructed with standard adaptive security and arbitrary number of corruptions in the communication network $\mathcal{N}$ using the protocol by Goldwasser and Lindell [GL02].[5]

Resource BC_i

Initialization

1: $m^* = \perp$
2: $m_1, \ldots, m_n \leftarrow \perp$
3: $\texttt{CorruptSender} = 0$

Party Interfaces

1: On input $(\texttt{bc}, m)$ at interface i, if $m^* = \perp$, set $m^* = m$. Output $\perp$ at interface i.
2: On input `receive` at interface $j \in [n]$, output m_j at interface j.

Adversary Interface

1: On input `leak` at interface A, output m^* at interface A.
2: On input $(\texttt{deliver}, j)$, $j \in [n]$, at interface A, set $m_j = m^*$. Output $\perp$ at interface A.
3: On input $(\texttt{inject}, m)$ at interface A, if $\texttt{CorruptSender} = 1$ and $m_j = \perp$ for all $j \in [n]$, set $m^* = m$.

Free Interface

1: On input $(\texttt{corrupt}, i)$ at interface W, set $\texttt{CorruptSender} = 1$. Output $\perp$ at interface W.

4.5.3 MPC with Abort

We briefly describe an ideal resource MPC capturing secure computation with abort (and no fairness). The resource has $n + 2$ interfaces, n party interfaces, an adversary interface A and a free interface W. Via the

[5] Note that in broadcast with abort, even when the sender is honest, it is allowed that parties output $\perp$.

free interface, the resource keeps track of the set of corrupted parties. The resource allows each party i to input a value x_i, and then once all honest parties provided its input, it evaluates a function $y = f(x_1, \ldots, x_n)$ (corrupted parties can change their input as long as the output was not evaluated). The adversary can then select which parties obtain output and which not.

Resource MPC_f

Initialization

1: $x_1, \ldots, x_n \leftarrow \perp$
2: $y, y_1, \ldots, y_n \leftarrow \perp$
3: $C = \varnothing$

Party Interfaces

1: On input $(\texttt{input}, x)$ at interface $i \in [n]$, if $x_i = \perp$, set $x_i = x$. Output $\perp$ at interface i. Moreover, if $x_j \neq \perp$ for each $j \notin C$, then set $y = f(x_1, \ldots, x_n)$.
2: On input `output` at interface $i \in [n]$, output y_i at interface i.

Adversary Interface

1: On input $(\texttt{deliver}, j)$, $j \in [n]$, at interface A, set $y_j = y$. Output $\perp$ at interface A.
2: On input $(\texttt{input}, x, i)$ at interface A, if $i \in C$ and $y = \perp$, then set $x_i = x$. Output $\perp$ at interface A.
3: On input $(\texttt{leak}, j)$, $j \in C$, at interface A, output x_j at interface A.
4: On input `leakOutput`, at interface A, output y at interface A.

Free Interface

1: On input $(\texttt{corrupt}, i)$ at interface W, set $C = C \cup \{i\}$. Output $\perp$ at interface W.

4.5.4 Multi-Party Zero-Knowledge

Let R be a binary relation, consisting of pairs (x, w), where x is the statement, and w is a witness to the statement. A zero-knowledge proof allows a prover to prove to a verifier knowledge of w such that $R(x, w) = 1$.

In our protocols we will consider the *multi-party* version, which allows a designated party i to prove a statement towards all parties according to relation R. Such a resource $\mathsf{ZK}_{i,R}$ can be seen as a special instance of MPC with abort MPC_f resource, where the function f simply takes as input (x, w) from the designated party i, and no input from any other party, and outputs x in the case $R(x, w) = 1$, and otherwise $\perp$. We denote ZK_R the parallel composition of $\mathsf{ZK}_{i,R}$, for $i \in [n]$.

Such a resource can be constructed assuming $\mathcal{BC}$ and a CRS even with standard adaptive security for arbitrary many corruptions (see e.g. [CLOS02]). Alternatively, the resource can be constructed less efficiently solely from $\mathcal{BC}$ for the case where $t < n/2$ (see e.g. [RB89] with [KLR06]).

4.5.5 Oblivious Transfer

An oblivious transfer resource involves a designated sender s, with input $(x_1, \ldots, x_\ell)$, and a designated receiver with input $i \in \{1, \ldots, \ell\}$. The output for the receiver is x_i, and the sender has no output. For our purposes, we can see the resource as a special instance of MPC with abort MPC_f resource, where the function f simply takes as input $(x_1, \ldots, x_\ell)$ from the designated sender s, an input i from the designated receiver r, and no other inputs, and it outputs x_i to the receiver, and no output to any other party.

4.6 Application to the CDN Protocol

In this section we show that the protocol by Cramer, Damgard and Nielsen [CDN01] based on threshold (additively) homomorphic encryption essentially achieves MPC with abort and with CC-adaptive security, in the communication network $\mathcal{N}$ of authenticated asynchronous channels. With similar techniques, one could achieve MPC with guaranteed output delivery and with CC-adaptive security in a synchronous communication model (assuming a broadcast specification).

The CDN protocol is perhaps the iconic example that suffers from the commitment problem, and the goal of this example is to conceptually distil out at which steps the protocol is subject to relevant adaptive attacks, and conclude that the CDN-approach of broadcasting encrypted inputs in the first step and computing on ciphertexts, actually achieves

strong adaptive security guarantees, even when the encryption commits to the plaintext.

Finally, note that the protocol is typically described assuming a synchronous network, where the protocol advances in a round to round basis, and messages send at round r are assumed to arrive by round $r+1$. However, our assumed network $\mathcal{N}$ is asynchronous. To address this, we follow the standard approach of executing a synchronous protocol over an asynchronous network (see [KLR06]). The idea is simply that each party waits for all round r messages before proceeding to round $r+1$. The consequence is that the CDN protocol, which achieves full security under a synchronous network, achieves security with abort under an asynchronous network.

4.6.1 Passive Corruption Case

The protocol relies on an adaptively secure threshold homomorphic encryption scheme (see for example the scheme by Lysyanskaya and Peikert [LP01], which is based on the Paillier cryptosystem [Pai99]). In such a scheme, given the public key, any party can encrypt a message. However, decrypting the ciphertext requires the collaboration of at least $t+1$ parties, where t is a parameter of the scheme. The scheme is additively homomorphic in the sense that one can perform additions on ciphertexts without knowing the underlying plaintexts (see Section 2.2.2).

For a plaintext a, let us denote $\overline{a}$ an encryption of a. Given encryptions $\overline{a}$, $\overline{b}$, one can compute (using homomorphism) an encryption of $a+b$, which we denote $\overline{a} \boxplus \overline{b}$. Similarly, from a constant plaintext α and an encryption $\overline{a}$ one can compute an encryption of αa, which we denote $\alpha \boxdot \overline{a}$. For concreteness, let us assume that the message space of the encryption scheme is the ring $R = \mathbf{Z}_N$, for some RSA modulus N.

Let us first describe a version of the protocol for the passive case (see the section below for a complete description in the active case). The protocol $\Pi_{\texttt{pcdn}}$ starts by having each party publish encryptions of its input values. Then, parties compute addition and multiplication gates to obtain a common ciphertext, which they jointly decrypt using threshold decryption. Any linear operation (addition or multiplication by a constant) can be performed non-interactively, due to the homomorphism property of the threshold encryption scheme. Given encryptions $\overline{a}$, $\overline{b}$ of input values to a multiplication gate, parties can compute an encryption

of $c = ab$ as follows:

1. Each party i chooses a random $d_i \in \mathbf{Z}_N$ and distribute encryptions $\overline{d_i}$ and $\overline{d_i b}$ to all parties.
2. Parties compute $\overline{a} \boxplus (\boxplus_i \overline{d_i})$ and decrypt it using a threshold decryption.
3. Parties set $\overline{c} = (a + \sum_i d_i) \boxdot \overline{b} \boxminus (\boxplus_i \overline{d_i b})$.

The main problem that arises when dealing with standard adaptive security, even in the passive case, is that of the commitment problem: the simulator has to first output encryptions on behalf of the so-far honest parties during the input stage, and then if one of these honest parties is later corrupted, the simulator learns the real input of this party and must reveal its internal state to the adversary. However, the simulator is now stuck, since the real input is not consistent with the encryption output earlier. To overcome this issue, protocols usually make use of non-committing encryption schemes. An exception to this, is the protocol by Damgard and Nielsen [DN03], which is a variant of the CDN protocol that even achieves standard adaptive security, and overcomes the commitment problem by assuming a CRS which is programmed in a very clever way.

We show that this issue does not arise when aiming for CC-adaptive security. Technically, for each subset of parties $X \subseteq \mathcal{P}$, the simulator only needs to lie about the inputs of parties in X, since it knows the inputs of the other parties. Moreover, the simulation is only until the point where a party in X gets corrupted, and so we do not need to justify the internal state of this party. We propose CC-adaptive security as a natural security goal to aim for, providing strong security guarantees against adaptive corruption, and at the same time overcoming the commitment problem. Conceptually, this example shows that the CDN-approach achieves such strong adaptive security guarantees, without the need to use non-committing encryption tools or erasures. Note that in the passive case, the protocol assumes solely a setup for threshold homomorphic encryption, whereas the protocol in [DN03] requires in addition a CRS.

Key Generation. As usual, we model the setup for the threshold encryption scheme with an ideal resource KeyGen that generates its keys. The resource KeyGen simply generates the public key ek and private key

$\mathsf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$ for the threshold encryption scheme, and outputs to each party i the public key ek and its private key share dk_i, and to the adversary the public key ek.

Theorem 4.6.1. *Protocol* $\Pi_{\mathtt{pcdn}}$ *CC-adaptively constructs* MPC_f *from* $[\mathcal{N}, \mathsf{KeyGen}]$*, with error* ϵ *and up to* $t < n/2$ *passive corruptions, where* ϵ *reduces distinguishers to the corresponding advantage in the security of the threshold encryption scheme and is described in the proof.*

Proof. Let $\mathcal{R} = [\mathcal{N}, \mathsf{KeyGen}]$ and $\mathcal{S} = \{\mathsf{MPC}_f\}$. Fix a set $X \subseteq \mathcal{P}$. We need to show that there is a simulator $\sigma_{\overline{X}}$ such that $\Pi_{\mathtt{pcdn}}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon}$. At any point in time, if the event $\mathcal{E}_X \vee \mathcal{E}_t = 1$, the simulator halts. The simulator works as follows.

Key Generation. The simulator $\sigma_{\overline{X}}$ simulates this step by invoking the simulator for the threshold encryption scheme. Let ek denote the public key, and dk_i denote the decryption key share for party i. It then outputs ek at the adversary interface.

Network Messages. The simulator simulates each step of the protocol, given that all messages before that step have been delivered by the adversary i.e., the simulator receives all the corresponding `deliver` messages at the adversary interface. If not, it simply keeps waiting. The messages in the steps below are output to the adversary at the corresponding steps, upon receiving the corresponding `leak` messages at the adversary interface.

Input Stage. For each party i that is honest at this step and gave input to the ideal resource, $\sigma_{\overline{X}}$ outputs an encryption c_i on behalf of this party at the adversary interface. If $i \in \mathcal{X}$, then the simulator does not know its input, and computes the ciphertext $c_i = \mathsf{Enc}_{\mathsf{ek}}(0)$ as an encryption of 0. Otherwise, $i \notin \mathcal{X}$ and the simulator knows its input x_i, so it computes $\overline{x_i} = \mathsf{Enc}_{\mathsf{ek}}(x_i)$ as the ciphertext.

For each party i that is corrupted at this point, the simulator knows its input x_i, and forwards this input to the ideal resource.

Addition Gates. This step can be simulated in a straightforward manner, performing local homomorphic operations on behalf of each honest party.

Multiplication Gates. Let $\overline{a}$ and $\overline{b}$ denote the ciphertexts input to the multiplication gate. The simulator $\sigma_{\overline{X}}$ can execute the honest protocol.

That is, it generates a random value d_i on behalf of each honest party i, and locally computes $\overline{d_i} = \mathsf{Enc}_{\mathsf{ek}}(d_i)$, and $\overline{d_i b} = d_i \boxdot \overline{b}$. It then outputs the pair of ciphertexts to the adversary interface. For each corrupted party at this step, the simulator obtains the pair of ciphertexts $\overline{d_i}$ and $\overline{d_i b}$.

Upon receiving the pairs of ciphertexts from all parties, compute the ciphertext $\overline{a} \boxplus (\boxplus_i d_i)$, and simulate an honest threshold decryption protocol of this ciphertext. That is, the simulator outputs a decryption share of the ciphertext to the adversary interface. Upon computing $t+1$ decryption shares, reconstruct the plaintext. Let $a + \sum_i d_i$ be the reconstructed plaintext. Then, compute the ciphertext $\overline{c} = (a + \sum_i d_i) \boxdot \overline{b} \boxminus (\boxplus_i \overline{d_i b})$.

Output. The simulator inputs to the ideal resource $(\texttt{deliver}, j)$, for each party j that obtains an output in the protocol. It also obtains the output with the instruction `leakOutput`. Then, upon obtaining an output y from the ideal resource, use the simulator of the (adaptively secure) threshold decryption protocol to compute decryption shares on behalf of the honest parties (see [LP01], where one can simply choose as the inconsistent party one of the parties in X).

Corruptions. On input a command `leak`, at interface $A.i$, if i is corrupted, the simulator outputs the internal state of party i. Note that this is easily done since for parties not in X, the simulator has access to its input. And if any party in X gets corrupted, the corresponding MBO is triggered, $\mathcal{E}_X = 1$, and the simulator halts.

We now prove that $\Pi_{\mathtt{pcdn}}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_t:\epsilon}$, for the simulator $\sigma_{\overline{X}}$ described above. For that, we first describe a sequence of hybrids.

Hybrid $\mathbf{H}_1$. In this system, we assume that the simulator has access to all inputs from the parties. It then executes the real-world protocol, except that the key generation and the decryption are executed using the respective simulators for the threshold encryption scheme. By security of the threshold encryption scheme, we have that $\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}\left(\Pi_{\mathtt{pcdn}}\mathcal{R}\right)$ is ϵ_1-close to $\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}(\mathbf{H}_1)$. That is:

$$\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}\left(\Pi_{\mathtt{pcdn}}\mathcal{R}\right) \subseteq \left(\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}(\mathbf{H}_1)\right)^{\epsilon_1},$$

where ϵ_1 is the advantage of the distinguisher (modified by the reduction) to the security of the threshold encryption scheme. Moreover, by definition we have that $\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}(\mathbf{H}_1) \in \mathbf{H}_1^{\mathcal{E}_X \vee \mathcal{E}_t]}$. Therefore:

$$\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}\left(\Pi_{\mathtt{pcdn}}\mathcal{R}\right) \subseteq \left(\mathbf{H}_1^{\mathcal{E}_X \vee \mathcal{E}_t]}\right)^{\epsilon_1} \iff \Pi_{\mathtt{pcdn}}\mathcal{R} \subseteq \mathbf{H}_1^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon_1}.$$

Hybrid $\mathbf{H}_2$. The simulator in addition sets the input encryption of the honest parties in X at the Input Stage to an encryption of 0. By semantic security of the threshold encryption scheme, and following the same reasoning as above, we have that $\mathbf{H}_1 \in \mathbf{H}_2^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon_2}$, where ϵ_2 is the advantage of the distinguisher (modified by the reduction) to the semantic security of the encryption scheme. Moreover, the hybrid specification $\{\mathbf{H}_2\}$ corresponds exactly to the ideal specification $(\sigma_{\overline{X}}\mathcal{S})$.

Combining the above steps, we have that $\Pi_{\mathtt{pcdn}}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon_X}$, where $\epsilon_X = \epsilon_1 + \epsilon_2$. Therefore, choosing the function ϵ where $\epsilon(D) = \sup_{X \subseteq \mathcal{P}}\{\epsilon_X(D)\}$, the statement follows. □

4.6.2 Active Corruption Case

To handle the case of active corruption, the CDN protocol makes use of a broadcast primitive to ensure consistency of distributed messages among the parties, and also zero-knowledge proofs at the appropriate steps of the protocol. As a side remark, we note that the original CDN protocol used a multi-party zero-knowledge proof based on Σ-protocols, where the challenge is chosen by a small randomly-selected committee. This step is subject to crucial adaptive attacks, since the adversary can trivially wait until the committee is selected and corrupt all members. With this step, the protocol would not satisfy CC-adaptive security. However, we will show that the protocol, assuming an ideal multi-party zero-knowledge resource (with abort) does achieve CC-adaptive security. Note that this resource guarantees consistency on whether the designated prover succeeded in the proof, and therefore also can be used as a broadcast resource. Therefore, in the protocol we do not need to assume the broadcast specification $\mathcal{BC}$ or the network $\mathcal{N}$ (these are used to construct $\mathcal{ZK}$). As noted in Section 4.5.4, such a resource may be instantiated even with standard adaptive security from $\mathcal{N}$ for $t < n/2$, or more efficiently from bilateral zero-knowledge proofs, assuming a CRS.

Multi-Party Zero-Knowledge Relations. We state the protocol assuming ZK_R resources for appropriate relations. More concretely, we are

interested in zero-knowledge proofs for three types of relations, parameterized by a threshold encryption scheme with public key ek:

1. *Proof of Plaintext Knowledge:* The statement consists of ek, and a ciphertext c. The witness consists of a plaintext m and randomness r such that $c = \mathsf{Enc}_{\mathsf{ek}}(m; r)$.

2. *Proof of Correct Multiplication:* The statement consists of ek, and ciphertexts c_1, c_2 and c_3. The witness consists of a plaintext m_1 and randomness r_1, r_3 such that $c_1 = \mathsf{Enc}_{\mathsf{ek}}(m_1; r_1)$ and $c_3 = m_1 \cdot c_2 + \mathsf{Enc}_{\mathsf{ek}}(0; r_3)$.

3. *Proof of Correct Decryption:* The statement consists of ek, a ciphertext c, and a decryption share d. The witness consists of a decryption key share dk_i, such that $d = \mathsf{Dec}_{\mathsf{dk}_i}(c)$.

Let us denote $\mathcal{ZK}$ the specification containing the parallel composition of $\mathsf{ZK}_{\mathtt{popk}}$, $\mathsf{ZK}_{\mathtt{pocm}}$ and $\mathsf{ZK}_{\mathtt{pocd}}$.

Protocol Description. We describe the protocol engine formally below. The (sub)-interfaces of the converter are self-explanatory and are connected to the resource that the interface is naming. For example, `in.ZKpopk`, indicates the inside sub-interface of the converter that is connected to resource $\mathsf{ZK}_{\mathtt{popk}}$.

Moreover, as noted above, the assumed resources have security with abort. The protocol steps are executed sequentially, where messages from step r are computed only if all messages from step $r-1$ have been received, in line with the standard way of executing a synchronous protocol in an asynchronous network (see [KLR06]).

Protocol $\Pi_{\mathtt{cdn}}(i)$

Key Setup

1: On input the public key ek and secret key share dk_i at interface `in.keygen`, store them.

Input Distribution

1: On input x_i at interface `out`, compute $\overline{x_i} = \mathsf{Enc}_{\mathsf{ek}}(x_i; r)$ and input $(\overline{x_i}, (x_i, r))$ at interface `in.ZKpopk`.

2: On input $\overline{x_j}$ at interface `in.ZKpopk`, assign this ciphertext as the input ciphertext of party j. Otherwise, assign a default ciphertext.

Addition Gates Input: $\overline{a}, \overline{b}$. Output: $\overline{c}$.

1: Locally compute $\overline{c} = \overline{a} \boxplus \overline{b}$.

Multiplication Gates Input: $\overline{a}, \overline{b}$. Output: $\overline{c}$.

1: Sample a random plaintext d_i and compute the ciphertexts $\overline{d_i} = \mathsf{Enc}_{\mathsf{ek}}(d_i; r_1)$ and $\overline{d_i b} = d_i \boxdot \overline{b} \boxplus \mathsf{Enc}_{\mathsf{ek}}(0; r_3)$. Then, output the triple $((\overline{d_i}, \overline{b}, \overline{d_i b}), (r_1, r_3))$ at interface `in.ZKpocm`.
2: On input $(\overline{d_j}, \overline{b}, \overline{d_j b})$ at interface `in.ZKpocm`, add j to the set S. That is, S is the set of parties that succeeded in the proof.
3: Compute $\overline{a} \boxplus \left(\boxplus_{i \in S} \overline{d_i}\right)$. Then, execute the Threshold Decryption sub-protocol on this ciphertext. Let $a + \sum_{i \in S} d_i$ be the decrypted plaintext.
4: Compute $\overline{c} = \left(a + \sum_{i \in S} d_i\right) \boxdot \overline{b} \boxminus \left(\boxplus_{i \in S} \overline{d_i b}\right)$.

Output

1: Upon obtaining c', the output ciphertext of the circuit, execute the Threshold Decryption sub-protocol on c'.

Threshold Decryption Input: ciphertext c. Output: y.

1: Compute a decryption share $s_i = \mathsf{DecShare}_{\mathsf{dk}_i}(c)$. Then, input $((\mathsf{ek}, c, s_i), \mathsf{dk}_i)$ at interface `in.ZKpocd`.
2: Upon receiving (ek, c, s_j) from $t+1$ different parties at interface `in.ZKpocd`, compute $y = \mathsf{Rec}(\{s_j\})$.

The following theorem states that the protocol $\Pi_{\mathtt{cdn}}$ achieves CC-adaptive security in the model assuming a threshold encryption setup and multi-party zero-knowledge resource. The main difference in the proof with respect to the passive protocol, is that the simulator extracts the inputs from corrupted parties from the inputs to the zero-knowledge resource, and also checks that the values received from the adversary interface satisfy the appropriate zero-knowledge relations. Details can be found in Section A.2.

Theorem 4.6.2. *Protocol $\Pi_{\mathtt{cdn}}$ CC-adaptively constructs MPC_f from $[\mathcal{ZK}, \mathsf{KeyGen}]$, with error ϵ and up to $t < n/2$ active corruptions, where ϵ reduces distinguishers to the corresponding advantage in the security of the threshold encryption scheme and is described in the proof.*

4.7 Application to the CLOS Protocol

In this section, we show another application of our new definition, with the iconic CLOS protocol [CLOS02], which is based on the classical GMW protocol [GMW87].

We show that the CLOS protocol can be used to achieve a CC-adaptively secure protocol for arbitrary number of active corruptions, assuming a CRS resource, and the existence of enhanced trapdoor permutations. Note that, since CC-adaptivity implies static security, and some form of setup is required even for static security, then it is impossible to achieve CC-adaptivity in the plain model (where only the network is assumed) for the dishonest majority setting. However, note that in contrast to the UC-adaptive version of the CLOS protocol, the construction does not require the use of augmented non-committing encryption. In fact, to the best of our knowledge, all UC-adaptively secure protocols in the dishonest majority setting require some form of non-committing encryption.

Theorem 4.7.1. *Assuming enhanced trapdoor permutations, there exists a non-trivial*[6] *protocol that CC-adaptively constructs* MPC_f *from* $[\mathcal{N}, \mathsf{CRS}]$*, for appropriate error* ϵ *(as defined by the steps below) and for any number of active corruptions.*

We only sketch the proof of the above theorem. We follow the steps of the CLOS protocol. First, a construction of a passively secure protocol assuming the asynchronous communication network $\mathcal{N}$ is shown. This construction is achieved by first constructing an ideal oblivious transfer (OT), and then designing a secure computation protocol assuming an ideal OT. The following lemma shows that the protocol $\Pi_{\mathtt{ot}}$ of [GMW87] achieves CC-adaptive security. We describe the protocol and the proof of the following lemma in Section A.3.

Lemma 4.7.2. *Assume that enhanced trapdoor permutations exist. Then,* $\Pi_{\mathtt{ot}}$ *CC-adaptively constructs* OT *from* $\mathcal{N}$*, for error* ϵ *(described in the*

[6] The ideal specification does not require any of the simulators to deliver the messages to the parties. This implies that a protocol that "hangs" (i.e., never sends any messages and never generates output) securely realizes any ideal resource, which is uninteresting. Following [CLOS02], we therefore let a non-trivial protocol be one for which all parties generate output if the adversary delivers all messages and all parties are honest.

proof), and for any number of passive corruptions.

Given that UC-adaptive security implies CC-adaptive security by Lemma 4.4.8, and there is a UC-adaptively secure MPC protocol assuming an ideal OT resource [CLOS02], we have the following lemma:

Lemma 4.7.3. *There exists a non-trivial protocol that CC-adaptively constructs* MPC_f *from* $[\mathcal{N}, \mathsf{OT}]$*, with no error, and for any number of passive corruptions.*

As a corollary of the above two lemmas and the composition guarantees from Lemma 4.4.2, we have:

Corollary 4.7.4. *Assume that enhanced trapdoor permutations exist. There exists a non-trivial protocol that CC-adaptively constructs* MPC_f *from* $\mathcal{N}$*, for error* ϵ *(defined by the composition Lemma 4.4.2 and error from Lemma 4.7.2), and for any number of passive corruptions.*

Second, we use the CLOS compiler that transforms any passively secure protocol operating in the network $\mathcal{N}$, to an actively secure protocol assuming in addition an ideal *commit-and-prove* CP resource (see Section A.4 for a description). One can see that the compiler preserves the adaptivity type in the sense that if the passive protocol is CC-adaptively secure, the compiled protocol is CC-adaptively secure.

Corollary 4.7.5. *Let* Π *be a multi-party protocol and let* Π' *be the protocol obtained by applying the CLOS compiler. Then, the following holds: if* Π *CC-adaptively constructs* MPC_f *from* $\mathcal{N}$ *for error* ϵ *and any number of passive corruptions, then* Π' *CC-adaptively constructs* MPC_f *from* $[\mathcal{N}, \mathsf{CP}]$ *for error* ϵ' *defined in the proof and any number of active corruptions.*

Sketch. The proof in CLOS (Proposition 9.6) shows that a malicious adversary cannot cheat in the compiled protocol because the resource CP checks the validity of each input received. In particular, they show that for any adversary interacting in the compiled protocol Π', there is a passive adversary interacting in protocol Π that simulates the same view.

Concretely, the proof shows that the specification $\Pi\tau\mathcal{N}$ and $\Pi'[\mathcal{N}, \mathsf{CP}]$ are the same, where τ is the translation converter attached at the adversary interface, which translates from the adversary in Π' to the adversary

in Π. (Typically the translation is called a simulator, and it happens between a real resource and an ideal resource. Here, the translation is between two real resources.)

Fix a set $X \subseteq \mathcal{P}$. Since Π CC-adaptively constructs MPC_f from $\mathcal{N}$ for error ϵ and any number of passive corruptions, we have that $\Pi\mathcal{N} \subseteq (\sigma_{\overline{X}}\mathsf{MPC}_f)^{\mathcal{E}_{X:\epsilon}}$.

Using Lemma 4.2.3, this implies the desired result:

$$\Pi'[\mathcal{N}, \mathsf{CP}] = \Pi\tau\mathcal{N} \subseteq \tau(\sigma_{\overline{X}}\mathsf{MPC}_f)^{\mathcal{E}_{X:\epsilon}} \subseteq (\tau\sigma_{\overline{X}}\mathsf{MPC}_f)^{\mathcal{E}_{X:\epsilon'}} := (\sigma'_{\overline{X}}\mathsf{MPC}_f)^{\mathcal{E}_{X:\epsilon'}},$$

where $\epsilon' = \epsilon_\tau$. □

It was also shown in [CLOS02] that CP can be constructed with UC-adaptive security assuming a zero-knowledge resource ZK and broadcast BC. Given that ZK can be constructed assuming a resource CRS and broadcast BC, and BC can be constructed from $\mathcal{N}$, the authors conclude that CP can be constructed from CRS and $\mathcal{N}$. Therefore, since UC-adaptive security implies CC-adaptive security, Lemma 4.4.8 shows:

Corollary 4.7.6. *There exists a non-trivial protocol that CC-adaptively constructs* CP *from* $[\mathcal{N}, \mathsf{CRS}]$*, for error* ϵ *(as in [CLOS02]), and for any number of active corruptions.*

From Corollaries 4.7.4, 4.7.5 and 4.7.6, and Lemma 4.4.2 we achieve the theorem statement.

Appendix A

Details of Chapter 4

A.1 Proof of Lemma 4.4.2

The proof of the lemma follows from the properties of the ϵ-relaxation and the until-relaxation, and is in line with the composition theorem for interval-wise relaxations in [JM20].

We start from the first property, which shows sequential composition. That is, if $\boldsymbol{\pi}$ constructs $\mathcal{S}$ from $\mathcal{R}$ with error ϵ, and $\boldsymbol{\pi'}$ constructs $\mathcal{T}$ from $\mathcal{S}$ with error ϵ', then one can construct $\mathcal{T}$ from $\mathcal{R}$ with a new error $\tilde{\epsilon}$, corresponding essentially to the sum of ϵ and ϵ'.

$$\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X \subseteq \mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z} : \epsilon} \;\wedge\; \boldsymbol{\pi'}\mathcal{S} \subseteq \bigcap_{X \subseteq \mathcal{P}} (\sigma'_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z} : \epsilon'}$$
$$\implies \boldsymbol{\pi'}\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X \subseteq \mathcal{P}} (\sigma'_{\overline{X}}\sigma_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z} : \tilde{\epsilon}},$$

for $\tilde{\epsilon} \coloneqq \sup_{X \subseteq \mathcal{P}} \{(\epsilon_{\pi'})_{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z}} + (\epsilon'_{\sigma_{\overline{X}}})_{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z}}\}$, where $(\epsilon_{\pi'})_{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z}}$ is the advantage of the distinguisher that first attaches π' to the given resource, and then interacts with the projected resource, and analogously for $(\epsilon'_{\sigma_{\overline{X}}})_{\mathcal{E}_X \vee \mathcal{E}_\mathcal{Z}}$.

Let $X \subseteq \mathcal{P}$ be a set. From the first part, Lemma 4.2.3 and composition order invariance, we have:

$$\boldsymbol{\pi'}\boldsymbol{\pi}\mathcal{R} \subseteq \boldsymbol{\pi'}\left((\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon}\right) \subseteq \left((\boldsymbol{\pi'}\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon_{\boldsymbol{\pi'}}}\right) \subseteq \left((\sigma_{\overline{X}}\boldsymbol{\pi'}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon_{\boldsymbol{\pi'}}}\right).$$

Moreover, from the second part we have:

$$(\sigma_{\overline{X}}\boldsymbol{\pi'}\mathcal{S}) \subseteq \sigma_{\overline{X}}\left((\sigma'_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon'}\right) \subseteq (\sigma_{\overline{X}}\sigma'_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon'_{\sigma_{\overline{X}}}}.$$

Combining both statements and using Lemma 4.2.2 yields:

$$\boldsymbol{\pi'}\boldsymbol{\pi}\mathcal{R} \subseteq \left((\sigma_{\overline{X}}\sigma'_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon'_{\sigma_{\overline{X}}}}\right)^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon_{\boldsymbol{\pi'}}} \subseteq (\sigma_{\overline{X}}\sigma'_{\overline{X}}\mathcal{T})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\tilde{\epsilon}}.$$

The second property ensures that the construction notion achieves parallel composition. That is, if $\boldsymbol{\pi}$ constructs $\mathcal{S}$ from $\mathcal{R}$, then it also constructs $[\mathcal{S},\mathcal{T}]$ from $[\mathcal{R},\mathcal{T}]$.

$$\boldsymbol{\pi}\mathcal{R} \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon} \implies \boldsymbol{\pi}[\mathcal{R},\mathcal{T}] \subseteq \bigcap_{X\subseteq\mathcal{P}} (\sigma_{\overline{X}}[\mathcal{S},\mathcal{T}])^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon_{\mathcal{T}}},$$

for $\epsilon_{\mathcal{T}}(D) \coloneqq \sup_{\mathbf{T}\in\mathcal{T}} \epsilon(D[\cdot,\mathbf{T}])$, where $D[\cdot,\mathbf{T}]$ denotes the distinguisher that emulates $\mathbf{T}$ in parallel to the given resource, and then executes D.

Let $X \subseteq \mathcal{P}$ be a set. From the composition order invariance property and Lemma 4.2.3, we have:

$$\boldsymbol{\pi}[\mathcal{R},\mathcal{T}] = [\boldsymbol{\pi}\mathcal{R},\mathcal{T}] \subseteq [(\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon},\mathcal{T}] \subseteq [\sigma_{\overline{X}}\mathcal{S},\mathcal{T}]^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon_{\mathcal{T}}} = \left(\sigma_{\overline{X}}[\mathcal{S},\mathcal{T}]\right)^{\mathcal{E}_X \vee \mathcal{E}_{\mathcal{Z}}:\epsilon_{\mathcal{T}}}.$$

A.2 Proof of Theorem 4.6.2

Let $\mathcal{R} = [\mathcal{ZK}, \mathsf{Keygen}]$ and $\mathcal{S} = \{\mathsf{MPC}_f\}$. Fix a set $X \subseteq \mathcal{P}$. We need to show that there is a simulator $\sigma_{\overline{X}}$ such that $\Pi_{\mathtt{cdn}}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_t:\epsilon}$. At any point in time, if the event $\mathcal{E}_X \vee \mathcal{E}_t = 1$, the simulator halts. The simulator works as follows.

Key Generation. The simulator $\sigma_{\overline{X}}$ simulates this step by invoking the simulator for the threshold encryption scheme. Let ek denote the public

key, and dk_i denote the decryption key share for party i. It then outputs ek at the adversary interface.

Network Messages. The simulator simulates each step of the protocol, given that all messages before that step have been delivered by the adversary i.e., the simulator receives all the corresponding `deliver` messages at the adversary interface. If not, it simply keeps waiting. The messages below in the steps below are output to the adversary at the corresponding steps, upon receiving the corresponding `leak` messages at the adversary interface.

Input Stage. For each party i that is honest at this step and gave input to the ideal resource, $\sigma_{\overline{X}}$ outputs an encryption c_i on behalf of this party at the adversary interface. If $i \in \mathcal{X}$, then the simulator does not know its input, and computes the ciphertext $c_i = \mathsf{Enc}_{\mathsf{ek}}(0)$ as an encryption of 0. Otherwise, $i \notin \mathcal{X}$ and the simulator knows its input x_i, so it computes $\overline{x_i} = \mathsf{Enc}_{\mathsf{ek}}(x_i)$ as the ciphertext. It then outputs the ciphertext c_i, indicating that the zero-knowledge proof was successful.

For each party i that is corrupted at this point, the simulator obtains (c, w) and checks that the witness w, consisting of a plaintext x and randomness r, satisfy the proof of plaintext knowledge relation, i.e., that $c = \mathsf{Enc}_{\mathsf{ek}}(x; r)$. It this holds, forward x to the ideal resource.

Addition Gates. This step can be simulated in a straightforward manner, performing local homomorphic operations on behalf of each honest party.

Multiplication Gates. Let $\overline{a}$ and $\overline{b}$ denote the ciphertexts input to the multiplication gate. The simulator $\sigma_{\overline{X}}$ can execute the honest protocol. That is, it generates a random value d_i on behalf of each honest party i, and locally computes $\overline{d_i} = \mathsf{Enc}_{\mathsf{ek}}(d_i)$, and $\overline{d_i b} = d_i \boxdot \overline{b}$. It then outputs the pair of ciphertexts to the adversary interface, indicating that the proof of correct multiplication is valid. For each corrupted party at this step, the simulator obtains (as input of the zero-knowledge proof of correct multiplication) the statement containing the ciphertexts c_1^i, $c_2 \coloneqq \overline{b}$ and c_3^i as the statement, and as witness the plaintexts d_i, and randomness r_1 and r_3. The simulator checks that $c_1^i = \mathsf{Enc}_{\mathsf{ek}}(d_i; r_1)$, and $c_3^i = d_i \boxdot c_2 + \mathsf{Enc}_{\mathsf{ek}}(0; r_3)$. If this holds, accept the pair of ciphertexts from party i.

Upon receiving the pairs of ciphertexts from all parties, let S be the set of accepted parties. Then, compute the ciphertext $\overline{a} \boxplus (\boxplus_{i \in S} c_1^i)$, and

simulate an honest threshold decryption protocol on this ciphertext. Let $a+\sum_{i\in S} d_i$ be the reconstructed plaintext. Then, compute the ciphertext $\overline{c} = (a + \sum_{i\in S} d_i) \boxdot c_2 \boxminus (\boxplus_{i\in S} c_3^i)$.

Output. The simulator inputs to the ideal resource $(\texttt{deliver}, j)$, for each party j that obtains an output in the protocol. It also obtains the output with the instruction `leakOutput`. Then, upon obtaining an output y from the ideal resource, use the simulator of the (adaptively secure) threshold decryption protocol to compute decryption shares on behalf of the honest parties (see [LP01], where one can simply choose as the inconsistent party one of the parties in X).

Corruptions. On input a command `leak`, at interface $A.i$, if i is corrupted, the simulator outputs the internal state of party i. Note that this is easily done since for parties not in X, the simulator has access to its input. And if any party in X gets corrupted, the corresponding MBO is triggered, $\mathcal{E}_X = 1$, and the simulator halts.

We prove that $\Pi_{\texttt{cdn}}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon}$ via a sequence of hybrids.

Hybrid $\mathbf{H}_1$. Here, we assume that the simulator has access to all inputs from the parties. It then executes the real-world protocol, except that in the zero-knowledge proofs, when the simulator has to output a proof on behalf of an honest party it simply outputs a valid response without checking the witness from the honest party. It is trivial to see that the real-world specification is the same as $\mathbf{H}_1$, since honest parties always send a valid witness to $\mathcal{ZK}$.

Hybrid $\mathbf{H}_2$. We modify the above hybrid to also change the key generation and the decryption, which are now executed using the respective simulators for the threshold encryption scheme. By security of the threshold encryption scheme, we have that $\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}(\mathbf{H}_1)$ is ϵ_1-close to $\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}(\mathbf{H}_2)$. With the same argument as in the passive case, we have that:

$$\mathsf{until}_{\mathcal{E}_X \vee \mathcal{E}_t}(\mathbf{H}_1) \in \left(\mathbf{H}_2^{\mathcal{E}_X \vee \mathcal{E}_t]}\right)^{\epsilon_1} \iff H_1 \in \mathbf{H}_2^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon_1}.$$

Hybrid $\mathbf{H}_3$. This hybrid is the same as above, except that the simulator sets the input encryption of the honest parties in X during the Input Stage as an encryption of 0. By semantic security of the threshold encryption scheme, we have that $\mathbf{H}_2 \in \mathbf{H}_3^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon_2}$, where ϵ_2 is the advantage

of the distinguisher (modified by the reduction) to the semantic security of the encryption scheme. Moreover, the hybrid $\mathbf{H}_2$ corresponds exactly to the ideal specification $(\sigma_{\overline{X}}\mathcal{S})$.

Combining the above steps, we have that $\Pi_{\mathtt{cdn}}\mathcal{R} \subseteq (\sigma_{\overline{X}}\mathcal{S})^{\mathcal{E}_X \vee \mathcal{E}_t : \epsilon_X}$, where $\epsilon_X = \epsilon_1 + \epsilon_2$. Therefore, choosing the function ϵ where $\epsilon(D) = \sup_{X \subseteq \mathcal{P}}\{\epsilon_X(D)\}$ concludes the proof.

A.3 Protocol and Proof of Lemma 4.11.1

The oblivious transfer protocol $\Pi_{\mathtt{ot}}$ from [GMW87] is presented below. We describe the protocol in the n-party setting, where two of the parties, the sender s and the receiver r exchange the usual messages, and other parties do not perform any instructions.

Protocol $\Pi_{\mathtt{ot}}$

Converter for Sender s

1: On input $(x_1 \ldots, x_\ell)$ at `out`, choose a trapdoor permutation f over $\{0,1\}^\kappa$, and its inverse f^{-1}. Then, output f to `in.net.`r.
2: On input $(y_1 \ldots, y_\ell)$ at `in.net.`r, output $(b_1, \ldots, b_\ell) := (x_1 \oplus B(f^{-1}(y_1)), \ldots, x_\ell \oplus B(f^{-1}(y_\ell)))$ to `in.net.`r, where $B(\cdot)$ is a hard-core predicate for f.

Converter for Receiver r

Set $f' = \bot$.
1: On input i at interface `out`, if $f' \neq \bot$, choose $y_1, \ldots, y_{i-1}, r, y_{i+1}, \ldots, y_\ell \in_R \{0,1\}^\kappa$, and compute $y_i = f(r)$, and output $(y_1, \ldots, y_\ell)$ at interface `in.net.`s.
2: On input f at interface `in.net.`s, set $f' = f$.
3: On input $(b_1, \ldots, b_\ell)$, output $b_i \oplus B(r)$ at `out`.

We prove that the protocol achieves CC-adaptive security against any number of passive corruptions.

Lemma A.3.1. *Assume that enhanced trapdoor permutations exist. Then, $\Pi_{\mathtt{ot}}$ CC-adaptively constructs OT from $\mathcal{N}$, for error ϵ (described in the proof), and for any number of passive corruptions.*

Proof. Fix a set $X \subseteq \mathcal{P}$. We divide four cases, depending on whether

the sender s or the receiver r are in the set X, and show that $\Pi_{\mathsf{ot}}\mathcal{N} \subseteq (\sigma_{\overline{X}}\mathsf{OT})^{\mathcal{E}_X:\epsilon}$. In all cases, the simulator halts at the point where a party in X is corrupted.

Case 1: $s \notin X$ and $r \notin X$. This case is trivial, since the simulator knows the inputs to both parties, and can therefore locally simulate all steps.

Case 2: $r \in X$. $\sigma_{\overline{X}}$ generates f, f^{-1} as in the protocol, outputs f at the adversary interface A. It then generates $y_1, \ldots, y_{i-1}, y_i, y_{i+1}, y_\ell$ randomly, where $y_j \in_R \{0,1^\kappa\}$. Output $y_1, \ldots, y_\ell$ at interface A. Finally, compute each bit $b_i = x_i \oplus B(f^{-1}(y_i))$, $i \in [1, \ell]$. Output $b_1, \ldots, b_\ell$ at interface A, input $x_1, \ldots, x_\ell$ to OT and deliver the output to the receiver.

Corruptions. At any point in time, on input `leak` from A output the sender's secret state.

Simulation is perfect in this case. Since f is a permutation, choosing z at random and computing $y_i = f(r)$, as occurs in the real protocol, gives a uniform random y_i. Moreover, the equality $b_i = x_i \oplus B(f^{-1}(y_i))$ is satisfied. Therefore, both real and ideal systems behave the same until the event $\mathcal{E}_X$ triggers. This means that $\Pi_{\mathsf{ot}}\mathcal{N} \subseteq (\sigma_{\overline{X}}\mathsf{OT})^{\mathcal{E}_X:0}$.

Case 3: $s \in X$. $\sigma_{\overline{X}}$ generates f, f^{-1} as in the protocol, outputs f at the adversary interface A. It then generates $y_1, \ldots, y_{i-1}, r, y_{i+1}, y_\ell$ randomly, where $y_j \in_R \{0,1^\kappa\}$, and $y_i = f(r)$. Output $y_1, \ldots, y_\ell$ at interface A. Finally, input i at OT, and obtain x_i. Then, compute $b_1, \ldots, b_{i-1}, b_{i+1}, b_\ell$ as uniform bits, and set $b_i = x_i \oplus B(f^{-1}(y_i))$. Output $b_1, \ldots, b_\ell$ at interface A.

Corruptions. On input `leak` from A before Step 2, if the receiver r is corrupted, output the input i. If the corruption is after Step 2, output $y_1, \ldots, y_\ell$.

The only difference between the real and ideal world, is in the third message $b_1, \ldots, b_\ell$. The bit b_i is identical in both worlds, and is set to $x_i \oplus B(f^{-1}(y_i))$. For the other bits b_j, $j \neq i$, the simulation chooses them uniformly, while the protocol chooses them as $x_j \oplus B(f^{-1}(y_j))$. However, since $B(\cdot)$ is a hard-core predicate and y_j is uniform random, these are distinguishable up to the hard-core property.

Therefore, we have that until $\mathcal{E}_X = 1$, the real system is the same as the ideal system, up to the security of the hard-core predicate. That is,

$\Pi_{\mathsf{ot}}\mathcal{N} \subseteq (\sigma_{\overline{X}}\mathsf{OT})^{\mathcal{E}_X:\epsilon_{hc}}$, where ϵ_{hc} is the advantage for the distinguisher modified by the reduction in distinguishing the hard-core bit from uniform.

Case 4: $s \in X$ and $r \in X$. In this case, the simulator $\sigma_{\overline{X}}$ simply generates f, f^{-1} as in the protocol, outputs f at the adversary interface A. It then generates $y_1, \ldots, y_\ell$, where $y_j \in_R \{0,1\}^\kappa$ and outputs this to interface A, and also sets the second messages $b_1, \ldots, b_\ell$ where $b_j \in_R \{0,1\}$ and outputs this to interface A.

This case can be argued similarly as the previous case. □

A.4 Commit-and-Prove Resource

The commit-and-prove resource [CLOS02] is a generalization of the commitment resource. It is parameterized by a relation R and a designated party i, the committer. It consists of two phases. In the first phase, party i can commit to a value w, and all parties receive a "committed" message. In the second phase, instead of opening the value, the resource receives some value x, and checks whether $R(x, w) = 1$. If so, it outputs x to all parties, and otherwise it ignores the input. In fact, the resource allows the committer to commit to multiple values, and the relation can depend on all these values.

We denote the resource CP_R the parallel composition of resources $\mathsf{CP}_{i,R}$ for each designated party $i \in [n]$, and omit writing the relation when it is clear from the context.

Resource $\mathsf{CP}_{i,R}$

Local Variable

$\overline{w}$ is initially an empty list.
$\mathcal{Q} = \varnothing$.

Commit Phase

1: On input $(\texttt{commit}, id, w)$ at interface i, append w to $\overline{w}$ and output $\perp$ at interface i.

2: On input id at interface $j \neq i$, if a value with this id was committed, make the `receipt` available to the adversary, who then can choose to deliver the message at interface j.
3: On input id at the adversary interface, if a value with this id was committed, output `receipt` at interface j.

Prove Phase

1: On input $(\texttt{prove}, id, x)$ at interface i, if $R(x, \overline{w}) = 1$, add (id, x) to $\mathcal{Q}$.
2: On input id at interface $j \neq i$ if a pair (id, x) was stored, make x available to the adversary, who then can choose to deliver the message at interface j.
3: On input id at the adversary interface, if a pair (id, x) was stored, output x at the same interface.

Chapter 5

Synchronous Constructive Cryptography

5.1 Introduction

5.1.1 Composable Security

One can distinguish two different types of security statements about multi-party protocols. *Stand-alone security* considers only the protocol at hand and does not capture (at least not explicitly) what it means to use the protocol in a larger context. This can cause major problems. For example, if one intuitively understands an r-round broadcast protocol as implementing a functionality where the sender inputs a value and r rounds later everybody learns this value, then one missed the point that a dishonest party learns the value already in the first round. Therefore a naive randomness generation protocol, in which each party broadcasts (using a broadcast protocol) a random string and then all parties compute the XOR of all the strings, is insecure even though naively it may look secure [HZ10]. There are also more surprising and involved examples of failures when using stand-alone secure protocols in larger contexts.

The goal of *composable security* frameworks is to capture all aspects

of a protocol that can be relevant in any possible application; hence the term *universal composability* [Can01]. While composable security is more difficult to achieve than some form of stand-alone security, one can argue that it is ultimately necessary. Indeed, one can sometimes reinterpret stand-alone results in a composable framework. There exist several frameworks for defining and reasoning about composable security (e.g. [PW00, Can01, DKMR05, MR11, MT13, KTR20, HUM13]).

5.1.2 Composable Synchronous Models

One can classify results on distributed protocols according to the underlying interaction model. Synchronous models, where parties are synchronized and proceed in rounds, were first considered in the literature because they are relatively simple in terms of the design and analysis of protocols. Asynchronous models are closer to the physical reality, but designing them and proving their security is significantly more involved, and the achievable results (e.g. the fraction of tolerable dishonest parties) are significantly weaker than for a synchronous model. However, synchronous models are nevertheless justified in settings where one can assume a maximal latency of all communication channels as well as sufficiently well-synchronized clocks.

Most composable treatments of synchronous protocols are in (versions of) the UC framework by Canetti [Can01], which is an inherently asynchronous model. The models presented in [Can01, Nie03, HM04, KMTZ13] propose different approaches to model synchronous communication on top of the UC framework [Can01]. These approaches inherit the complexity of the UC framework designed to capture full asynchrony. Another approach was introduced with the Timing Model [DNS98, Gol02, KLP05]. This model integrates a notion of time in an intuitive manner, but as noted in [KMTZ13] fails to exactly capture the guarantees expected from a synchronous network. A similar approach was proposed in [BHMU05], which modifies the asynchronous reactive-simulatability framework [BPW07] by adding an explicit time port to each automaton.

Despite the large number of synchronous composable frameworks, the overhead created when using them is still too large. For example, when using a model built on top of UC, one typically needs to consider clock/synchronization functionalities, activation tokens, message scheduling, etc. Researchers wish to make composable statements, but using

these models often turn out to be a burden and create huge overhead. As a consequence, papers written in synchronous UC models tend to be rather informal: the descriptions of the functionalities are incomplete, clock functionalities are missing, protocols are underspecified and the proofs are often made at an intuitive level. This leaves the question:

Can one design a composable framework targeted to minimally capture synchronous protocols?

People have considered capturing composable frameworks for restricted settings (e.g. [Wik16, CCL15]), but to the best of our knowledge, there is no composable framework that is targeted to minimally capture any form of synchronous setting.

5.1.3 Multi-Party Computation

In the literature on secure multi-party computation (MPC) protocols, of which secure function evaluation (SFE) is a special case, most of the results are for the synchronous model as well as stand-alone security, even though intuitively most protocols seem to provide composable security. To the best of our knowledge, the first paper proving the composable security of a classical SFE protocol is [CLOS02], where the security of the seminal GMW-protocol [GMW87] is proved. The protocol assumes trusted setup, and security is obtained in the UC framework. In [AL17], the security of the seminal BGW-protocol [BGW88] is proved in the plain model. With the results in [KLR06, KMTZ13], one can prove security in the UC framework.

5.1.4 Contributions

A guiding principle is to strive for minimality and to avoid unnecessary artefacts, thus lowering the entrance fee for getting into the field of composable security and also bringing the reasoning about composable security for synchronous protocols closer to being tractable by formal methods.

The contributions in this chapter are two-fold. First, we introduce a new composable framework to capture settings where parties have synchronized clocks (in particular, traditional synchronous protocols), and

illustrate the framework with a few simple examples. Our focus is on the meaningful class of information-theoretic security as well as static corruption.

As a second contribution, we prove the composable security of Maurer's simple-MPC protocol [Mau06] and demonstrate that it perfectly constructs a versatile computer resource which can be (re-)programmed during the execution. Compared to [CLOS02, AL17], our treatment is significantly simpler for two reasons. First, the protocol of [Mau06] is simpler than the BGW-protocol. Second, and more importantly, the simplicity of our framework allows to prove security of the protocols without the overhead of asynchronous models: we do not deal with activation tokens, message scheduling, running time, etc.

Synchronous Constructive Cryptography. Our framework is an instantiation of the Constructive Cryptography framework [Mau11, MR11, MR16], for specific instantiations of the resource and converter concepts. Moreover, we introduce a new type of construction notion, parameterized by the set Z of potentially dishonest parties, allowing to capture the guarantees for every such dishonest set Z. An often considered special case is that nothing is guaranteed if Z contains too many parties.

Synchronous resources are very simple: They are (random) systems where the alphabet is list-valued. That is, a system takes a complete input list and produces a complete output list. Parallel composition of resources is naturally defined. There is no need to talk about a scheduler or activation patterns.

To allow that dishonest parties can potentially make their inputs depend on some side information of the round, we let one round r of the protocol correspond to two rounds, $r.a$ and $r.b$ (called semi-rounds). Honest parties provide the round input in semi-round $r.a$ and the dishonest parties receive some information already in the same semi-round $r.a$. In semi-round $r.b$, the dishonest parties give their inputs and everybody receives the round's output.[1]

The framework is aimed at being minimal and differs from other frameworks in several ways. One aspect is that the synchronous communication network is simply a resource and not part of the framework; hence

[1] What is known as a rushing adversary in the literature is the special case of communication channels where a dishonest receiver sees the other parties' inputs of a round before choosing his own input for that round.

it can be modelled arbitrarily, allowing to capture incomplete networks and various types of channels (e.g., delay channels, secure, authenticated, insecure, etc).

We demonstrate the usage of our model with three examples: a two-party protocol to construct a common randomness resource (Section 5.4), the protocol introduced in [BGP89] to construct a broadcast resource (Section B.1), and the simple MPC protocol [Mau06] as the construction of a computer resource (Sections 5.6 and 5.7).

The Computer Resource. We introduce a system Computer which captures intuitively what traditional MPC protocols like GMW, BGW or CCD [GMW87, BGW88, CCD88, RB89, GRR98, Mau06] achieve. Traditionally, in a secure function evaluation protocol among n parties, the function to compute is modelled as an arithmetic circuit assumed to be known in advance. However, the same protocols are intuitively secure even if parties do not know in advance the entire circuit. It is enough that parties have agreement on the next instruction to execute.

We capture such guarantees in an interactive computer resource, similar to a (programmable) old-school calculator with a small instruction set (read, write, addition, and multiplication in our case), an array of value-registers, and an instruction queue. The resource has n interfaces. The interfaces $1, \ldots, n-1$ are used to give inputs to the resource and receive outputs from the resource. Interface n is used to write instructions into the queue. A read instruction (INPUT, i, p) instructs the computer to read a value from a value space $\mathcal{V}$ at interface i and store it at position p of the value register. A write instruction (OUTPUT, i, p) instructs the computer to output the value stored at position p to interface i. A computation instruction $(\text{OP}, p_1, p_2, p_3)$, $\text{OP} \in \{\text{ADD}, \text{MULT}\}$ instructs the computer to add or to multiply the values at positions p_1 and p_2 and store it at position p_3. We then show how to construct the computer resource using the Simple MPC protocol [Mau06]. A similar statement could be obtained using other traditional MPC protocols.

5.2 Synchronous Systems

To instantiate the Constructive Cryptography framework at the level of synchronous discrete systems, we need to instantiate the notions of a

resource $\mathbf{R} \in \Phi$ and a converter $\pi \in \Sigma$. We define each of them as special types of random systems [Mau02, MPR07].

5.2.1 Resources

A resource (the mathematical type) is a special type of random system [Mau02, MPR07]. A resource with n interfaces takes one input per interface and produces an output at every interface (see Figure 5.1). Without loss of generality, we assume that the alphabets at all interfaces and for all indices i are the same.[2] An $(n, \mathcal{X}, \mathcal{Y})$-resource is a resource with n interfaces and input (resp. output) alphabet $\mathcal{X}$ (resp. $\mathcal{Y}$).

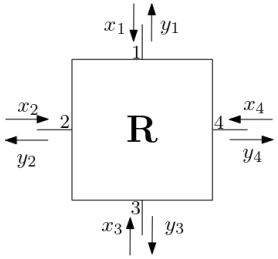

Figure 5.1: An example resource with 4 interfaces. At each invocation, the resource takes an input $x_j \in \mathcal{X}$ at each interface j, and it outputs a value $y_j \in \mathcal{Y}$ at each interface j.

Definition 5.2.1. An $(n, \mathcal{X}, \mathcal{Y})$-resource is an $(\mathcal{X}^n, \mathcal{Y}^n)$-random system.

Parallel Composition. One can take several independent $(n, \mathcal{X}_j, \mathcal{Y}_j)$-resources $\mathbf{R}_1, \ldots, \mathbf{R}_k$ and form an $(n, \bigtimes_{j=1}^{k} \mathcal{X}_j, \bigtimes_{j=1}^{k} \mathcal{Y}_j)$-resource, denoted $[\mathbf{R}_1, \ldots, \mathbf{R}_k]$. A party interacting with the composed resource $[\mathbf{R}_1, \ldots, \mathbf{R}_k]$ can give an input $\mathbf{a} = (a^1, \ldots, a^k)$, which is interpreted as giving each input $a^j \in \mathcal{X}_j$ to resource $\mathbf{R}_j$, and then receive an output $\mathbf{b} = (b^1, \ldots, b^k)$ containing the output from each of the resources.

In the following definition, we denote by $x_i = (\mathbf{a}_{1,i}, \ldots, \mathbf{a}_{n,i})$ the i-th input to the resource, and by $y_i = (\mathbf{b}_{1,i}, \ldots, \mathbf{b}_{n,i})$ the i-th output from the resource. We further let $[[x_i]]_j = ([\mathbf{a}_{1,i}]_j, \ldots, [\mathbf{a}_{n,i}]_j)$ be the tuple with

[2] The alphabets are large enough to include all values that can actually appear.

the j-th component of each tuple $\mathbf{a}_{\cdot,i}$; and let $[[x^i]]_j$ be the finite sequence $[[x_1]]_j, \ldots, [[x_i]]_j$. We let $[[y_i]]_j$ and $[[y^i]]_j$ be defined accordingly.

Definition 5.2.2. Given a tuple of resources $(\mathbf{R}_1, \ldots, \mathbf{R}_k)$, where $\mathbf{R}_j$ is an $(n, \mathcal{X}_j, \mathcal{Y}_j)$-resource. The parallel composition $\mathbf{R} := [\mathbf{R}_1, \ldots, \mathbf{R}_k]$, is an $(n, \bigtimes_{j=1}^k \mathcal{X}_j, \bigtimes_{j=1}^k \mathcal{Y}_j)$-resource, defined as follows:

$$\mathrm{p}^{\mathbf{R}}_{Y_i|X^iY^{i-1}}(y_i, x^i, y^{i-1}) = \prod_{j=1}^{k} \mathrm{p}^{\mathbf{R}_j}_{Y_i|X^iY^{i-1}}([[y_i]]_j, [[x^i]]_j, [[y^{i-1}]]_j)$$

5.2.2 Converters

An $(\mathcal{X}, \mathcal{Y})$-converter is a system (of a different type than resources) with two interfaces, an outside interface `out` and an inside interface `in`. The inside interface is connected to the $(n, \mathcal{X}, \mathcal{Y})$-resource, and the outside interface serves as the interface of the combined system. When an input is given (an input at the outside), the converter invokes the resource (with an input on the inside), and then converts its response into a corresponding output (an output on the outside). When a converter is connected to several resources in parallel $[\mathbf{R}_1, \ldots, \mathbf{R}_k]$, we address the corresponding sub-interfaces with the name of the resource, i.e, `in.R1` is the sub-interface connected to $\mathbf{R}_1$.

More concretely, an $(\mathcal{X}, \mathcal{Y})$-converter is an $(\mathcal{X} \cup \mathcal{Y}, \mathcal{X} \cup \mathcal{Y})$-random system whose input and output alphabets alternate between $\mathcal{X}$ and $\mathcal{Y}$. That is,

- On the first input, and further odd inputs, it takes a value $x \in \mathcal{X}$ and produces a value $x' \in \mathcal{X}$.
- On the second input, and further even inputs, it takes a value $y' \in \mathcal{Y}$, and produces a value $y \in \mathcal{Y}$.

Definition 5.2.3. An $(\mathcal{X}, \mathcal{Y})$-converter π is a pair of sequences of conditional probability distributions $\mathrm{p}^{\pi}_{X'_i|X^iX'^{i-1}Y'^{i-1}Y^{i-1}}$ and $\mathrm{p}^{\pi}_{Y_i|X^iX'^iY'^iY^{i-1}}$, for $i \geq 1$. Equivalently, a converter can be characterized by the sequence $\mathrm{p}^{\pi}_{X'^iY'^i|X^iY'^i} = \prod_{k=1}^{i} \mathrm{p}^{\pi}_{X'_k|X^kX'^{k-1}Y'^{k-1}Y^{k-1}} \cdot \mathrm{p}^{\pi}_{Y_k|X^kX'^kY'^kY^{k-1}}$, for $i \geq 1$.

Application of a Converter to a Resource Interface.

The application of a converter π to a resource $\mathbf{R}$ at interface j can be naturally understood as the resource that operates as follows (see Figure 5.2):

- On input $(x_1, \ldots, x_n) \in \mathcal{X}^n$: input x_j to π, and let x'_j be the output.

  Then, input $(x_1, \ldots, x_{j-1}, x'_j, x_{j+1}, \ldots, x_n) \in \mathcal{X}^n$ to $\mathbf{R}$.

- On output $(y_1, \ldots, y_{j-1}, y'_j, y_{j+1}, \ldots, y_n) \in \mathcal{Y}^n$ from $\mathbf{R}$, input y'_j to π, and let y_j be the output.

  The output is $(y_1, \ldots, y_n) \in \mathcal{Y}^n$.

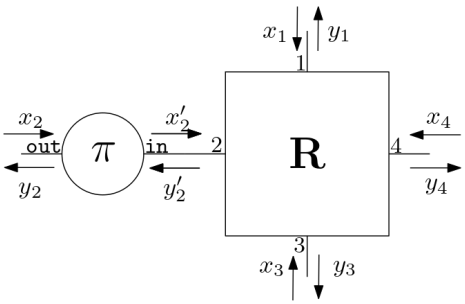

Figure 5.2: The figure shows the application of a converter π to the interface 2 of a resource $\mathbf{R}$. On input a value $x_2 \in \mathcal{X}$ to interface `out` of π, the converter π outputs a value $x'_2 \in \mathcal{X}$ at interface `in`. The resource $\mathbf{R}$ takes as input $(x_1, x'_2, x_3, x_4) \in \mathcal{X}^4$, and outputs $(y_1, y'_2, y_3, y_4) \in \mathcal{Y}^4$. On input y'_2 to interface `in` of π, the converter outputs a value y_2 at interface `out`.

Given a tuple $a = (a_1, \ldots, a_n)$, we denote $a_{\{j \to b\}}$ the tuple where the j-th component is substituted by b, i.e. $(a_1, \ldots, a_{j-1}, b, a_{j+1}, \ldots, a_n)$. Moreover, given a sequence a^i of tuples $t^1, \ldots, t^i$ and a sequence b^i of values $b_1, \ldots, b_i$, we denote $a^i_{\{j \to b^i\}}$, the sequence of tuples $t^1_{\{j \to b_1\}}$, $\ldots$, $t^i_{\{j \to b_i\}}$.

Definition 5.2.4. The application of an $(\mathcal{X}, \mathcal{Y})$-converter π at interface j of an $(n, \mathcal{X}, \mathcal{Y})$-resource $\mathbf{R}$ is the $(n, \mathcal{X}, \mathcal{Y})$-resource $\pi^j\mathbf{R}$ defined as follows:

$$\mathrm{p}^{\pi^j\mathbf{R}}_{Y^i|X^i}\left(y^i, x^i\right) = \sum_{x'^i, y'^i} \mathrm{p}^{\pi}_{X'^iY^i|X^iY'^i}\left(x'^i, [y^i]_j, [x^i]_j, y'^i\right) \mathrm{p}^{\mathbf{R}}_{Y^i|X^i}\left(y^i_{\{j\to y'^i\}}, x^i_{\{j\to x'^i\}}\right)$$

One can see that applying converters at distinct interfaces commutes. That is, for any converters π and ρ, any resource $\mathbf{R}$ and any disjoint interfaces j, k, we have that $\pi^j\rho^k\mathbf{R} = \rho^k\pi^j\mathbf{R}$.

For a tuple of converters $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_n)$, we denote by $\boldsymbol{\pi}\mathbf{R}$ the resource where each converter π_j is attached to interface j. Given a subset of interfaces I, we denote by $\boldsymbol{\pi}_I\mathbf{R}$ the resource where each converter π_j with $j \in I$, is attached to interface j.

5.3 Resources with Specific Round-Causality Guarantees

The resource type of Definition 5.2.1 captures that all parties act in a synchronized manner. The definition also implies that any (dishonest) party's input depends solely on the previous outputs seen by the party.

In practice this assumption is often not justified. For example, consider a resource consisting of two parallel communication channels (in a certain round) between two parties, one in each direction. Then it is typically unrealistic to assume that a dishonest party can not delay giving its input until having seen the output on the other channel. Such adversarial behavior is typically called "rushing" in the literature. More generally, a dishonest party's input can depend on partial information of the current round inputs from honest parties.

To model such causality guarantees, we introduce resources that proceed in two rounds (called semi-rounds) per actual protocol round.[3] This makes explicit what a dishonest party's input can (and can not) depend on.

[3]This type of resource is similar to the notion of *canonical synchronous functionalities* in [CCGZ16].

More concretely, each round r consists of two semi-rounds, denoted $r.a$ and $r.b$. In the first semi-round, $r.a$, the resource takes inputs from the honest parties and gives an output to the dishonest parties. No output is given to honest parties, and no input is taken from dishonest parties. In the second semi-round, $r.b$, the resource takes inputs from the dishonest parties and gives an output to all parties. Figure 5.3 illustrates the behavior of such a resource within one round. When describing such resources, we often omit specifying the semi-round when it is clear from the context.

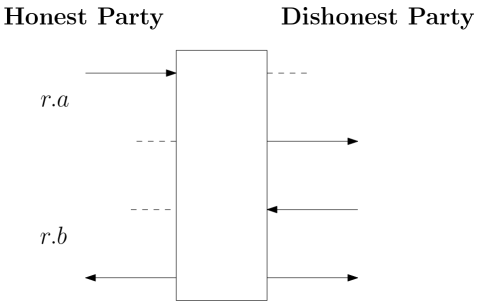

Figure 5.3: Figure depicts a resource operating in a round. The dashed lines indicate that no value is taken as input to the resource, and is output from the resource. The honest (resp. dishonest) parties give inputs to the resource in the first (resp. second) invocation, and all parties receive an output in the second invocation. The dishonest parties receive in addition an output in the first invocation.

When applying a protocol converter to such a resource, we formally attach the corresponding converter that operates in semi-rounds, where round-r inputs are given to the resource at $r.a$, and round-r outputs are obtained at $r.b$.

5.4 A First Example

We demonstrate the usage of our model to describe a very simple 2-party protocol which uses delay channels to generate common randomness. The protocol uses a channel with a known lower and upper bound on the

delay, and proceeds as follows: Each party generates a random value and sends it to the other party via a delay channel. Then, once the value is received, each party outputs the sum of the received value and the previously generated random value. It is intuitively clear that the protocol works because 1) a dishonest party does not learn the message before round r, and 2) an honest party is guaranteed to learn the message at round R.

Bounded-Delay Channel with Known Lower and Upper Bound. We model a simple delay channel $\overrightarrow{\mathcal{DC}}$ (resp. $\overleftarrow{\mathcal{DC}}$) from party 1 to party 2 (resp. party 2 to party 1) with known lower and upper bound on the delay. It takes a message at round 1, and is guaranteed to not deliver the message until round r to a dishonest party, but is guaranteed to deliver it at round R to an honest party. To model such a delay channel, we define a delay channel $\overrightarrow{\mathsf{DC}}_{r,R,Z}$ with message space $\mathcal{M}$ from party 1 to party 2 with fixed delay that takes a message at round 1 and delivers it at round r if the receiver is dishonest, and at round R if the receiver is honest. The set Z indicates the set of dishonest parties. The channel $\overleftarrow{\mathsf{DC}}_{r,R,Z}$ in the other direction is analogous.

Resource $\overrightarrow{\mathsf{DC}}_{r,R,Z}$

$\mathtt{msg} \leftarrow 0$
On input $m \in \mathcal{M}$ at interface 1 of round 1, set $\mathtt{msg} \leftarrow m$.
if $2 \in Z$ **then**
  Output $\mathtt{msg}$ at interface 2 at round r.
else
  Output $\mathtt{msg}$ at interface 2 at round R.

To capture that the delay channel is not guaranteed to deliver the message to a dishonest receiver exactly at round r, we consider the $*$-relaxation $(\overrightarrow{\mathsf{DC}}_{r,R,Z})^{*Z}$ on the delay channel at the dishonest interfaces Z. This specification includes resources with no guarantees at Z. For example, the resource may deliver the message later than r, or garbled, or not at all.

Common Randomness Resource. The sketched protocol constructs a common randomness resource CRS that outputs a random string. We

would like to model a CRS that is guaranteed to output the random string at round R to an honest party, but does not output the random string before r to a dishonest party. For that, we first consider a resource which outputs a random string to each honest (resp. dishonest) party at round R (resp. r).

Resource $\mathsf{CRS}_{r,R,Z}$

$\mathtt{rnd} \leftarrow_{\$} \mathcal{M}$
For each party $i \in \{1, 2\}$:
if $i \in Z$ **then**
    Output $\mathtt{rnd}$ at interface i at round r.
else
    Output $\mathtt{rnd}$ at interface i at round R.

With the same idea as with the delay channels, we can model a common randomness resource that is guaranteed to deliver the randomness to the honest parties at round R but is not guaranteed to deliver the output to the dishonest parties at round r, by considering a $*$-relaxation on the resource over the dishonest interfaces Z, $(\mathsf{CRS}_{r,R,Z})^{*Z}$.

Two-Party Construction. We describe the 2-party protocol $\boldsymbol{\pi} = (\pi_1, \pi_2)$ sketched at the beginning of the section and show that it constructs a common randomness resource.

Converter π_1

Local variable: $\mathtt{rnd}$

Round 1

$\mathtt{rnd} \leftarrow_{\$} \mathcal{M}$
Output $\mathtt{rnd}$ at $\mathtt{in.dc}$. // in.dc for π_2

Round R

On input $v \in \mathcal{M}$ at $\mathtt{in.dc}$, output $\mathtt{rnd} + v$ at $\mathtt{out}$. // in.dc for π_2

Lemma 5.4.1. $\boldsymbol{\pi} = (\pi_1, \pi_2)$ *constructs the specification* $(\mathsf{CRS}_{r,R,Z})^{*Z}$ *from the specification* $[(\overrightarrow{\mathsf{DC}}_{r,R,Z})^{*Z}, (\overleftarrow{\mathsf{DC}}_{r,R,Z})^{*Z}]$.

Proof. We prove each case separately.
1) $Z = \varnothing$*:* In this case, it is easy to see that $\pi_1\pi_2[\overrightarrow{\mathsf{DC}}_{r,R,\varnothing}, \overleftarrow{\mathsf{DC}}_{r,R,\varnothing}] = \mathsf{CRS}_{r,R,\varnothing}$ holds, since the sum of two uniformly random messages is uniformly random.
2) $Z = \{2\}$*:* Consider now the case where party 2 is dishonest (the case where party 1 is dishonest is similar). Let $\mathbf{S} \coloneqq [\overrightarrow{\mathsf{DC}}_{r,R,Z}, \overleftarrow{\mathsf{DC}}_{r,R,Z}]$. It suffices to prove that $\pi_1\mathbf{S} \in (\mathsf{CRS}_{r,R,Z})^{*Z}$ because:

$$\begin{aligned}\pi_1[(\overrightarrow{\mathsf{DC}}_{r,R,Z})^{*Z}, (\overleftarrow{\mathsf{DC}}_{r,R,Z})^{*Z}] &\subseteq (\pi_1[\overrightarrow{\mathsf{DC}}_{r,R,Z}, \overleftarrow{\mathsf{DC}}_{r,R,Z}])^{*Z} \\ &= (\pi_1\mathbf{S})^{*Z} \subseteq ((\mathsf{CRS}_{r,R,Z})^{*Z})^{*Z} = (\mathsf{CRS}_{r,R,Z})^{*Z},\end{aligned}$$

where the last equality holds because the $*$-relaxation is idempotent. Hence, we show that the converter σ described below is such that $\pi_1\mathbf{S} = \sigma^2\mathsf{CRS}_{r,R,Z}$.

Converter σ

Initialization

$\texttt{rcv} \leftarrow 0$.

Round $1.b$

On input $v \in \mathcal{M}$ at $\texttt{out.dc}$, set $\texttt{rcv} \leftarrow v$.

Round $r.a$

On input $\texttt{rnd}$ at $\texttt{in}$, output $\texttt{rnd} - \texttt{rcv}$ at $\texttt{out.dc}$.

Consider the system $\pi_1\mathbf{S}$. The system outputs at interface 1 of round $R.b$, a value $\texttt{rnd} + v$, where $\texttt{rnd}$ is a random value and v is the value received at interface 2 of round $1.b$ (and $v = 0$ if no value was received). Moreover, the system outputs at interface 2 of round $r.a$, the value $\texttt{rnd}$.

Now consider the system $\sigma^2\mathsf{CRS}_{r,R,Z}$. The system outputs at interface 1 of round $R.b$, a random value $\texttt{rnd}'$. Moreover, the system outputs at interface 2 of round $r.a$, the value $\texttt{rnd}' - v$, where v is the same value received at interface 2 of round $1.b$ (and $v = 0$ if no value was received).

Since the joint distribution $\{\texttt{rnd} + v, \texttt{rnd}\}$ and $\{\texttt{rnd}', \texttt{rnd}' - v\}$ are exactly the same, we conclude that $\pi_1\mathbf{S} = \sigma^2\mathsf{CRS}_{r,R,Z}$.

□

5.5 Communication Resources

5.5.1 Point-to-Point Channels

We model the standard synchronous communication network, where parties have the guarantee that messages input at round k are received by round $k+1$, and dishonest parties' round-k messages potentially depend on the honest parties' round-k messages. Let $\mathsf{AUTH}_{\ell,Z}(s,r)$ be a bilateral channel resource with n interfaces, one designated to each party $i \in \mathcal{P}$, and where two of the interfaces, s and r are designated to the sender and the receiver. The channel is parameterized by the set of dishonest parties $Z \subseteq \mathcal{P}$. The privacy guarantees are formulated by a leakage function $\ell(\cdot)$ that determines the information leaked to dishonest parties. For example, in an authenticated channel $\ell(m) = m$, and in a secure channel $\ell(m) = |m|$.

In the following, we formally describe the channel resource.

Resource $\mathsf{AUTH}_{\ell,Z}(s,r)$

Round $k, k \geq 1$

On input m at interface s, output m at interface r.
Output $\ell(m)$ at each interface $i \in Z$.

Let $\mathcal{N}_Z$ be the complete network of pairwise secure channels. That is, $\mathcal{N}_Z$ is the parallel composition of secure channels $\mathsf{AUTH}_{\ell,Z}(i,j)$ with $\ell(m) = |m|$, for each pair of parties $i, j \in \mathcal{P}$.

5.5.2 Broadcast Resource Specification

Broadcast is an important building block that many distributed protocols use. It allows a specific party, called the sender, to consistently distribute a message. More formally, it provides two guarantees: 1) Every honest party outputs the same value (consistency), and 2) the output value is the sender's value in case the sender is honest (validity).

The broadcast specification $\mathsf{BC}_{k,l,Z}(s)$ involves a set of parties $\mathcal{P}$, where one of the parties is the sender s. It is parameterized by the round numbers k and l indicating when the sender distributes the mes-

sage and when the parties are guaranteed to receive it. The specification $\mathsf{BC}_{k,l,Z}(s)$, is the set of all resources satisfying both validity and consistency. That is, there is a value v such that the output at each interface j for $j \notin Z$ at round $l.b$ is $y_j^{l.b} = v$, and if the sender is honest, this value is the sender's input $x_s^{k.a}$ at round $k.a$. That is:

$$\mathsf{BC}_{k,l,Z}(s) \coloneqq \left\{ \mathbf{R} \in \Phi \;\middle|\; \exists v \Big[\Big(\forall j \in \overline{Z}\ y_j^{l.b} = v \Big) \wedge \Big(s \in \overline{Z} \to v = x_s^{k.a} \Big) \Big] \right\}$$

We show how to construct such a broadcast specification in Section B.1. Let $\mathsf{BC}_{\Delta,Z}(s)$ be the parallel composition of $\mathsf{BC}_{k,k+\Delta,Z}(s)$, for each $k \geq 1$, and let $\mathsf{BC}_{\Delta,Z}$ be the parallel composition of $\mathsf{BC}_{\Delta,Z}(s)$, for each party $s \in \mathcal{P}$.

5.6 The Interactive Computer Resource

In this section, we introduce a simple ideal *interactive computer* resource with n interfaces. Interfaces $1, \ldots, n-1$ are used to give input values and receive output values. Interface n allows to input *instruction commands*. The resource has a memory which is split into two parts: an array storing values S and a queue C storing instruction commands to be processed. We describe the functionality of the resource in two parts: Storing the instructions that are input at n, and processing the instructions.

Store Instructions. On input an instruction at interface n at round r, the instruction is stored in the queue C. Then, after a fixed number of rounds, the input instruction is output at each honest interface i, and at dishonest interfaces at round $r.a$.

Instruction Processing. The interactive computer processes instructions sequentially. There are three types of instructions that the resource can process. Each instruction type has a fixed number of rounds.

1. An input instruction $(\textsc{input}, i, p)$ instructs the resource to read a value from a value space $\mathcal{V}$ at interface i and store it at position p of the array S. If party i is honest, it inputs the value at the first round of processing the input instruction, otherwise it inputs the value at the last round. This models the fact that a dishonest party

i can defer the choice of the input value to the end of processing the instruction.

2. An output instruction (OUTPUT, i, p) instructs the computer to output the value stored at position p to interface i. If party i is dishonest, it receives the value at the first round of processing the output instruction. Otherwise, the value is output at the last round of processing the instruction.

3. A computation instruction $(\text{OP}, p_1, p_2, p_3))$, $\text{OP} \in \{\text{ADD}, \text{MULT}\}$ instructs the computer to add or to multiply the values at positions p_1 and p_2 and store it at p_3.

One could consider different refinements of the interactive computer. For example, a computer that receives lists of instructions, processes instructions in parallel, or that allows instructions to be the result of a computation using values from S. For simplicity, we stick to a simple version of the computer.

Resource Computer_Z

Parameters: r_i,r_o,r_a,r_m,r_s. // #rounds to process an input, output, addition or multiplication instruction, and to store an instruction

Initialization

$L \leftarrow$ empty array. // Store values
$C \leftarrow$ empty queue. // Store instructions
`Next2Read` $\leftarrow 1$. // Counter indicating when to read the next instruction
`Current` $\leftarrow \perp$. // Contains the current instruction being processed

Round $k, k \geq 1$

// Read next instruction
if `Next2Read` $= k$ **then**
  `Current` $\leftarrow C$.`pop`()
  if `Current` $\neq \perp$ **then**
    `Next2Read` $\leftarrow k + r_j$, where r_j, for $j \in \{i, o, a, m\}$, is the round delay of the instruction in `Current`.
  else

$\texttt{Next2Read} \leftarrow k+1$

// Process instruction
if $\texttt{Current} = (\text{INPUT}, i, p)$ **then**
  if $i \notin Z$ **then**
    Read $x \in \mathcal{V}$ at interface i at round $\texttt{Next2Read} - r_i$.
  else
    Read $x \in \mathcal{V}$ at interface i at round $\texttt{Next2Read} - 1$.
  $L[p] \leftarrow x$
else if $\texttt{Current} = (\text{OP}, p_1, p_2, p_3)$, $\text{OP} \in \{\text{ADD}, \text{MULT}\}$ **then**
  $L[p_3] \leftarrow L[p_1]$ OP $L[p_2]$
else if $\texttt{Current} = (\text{OUTPUT}, i, p)$ **then**
  if $i \notin Z$ **then**
    Output $L[p]$ at interface i at round $\texttt{Next2Read} - 1$.
  else
    Output $L[p]$ at interface i at round $\texttt{Next2Read} - r_o$.
$\texttt{Current} \leftarrow \perp$

// Store instruction in queue
Read instruction I at interface n.
If I is a valid instruction, output I at each interface $i \in Z$. Then, at round $k + \Delta$ introduce the instruction in the queue $C.\texttt{push}(I)$, and output I at each interface $i \notin Z$. // If party n is honest, output to honest parties at $(k + \Delta).b$ and dishonest parties at $k.a$. Otherwise, output to all parties at $(k + \Delta).b$

5.7 Protocol Simple MPC

We adapt Maurer's Simple MPC protocol [Mau06], originally described for SFE in the stand-alone setting, to realize the resource Computer from Section 5.6, thereby proving a much stronger (and composable) statement. The protocol is run among a set $\mathcal{P} = \{1, \ldots, n\}$ of n parties. Parties $1, \ldots, n-1$ process the instructions, give input values and obtain output values. Party n has access to the instructions that the other parties needs to execute.

General Adversaries. In many protocols, the sets of possible dishonest parties are specified by a threshold t, that indicates that any set of dishonest parties is of size at most t. However, in this protocol, one specifies

a so-called *adversary structure* $\mathcal{Z}$, which is a monotone[4] set of subsets of parties, where each subset indicates a possible set of dishonest parties. We are interested in the condition that no three sets in $\mathcal{Z}$ cover $[n-1]$, also known as $\mathcal{Q}^3([n-1], \mathcal{Z})$ [HM00].

5.7.1 Protocol Description

Let $\mathcal{Z}$ be an adversary structure that satisfies $\mathcal{Q}^3([n-1], \mathcal{Z})$. Protocol $\mathsf{sMPC} = (\pi_1, \ldots, \pi_n)$ constructs the resource $\mathsf{Computer}_Z$, introduced in Section 5.6, for any $Z \in \mathcal{Z}$. For sets $Z \notin \mathcal{Z}$, the protocol constructs the trivial specification Φ.

Assumed Specifications. The protocol assumes the following specifications: a network specification $\mathcal{N}_Z$ among the parties in $\mathcal{P}$ (see Section 5.5.1) and a parallel broadcast specification $\mathsf{BC}_{\Delta,Z}$ which is the parallel composition of broadcast channels where any party in $\mathcal{P}$ can be a sender and the set of recipients is $\mathcal{P}$ (see Section 5.5.2).

Converters. The converter π_n is the identity converter. It allows to give direct access to the flow of instructions that the parties need to process. Because the instructions are delivered to the parties in $\mathcal{P}$ via the broadcast specification $\mathsf{BC}_{\Delta,Z}(n)$, parties have agreement on the next instruction to execute.

We now describe the converters $\pi_1, \ldots, \pi_{n-1}$. Each converter π_i keeps an (initially empty) array L with the current stored values, and a queue C of instructions to be executed. Each time an instruction is received from $\mathsf{BC}_{\Delta,Z}(n)$, it is added to C and also output. Each instruction in C is processed sequentially.

In order to describe how to process each instruction, we consider the adversary structure $\mathcal{Z}' \coloneqq \{Z \setminus \{n\} : \ Z \in \mathcal{Z}\}$. Let the maximal sets in $\mathcal{Z}'$ be $\max(\mathcal{Z}') \coloneqq \{Z_1, \ldots, Z_m\}$.

Input Instruction (input, i, p), for $i \in [n-1]$. Converter π_i does as follows: On input a value s from the outside interface, compute shares $s_1, \ldots, s_m$ using a m-out-of-m secret-sharing scheme (m is the number of maximal sets in $\mathcal{Z}'$). That is, compute random summands such that $s = \sum_{j=1}^{m} s_j$. Then, output s_j to the inside interface $\texttt{in.net.ch}_{i,k}$, for each party $k \in \overline{Z_j}$.

[4] If $Z \in \mathcal{Z}$ and $Z' \subseteq Z$, then $Z' \in \mathcal{Z}$.

Then each converter for party in $\overline{Z_j}$, echoes the received shares to all parties in $\overline{Z_j}$, i.e. outputs the received shares to $\texttt{in.net.ch}_{i,k}$, for each party $k \in \overline{Z_j}$. If a converter obtained different values, it broadcasts a complaint message, i.e. it outputs a complaint message at `in.bc`. In such a case, π_i broadcasts the share s_j. At the end of the process, the converters store the received shares in their array, along with the information that the value was assigned to position p. Intuitively, a consistent sharing ensures that no matter which set Z_k of parties is dishonest, they miss the share s_k, and hence s remains secret.

Output Instruction $(\textbf{output}, i, p)$**, for** $i \in [n-1]$**.** Each converter π_l, $l \in [n-1]$, outputs all the stored shares assigned to position p at interface $\texttt{in.net.ch}_{l,i}$. Converter π_i does: Let v_j^l be the value received from party l as share j at $\texttt{in.net.ch}_{l,i}$. Then, converter π_i reconstructs each share s_j as the value v such that $\{l \mid v_j^l \neq v\} \in \mathcal{Z}$, and outputs $\sum_j s_j$.

Addition Instruction $(\textbf{add}, p_1, p_2, p_3)$**.** Each converter for a party in $\overline{Z_j}$ adds the j-th shares of the values assigned to positions p_1 and p_2, and stores the result as the j-th share of the value at position p_3.

Multiplication Instruction $(\textbf{mult}, p_1, p_2, p_3)$**.** The goal is to compute a share of the product ab, assuming that the converters have stored shares of a and of b respectively. Given that $ab = \sum_{p,q=1}^{m} a_p b_q$, it suffices to compute shares of each term $a_p b_q$, and add the shares locally. In order to compute a sharing of $a_p b_q$, the converter for each party $i \in \overline{Z_p} \cap \overline{Z_q}$ executes the same steps as the input instruction, with the value $a_p b_q$. Then, converters for parties in $\overline{Z_p} \cap \overline{Z_q}$ check that they all shared the same value by reconstructing the difference of every pair of shared values. In the case that all differences are zero, they store the shares of a fixed party (e.g. the shares from the party in $\overline{Z_p} \cap \overline{Z_q}$ with the smallest index). Otherwise, each term a_p and b_q is reconstructed, and the default sharing $(a_p b_q, 0, \ldots, 0)$ is adopted.

Theorem 5.7.1. *Let $\mathcal{P} = \{1, \ldots, n\}$, and let $\mathcal{Z}$ be an adversary structure that satisfies $\mathcal{Q}^3([n-1], \mathcal{Z})$. Protocol* sMPC *constructs* $(\mathsf{Computer}_Z)^{*z}$ *with parameters $(r_i, r_o, r_a, r_m, r_s) = (2\Delta + 2, 1, 0, 2\Delta + 4, \Delta)$ from the assumed specification $[\mathcal{N}_Z, \mathsf{BC}_{\Delta,Z}]$, for any $Z \in \mathcal{Z}$, and constructs Φ otherwise.*

Proof. **Case** $Z = \varnothing$**:** In this case all parties are honest. We need to argue that:

$$\mathcal{R}_{\varnothing} \coloneqq \mathsf{sMPC}[\mathcal{N}_{\varnothing}, \mathsf{BC}_{\Delta,\varnothing}] = \mathsf{Computer}_{\varnothing}.$$

At the start of the protocol, the computer resource $\mathsf{Computer}_{\varnothing}$, and each protocol converter has an empty queue C of instructions and empty array L of values. Consider the system $\mathcal{R}_{\varnothing}$. Each time party n inputs an instruction I to $\mathsf{BC}_{\Delta,\varnothing}(n)$, because of validity, it is guaranteed that after Δ rounds each protocol converter receives I, stores I in the queue C and outputs I at interface `out`. Each converter processes the instructions in its queue sequentially, and each instruction takes the same constant amount of rounds to be processed for all parties. Hence, all honest parties keep a queue with the same instructions throughout the execution of the protocol.

Now consider the system $\mathsf{Computer}_{\varnothing}$. It stores each instruction input at interface n in its queue C, and outputs the instruction I at each party interface $i \in \mathcal{P}$ after Δ rounds. The instructions are processed sequentially, and it takes the same amount of rounds to process each instruction as in $\mathcal{R}_{\varnothing}$.

We then conclude that each queue for each protocol converter in $\mathcal{R}_{\varnothing}$ contains exactly the same instructions as the queue in $\mathsf{Computer}_{\varnothing}$.

We now argue that the behavior of both systems is identical not only when storing the instructions, but also when processing them.

Let us look at the content of the arrays L that $\mathsf{Computer}_{\varnothing}$ and each protocol converter in $\mathcal{R}_{\varnothing}$ has. Whenever a value s is stored in the array L of $\mathsf{Computer}_{\varnothing}$ at position p, there are values s_l, such that $s = \sum_{l=1}^{m} s_l$ and s_l is stored in each converter π_j such that $j \notin Z_l$. For each value s_l, the converters that store s_l, also stores additional information containing the position p and the index l.

Consider an input instruction, (INPUT, i, p) at round k, and a value x is input at the next round at interface i. In the system $\mathcal{R}_{\varnothing}$, the converter π_i computes values s_l, such that $s = \sum_{l=1}^{m} s_l$ and sends each s_l to each converter π_j such that $j \notin Z_l$. All broadcasted messages are 0, i.e. there are no complaints, and as a consequence s_l is stored in each converter π_j, where $j \notin Z_j$. In the system $\mathsf{Computer}_{\varnothing}$, the value x is stored at the p-th register of the array L.

Consider an output instruction, (OUTPUT, i, p). In the system $\mathcal{R}_{\varnothing}$, each converter π_j sends the corresponding previously stored values s_l associated with position p, and π_i outputs $s = \sum_{l=1}^{m} s_l$. In the system $\mathsf{Computer}_{\varnothing}$, the value x stored at the p-th register of the array L is output

at interface i.

Consider an addition instruction, $(\text{ADD}, p_1, p_2, p_3)$. In the system $\mathcal{R}_\varnothing$ each converter adds, for each share index l, the corresponding values associated with position p_1 and p_2, and stores the result as a value associated with position p_3 and index l. In the system $\mathsf{Computer}_\varnothing$, the sum of the values a and b stored at the p_1-th and p_2-th positions is stored at position p_3.

Consider a multiplication instruction, $(\text{MULT}, p_1, p_2, p_3)$. In the ideal system $\mathsf{Computer}_\varnothing$, the product of the values a and b stored at positions p_1-th and p_2-th is stored at position p_3. In the system $\mathcal{R}_\varnothing$, let a_p (resp. b_q) be the value associated with position p_1 (resp. p_2) and with index p (resp. q), that each converter for party in $\overline{Z_p}$ (resp. $\overline{Z_q}$) has. For each $1 \leq p, q \leq m$, consider each protocol converter for party $j \in \overline{Z_p} \cap \overline{Z_q}$. (Note that since the adversary structure satisfies $Q^3(\mathcal{P}, \mathcal{Z})$, then, for any two sets $Z_p, Z_q \in \mathcal{Z}$, $\overline{Z_p} \cap \overline{Z_q} \neq \varnothing$.) The converter does the following steps:

1. Input instruction steps with the value $a_p b_q$ as input. As a result, each converter in $\overline{Z_u}$ stores a value, which we denote v_j^u, from $j \in \overline{Z_p} \cap \overline{Z_q}$.

2. Execute the output instruction, with the value $v_j^u - v_{j_0}^u$ and towards all parties in $[n-1]$. As a result, every party obtains 0, and the value $v_{j_0}^u$ is stored.

3. The value associated with position p_3 and index p, stored by each converter for party in $\overline{Z_p}$, is the sum $w_p = \sum_{j_0} v_{j_0}^p$.

As a result, each party in $\overline{Z_p}$ stores w_p, and $\sum_p w_p = ab$.

Case $Z \neq \varnothing$: In this case, the statement is only non-trivial if $Z \in \mathcal{Z}$, because otherwise the ideal system specification is $\mathcal{S}_Z = \Phi$, i.e. there are no guarantees.

We need to show that when executing sMPC with the assumed specification, we obtain a system in the specification $(\mathsf{Computer}_Z)^{*_Z}$. That is, for each network resource $\mathsf{N} \in \mathcal{N}_Z$ and parallel broadcast resource $\mathsf{PBC} = [\mathsf{BC}_1, \ldots, \mathsf{BC}_n] \in \mathsf{BC}_{\Delta,Z}$ we need to find a system σ such that:

$$\mathbf{R} \coloneqq \mathsf{sMPC}_{\mathcal{P}\setminus Z}[\mathsf{N}, \mathsf{PBC}] = \mathbf{S} \coloneqq \sigma^Z \mathsf{Computer}_Z.$$

Converter σ

Round k, $k \geq 1$

// Dishonest party n

1: Emulate the behavior of the assumed broadcast resource BC_n, where honest parties' inputs are $\perp$, and dishonest parties' inputs are given at `out`.
2: On input an instruction at `in`, store it.
// Instruction emulation
3: Read the next instruction I to execute.
4: **if** $I = (\text{INPUT}, i, p)$ **then**
5: **if** $i \notin Z$ **then**
6: Compute and output a random value s_q at each interface `out`.i, $i \in Z \cap \overline{Z_q}$. Store s_q, q and p.
7: On input a complaint message on q at `out`.i, output the stored value s_q at interface `out.bc`.
8: **else**
9: If there is exactly one value received s_q for each $\overline{Z} \cap \overline{Z_q}$, then store the values s_q with q and p.
10: Otherwise, emulate the behavior of BC_j for a complaint message, for each q such that there are zero or more than a different value, and for each $j \in \overline{Z} \cap \overline{Z_q}$.
11: On input s_q at `out`, store it.
12: Input at `in` the sum of values s_q stored.
13: **else if** $I = (\text{OUTPUT}, i, p)$ **then**
14: Read $x \in \mathcal{V}$ at interface `in`.
15: Output at `out`, random values s'_q such that $\sum_{q:Z\cap\overline{Z_q}=\varnothing} s'_q + \sum_{q:Z\cap\overline{Z_q}\neq\varnothing} s_q = x$. // The dishonest values s_q associated with position p are stored
16: **else if** $I = (\text{ADD}, p_1, p_2, p_3)$ **then**
17: For each q, add the values s_q, s'_q associated with p_1 and p_2 respectively, and store the result as well as the position p_3.
18: **else if** $I = (\text{MULT}, p_1, p_2, p_3)$ **then**
19: Consider, for each $1 \leq p, q \leq m$, two possible cases:
20: **if** $Z \cap \overline{Z_p} \cap \overline{Z_q} \neq \varnothing$ **then**
21: The values a_p and b_q, where a_p (resp. b_q) is the value associated with position p_1 (resp. p_2).

22: For each $i \in Z \cap \overline{Z_p} \cap \overline{Z_q}$, follow the same steps as with the input instruction (Steps 9-11). // Check that the dishonest parties in $\overline{Z_p} \cap \overline{Z_q}$ input a consistent sharing
23: Check that the values from party i add up to $a_p b_q$. If so, store the values from the party with the smallest index. Otherwise, define the sharing of $a_p b_q$ as $(a_p b_q, \ldots, 0)$, and output the corresponding shares to the dishonest parties.
24: **else**
25: If all parties in $\overline{Z_p} \cap \overline{Z_q}$ are honest, generate random values as shares of $a_p b_q$, store them and answer complaints, according to Steps 6-7.
26: Output 0s as the reconstructed differences.
27:

We first argue that the instructions written at the queue C in resource $\sigma^Z \mathsf{Computer}_Z$ follow the same distribution as the instructions that the honest parties store in their queue in the system $\mathcal{R}_Z$. If party n is honest, this is true, as argued in the previous case for $Z = \varnothing$. In the case that party n is dishonest, the converter σ inputs (equally distributed) instructions as BC_n outputs to honest parties in $\mathcal{R}_Z$ by emulating the behavior of BC_n, taking into account the inputs from dishonest parties provided at the outside interface, and the honest parties' inputs are $\perp$.

Now we need to show that the messages that dishonest parties receive in both systems are equally distributed. We argue about each single instruction separately. Let I be the next instruction to be executed.

Input instruction: $I = (\text{INPUT}, i, p)$. We consider two cases, depending on whether party i is honest.

Dishonest party i. In the system $\mathbf{R}$, if a complaint message is generated from an honest party, the exact same complaint message will be output by σ in the system $\mathbf{S}$. This is because σ stores the shares received at the outside interface by the dishonest parties, and checks that the shares are consistent. Moreover, at the end of the input instruction it is guaranteed that all shares are consistent (i.e., all honest parties in each $\overline{Z_q}$ have the same share), and hence the sum of the shares is well-defined. This exact sum is input at $\mathsf{Computer}_Z$ by σ.

Honest party i. In this case, the converter σ generates and outputs random consistent values as the shares for dishonest parties. On input a complaint from a dishonest party, output at the broadcast interface its

share to all dishonest parties. In the system $\mathbf{R}$, dishonest parties also receive shares that are randomly distributed. Observe that in this case, the correct value is stored in the queue of $\mathsf{Computer}_Z$, but σ only has the shares of dishonest parties.

Output instruction: $I = (\text{OUTPUT}, i, p)$. In this case, the emulation is only non-trivial if party i is dishonest. The converter outputs random shares such that the sum of the random shares and the corresponding shares from dishonest parties that are stored, corresponds to the output value x obtained from $\mathsf{Computer}_Z$. Observe that in the system $\mathbf{R}$, the shares sum up to the value x as well, because of the $\mathcal{Q}^3$ condition. Given that the correct value was stored in the queue in every input instruction, the same shares that are output by σ follow the same distribution as the shares received by dishonest parties in $\mathbf{R}$ (namely, random shares subject to the fact that the sum of the random shares and the dishonest shares is equal to x).

Addition instruction: $I = (\text{ADD}, p_1, p_2, p_3)$. The converter σ simply adds the corresponding shares and stores them in the correct location.

Multiplication instruction: $I = (\text{MULT}, p_1, p_2, p_3)$. Consider each $1 \leq p, q \leq m$. Consider the following steps in the execution of the multiplication instruction in $\mathbf{R}$:

1. Honest parties execute the input instruction steps with the value $a_p b_q$ as input. Dishonest parties can use any value as input. However, it is guaranteed that the sharing is consistent. That is, each converter for an honest party in $\overline{Z_u}$ stores a value, which we denote v_j^u, from $j \in \overline{Z_p} \cap \overline{Z_q}$.

2. Execute the output instruction, with the value $v_j^u - v_{j_0}^u$ and towards all parties in $\mathcal{P}$. If any dishonest party used a value different than $a_p b_q$ in the previous step, one difference will be non-zero, and the default sharing $(a_p b_q, 0, \ldots, 0)$ is adopted. Otherwise, the sharing from P_{j_0}, i.e. the values $v_{j_0}^u$, is adopted.

Case $Z \cap \overline{Z_p} \cap \overline{Z_q} \neq \varnothing$: If there is a dishonest party in $\overline{Z_p} \cap \overline{Z_q}$, then the converter σ has the values a_p and b_q stored.

Step 1: For each dishonest party $i \in \overline{Z_p} \cap \overline{Z_q}$, the converter σ checks whether the shares are correctly shared (it checks that the dishonest parties in $\overline{Z_p} \cap \overline{Z_q}$ input a consistent sharing), in the same way as when emulating the input instruction.

Step 2: After that, σ checks that the shares from party i add up to $a_p b_q$. If not, the converter σ defines the sharing of $a_p b_q$ as $(a_p b_q, 0, \ldots, 0)$, and outputs the corresponding shares to the dishonest parties.

Observe that given that the adversary structure satisfies the $\mathcal{Q}^3$ condition, there is always an honest party in $\overline{Z_p} \cap \overline{Z_q}$. Then, in the system $\mathbf{R}$, it is guaranteed that the value $a_p b_q$ is shared. Moreover, as in $\mathbf{S}$, the default sharing is adopted if and only if a dishonest party shared a value different from $a_p b_q$.

Case $Z \cap \overline{Z_p} \cap \overline{Z_q} = \varnothing$*:* If all parties in $\overline{Z_p} \cap \overline{Z_q}$ are honest, dishonest parties receive random shares in $\mathbf{R}$. Moreover, all reconstructed differences are 0, since honest parties in $\overline{Z_p} \cap \overline{Z_q}$ share the same value. In $\mathbf{S}$, σ generates random values as shares of $a_p b_q$ as well, and then open 0s as the reconstructed differences.

□

Appendix B

Details of Chapter 5

B.1 Broadcast Construction

We show how to construct the broadcast resource specification introduced in Section 5.5.2, using the so-called *king-phase* paradigm [BGP89]. The construction consists of several steps, each providing stronger consistency guarantees.

B.1.1 Weak-Consensus

Let Z be a set of parties. The primitive *weak-consensus* provides two guarantees:

- Validity: If all parties in $\overline{Z}$ input the same value, they agree on this value.
- Weak Consistency: If some party $i \in \overline{Z}$ decides on an output $y_i \in \{0, 1\}$, then every other party $j \in \overline{Z}$ decides on a value $y_j \in \{y_i, \perp\}$.

A specification $\mathcal{WC}_{k,l,Z,t}$ capturing the guarantees of a weak-consensus primitive (up to t dishonest parties, and where parties input at round k and output at round l) can be naturally defined as the set of all resources satisfying validity and weak consistency. More concretely, for $|Z| \leq t$, $\mathcal{WC}_{k,l,Z,t}$, is the set of all resources which output a value at round $l.b$

that satisfy the validity and weak consistency properties, according to the inputs from round $k.a$. That is:

$$\mathcal{WC}_{k,l,Z,t} \coloneqq \Big\{ \mathbf{R} \in \Phi \;\Big|\; \exists v \Big(\forall j \in \overline{Z}\ y_j^{l.b} \in \{v, \perp\} \Big) \wedge \Big(\exists v'\ \forall j \in \overline{Z}\ x_j^{k.a} = v' \rightarrow \forall j \in \overline{Z}\ y_j^{l.b} = x_j^{k.a} \Big) \Big\}$$

And when $|Z| > t$, $\mathcal{WC}_{k,l,Z,t} = \Phi$.

Protocol $\Pi^k_{\texttt{wc}} = (\pi_1^{\texttt{wc}}, \ldots, \pi_n^{\texttt{wc}})$ constructs specification $\mathcal{WC}_{k,k,Z,t}$ from $\mathcal{N}_Z$. The protocol is quite simple: At round k each party sends its input message to every other party via each channel. Then, if there is a bit b that is received at least $n - t$ times, the output is b. Otherwise, the output is $\perp$. At a very high level, the protocol meets the specification because, if a party i outputs a bit b, it received b from at least $n - t$ parties, and hence it received b from at least $n - 2t$ honest parties. This implies that every other party received the bit $1 - b$ at most $2t < n - t$ times (since $t < \frac{n}{3}$). Hence, no honest party outputs $1 - b$.

Converter $\pi_i^{\texttt{wc}}$

Local Variable: y.

Round k

On input x_i at `out`, output x_i to each $\texttt{in.net.ch}_{i,j}$, where $j \in \mathcal{P}$.
On input values y_j at each $\texttt{in.net.ch}_{i,j}$:
if $\big|\{j \in \mathcal{P} \mid y_j = 0\}\big| \geq n - t$ **then**
  $y \leftarrow 0$
else if $\big|\{j \in \mathcal{P} \mid y_j = 1\}\big| \geq n - t$ **then**
  $y \leftarrow 1$
else
  $y \leftarrow \perp$
Output y at `out`.

Theorem B.1.1. *Let $t < \frac{n}{3}$. $\Pi^k_{\texttt{wc}}$ constructs $\mathcal{WC}_{k,k,Z,t}$ from $\mathcal{N}_Z$, for any $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$, and constructs Φ otherwise.*

Proof. Let $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$. We want to prove that the system specification $\mathcal{R}_Z \coloneqq (\Pi^k_{\texttt{wc}})_{\overline{Z}}\mathcal{N}_Z \subseteq \mathcal{WC}_{k,k,Z,t}$.

For that, all we need to prove is that at round $k.b$, the outputs from the honest parties satisfy both the weak-consistency and the validity property, where the inputs to be taken into account are those at round $k.a$. We divide two cases:

- If every party $i \in \overline{Z}$ had as input value b at round k (there was pre-agreement): In the system specification $\mathcal{WC}_{k,k,Z,t}$, the parties output the bit b by definition. In the system specification $\mathcal{R}_Z$, each party $i \in \overline{Z}$ receives the bit b at least $n-t$ times. Hence, each party $i \in \overline{Z}$ also outputs b.

- Otherwise, in $\mathcal{R}_Z$, either every party $i \in \overline{Z}$ outputs $\perp$ (in which case the parties meet the specification $\mathcal{WC}_{k,k,Z,t}$), or some party i outputs a bit b. In this case, we observe that it received b from at least $n-t$ parties, and hence it received b from at least $n-2t$ honest parties. This implies that every other party received the bit $1-b$ at most $2t < n-t$ times (since $t < \frac{n}{3}$). In conclusion, no honest party outputs $1-b$, and the parties output a value $v_i \in \{\perp, b\}$.

□

B.1.2 Graded-Consensus

We define *graded-consensus* with respect to a set of parties Z. In this protocol, each party inputs a bit $x_i \in \{0,1\}$ and outputs a pair value-grade $(y_i, g_i) \in \{0,1\}^2$. The primitive provides two guarantees:

- Validity: If all parties in $\overline{Z}$ input the same value, they agree on this value with grade 1.

- Graded Consistency: If some party $i \in \overline{Z}$ decides on a value $y_i \in \{0,1\}$ with grade $g_i = 1$, then every other party $j \in \overline{Z}$ decides on the same value $y_j = y_i$.

Specification $\mathcal{GC}_{k,l,Z,t}$ captures the guarantees of a graded-consensus primitive secure up to t dishonest parties, and where parties give input at round k and output at round l. If $|Z| \leq t$:

$$\mathcal{GC}_{k,l,Z,t} \coloneqq \Big\{ \mathbf{R} \in \Phi \;\Big|\; \forall v \left(\exists j \in \overline{Z}\ y_j^{l.b} = (v,1) \rightarrow \forall i \in \overline{Z}\ y_i^{l.b} = (v,g) \ \wedge\ g \in \{0,1\} \right) \wedge \left(\exists v\ \forall j \in \overline{Z}\ x_j^{k.a} = v \rightarrow \forall j \in \overline{Z}\ y_j^{l.b} = (x_j^{k.a}, 1) \right) \Big\}$$

And when $|Z| > t$, $\mathcal{GC}_{k,l,Z,t} = \Phi$.

We show a protocol $\Pi^k_{\mathsf{gc}} = (\pi_1^{\mathsf{gc}}, \ldots, \pi_n^{\mathsf{gc}})$ that constructs specification $\mathcal{GC}_{k,k+1,Z,t}$ from the assumed specification $[\mathcal{WC}_{k,k,Z,t}, \mathcal{N}_Z]$: At round k, each party i invokes the weak consensus protocol on its input x_i. Then, at round $k+1$, each party sends the output from the weak consensus protocol to every other party via the network. After that, each party i sets the output value y_i to be the most received bit, and the grade $g_i = 1$ if and only if the value was received at least $n - t$ times.

If any party i decides on an output y_i with $g_i = 1$, it means that the party received y_i from at least $n - t$ parties, where at least $n - 2t$ are honest parties. Hence, every other honest party received the value y_i at least $n - 2t$ times. Given that $n - 2t > t$, at least one honest party obtained y_i as output of $\mathcal{WC}_{k,k,Z,t}$. Therefore, by weak consistency, no honest party obtained $1 - y_i$ as output from $\mathcal{WC}_{k,k,Z,t}$, from which it follows that each honest party j received it at most $t < n - 2t$ times and therefore outputs $y_j = y_i$.

Converter π_i^{gc}

Local Variables: y, g.

Round k

On input x_i at `out`, output x_i at `in.wc`. // Output the value to $\mathcal{WC}_{k,k,Z,t}$
On input z_i at `in.wc`, store the value.

Round $k+1$

Output z_i at each interface $\texttt{in.net.ch}_{i,j}$, for $j \in \mathcal{P}$.
On input a message z_j from each $\texttt{in.net.ch}_{j,i}$: // Value from each party j
if $\big|\{j \in \mathcal{P} \mid z_j = 0\}\big| \geq \big|\{j \in \mathcal{P} \mid z_j = 1\}\big|$ **then**

$y \leftarrow 0$
else
  $y \leftarrow 1$
if $\left|\{j \in \mathcal{P} \mid z_j = y\}\right| \geq n - t$ **then**
  $g \leftarrow 1$
else
  $g \leftarrow 0$
Output (y, g) at `out`.

Theorem B.1.2. *Let $t < \frac{n}{3}$. $\Pi^k_{\mathtt{gc}}$ constructs $\mathcal{GC}_{k,k+1,Z,t}$ from specification $[\mathcal{WC}_{k,k,Z,t}, \mathcal{N}_Z]$, for any $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$, and constructs Φ otherwise.*

Proof. Let $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$. We want to prove that the system specification $\mathcal{R}_Z \coloneqq (\Pi^k_{\mathtt{gc}})_{\overline{Z}}[\mathcal{WC}_{k,k,Z,t}, \mathcal{N}_Z] \subseteq \mathcal{GC}_{k,k+1,Z,t}$.

For that, all we need to prove is that at round $(k+1).b$, the outputs from the honest parties satisfy both the graded-consistency and the validity property, where the inputs to be taken into account are those at round $k.a$.

At round $k.a$, each party $i \in \overline{Z}$ inputs the message x_i to $\mathcal{WC}_{k,k,Z,t}$. Then, it is guaranteed that at round $k.b$, honest parties obtain an output that satisfies validity and weak-consistency. At round $(k+1).b$, we divide two cases:

- If every party $i \in \overline{Z}$ had as input value b at round k (there was pre-agreement): In $\mathcal{GC}_{k,k+1,Z,t}$, the parties output the bit $(b, 1)$ by definition. In $\mathcal{R}_Z$, each party $i \in \overline{Z}$ outputs the bit b as z_j because of the validity of $\mathcal{WC}_{k,k,Z,t}$. Then, party i receives at least $n - t$ times the bit b. Hence, each party $i \in \overline{Z}$ also outputs b.

- If an honest party i decides on an output y_i with $g_i = 1$, then it means that the party received y_i from at least $n - t$ parties, where at least $n - 2t$ are honest parties. This implies that every other honest party received the value y_i at least $n - 2t$ times. Given that $n - 2t > t$, at least one honest party obtained y_i as output of $\mathcal{WC}_{k,k,Z,t}$ at round $(k+1).b$. Therefore, by weak consistency, no honest party obtained $1 - y_i$ as output from $\mathcal{WC}_{k,k,Z,t}$, from which it follows that each honest party j received at most $t < n - 2t$ times and therefore outputs $y_j = y_i$.

□

B.1.3 King-Consensus

We first define a specification that achieves *king-consensus* with respect to a set of parties Z. In the king-consensus primitive, there is a party K, the king, which plays a special role. The primitive provides two guarantees:

- Validity: If all parties in $\overline{Z}$ input the same value, they agree on this value.
- King Consistency: If party $K \in \overline{Z}$, then there is a value y such that every party $j \in \overline{Z}$ decides on the value $y_j = y$.

We describe a specification $\mathcal{KC}_{k,l,Z,t,K}$ that models a king-consensus primitive where K has the role of king, and is secure up to t dishonest parties, which starts at round k and ends at round l. If $|Z| \leq t$:

$$\mathcal{KC}_{k,l,Z,t,K} \coloneqq \Bigg\{ \mathbf{R} \in \Phi \;\Big|\; \Big(K \in \overline{Z} \rightarrow \exists v\ \forall i \in \overline{Z}\ y_i^{l.b} = v\Big) \wedge \Big(\exists v\ \forall j \in \overline{Z}\ x_j^{k.a} = v \rightarrow \forall j \in \overline{Z}\ y_j^{l.b} = x_j^{k.a}\Big) \Bigg\}$$

And when $|Z| > t$, $\mathcal{KC}_{k,l,Z,t,K} = \Phi$.

Protocol $\Pi^k_{\mathtt{kc}} = (\pi^{\mathtt{kc}}_1, \ldots, \pi^{\mathtt{kc}}_n)$ constructs specification $\mathcal{KC}_{k,k+2,Z,t,K}$ from the assumed specification $[\mathcal{GC}_{k,k+1,Z,t}, \mathcal{N}_Z]$: At round k, each party i invokes the graded consensus protocol on its input x_i. Then, at round $k+2$, the king K sends the output z_K from the graded consensus protocol to every other party. Finally, each party i sets the value $y_i = z_i$ to the output of graded consensus if the grade was $g_i = 1$, and otherwise to the value of the king $y_i = z_K$. Note that consistency is guaranteed to hold only in the case the king is honest: if every honest party i has grade $g_i = 0$, they all adopt the king's value. Otherwise, there is a party j with grade $g_j = 1$, and graded consistency ensures that all honest parties (in particular the king) have the same output.

Converter $\pi_i^{\mathtt{kc}}$

Local Variable: y.

Round k

On input x_i at `out`, output x_i at $\mathtt{in.gc}$. // Output to $\mathcal{GC}_{k,k+1,Z,t}$

Round $k+1$

On input (z_i, g_i) from `in.gc`, store the pair.

Round $k+2$

If $i = K$, output z_K to each $\mathtt{in.net.ch}_{K,j}$, for $j \in \mathcal{P}$. // Party i is the king

On input z_K from $\mathtt{in.net.ch}_{K,i}$:

if $g_i = 0$ **then**

  $y \leftarrow z_K$

else

  $y \leftarrow z_i$

Output y at `out`.

Theorem B.1.3. *Let $t < \frac{n}{3}$. $\Pi_{\mathtt{kc}}^k$ constructs $\mathcal{KC}_{k,k+2,Z,t,K}$ from specification $[\mathcal{GC}_{k,k+1,Z,t}, \mathcal{N}_Z]$, for any $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$, and constructs Φ otherwise.*

Proof. Let $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$. We want to prove that the system specification $\mathcal{R}_Z := (\Pi_{\mathtt{kc}}^k)_{\overline{Z}}[\mathcal{GC}_{k,k+1,Z,t}, \mathcal{N}_Z] \subseteq \mathcal{KC}_{k,k+2,Z,t,K}$.

At round $k.a$, each party $i \in \overline{Z}$ inputs the message x_i to $\mathcal{GC}_{k,k+1,Z,t}$. Then, it is guaranteed that at round $(k+1).b$, honest parties obtain an output that satisfies validity and graded-consistency. We divide two cases:

- If every party $i \in \overline{Z}$ had as input value b at round k (there was pre-agreement): In $\mathcal{KC}_{k,k+2,Z,t,K}$, the parties output the bit b at round $k+2$ by definition. In the system specification $\mathcal{R}_Z$, each party $i \in \overline{Z}$ receives the bit $(b, 1)$ at round $k+1$, because of the validity of $\mathcal{GC}_{k,k+1,Z,t}$. Hence, each party $i \in \overline{Z}$ also outputs b at round $k+2$.

- Otherwise, assume the king is honest. If every honest party i obtains an output $(z_i, 0)$, then at round $(k+2).b$, every party takes the value of the king z_K. Otherwise, there is a party j that obtained an

output $(z_j, 1)$ at round $(k+1).b$. In this case, graded consistency implies that all honest parties have the same output. In particular, this holds for the honest king. Thus, all parties decide on the same output.

□

B.1.4 Consensus

We define a specification that achieves *consensus* with respect to a set of parties Z. The primitive provides two guarantees:

- Validity: If all parties in $\overline{Z}$ input the same value, they agree on this value.
- Consistency: There is a value y such that every party $j \in \overline{Z}$ decides on the value $y_j = y$.

We describe a specification $\mathcal{C}_{k,l,Z,t}$ that models consensus, secure up to t dishonest parties, which starts at round k and ends at round l. If $|Z| \leq t$:

$$\mathcal{C}_{k,l,Z,t} \coloneqq \Bigg\{ \mathbf{R} \in \Phi \;\Bigg|\; \Big(\exists v\ \forall i \in \overline{Z}\ y_i^{l.b} = v\Big) \wedge \Big(\exists v\ \forall j \in \overline{Z}\ x_j^{k.a} = v \rightarrow \forall j \in \overline{Z}\ y_j^{l.b} = x_j^{k.a}\Big) \Bigg\}$$

And when $|Z| > t$, $\mathcal{C}_{k,l,Z,t} = \Phi$.

Protocol $\Pi^k_{\texttt{cons}} = (\pi_1^{\texttt{cons}}, \ldots, \pi_n^{\texttt{cons}})$ constructs $\mathcal{C}_{k,k+3(t+1)-1,Z,t}$ from specification $[\mathcal{KC}_{k,k+2,Z,t,1}, \ldots, \mathcal{KC}_{k+3t,k+3(t+1)-1,Z,t,t+1}]$. The idea is simply to execute the king consensus protocol sequentially $t+1$ times with different kings. More concretely, at round $k+3j$, $j \in [0,t]$, parties execute the king consensus protocol, where the king is $j+1$. If parties start with the same input bit, validity of king consensus guarantees that this bit is kept until the end. Otherwise, since the number of dishonest parties is at most t, one of the executions has an honest king. After the execution with the honest king, consistency is reached, and validity ensures that consistency is maintained until the end of the execution.

Converter π_i^{cons}

Local Variable: y.

On input x at round k, $y \leftarrow x$.
for $j = 0$ to t **do**
  Output y at `in.kc` at round $k + 3j$. // Output to $\mathcal{KC}_{k+3j,k+3j+2,Z,t,j+1}$
  On input x' at `in.kc` at round $k + 3j + 2$, set $y \leftarrow x'$.
Output y at `out`.

Theorem B.1.4. *Let $t < n$. Π^k_{cons} constructs $\mathcal{C}_{k,k+3t+2,Z,t}$ from specification $[\mathcal{KC}_{k,k+2,Z,t,1}, \ldots, \mathcal{KC}_{k+3t,k+3t+2,Z,t,t+1}]$, for any $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$, and constructs Φ otherwise.*

Proof. Let $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$. We divide two cases:

- If every party $i \in \overline{Z}$ had as input value b at round k (there was pre-agreement): After each input to $\mathcal{KC}_{k+3j,k+3j+2,Z,t,j+1}$, the parties obtain the bit b because of validity. This is the same in $\mathcal{C}_{k,k+3t+2,Z,t}$ by definition.

- Otherwise, given that there are up to t dishonest parties and there are $t + 1$ different kings, there is an honest king K. The output of any system in the specification $\mathcal{KC}_{k+3(K-1),k+3K-1,Z,t,K}$ is the same value v for all honest parties because of the king consistency. All the following invocations to king consensus keep the value v as the output because of the validity property. Thus, all parties decide on the same output.

□

B.1.5 Broadcast

In Section 5.5.2 we introduced a broadcast resource specification. We show how to achieve such a specification from $\mathcal{C}_{k,l,Z,t}$, as long as $|Z| \leq t$, for any $t \leq \frac{n}{3}$.
We recall the broadcast specification resource secure up to t dishonest parties, which starts at round k and ends at round l. If $|Z| \leq t$:

$$\mathsf{BC}_{k,l,Z,t} \coloneqq \left\{ \mathbf{R} \in \Phi \;\middle|\; \exists v \Big[\Big(\forall j \in \overline{Z}\ y_j^{l.b} = v \Big) \wedge \Big(s \in \overline{Z} \to v = x_s^{k.a} \Big) \Big] \right\}$$

And when $|Z| > t$, $\mathsf{BC}_{k,l,Z,t} = \Phi$.

Protocol $\Pi^k_{\mathsf{bc}} = (\pi_1^{\mathsf{bc}}, \ldots, \pi_n^{\mathsf{bc}})$ constructs specification $\mathsf{BC}_{k,k+3t+3,Z,t}$ from the assumed specification $[\mathcal{C}_{k+1,k+3t+3,Z,t}, \mathcal{N}_Z]$. The sender simply sends its input value x to every party, and then parties execute the consensus protocol on the received value from the sender.

Theorem B.1.5. *Let $t < \frac{n}{2}$. Π^k_{bc} constructs $\mathsf{BC}_{k,k+3t+3,Z,t}$ from specification $[\mathcal{C}_{k+1,k+3t+3,Z,t}, \mathcal{N}_Z]$, for any $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$, and constructs Φ otherwise.*

Proof. Let $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$. We divide two cases:

- If the sender is honest, every honest party receives the sender's input x_s and inputs this value into the consensus resource. Because of the validity of consensus, every honest party obtains x_s from the consensus resource and outputs it. This is the same in $\mathsf{BC}_{k,k+3t+3,Z,t}$ by definition.
- Otherwise, the consistency of the consensus resource guarantees that every honest party receives the same value from the consensus resource, and hence every honest party outputs the same value.

□

As a corollary of composing all the previous protocols, we obtain that there is a protocol which constructs broadcast from a network of bilateral channels.

Corollary B.1.6. *Let $t < \frac{n}{3}$. There exists a protocol that constructs $\mathsf{BC}_{k,k+3t+3,Z,t}$ from $\mathcal{N}_Z$, for any $Z \subseteq \mathcal{P}$ such that $|Z| \leq t$, and constructs Φ otherwise.*

Part II

MPC with Enhanced Security Guarantees

Chapter 6

Universal Composability Framework

6.1 Overview

In this part, the results are proven in the universal composability (UC) framework introduced by Canetti [Can01]. As many other composable frameworks, it follows the real/ideal paradigm. All entities, including parties and the adversary, are modelled via interactive Turing machines (ITMs).

The goal of a protocol is to emulate an *ideal functionality*, which models a trusted party that receives inputs and provides outputs to the parties. Intuitively, a protocol is proven secure if one shows that for any attack that an adversary can perform in the real protocol, one can construct a corresponding ideal adversary which can perform the same attack in the ideal world via what is called the simulator. The simulator runs in the ideal world, interacting only with the ideal functionality and the real adversary, and has to be such that the distributions of messages seen in the real world and ideal world executions are indistinguishable from the point of view of an external entity called *the environment*. The environment has total control over the adversary, and can choose the inputs, and see the outputs of all parties.

6.1.1 Real and Hybrid Worlds

In the real-world, the protocol π, consisting of a set of ITMs, one per party, interacts with two additional ITM entities: the environment machine $\mathcal{Z}$, and the adversary machine $\mathcal{A}$.

The environment $\mathcal{Z}$ represents the interaction via inputs given to the protocol and outputs received from it, i.e., can pass inputs to and read outputs from the parties. The adversary $\mathcal{A}$ represents information leakage from the protocol execution. The adversary can also issue corruption commands, via which one can model the specific type of corruption. Importantly, the corruption commands are externalized, meaning that the environment can check which parties are corrupted at any point.

Protocol Execution. The execution consists of a sequence of activations. At any point, only one entity is activated. First, $\mathcal{Z}$ is activated. From then, machines take turns in the execution, where the activated machine performs an instruction to transmit information to another machine, and loses its activation token when doing so. There are different ways to transmit information. The machine can write an output on its output tape, give input to a sub-routine machine, or pause without sending information (in which case, the environment is activated). The execution ends when the environment $\mathcal{Z}$ halts.

Let us assume that the output of $\mathcal{Z}$ is a bit, and let us denote by $\mathbf{REAL}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$ the output distribution of the environment $\mathcal{Z}$ in the real world execution of protocol Π, with the set $\mathcal{P}$ of parties participating in protocol Π and real-world adversary $\mathcal{A}$, and κ is the security parameter and z is the auxiliary input to $\mathcal{Z}$.

Setup and Assumed Functionalities. In order to model setup, or the ideal modules a protocol assumes, the UC framework introduces so-called hybrid worlds. This is modeled by introducing ideal functionalities that are available in a protocol execution. One usually denotes the $\mathcal{G}$-hybrid world, which is identical to the real world, with the exception that parties can interact with an unbounded number of instances of $\mathcal{G}$. The interaction with the functionality is as usual: parties can send messages to it, and receive outputs from it.

Let us denote the hybrid execution of a protocol Π, which is given access to an ideal functionality $\mathcal{G}$, by $\mathbf{HYB}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}}(1^\kappa, z)$. Note that the real world can be understood as an instance of a hybrid world where only

a default network is available.[1] Without loss of generality we may make all hybrids explicit and only think about the hybrid world.

6.1.2 Ideal World

Security of a protocol is defined by comparing the protocol execution (with its assumed functionalities) to an ideal world. In the ideal world, a key ingredient is the ideal functionality. This is a single machine that models the desired functionality of a trusted party. The ideal world consists of the ideal functionality, an ideal-world adversary $\mathcal{S}$ (called the simulator), plus a set of dummy parties, which simply forward the values from the environment to the functionality and viceversa. Note that the ideal functionality is a single machine which models all aspects that one would want from a protocol execution.

The execution proceeds similarly as in the real-world. We denote the output distribution of $\mathcal{Z}$ when interacting with the ideal world as $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(1^\kappa, z)$, where $\mathcal{F}$ is the ideal functionality and $\mathcal{S}$ is the simulator.

6.1.3 Security of a Protocol

Security of a protocol is then defined by comparing the real world with the ideal world. As pointed out above, we consider the security statement rather directly for a hybrid world.

Definition 6.1.1. A protocol Π UC-securely realizes an ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model if for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that for any PPT environment $\mathcal{Z}$, it holds that:

$$\mathbf{HYB}^{\mathcal{G}}_{\Pi,\mathcal{A},\mathcal{Z}} \approx_c \mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}.$$

The composition theorem provides security guarantees when protocols are composed in an arbitrary way. This means that if ρ is a UC-secure protocol realizing $\mathcal{G}$, then the protocol Π in the $\mathcal{G}$-hybrid model can be replaced by the composition $\Pi \circ \rho$. Informally, the composition theorem then guarantees that $\mathbf{REAL}_{\Pi\circ\rho,\mathcal{A},\mathcal{Z}}$ is indistinguishable from $\mathbf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

[1] In the default UC framework, there is a built-in asynchronous network, without guaranteed delivery.

6.1.4 Synchronous and Asynchronous Models

The default real-world in the UC framework considers by default an asynchronous network communication model, where the adversary is allowed to drop messages. In Chapters 7, 8 and 9 we are, however, interested in a synchronous model and an asynchronous model with *eventual delivery*. For such settings, proper extended hybrid UC-functionalities have been made in the literature. We include some of the references to synchronous UC [KMTZ13, Can01, LLM+20b], and asynchronous UC with eventual delivery [CGHZ16, LLM+20b].

While Chapters 9 and 8 use a purely synchronous model, and therefore one can consider a synchronous model similar to the one introduced by Katz et al. [KMTZ13], Chapter 7 makes statements for a model where parties have access to clocks, and parties have access to a network with eventual delivery.[2]

In the following, we introduce a clock functionality and a network functionality that is enough to capture synchronous, partially synchronous and asynchronous with eventual delivery models. This is based on the published work [LLM+20b].

Communication Network and Clocks

We borrow ideas from a standard model for UC synchronous communication [KMTZ13, KZZ16]. Parties have access to functionalities and global functionalities [CDPW07]. More concretely, parties have access to a synchronized global clock functionality $\mathcal{G}_{\text{CLK}}$, and a network functionality $\mathcal{N}^{\delta}$ of pairwise authenticated channels with an unknown upper bound on the message delay δ.

At a high level, the model captures the two guarantees that parties have in the synchronous model of communication. First, every party must be activated each clock tick, and second, every party is able to perform all its local computation before the next tick. Both guarantees are captured via the clock functionality $\mathcal{G}_{\text{CLK}}$. It maintains the global time τ, initially set to 0, and a round-ready flag $d_i = 0$, for each party P_i. Each clock tick, $\mathcal{G}_{\text{CLK}}$ sets the flag to $d_i = 1$ whenever a party sends a confirmation (that it is ready) to the clock. Once the flag is set for every honest party,

[2] Note that the model by [CGHZ16] introduces a network with eventual delivery, but parties do not have access to clocks.

the clock counter is increased and the flags are reset to 0 again. This ensures that all honest parties are activated in each clock tick.

Functionality $\mathcal{G}_{\text{CLK}}$

The clock functionality stores a counter τ, initially set to 0. For each honest party P_i it stores flag d_i, initialized to 0.

ReadClock:

1: On input (ReadClock), return τ.

Ready:

1: On input (ClockReady) from honest party P_i set $d_i = 1$ and notify the adversary.

ClockUpdate:

Every activation, the functionality runs the following code before doing anything else:

1: **if** for every honest party P_i it holds $d_i = 1$ **then**
2: Set $d_i = 0$ for every honest party P_i and $\tau = \tau + 1$.

As mentioned above, the UC standard communication network does not consider any delivery guarantees. Hence, we consider the functionality $\mathcal{N}^{\delta}$ which models a complete network of pairwise authenticated channels with an upper bound δ parameter corresponding to an upper bound delay in the network.[3]

The network keeps track of the activations, so one can think that the network knows at any point in time the actual time. The network works in a *fetch-based* mode: parties need to actively query for the messages in order to receive them. For each message m sent from P_i to P_j, $\mathcal{N}$ creates a unique identifier $\mathtt{id}_m$ for the tuple $(T_{\mathtt{init}}, T_{\mathtt{end}}, P_i, P_j, m)$. This identifier is used to refer to a message circulating the network in a concise way. The field $T_{\mathtt{init}}$ indicates the time at which the message was sent,

[3] If $\delta = \Delta$ is set to the a worst-case assumed public network delay, this corresponds to a synchronous network. In the case where δ is unknown, this may model a real network upper bound that parties do not know. An asynchronous network can be understood as $\delta = \infty$.

whereas $T_{\texttt{end}}$ is the time at which the message is made available to the receiver. At first, the time $T_{\texttt{end}}$ is initialized to $T_{\texttt{init}} + 1$.

Whenever a new message is input to the buffer of $\mathcal{N}$, the adversary is informed about both the content of the message and its identifier. It is then allowed to modify the delivery time $T_{\texttt{end}}$ by any finite amount. For that, it inputs an integer value T along with some corresponding identifier $\texttt{id}_m$ with the effect that the corresponding tuple $(T_{\texttt{init}}, T_{\texttt{end}}, P_i, P_j, m)$ is modified to $(T_{\texttt{init}}, T_{\texttt{end}} + T, P_i, P_j, m)$. Moreover, to capture that there is an upper bound on the delay of the messages, the network does not accept more than δ accumulated delay for any identifier $\texttt{id}_m$. That is, $\mathcal{N}$ checks that $T_{\texttt{end}} \leq T_{\texttt{init}} + \delta$. Also, observe that the adversary has the power to schedule the delivery of messages: we allow it to input delays more than once, which are added to the current amount of delay. If the adversary wants to deliver a message during the next activation, it can input a negative delay.

Functionality $\mathcal{N}^{\delta}$

It is parameterized by a positive constant δ. It also stores the current time τ and keeps a buffer of messages `buffer` which initially is empty.
The functionality keeps track of the current time and updates τ accordingly.

Message transmission:

1: At the onset of the execution, output δ to the adversary.
2: On input (SEND, i, j, m) from party P_i, $\mathcal{N}$ creates a new identifier $\texttt{id}_m$ and records the tuple $(\tau, \tau+1, P_i, P_j, m, \texttt{id}_m)$ in `buffer`. Then, it sends the tuple $(\text{SENT}, P_i, P_j, m, \texttt{id}_m)$ to the adversary.
3: On input $(\text{FETCHMESSAGES}, i)$ from P_i, for each message tuple $(T_{\texttt{init}}, T_{\texttt{end}}, P_k, P_i, m, \texttt{id}_m)$ from `buffer` where $T_{\texttt{end}} \leq \tau$, the functionality removes the tuple from `buffer` and outputs (k, m) to P_i.
4: On input $(\text{DELAY}, D, \texttt{id})$ from the adversary, if there exists a tuple $(T_{\texttt{init}}, T_{\texttt{end}}, P_i, P_j, m, \texttt{id}')$ in `buffer` such that $\texttt{id}' = \texttt{id}$ and $T_{\texttt{end}} + D \leq T_{\texttt{init}} + \delta$, then set $T_{\texttt{end}} = T_{\texttt{end}} + D$ and return (DELAY-OK) to the adversary. Otherwise, ignore the message.

Chapter 7

Synchronous MPC with Asynchronous Fallback

7.1 Introduction

One can very roughly classify the results in MPC according to the underlying communication model. The *synchronous* model assumes that there is some parameter Δ known to all parties such that whenever a party sends a message, the recipient is guaranteed to receive it within time at most Δ. It is possible to achieve very strong security guarantees in this model; for example, prior work has shown how to achieve MPC with *full security*, where parties are guaranteed to obtain the correct output, for up to $t_s < \frac{n}{2}$ corruptions (e.g. [GMW87, RB89, BIB89, BMR90, CDN01, DN03, DI05]). However, one can argue that the synchrony assumption is too strong: if an honest party P doesn't manage to send a message within Δ delay, it is considered dishonest in the synchronous model. As a consequence, synchronous protocols generally lose all security guarantees (e.g., parties can jointly reconstruct P's secret-shared input) if the network delays are greater than expected. This is of particular concern in real-world deployments, where it may not be possible to guarantee ideal network conditions at all times.

In the asynchronous model, there is no assumptions on delay upper bounds, so the network delay can be arbitrarily large. The asynchronous

model is therefore a safe choice for modeling even the most unpredictable real-world networks; however, prior work has shown that optimal security guarantees in this model are necessarily weaker than in the synchronous model: MPC can be achieved in the asynchronous model only for $t_a < \frac{n}{3}$ corruptions, and the output is not guaranteed to take into account all inputs into the computation [BCG93, BKR94, HNP05, BH07, CGHZ16, Coh16].

In this chapter, we investigate MPC protocols that keep strong security guarantees under both communication models. More specifically, we ask the following question:

Is there a protocol for MPC that is secure against t_s corruptions under a synchronous network, and t_a corruptions under an asynchronous network?

We completely answer this question by showing tight feasibility and impossibility results:

Feasibility result. We give an MPC protocol that is fully secure up to t_s corruptions under a synchronous network and up to t_a corruptions under an asynchronous network, for any $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ satisfying $t_a + 2t_s < n$. The number of inputs taken into account in the latter case is $n - t_s$.

Note that for the regime where $t_s < \frac{n}{3}$, existing asynchronous MPC protocols (e.g. [HNP05]) already achieve such security guarantees, i.e., are fully secure under an asynchronous network (and hence also a synchronous network), and moreover take into account all inputs when the network is synchronous.

Optimality of our protocol. We show that our protocol is tight with respect to both the threshold tradeoffs t_a and t_s, and also the number of inputs taken into account. More concretely, we show:

- For any t_s, any MPC protocol which achieves full security up to t_s corruptions under a synchronous network cannot take into account more than $n - t_s$ inputs when run over an asynchronous network, even if all parties are guaranteed to be honest in this case.
- For any $t_a + 2t_s \geq n$, there is no MPC protocol which gives full security up to t_s corruptions under a synchronous network, and

where all parties output the same value up to t_a corruptions under an asynchronous network.

7.1.1 Technical Overview

In this section, we briefly sketch our protocol for MPC that achieves full security up to t_s corruptions under a synchronous network and up to t_a corruptions under an asynchronous network, for any $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ satisfying $t_a + 2t_s < n$.

At a very high level, we run two sub-protocols $\Pi_{\texttt{smpc}}$ and $\Pi_{\texttt{ampc}}$ one after the other, where $\Pi_{\texttt{smpc}}$ is a t_s-secure synchronous protocol and $\Pi_{\texttt{ampc}}$ is a t_a-secure asynchronous protocol. Conceptually, a key challenge is that parties are not able to obtain output in both protocols, as this would violate privacy. Thus, parties need to agree on whether to run the second sub-protocol. For that, the key is that the protocol $\Pi_{\texttt{smpc}}$ gives some guarantees even when the network is asynchronous. More concretely, $\Pi_{\texttt{smpc}}$ achieves unanimous abort up to t_a corruptions under an asynchronous network. Intuitively, this means that the protocol is secure, except the fact that either all parties learn the correct output, or all parties obtain $\perp$ as the output.

When the network is synchronous, security of the overall protocol is inherited from the first sub-protocol. In the case where the network is asynchronous, parties either learn the correct output from the first sub-protocol or all parties obtain $\perp$ and can safely execute the second sub-protocol.

Synchronous MPC with Asynchronous Unanimous Abort. In order to construct the first sub-protocol, we modify a synchronous MPC protocol that uses threshold homomorphic encryption [CDN01, DN03]. The original protocol provides full security up to $t_s < \frac{n}{2}$ corruptions in a synchronous network.

Let us briefly recall the high-level structure of the original protocol [CDN01, DN03]. The protocol is based on a threshold version of the Paillier cryptosystem [Pai99]. For a plaintext a, let us denote $\overline{a}$ an encryption of a. The cryptosystem is homomorphic: given encryptions $\overline{a}$, $\overline{b}$, one can compute an encryption of $a+b$, which we denote $\overline{a} \boxplus \overline{b}$. Similarly, from a constant plaintext α and an encryption $\overline{a}$ one can compute an encryption of αa, which we denote $\alpha \boxdot \overline{a}$.

The protocol starts by having each party publish encryptions of its input values, as well as zero-knowledge proofs that it knows these values. Then, parties compute addition and multiplication gates to obtain a common ciphertext, which they jointly decrypt using threshold decryption. Any linear operation (addition or multiplication by a constant) can be performed non-interactively, due to the homomorphism property of the threshold encryption scheme. Given encryptions $\overline{a}$, $\overline{b}$ of input values to a multiplication gate, parties can compute an encryption of $c = ab$ as follows:

1. Each P_i chooses a random $d_i \in \mathbf{Z}_n$ and uses a Byzantine broadcast protocol to distribute encryptions $\overline{d_i}$ and $\overline{d_i b}$.
2. Parties prove (in zero-knowledge) knowledge of the plaintext of $\overline{d_i}$ and that $\overline{d_i b}$ encrypts the correct value. Let S be the subset of parties succeeding in both proofs.
3. Parties compute $\overline{y} = \overline{a} \boxplus (\boxplus_{i \in S} \overline{d_i})$ and decrypt it using a threshold decryption.
4. Parties set $\overline{c} = y \boxdot \overline{b} \boxminus ((\boxplus_{i \in S} \overline{d_i b}))$.

Intuitively, the protocol works because 1) honest parties have agreement on the ciphertext to decrypt after evaluating the circuit, and 2) only ciphertexts or random values are revealed.

When the above protocol is executed over an asynchronous network, all security guarantees are lost. This is because synchronous broadcast protocols do not necessarily give any guarantees when run over an asynchronous network. As a result, parties lose agreement in critical points in the protocol. For example, parties can receive different sets of encrypted inputs during input distribution, which can lead to privacy violations if the mismatching inputs are decrypted. Moreover, parties must reach agreement on S, and S must contain at least one honest party contributing to the reconstructed random value to ensure that the value is random and unknown to the adversary. For this, it is essential that parties have agreement on whether a zero-knowledge proof was successful or not. Finally, parties need to reach agreement on which ciphertext to decrypt, or whether to decrypt at all.

To solve the problems above, we replace the problematic sub-protocols with versions that achieve certain guarantees even when the network is

asynchronous. More concretely, we will make use of broadcast, Byzantine agreement and asynchronous common subset sub-protocols. The broadcast protocol will ensure that encrypted inputs from honest parties can only lead to correct ciphertexts. When used with the Byzantine agreement protocol proposed in [BKL19], it will allow parties to reach agreement on the set S for the multiplication gates. Finally, we make use of the enhanced asynchronous common subset sub-protocol in [BKL20] at the end of the circuit computation to decide whether or not parties should proceed to decrypt a ciphertext, or output $\perp$.

7.1.2 Related Work

Despite being a very natural direction of research, protocols resilient to both synchronous and asynchronous networks have only begun to be studied in relatively recent works. The closest related work is the recent work by Blum et al. [BKL19] which considers the problem of Byzantine agreement in a 'hybrid' network model. The authors prove that Byzantine agreement t_s-secure under a synchronous network and t_a-secure under an asynchronous network is possible if and only if $t_a + 2t_s < n$. The work was recently further extended to the problem of state-machine replication [BKL20]. Our work extends both above works to the problem of secure multi-party computation, and in particular, introduces techniques to protect privacy of inputs in the hybrid network setting.

Another close related work is the work by Guo et al. [GPS19], which considers a weakened variant of the classical synchronous model. Here, an attacker can temporarily disconnect a subset of parties from the rest of the network. Guo et al. gave Byzantine agreement and multi-party computation protocols tolerating the optimal corruption threshold in this model, and Abraham et al. [AMN+19] achieve similar guarantees for state-machine replication. The main difference between these works and ours is that their protocols need to assume synchrony in part of the network. In contrast, our protocols give guarantees even if the network is fully asynchronous.

Further related work for the problem of Byzantine agreement protocols include the work by Malkhi et al. [MNR19] which considers protocols that provide guarantees when run in synchronous or partially synchronous networks, and the work by Liu et al. [LVC+16] which designs protocols resilient to malicious corruptions in a synchronous network, and

fail-stop corruptions in an asynchronous network. Kursawe [Kur02] shows a protocol for asynchronous Byzantine agreement that reaches agreement more quickly in case the network is synchronous.

A line of works [PS17a, PS18, LM18, LLM+20b] has recently investigated protocols that achieve *responsiveness*. These protocols operate under a synchronous network, but in addition give the guarantee that parties obtain output as fast as the actual network delay allows. None of these works provide security guarantees when the network is not synchronous.

7.2 Model

Our protocols are proven secure in the universally composable (UC) framework [Can01] (see Section 6.1 for a summary).

7.2.1 Setup

We consider a setting with n parties $\mathcal{P} = \{P_1, \ldots, P_n\}$. We denote κ the security parameter.

Common reference string. We assume that the parties have a common reference string (CRS). The CRS is used to realize the bilateral zero-knowledge UC functionalities.

Digital signatures. We assume that parties have a public-key infrastructure available, i.e., all parties hold the same vector of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, and each party P_i holds the secret key sk_i associated with pk_i. This allows parties to sign values.

We require that the signature scheme is correct and unforgeable against chosen message attacks.

Threshold encryption. We assume that parties have a threshold additively homomorphic encryption setup available. That is, it provides to each party P_i a global public key ek and a private key share dk_i.

Such a threshold encryption scheme can be based on, for example, the Paillier cryptosystem [Pai99] (see Section C.1). We use the threshold encryption scheme as a basic tool in the MPC protocol, following the approach in [CDN01, DN03].

7.2.2 Communication Network and Adversary

We consider a complete network of authenticated channels. Our protocols operate in two possible settings: synchronous or asynchronous.

In the synchronous setting, all parties have access to synchronized clocks and all messages are guaranteed to be delivered within some known upper bound delay Δ. Within Δ, the adversary can schedule the messages arbitrarily. In particular, the adversary is *rushing*, i.e., within the same round, the adversary is allowed to send its messages after seeing the honest parties' messages. Sometimes it is convenient to describe a protocol in rounds, where each round r refers to the interval of time $(r-1)\Delta$ to $r\Delta$. In such case, we say that a party receives a message in round r if it receives the message within that time interval. Moreover, we say a party sends a message in round r when it sends the message at the beginning of the round, i.e., at time $(r-1)\Delta$.

In the asynchronous setting, both assumptions above are removed. That is, parties do not have access to synchronized clocks, and the adversary is allowed to arbitrarily schedule the delivery of the messages. However, we assume that all messages are eventually delivered (i.e., the adversary cannot drop messages). Note that the model introduced in Section 6.1.4 assumes synchronized clocks for the parties, but one can also model similarly local clocks that are not synchronized. In the asynchronous setting, our protocols only need that the clocks advance.

We consider a static adversary who corrupts parties in an arbitrary manner at the beginning of the protocol.

7.3 Definitions

7.3.1 Broadcast

Broadcast allows a designated party called the *sender* to consistently distribute a message among a set of parties.

Definition 7.3.1. (Broadcast) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender P_s initially holds an input v, and parties terminate upon generating output.

- Validity: Π is t-valid if the following holds whenever up to t parties are corrupted: if P_s is honest, then every honest party which

outputs, outputs v.

- Weak-validity: Π is t-weakly valid if the following holds whenever up to t parties are corrupted: if P_s is honest, then every honest party which outputs, outputs v or $\perp$.
- Consistency: Π is t-consistent if the following holds whenever up to t parties are corrupted: every honest party which outputs, outputs the same value.
- Liveness: Π is t-live if the following holds whenever up to t parties are corrupted: every honest party outputs a value.

If Π is t-valid, t-consistent and t-live, we say that it is t-secure.

In the asynchronous setting, one can formally prove that the strong broadcast guarantees as in Definition 7.3.1 cannot be achieved [Bra87, BT85]. Intuitively, the reason is that one cannot distinguish between a dishonest sender not sending messages, or an honest sender's messages being delayed. Hence, a useful primitive is a *reliable broadcast* protocol, which achieves the same guarantees as a broadcast protocol, except that the liveness property is relaxed and divided into two properties.

Definition 7.3.2. (Reliable Broadcast) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender P_s initially holds an input v, and parties terminate upon generating output.

- Validity: Π is t-valid if the following holds whenever up to t parties are corrupted: if P_s is honest, then every honest party outputs v.
- Consistency: Π is t-consistent if the following holds whenever up to t parties are corrupted: either no honest party terminates, or else all honest parties output the same value.

Observe that, in contrast to Definition 7.3.1, when the sender is dishonest, it is allowed that no honest party terminates.

7.3.2 Byzantine Agreement

In a Byzantine agreement protocol, each party P_i starts with a value v_i. The protocol allows the set of parties to agree on a common value. The

achieved guarantees are the same as in broadcast (see Definition 7.3.1), except that validity is adapted accordingly.

Definition 7.3.3. (Byzantine Agreement) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i initially holds an input v_i, and parties terminate upon generating output.

- Validity: Π is t-valid if the following holds whenever up to t parties are corrupted: if every honest party has the same input value v, then every honest party that outputs, outputs v.
- Consistency: Π is t-consistent if the following holds whenever up to t parties are corrupted: every honest party which outputs, outputs the same value.
- Liveness: Π is t-live if the following holds whenever up to t parties are corrupted: every honest party outputs a value.

If Π is t-valid, t-consistent and t-live, we say that it is t-secure.

7.3.3 Asynchronous Common Subset

A protocol for the asynchronous common subset (ACS) problem [BCG93, BKR94, Can96, MXC+16] allows n parties, each with an initial input, to agree on a subset of the inputs. For this primitive, we do not assume that parties terminate upon generating output, that is, even after generating output parties are allowed to keep participating in the protocol indefinitely.

Definition 7.3.4. (ACS) Let Π be a protocol that is executed by parties $P_1, \ldots, P_n$, where each party initially holds an input v, and parties output sets of size at most n.

- Validity: Π is t-valid if the following holds whenever up to t parties are corrupted: if all honest parties start with the same input v, then every honest party which outputs, outputs $\{v\}$.
- Consistency: Π is t-consistent if the following holds whenever up to t parties are corrupted: every honest party which outputs, outputs the same set.

- Liveness: Π is t-live if the following holds whenever up to t parties are corrupted: every honest party outputs.
- Validity liveness: Π is t-live valid if the following holds whenever up to t parties are corrupted: If all honest parties start with the same input, then every honest party outputs.
- Set quality: Π has (t, h)-set quality if the following holds whenever up to t parties are corrupted: if an honest party outputs a set, it contains the inputs of at least h honest parties.

7.3.4 Multi-Party Computation

At a high level, a protocol for multi-party computation (MPC) allows n parties $P_1, \ldots, P_n$, where each party P_i has an initial input x_i, to jointly compute a function over the inputs $f(x_1, \ldots, x_n)$ in such a way that nothing beyond the output is revealed.

We consider different types of security guarantees for our MPC protocols. The first one is the strongest guarantee that an MPC protocol can offer: MPC with guaranteed output delivery, or full security (cf. [GMW87, BGW88, CCD88, RB89, BMR90, CDN01]). Here, honest parties are guaranteed to obtain the correct output. Formally, in UC this is modeled as the protocol realizing the ideal functionality where each party P_i inputs x_i to the functionality, and it then outputs $f(x_1, \ldots, x_n)$ to the parties.

When the network is asynchronous, it is provably impossible that the computed function takes into account all inputs from honest parties [BCG93, BKR94, HNP05, BH07, CGHZ16, Coh16]. The reason is that one cannot distinguish between a dishonest party not sending its input, or an honest party's input being delayed. Hence, we say that a protocol achieves ℓ-output quality, if the output to be computed contains the inputs from at least ℓ parties. Traditional asynchronous protocols in the literature (e.g. [BCG93, BKR94, HNP05]) achieve $(n - t)$-output quality under t corruptions, since the computed output ignores up to t inputs. Formally this is modelled in the ideal functionality as allowing the ideal adversary to choose a subset S of ℓ parties. The functionality then computes $f(x_1, \ldots, x_n)$, where $x_i = v_i$ is the input of P_i in the case that $P_i \in S$, and otherwise $x_i = \perp$.

Functionality $\mathcal{F}_{\text{SFE}}^{\text{sec},\ell}$

$\mathcal{F}_{\text{SFE}}$ is parameterized by a set $\mathcal{P}$ of n parties and a function $f : (\{0,1\}^* \cup \{\bot\})^n \to (\{0,1\}^*)^n$. For each $P_i \in \mathcal{P}$, initialize the variables $x_i = y_i = \bot$. Set $S = \mathcal{P}$.

1: On input (INPUT, v) from $P_i \in \mathcal{P}$, set $x_i = v$ and send a message (INPUT, P_i) to the adversary.
2: On input $(\text{OUTPUTSET}, S')$ from the ideal adversary, where $S' \subseteq \mathcal{P}$ and $|S'| \geq \ell$, set $S = S'$.
3: Once all input variables from honest parties in S have been stored, set each $y_i = f(x'_1, \ldots, x'_n)$, where $x'_i = x_i$ for each $P_i \in S$, and otherwise $x'_i = \bot$. Ignore further instructions from Steps 1 and 2.
4: On input (GETOUTPUT) from P_i, output (OUTPUT, y_i) to P_i.

In addition to MPC with full security, we also consider weaker notions of security. In MPC with selective abort [IOZ14, CL17], the ideal world adversary can choose any subset of parties to receive $\bot$, instead of the correct output. The last type of security we consider is called MPC with unanimous abort [GMW87, FGH+02]. Under this definition, the adversary is permitted to choose whether all honest parties receive the correct output or all honest parties receive $\bot$ as output; as such it is is slightly stronger than MPC with selective abort, but weaker than full security.

Let us denote the functionality $\mathcal{F}_{\text{SFE}}^{\texttt{sout},\ell}$ (resp. $\mathcal{F}_{\text{SFE}}^{\texttt{uout},\ell}$), the above functionality, where the adversary can selectively choose any subset of parties to obtain $\bot$ as the output (resp. choose that either all honest parties receive $f(x_1, \ldots, x_n)$ or $\bot$).

Definition 7.3.5. A protocol π achieves full security (resp. selective abort; unanimous abort) with ℓ output-quality if it UC-realizes functionality $\mathcal{F}_{\text{SFE}}^{\texttt{sec},\ell}$ ($\mathcal{F}_{\text{SFE}}^{\texttt{sout},\ell}$; $\mathcal{F}_{\text{SFE}}^{\texttt{uout},\ell}$).

Since protocols run in a synchronous network typically achieve n-output quality, we implicitly assume that all synchronous protocols discussed achieve n-output quality (unless otherwise specified).

Weak termination. In general, traditional protocols for MPC require that the protocol terminates (halts). In this chapter, we capture a slightly weaker version as a property of a protocol: we say that a protocol has weak termination, if parties are guaranteed to terminate upon receiving

an output different than $\perp$, but do not necessarily terminate if the output is $\perp$.

7.4 Synchronous MPC with Asynchronous Unanimous Abort and Weak Termination

In this section, we show a protocol $\Pi_{\texttt{smpc}}^{t_s,t_a}$ that achieves full security up to t_s corruptions when the network is synchronous, and achieves unanimous abort with weak termination up to t_a corruptions when the network is asynchronous, for any $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ satisfying $t_a + 2t_s < n$. The protocol relies on a number of primitives:

- $\Pi_{\texttt{bc}}^{t_s,t_a}$ is a broadcast protocol that is t_s-secure when run in a synchronous network, and is t_a-weakly valid and t_a-live when run in an asynchronous network.
- $\Pi_{\texttt{ba}}^{t_s,t_a}$ is a Byzantine agreement protocol that is t_s-secure when run in a synchronous network, and is t_a-secure when run in an asynchronous network.
- $\Pi_{\texttt{acs}}^{t_s,t_a}$ is an asynchronous common subset protocol that is t_s-valid and t_s-live valid when run in a synchronous network, and is t_a-consistent, t_a-live and has $(t_a, 1)$-set quality when run in an asynchronous network.
- $\Pi_{\texttt{zk}}^{t_s,t_a}$ is a multi-party zero-knowledge protocol that allows a party P_i to prove knowledge of a witness w for a statement x satisfying a certain relation R towards all parties. The protocol achieves full security up to t_s corruptions when the network is synchronous, and achieves security with selective abort up to t_a corruptions when the network is asynchronous.

In the following, we show instantiations for each of the sub-protocols.

7.4.1 Broadcast

We use the Dolev-Strong protocol [DS83, BKL19] to achieve a broadcast protocol that is t_s-secure when run in a synchronous network, and is t_a-

weakly valid and t_a-live when run in an asynchronous network. The idea is quite simple: we run the Dolev-Strong protocol for $t_s + 1$ rounds and output v if v is the only value accepted, and otherwise $\perp$. In the protocol, we say that a message (v, Σ) at round r is valid if Σ contains r signatures, where one of them is from the sender and the other $r - 1$ from distinct additional parties.

Protocol $\Pi_{\mathsf{bc}}^{t_s,t_a}$

Sender P_s has input v. Each party P_i keeps local variables $\Sigma_i, \Omega_i \coloneqq \varnothing$.

Round 1. P_s signs its input v to obtain a signature σ_s, and sends $(v, \{\sigma_s\})$ to all parties.

Round $1 \leq r \leq t_s$. Each P_i does: Upon receiving a valid message (v, Σ), add v to Ω_i. Compute a signature σ_i on v and let $\Sigma_i \coloneqq \Sigma_i \cup \{\sigma_i\}$. Send (v, Σ_i) to all parties in the next round.

Output determination

Round $t_s + 1$. Each P_i does: Upon receiving a valid message (v, Σ), add v to Ω_i. Then, if Ω_i contains exactly one value v', output v' and terminate. Otherwise, output $\perp$ and terminate.

Lemma 7.4.1. *Let n, t_s, t_a be such that $t_a, t_s < n$. $\Pi_{\mathsf{bc}}^{t_s,t_a}$ is a broadcast protocol that is t_s-secure when run in a synchronous network, and is t_a-weakly valid and t_a-live when run in an asynchronous network.*

Proof. Security under a synchronous network is achieved via the standard analysis of the Dolev-Strong protocol: If the sender is honest, each honest party P_i adds the sender's input v to Ω_i, and no honest party adds any other value. Moreover, if an honest P_i adds v to Ω at round $r \leq t_s$, every honest P_j adds v at round $r + 1$. And if P_i adds v at round $t_s + 1$, then there are $t_s + 1$ signatures on v and hence an honest P_k added v at some round $r' \leq t_s$ and every honest party added v at round $r' + 1$. If the network is asynchronous, t_a-liveness is trivial, since every honest party outputs at (local) time $(t_s + 1)\Delta$. The protocol is also t_a-weakly valid because the adversary cannot forge signatures from the sender P_s.

□

7.4.2 Byzantine Agreement

In [BKL19], the authors show a Byzantine agreement protocol that is t_s-secure when run in a synchronous network, and is t_a-secure when run in an asynchronous network. We briefly sketch the construction here.

At a high level, their protocol consists of two phases: a round-based BA followed by an event-based BA. An honest party P_i with input v_i uses v_i as their input for the round-based phase. If the round-based phase terminates with output $v' \in \{0, 1\}$ within some (local) time limit, P_i uses v' as input for the event-based phase. (The timeout is chosen such that the honest parties are guaranteed to receive output from the round-based BA before the timeout when the network is synchronous and at most t_s parties are corrupted.) Otherwise, if the round-based phase times out without producing boolean output, P_i proceeds directly to the event-based phase, using their original input v_i as their input. P_i then outputs the output they receive from the event-based phase.

Intuitively, when the network is synchronous and there are t_s corruptions, the security guarantees for the full protocol are primarily inherited from the round-based BA sub-protocol (with the caveat that the event-based BA sub-protocol guarantees t_s-validity and therefore preserves the results of the first phase). When the network is asynchronous and there are t_a corruptions, the round-based BA protocol need only be t_a-weakly valid, after which the desired security guarantees follow from the security properties of the event-based BA sub-protocol. We state the following lemma. The proof can be found in [BKL19].

Lemma 7.4.2. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2t_s < n$. There is a protocol $\Pi_{\mathsf{ba}}^{t_s,t_a}$ satisfying the following properties:*

1. *When run in a synchronous network, it is t_s-secure.*
2. *When run in an asynchronous network, it is t_a-secure.*

7.4.3 Asynchronous Common Subset

We describe the protocol $\Pi_{\mathsf{acs}}^{t_s,t_a}$ presented in [BKL20], which is an asynchronous common subset protocol that is t_s-valid and t_s-live valid when run in a synchronous network, and is t_a-consistent, t_a-live and has $(t_a, 1)$-set quality when run in an asynchronous network.

The protocol is based on previous asynchronous common subset protocols [BKR94, Can96, MXC+16], but the output decision differs. The general idea is that parties run n executions of Bracha's reliable broadcast protocol [Bra87], where each party P_i acts as the sender in each execution, followed by n executions of Byzantine agreement to agree on a subset of parties that finished the reliable broadcast protocol. If a party sees $n - t_s$ broadcasts terminate on the same value, it outputs this value. Otherwise, it waits until all Byzantine agreement protocols have terminated and then outputs based on the set C of senders for whom the corresponding BA output 1: If there is a majority v of broadcasted values from parties in C, output v, and otherwise output the union of all broadcasted values from parties in C.

In order to achieve the guarantees described above, the protocol needs a reliable broadcast protocol which, under an asynchronous network, achieves validity up to t_s corruptions, and consistency up to t_a corruptions. Let us denote $\texttt{RBC}_i$ the reliable broadcast protocol where P_i acts as the sender, and $\texttt{BA}_i$ the Byzantine agreement protocol which outputs whether $\texttt{RBC}_i$ has terminated or not.

Protocol $\Pi_{\mathsf{acs}}^{t_s,t_a}(P_i)$

1: Participate in each protocol $\texttt{RBC}_j$, $j \neq i$, as the receiver, and participate in $\texttt{RBC}_i$ as the sender.
2: On output from $\texttt{RBC}_j$, if an input has not yet been provided to $\texttt{BA}_j$, then input 1 to $\texttt{BA}_j$.
3: When $n - t_a$ of the protocols $\texttt{BA}_j$ have output 1, provide input 0 to each instance $\texttt{BA}_j$ that has not yet been provided input.

Output determination

1: **if** at least $n - t_s$ executions of $\texttt{RBC}_j$ output a value v **then**
2: Output $\{v\}$.
3: **else**
4: let $C \coloneqq \{j \mid \texttt{BA}_j \text{ output } 1\}$. Once all instances $\texttt{BA}_j$ have been completed and $|C| \geq n - t_a$, wait for the output v_j of each $\texttt{RBC}_j$, $j \in C$.
5: **if** A majority of the executions $\{\texttt{RBC}_j\}_{j \in C}$ output a value v **then**
6: Output $\{v\}$.
7: **else**
8: Output $\bigcup_{j \in C} \{v_j\}$.

We state the following lemma. The proof can be found in [BKL20].

Lemma 7.4.3. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2t_s < n$. Protocol $\Pi_{\mathsf{acs}}^{t_s,t_a}$ satisfies the following properties:*

1. *When run in a synchronous network, it is t_s-valid and t_s-live valid.*
2. *When run in an asynchronous network, it is t_a-consistent, t_a-live and has $(t_a, 1)$-set quality.*

7.4.4 Zero-Knowledge Proofs

Let us assume a binary relation R, consisting of pairs (x, w), where x is the statement, and w is a witness to the statement. A zero-knowledge proof allows a prover P to prove to a verifier V knowledge of w such that $R(x, w) = 1$. We are interested in zero-knowledge proofs for three types of relations, parameterized by a threshold encryption scheme with public encryption key ek:

1. *Proof of Plaintext Knowledge:* The statement consists of ek, and a ciphertext c. The witness consists of a plaintext m and randomness r such that $c = \mathsf{Enc}_{\mathsf{ek}}(m, r)$.
2. *Proof of Correct Multiplication:* The statement consists of ek, and ciphertexts c_1, c_2 and c_3. The witness consists of a plaintext m_1 and randomness r_1, r_3 such that $c_1 = \mathsf{Enc}_{\mathsf{ek}}(m_1, r_1)$ and $c_3 = m_1 \cdot c_2 + \mathsf{Enc}_{\mathsf{ek}}(0; r_3)$.
3. *Proof of Correct Decryption:* The statement consists of ek, a ciphertext c, and a decryption share d. The witness consists of a decryption key share dk_i, such that $d = \mathsf{Dec}_{\mathsf{dk}_i}(c)$.

Examples of bilateral zero-knowledge proofs of knowledge can be found for example in [Dam00, CDN01]. The bilateral UC zero-knowledge functionality ZK for a relation R and a pair prover P and a verifier V is defined as follows: P inputs a pair (x, w) instance-witness, and the functionality outputs (x, b) to the verifier, where $b = 1$ if and only if $R(x, w) = 1$. It is known that assuming a CRS, one can realize a bilateral UC zero-knowledge functionality ZK [CF01, DN02, Coh16].

Multi-party zero-knowledge protocols. A multi-party ZK protocol allows a prover P to prove towards all parties knowledge of a witness w

for a statement x such that $R(x, w) = 1$. The ideal functionality can be seen as a special case of secure function evaluation, where the prover inputs (x, w), and the parties obtain the statement x and 1 if and only if $R(x, w) = 1$.

Assuming a bilateral UC zero-knowledge functionality ZK, one can construct a UC multi-party zero-knowledge functionality $\mathcal{F}_{\text{MZK}}$ using so-called *certificates* [HNP05] as follows: The prover bilaterally performs the zero-knowledge proofs towards each of the recipients, who upon a successful proof, send a signature that the proof was correct. Once the prover collects a list L of $t_s + 1$ signatures, the list works as a certificate that proves non-interactively that at least one honest party accepted the proof. The prover can hence broadcast the list L to let all honest parties know that the proof is correct. If the last broadcast is executed with the protocol $\Pi_{\mathsf{bc}}^{t_s,t_a}$, it is easy to see that under t_s corruptions and a synchronous network the multi-party zero-knowledge functionality achieves full security. Moreover, if there are up to t_a corruptions and an asynchronous network, broadcast guarantees weak validity, so the protocol achieves security with selective abort (in the last step, if the prover has a certificate, it is guaranteed that parties receive the certificate or $\perp$, and a dishonest party who did not collect such certificate cannot make the parties accept the proof).

Protocol $\Pi_{\mathsf{zk}}^{t_s,t_a}$

Prover P proves knowledge of a witness w for a statement x satisfying a certain relation R towards all parties.

1: P inputs (x, w) to each bilateral ZK.
2: Each P_i does: Upon a successful proof, compute $\sigma_i = \mathsf{Sign}_{sk_i}(x)$ and send σ_i to P.
3: P collects a list L of $t_s + 1$ signatures and broadcasts using protocol $\Pi_{\mathsf{bc}}^{t_s,t_a}$ the list L.
4: Each P_i does: Upon receiving a list L as output of the broadcast protocol, if L contains $t_s + 1$ signatures on the same instance x, output $(x, 1)$. In any other case, output $\perp$.

Lemma 7.4.4. *Let R be a relation. Let n, t_s, t_a be such that $t_a, t_s < n$. $\Pi_{\mathsf{zk}}^{t_s,t_a}$ realizes the multi-party zero-knowledge functionality for P as prover with the following guarantees:*

1. *When run in a synchronous network, it achieves full security up to t_s corruptions.*

2. *When run in an asynchronous network, it achieves security with selective abort up to t_a corruptions.*

Proof. We prove each of the cases separately. We simulate in the hybrid where there is a trusted setup generating the keys in the real world. In the ideal world, the simulator $\mathcal{S}$ generates the PKI keys, and outputs the public keys to the adversary along with its secret keys.

Synchronous network and up to t_s corruptions. We describe the simulator $\mathcal{S}$ for the case where the network is synchronous and there are up to t_s corruptions. Let us first consider the case where the prover P is honest.

- $\mathcal{S}$ forwards the result from $\mathcal{F}_{\text{MZK}}$ to the adversary. If the result is positive, generate a signature σ_i on behalf of each honest party. Let L be list of signatures.

- On input correct signatures from the dishonest parties, it adds it to L.

- $\mathcal{S}$ emulates the messages of the broadcast protocol.

Now assume that P is dishonest.

- $\mathcal{S}$ gets the instance-witness pairs that P inputs to prove to each party. To the dishonest parties, output the instance and the bit 1 if and only if the witness is correct.

- For each of the pairs, forward a signature on behalf of the honest party if the witness is a correct witness to the corresponding instance.

- $\mathcal{S}$ receives a list L of $t_s + 1$ signatures on the same instance: input the instance and the witness to $\mathcal{F}_{\text{MZK}}$.

Asynchronous network and up to t_a corruptions. The only difference with respect to the case where the network is synchronous, is that

the protocol $\Pi_{\mathtt{bc}}^{t_s,t_a}$ only provides weak-validity. In the simulation, it implies that the simulator will also need to simulate the $\perp$ messages from the broadcast protocols.

It is easy to see that the simulation goes through. In the case of a synchronous network and t_s corruptions, an honest prover collects at least $t_s + 1$ signatures and every honest receiver outputs 1. In the case the prover is dishonest, it cannot collect $t_s + 1$ signatures for an instance without having succeeded in one of the proofs, and hence each honest party outputs $\perp$. If the network is asynchronous, when the prover is honest, every honest party outputs 1 or $\perp$, where the set of parties that output $\perp$ is chosen by the adversary. In the case the prover is dishonest, the case is analogous as the synchronous case and every honest party outputs $\perp$.

□

7.4.5 Description of the Synchronous MPC Protocol

We start from the MPC protocol that uses homomorphic encryption presented in [CDN01, DN03]. The protocol was originally designed for the synchronous setting and guarantees full security up to $t_s < \frac{n}{2}$ corruptions. We modify the protocol to also achieve unanimous abort up to t_a corruptions even when the network is asynchronous, as long as $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ satisfies $t_a + 2t_s < n$.

We assume that the computation is specified as a circuit with addition and multiplication gates. We assume that the plaintext space does not contain a special symbol $\perp$. For example, we can assume that the plaintext space is $\mathbf{Z}_N$ for some RSA modulus N and that we use a threshold version of the Paillier cryptosystem (see Section C.1).

When the network is synchronous, we need to ensure that parties start simultaneously in each of the sub-protocols in order to ensure that the security guarantees are preserved. For example, in $\Pi_{\mathtt{ba}}^{t_s,t_a}$ there is a timeout chosen such that honest parties are guaranteed to receive output when the network is synchronous. As a consequence, if parties start at different times, we lose the security guarantees in the synchronous case. In order to solve this, we wait at least for an upper bound on the running time of each sub-protocol. This allows parties to simultaneously start at each sub-protocol when the network is synchronous. Let us denote T_{bc}, T_{zk}, T_{ba}, T_{dec} upper bounds on the running time of $\Pi_{\mathtt{bc}}^{t_s,t_a}$, $\Pi_{\mathtt{zk}}^{t_s,t_a}$, n

parallel executions of $\Pi_{\mathtt{ba}}^{t_s,t_a}$, and the Threshold Decryption sub-protocols respectively, in the case the network is synchronous.

Protocol $\Pi_{\mathtt{smpc}}^{t_s,t_a}(P_i)$

Let x_i denote the input value of party P_i. Let $\mathtt{abort} = 0$.

Input Distribution

1: P_i computes $\overline{x_i}$ and broadcasts using $\Pi_{\mathtt{bc}}^{t_s,t_a}$ the ciphertext $\overline{x_i}$ and uses the multi-party zero-knowledge functionality $\mathcal{F}_{\text{MZK}}$ to prove knowledge of the plaintext of $\overline{x_i}$ towards all parties. Wait until $\max\{T_{bc}, T_{zk}\}$ clock ticks passed.

2: If there is a broadcast or zero-knowledge proof that has not terminated, or the number of correct encryptions received is less than $n - t_s$ inputs, set $\mathtt{abort} = 1$. Continue participating in the sub-protocols, but do not compute any ciphertext.

Addition Gates Input: $\overline{a}, \overline{b}$. Output: $\overline{c}$.

1: P_i locally computes $\overline{c} = \overline{a} \boxplus \overline{b}$.

Multiplication Gates Input: $\overline{a}, \overline{b}$. Output: $\overline{c}$.

1: P_i chooses a random plaintext d_i and broadcasts using $\Pi_{\mathtt{bc}}^{t_s,t_a}$ the ciphertexts $\overline{d_i}$ and $\overline{d_i b}$ and uses the multi-party zero-knowledge functionality $\mathcal{F}_{\text{MZK}}$ to prove knowledge of d_i and that $\overline{d_i b}$ is a correct encryption of the multiplication. Wait for $\max\{T_{bc}, T_{zk}\}$.

2: Let S_i be the subset of the parties succeeding with both proofs. Run n times the protocol $\Pi_{\mathtt{ba}}^{t_s,t_a}$, each one to decide for each party P_j's proof. Input 1 to party j's BA if and only if $j \in S_i$. Wait for T_{ba}. // Crucial to agree on the same S, otherwise privacy breaks.

3: Let S be the subset of the parties for which $\Pi_{\mathtt{ba}}^{t_s,t_a}$ outputs 1.

4: **if** $|S| > t_s$ **then**

5: P_i computes $\overline{a} \boxplus \left(\boxplus_{i \in S} \overline{d_i}\right)$. P_i executes the Threshold Decryption sub-protocol on this ciphertext. Wait for T_{dec}.

6: P_i learns $a + \sum_{i \in S} d_i$ and computes $\overline{c} = \left(a + \sum_{i \in S} d_i\right) \boxdot \overline{b} \boxminus \left(\boxplus_{i \in S} \overline{d_i b}\right)$.

7: **else**

8: Set $\mathtt{abort} = 1$.

Output Determination Input x, where $x = c_i$ is the output ciphertext of the circuit if $\mathtt{abort} = 0$, and otherwise $x = \perp$.

1: P_i executes the protocol $\Pi_{\mathsf{acs}}^{t_s,t_a}$ with x as input. Let S_i be the output of the protocol.
2: **if** $S_i = \{c\}$ **then**
3: Execute the Threshold Decryption sub-protocol on c.
4: After an output is given, terminate.
5: **else**
6: Output $\perp$. // Observe that parties do not terminate, since $\Pi_{\mathsf{acs}}^{t_s,t_a}$ does not guarantee termination.

Threshold Decryption Input: ciphertext c.

1: P_i computes its decryption share s_i sends it to every other party.
2: P_i proves that the value s_i is a correct decryption share of c bilaterally.
3: Once t_s+1 correct decryption shares are collected, send the list to every party and output the corresponding plaintext.

Theorem 7.4.5. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2t_s < n$. Protocol $\Pi_{\mathsf{smpc}}^{t_s,t_a}$ satisfies the following properties:*

1. *When run in a synchronous network, it achieves full security up to t_s corruptions.*
2. *When run in an asynchronous network, it achieves unanimous abort with weak termination up to t_a corruptions and has $n - t_s$ output quality.*

Proof. We prove each of the cases individually. We simulate in the hybrid where there is a trusted setup generating the keys for the PKI, the threshold encryption scheme and the CRS in the real world. In the ideal world, the simulator $\mathcal{S}$ generates the PKI keys, threshold encryption keys and CRS, and outputs the corresponding public keys and CRS to the adversary along with its secret keys.

Case 1: Synchronous network. We describe the simulator $\mathcal{S}$ for the case where the network is synchronous and there are up to t_s corruptions.

- *Input Distribution*: Emulate the messages of the broadcast protocol. This means that, on behalf of each honest party, emulate the broadcast protocol using an encryption of 0 as the input. Also, emulate the $\mathcal{F}_{\text{MZK}}$ functionality by outputting 1 on behalf of each honest parties, and from each corrupted party, on input $(c, (x, r))$ check that $c = \mathsf{Enc}_{\mathsf{ek}}(x, r)$ and output 1 to the adversary and 0

otherwise. The simulator waits for $\max\{T_{bc}, T_{zk}\}$. For each honest party P_i, it keeps track of the correct encrypted inputs I_i that P_i received. If the number of correct ciphertexts is less than $n - t_s$, the simulator does not compute on its ciphertexts on his behalf and sets a local variable $\texttt{abort}_i = 1$.

- *Addition Gates*: $\mathcal{S}$ simply adds the corresponding ciphertexts locally.

- *Multiplication Gates*: $\mathcal{S}$ emulates the broadcast protocols on random encryptions, and outputs 1 when emulating $\mathcal{F}_{\text{MZK}}$ on behalf of them. For each honest party P_i, keep track of the set of parties S_i succeeding in the proofs. The simulator waits for $\max\{T_{bc}, T_{zk}\}$. Then, emulate the messages in the Byzantine agreement protocols and compute the set S. Then it waits for waits for T_{ba}. If the set S is greater than t_s, it computes $\overline{a} \boxplus \left(\boxplus_{i \in S} \overline{d_i}\right)$ and emulates the threshold decryption sub-protocol. After waiting for T_{dec}, it computes the output ciphertext of the multiplication gate. Otherwise, it sets $\texttt{abort}_i = 1$.

- *Output Determination*: For each party P_i, emulate the messages in the asynchronous common subset protocol with the corresponding input (either a ciphertext, which is the result of the computation, or $\perp$ in the case $\texttt{abort}_i = 1$). If the output is a single ciphertext c, emulate the threshold decryption sub-protocol.

- *Threshold Decryption*: In a multiplication gate, simply compute the decryption shares and emulate the sending messages. In the Output Determination stage, $\mathcal{S}$ obtains the output y of the computation, and adjusts the shares such that the shares decrypt to y. In both cases, the simulator always outputs 1 on behalf of the honest parties indicating that the proofs of correct decryptions are correct.

Case 2: Asynchronous network. The only difference with respect to the case where the network is synchronous, is that the protocol $\Pi_{\texttt{bc}}^{t_s, t_a}$ only provides weak-validity. In the simulation, it implies that the simulator will also need to simulate the $\perp$ messages from the broadcast protocols, and not simulate on behalf of the honest parties which stop participating in the protocol after they aborted.

We define a series of hybrids to argue that no environment can distinguish between the real world and the ideal world.

Hybrids and security proof.

Hybrid 1**.** This corresponds to the real world execution. Here, the simulator knows the inputs and keys of all honest parties.

Hybrid 2**.** We modify the real-world execution in the zero-knowledge proofs. In the case of a synchronous network, when a corrupted party requests a proof of any kind from an honest party, the simulator simply gives a valid response without checking the witness from the honest party. In the case of an asynchronous network, the simulator is allowed to set outputs to $\perp$ as the adversary does.

Hybrid 3**.** This is similar to Hybrid 2, but the computation of the decryption shares is different. Here, the simulator obtains the output y from the ideal functionality, and if it is not $\perp$, it computes the decryption shares of corrupted parties, and then adjusts the decryption shares of honest parties such that the decryption shares $(d_1, \ldots, d_n)$ reconstruct to the output value y.

Hybrid 4**.** We modify the previous hybrid in the Input Stage. Here, the honest parties, instead of sending an encryption of the actual input, they send an encryption of 0.

Hybrid 5**.** This corresponds to the ideal world execution.

In order to prove that no environment can distinguish between the real world and the ideal world, we prove that no environment can distinguish between any two consecutive hybrids.

Claim 1**.** No efficient environment can distinguish between Hybrid 1 and Hybrid 2.

Proof of claim. This follows trivially, since the honest parties always send a valid witness to $\mathcal{F}_{\text{MZK}}$ in the case of a synchronous network. In the case of an asynchronous network, the simulator chooses the set of parties that get $\perp$ as the real-world adversary. $\Diamond$

Claim 2**.** No efficient environment can distinguish between Hybrid 2 and Hybrid 3.

Proof of claim. This follows from properties of a secret sharing scheme and the security of the threshold encryption scheme. Given that the threshold is $t_s + 1$, any number corrupted decryption shares below $t_s + 1$ does not reveal anything about the output y. Moreover, one can find shares for honest parties such that $(d_1, \ldots, d_n)$ is a sharing of y. ◊

Claim 4. No efficient environment can distinguish between Hybrid 3 and Hybrid 4.

Proof of claim. This follows from the semantic security of the used threshold encryption scheme. ◊

Claim 5. No efficient environment can distinguish between Hybrid 4 and Hybrid 5.

Proof of claim. The simulator in the ideal world and the simulator in Hybrid 4 emulate the joint behavior of the ideal functionalities exactly in the same way. ◊

We conclude that the real world and the ideal world are indistinguishable.
Finally, let us argue why the protocol has weak termination. Observe that when the protocol outputs $\perp$, parties do not terminate. This is because the protocol $\Pi_{\mathtt{acs}}^{t_s,t_a}$ does not guarantee termination, i.e. might need to run forever (see [BKL20]). However, when parties have agreement on a ciphertext to decrypt (in particular, this is the case when the network is synchronous), the threshold decryption sub-protocol ensures that honest parties can jointly collect $t_s + 1 \leq n - t_s \leq n - t_a$ decryption shares, decrypt the ciphertext and terminate.

□

7.5 Main Protocol

In this section, we present the protocol $\Pi_{\mathtt{mpc}}^{t_s,t_a}$ for secure function evaluation which tolerates up to t_s (resp. t_a) corruptions when the network is synchronous (resp. asynchronous), for any $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ satisfying $t_a + 2t_s < n$. The protocol is based on two sub-protocols:

- $\Pi^{t_s,t_a}_{\texttt{smpc}}$ is a secure function evaluation protocol which gives full security up to t_s corruptions when run in a synchronous network, and achieves unanimous abort with weak termination up to t_a corruptions and has $n - t_s$ output quality when run in an asynchronous network.
- $\Pi^{t_a}_{\texttt{ampc}}$ is a secure function evaluation protocol which gives full security up to t_a corruptions and has $n - t_a$ output quality when run in an asynchronous network.

Protocol $\Pi^{t_s,t_a}_{\texttt{mpc}}(P_i)$

Let x_i denote the input value of party P_i.

1: Run $\Pi^{t_s,t_a}_{\texttt{smpc}}$ using x_i as input. Let y_i be the output of P_i.
2: If $y_i \neq \perp$, output y_i and terminate. Otherwise, run $\Pi^{t_a}_{\texttt{ampc}}$ using x_i as input, output the result and terminate.

Theorem 7.5.1. *Let n, t_s, t_a be such that $0 \leq t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $t_a + 2t_s < n$. Protocol $\Pi^{t_s,t_a}_{\texttt{mpc}}$ satisfies the following properties:*

1. *When run in a synchronous network, it achieves full security up to t_s corruptions.*
2. *When run in an asynchronous network, it achieves full security up to t_a corruptions and has $n - t_s$ output quality.*

Proof. The case where the network is synchronous and there are up to t_s corruptions is trivial, since $\Pi^{t_s,t_a}_{\texttt{smpc}}$ is guaranteed to provide full security, and $\Pi^{t_a}_{\texttt{ampc}}$ is never executed. In the other case where the network is asynchronous and there are up to t_a corruptions, observe that after $\Pi^{t_s,t_a}_{\texttt{smpc}}$ gives output (which is guaranteed to happen), in the case where there is a non-$\perp$ output, every honest party is guaranteed to get this output (which take into account at least $n - t_s$ inputs) and also terminate. If the output is $\perp$, the adversary learned no information so far about the inputs, so it is safe to execute $\Pi^{t_a}_{\texttt{ampc}}$. In this case, since $\Pi^{t_a}_{\texttt{ampc}}$ has output quality $n - t_a$, the overall protocol also has $n - t_s \leq n - t_a$ output quality. Observe that in this case the honest parties terminate as soon as $\Pi^{t_a}_{\texttt{ampc}}$ terminates, since $\Pi^{t_a}_{\texttt{ampc}}$ guarantees termination.

□

7.6 Impossibility Proof

We now discuss two lower bounds in this setting. Our first result shows that our feasibility result in Section 7.5 is tight with respect to the output quality. More concretely, we show that there are basic functions for which it is impossible to achieve both (1) full security up to t corruptions in a synchronous network and (2) $(n - t + 1)$-output quality for even 0 *corruptions* in an asynchronous network. Put simply, a protocol secure against t corruptions cannot rely on receiving more than $n - t$ inputs, even in executions in which all participants happen to be honest.

Our second result shows that the construction presented in Section 7.5 is tight with respect to the corruption thresholds. That is, we show that there is no protocol for secure function evaluation achieving the guarantees of Theorem 7.5.1 when $t_a + 2t_s \geq n$. As an example, we show that the majority function cannot be computed with full security up to t_s corruptions in a synchronous network as well as security up to t_a corruptions in an asynchronous network (in fact, in an asynchronous network, it cannot be computed even if we require only unanimous abort).

Theorem 7.6.1. *Fix any t. There is no protocol Π for MPC with the following properties:*

- *When run in a synchronous network, it achieves full security up to t corruptions.*
- *When run in an asynchronous network, it achieves $(n-t+1)$-output quality when every party is honest.*

Proof. We show the proof for the case of the OR function. More concretely, the function computes the OR of all the inputs that are received by the ideal functionality (i.e. all inputs that are not $\perp$).

We partition the n parties into two sets S_t, S_{n-t}, where $|S_t| = t$ and $|S_{n-t}| = n - t$. Consider an execution of Π in a synchronous network where parties in S_t are corrupted and abort, and parties in S_{n-t} input 0. In this case, since the protocol achieves full security, all honest parties obtain 0 as output and terminate by some time T.

Next consider an execution of Π in an asynchronous network where all parties are honest, parties in S_t have input 1, and parties in S_{n-t} have input 0. All communication between S_t and S_{n-t} is delayed for more

than T clock ticks. Since the view of the parties in S_{n-t} is exactly the same, these parties output 0. This contradicts the fact that Π achieves $(n-t+1)$-output quality. □

Theorem 7.6.2. *Fix any t_a, t_s such that $t_a + 2t_s \geq n$. There is no protocol Π for MPC with the following properties:*

- *When run in a synchronous network, it achieves full security up to t_s corruptions.*
- *When run in an asynchronous network, it achieves unanimous abort up to t_a corruptions.*

Proof. **Case 1:** $t_s \geq n/2$ or $t_a \geq n/3$. These bounds follow from classical impossibility results for synchronous and asynchronous MPC protocols with full security (c.f. [Cle86, BKR94]).

Case 2: $t_s < n/2$, $t_a < n/3$, and $t_a + 2t_s \geq n$.

Assume without loss of generality that $t_a + 2t_s = n$. We prove the impossibility for the case of the majority function. Partition the n parties into three sets, $S^0_{t_s}$, $S^1_{t_s}$, and S_{t_a}, where $|S^0_{t_s}| = |S^1_{t_s}| = t_s$ and $|S_{t_a}| = t_a$.

First, consider an execution of Π in which the network is synchronous and the t_s parties in $S^1_{t_s}$ are corrupted and crash, and furthermore the honest parties all input 0. Since t_s is less than $n/2$, the protocol must output 0.

Next, consider an execution of Π in which the network is asynchronous, the t_a parties in S_{t_a} are corrupted, and the parties in $S^0_{t_s}$ and $S^1_{t_s}$ input 0 and 1, respectively. In the real world, the adversary can use the following attack: block all messages between $S^0_{t_s}$ and $S^1_{t_s}$ throughout, and have all corrupted parties simulate an honest protocol execution with input $b \in \{0,1\}$ with the parties in $S^b_{t_s}$. A party in $S^0_{t_s}$ cannot distinguish between this execution and the first execution, and thus the protocol outputs 0; for the same reason a party in $S^1_{t_s}$ outputs 1. By contrast, in the ideal world, the output will of course be the same for all parties. This proves that there is no protocol for the majority function Π that achieves both properties. □

Appendix C

Details of Chapter 7

C.1 Paillier Cryptosystem

In this section we describe the Paillier cryptosystem [Pai99]. The public key pk is a k-bit RSA modulus $N = pq$, where p, q have $\frac{k}{2}$ bits and are such that $p = 2p' + 1$, $q = 2q' + 1$ for p', q' primes. The secret key is $\mathsf{sk} = \phi(N)(\phi(N)^{-1} \mod N)$.

In order to encrypt a message $a \in \mathbf{Z}_N$, one computes the ciphertext $\overline{a} = \mathsf{Enc}_{\mathsf{pk}}(a, r) = g^a r^N \mod N^2$, where $r \in \mathbf{Z}_N^*$ is chosen uniformly at random, and $g = N + 1$. To decrypt a message, one simply computes $c^{\mathsf{sk}} \mod N^2 = Na + 1$, from which $a \mod N$ can be obtained.

The encryption scheme is additively homomorphic in the sense that $\overline{a} \boxplus \overline{b} = \mathsf{Enc}_{\mathsf{pk}}(a, r_a) \cdot \mathsf{Enc}_{\mathsf{pk}}(b, r_b) = \mathsf{Enc}_{\mathsf{pk}}(a + b, r_a r_b)$.

Semantic security can be shown under the so-called decisional composite residual assumption (DCRA), which states that random elements in $\mathbf{Z}_{N^2}^*$ are computationally indistinguishable from random elements of the form r^N.

A threshold version of this cryptosystem can be found in [DJ01], based on a variant of Shoup's technique [Sho00] for threshold RSA.

Chapter 8

Synchronous MPC with Asynchronous Responsiveness

8.1 Introduction

In the context of multiparty computation (MPC), a set of mutually distrustful parties wish to jointly compute a function by running a distributed protocol. The protocol is deemed secure if every party obtains the correct output and if it does not reveal any more information about the parties' inputs than what can be inferred from the output. Moreover, these guarantees should be met even if some of the parties can maliciously deviate from the protocol description. Broadly speaking, MPC protocols exist in two regimes of synchrony. First, there are *synchronous* protocols which assume that parties share a common clock and messages sent by honest parties can be delayed by at most some a priori known bounded time. Synchronous protocols typically proceed in rounds of length Δ, ensuring that any message sent at the beginning of a round by an honest party will arrive by the end of that round at its intended recipient. On the upside, such strong timing assumptions allow to obtain protocols with an optimal resilience of $\frac{1}{2}n$ corruptions for the case of *full security*

[RB89, GMW87, BMR90, CDN01], and of arbitrary number of corruptions for the case of *security with (unanimous) abort* and no fairness [FGH+02, GL02]. On the downside, especially in real-world networks where the *actual* maximal network delay δ is hard to predict, Δ has to be chosen rather pessimistically, and synchronous protocols fail to take advantage of a fast network.

The second type of protocols that we will study in this work are *asynchronous* protocols. Such protocols do not require synchronized clocks or an a priori known bounded network delay to work properly. As such, they function correctly under much more realistic network assumptions. Moreover, asynchronous protocols have the benefit of running at the *actual speed of the network*, i.e., they run in time that depends only on δ, but *not* on Δ; a notion that we shall refer to as *responsiveness* [PS17a]. This speed and robustness comes at a price, however: it can easily be seen that no asynchronous protocol that implements an arbitrary function can tolerate $\frac{1}{3}n$ maliciously corrupted parties [BKR94]. We ask the natural question of whether it is possible to leverage synchronous MPC protocols to also achieve responsiveness:

Is there a (synchronous) MPC protocol that allows to simultaneously achieve full security with responsiveness up to t corruptions, and some form of extended security (full security, unanimous abort) up to $T \geq t$ corruptions?

We settle the question with tight feasibility and impossibility results:

- For the case where unanimous abort is required as extended security, this is possible if and only if $T + 2t < n$.
- For the case where full security is required as extended security, this is possible if and only if $T < \frac{n}{2}$ and $T + 2t < n$.

8.1.1 Technical Overview of Our Results

The Model. We first introduce a new composable model of functionalities in the UC framework [Can01], which captures the guarantees that protocols from both asynchronous and synchronous worlds achieve in a

very general fashion. Our model allows to capture multiple distinct guarantees such as privacy, correctness, or responsiveness, each of which is guaranteed to hold for (possibly) different thresholds of corruption. In contrast to previous works, we do not capture the guarantees as protocol properties, but rather as part of the ideal functionality. This allows to use the ideal functionality as an assumed functionality in further steps of the composition, without the need to keep track of the properties of the real-world protocols.

Real World Functionalities. Our protocols work with public-key infrastructure (PKI) and common-reference string (CRS) as setup. Parties have access to a synchronized global clock functionality $\mathcal{G}_{\text{CLK}}$ and a communication network of authenticated channels with *unknown* upper bound δ, corresponding to the maximal network delay. This value is unknown to the honest parties. Instead, protocols make use of a conservatively assumed worst-case delay $\Delta \gg \delta$. Within δ, the adversary can schedule the messages arbitrarily.

Ideal Functionality. In order to capture the guarantees that asynchronous and synchronous protocols achieve in a fine-grained manner, we describe an ideal functionality $\mathcal{F}_{\text{HYB}}$ which allows parties to jointly evaluate a function. At a high level, $\mathcal{F}_{\text{HYB}}$ is composed of two phases; an asynchronous and a synchronous phase, separated by some pre-defined time-out. Each party can obtain a unique identical output in either phase. As in asynchronous protocols, the outputs obtained during the asynchronous phase are obtained fast, i.e., at a time which depends on the actual maximal network delay δ, but not on the conservatively assumed worst-case network delay Δ. Let us describe the guarantees that $\mathcal{F}_{\text{HYB}}$ provides.

If there are up to t corruptions, $\mathcal{F}_{\text{HYB}}$ achieves full security with responsiveness. That is, honest parties obtain a correct and identical output, and honest parties' inputs remain private. Moreover, they obtain an output $y_{\texttt{asynch}}$ by a time proportional to the actual network delay δ. Unavoidably, this means that $\mathcal{F}_{\text{HYB}}$ may ignore up to t inputs from honest parties.

If there are up to $T \geq t$ corruptions, $\mathcal{F}_{\text{HYB}}$ can give output at two different points in time $\tau_1 \leq \tau_2$. Either all parties obtain $y_{\texttt{asynch}}$ before time τ_1 (there might be some parties which obtained $y_{\texttt{asynch}}$ in the asynchronous phase), or all parties obtain the output $y_{\texttt{sync}}$ by time τ_2, which is guaran-

teed to take into account all inputs from honest parties. For the output $y_{\texttt{sync}}$, we consider two versions: $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$ which guarantees full security up to T corruptions implying that $y_{\texttt{sync}}$ is the correct output, and $\mathcal{F}_{\text{HYB}}^{\texttt{ua}}$ which guarantees security with unanimous abort up to T corruptions, meaning that the adversary can set $y_{\texttt{sync}}$ to $\perp$.

We depict in Figure 8.1 a time-line showing the point in time at which the honest parties obtain the output, depending on the number of corruptions.

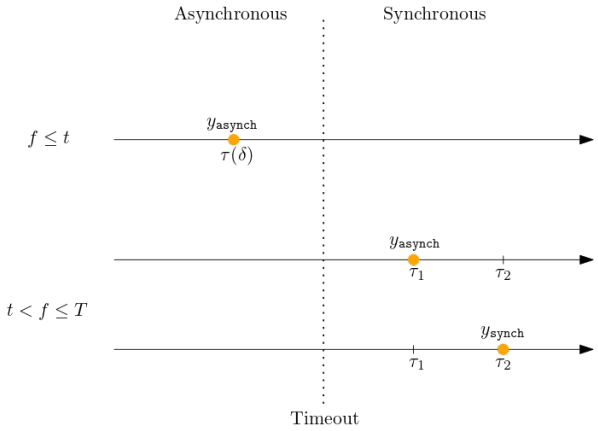

Figure 8.1: The dotted vertical line separates the asynchronous and the synchronous phase. The orange dot shows the latest point in time when honest parties get output. The output $y_{\texttt{asynch}}$ takes into account $n - t$ inputs, whereas $y_{\texttt{sync}}$ takes into account all inputs. Up to t corruptions all parties obtain $y_{\texttt{asynch}}$ fast. In the other case, either all parties obtain $y_{\texttt{asynch}}$ by τ_1, or all parties obtain $y_{\texttt{sync}}$ by τ_2, which is the correct output for $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$, and may be $\perp$ for $\mathcal{F}_{\text{HYB}}^{\texttt{ua}}$.

Black-Box Compiler. We give a generic black-box compiler that combines an asynchronous MPC protocol with a synchronous MPC protocol and gives a hybrid protocol that combines beneficial properties from both the synchronous and asynchronous regime, very roughly in the following way: Using threshold encryption and assuming 1) a *two-threshold* asynchronous protocol with full security up to t corruptions and security with

no termination (correctness and privacy) up to $T \geq t$ corruptions, and 2) a synchronous protocol with *extended* security (full security or security with unanimous abort) up to T corruptions, the compiler provides full security with responsiveness up to t corruptions, and extended security up to T corruptions, for any $T + 2t < n$.

For the first sub-protocol 1), we show how to modify the asynchronous MPC protocol by Cohen [Coh16] to obtain the trade-off mentioned above when used in our aforementioned compiler. We separate the termination threshold from all other security guarantees. That is, we achieve an asynchronous protocol that terminates (in a responsive and fully-secure manner) for any $t < \frac{1}{3}n$, and provides security without termination up to $T < n - 2t$ corruptions.

The second sub-protocol 2) can be achieved with known protocols; for $T < n$ in the case of security with unanimous abort (e.g. [FGH+02, GL02]) and for $T < n/2$ for full security (e.g. [RB89, GMW87, BMR90, CDN01]).

Compiler Description. We now give an outline of our compiler. At a high level, the idea of our compiler is to first run an asynchronous protocol until some pre-defined timeout. Upon timing out, the parties switch to a synchronous computation. If sufficiently many parties are honest, the honest parties obtain their output at the actual speed of network. The main challenge is to ensure that if even a single party obtains output during the asynchronous phase, the output will not be changed during the synchronous phase. This would be problematic for two reasons: First, because the combined protocol would offer no improvement over a standard synchronous protocol in terms of responsiveness; if a party does not know if the output it obtains during the asynchronous phase will be later changed during the synchronous phase, then this output is essentially useless to that party. Therefore, if this were indeed the case, then one could run *just* the synchronous part of the protocol. Second, computing two different outputs may be problematic for privacy reasons, as two different outputs give the adversary more information about the honest parties' inputs than what it should be able to infer. Our solution to this problem is to have the asynchronous protocol output a *threshold ciphertext* $[y]$ *of the actual output* y. Prior to running the hybrid protocol, the parties each obtain a key share d_i such that k out of n parties can jointly decrypt the ciphertext by pooling their shares. This way, if we set $k = n - t$, where

t is the responsiveness threshold, we are ensured that sufficiently many parties will pool their shares during the asynchronous phase, given that fewer than t parties are corrupt. Therefore, every honest party should be able to decrypt and learn the output during the asynchronous phase, thus ensuring responsiveness. On the other hand, our compiler ensures that if any honest party gives out its share during the asynchronous phase after seeing the ciphertext $[y]$ being output by the asynchronous protocol, then the only possible output during the synchronous phase can be y. Finally, our compiler has a mechanism to detect whether no honest party has made its share public yet. In this case, we can safely recompute the result during the synchronous phase of the hybrid protocol, as we can be certain that the adversary does not have sufficient shares to learn the output from the asynchronous phase.

Two-Threshold Asynchronous MPC Protocol. Finally, in Section 8.5, we show how to obtain an asynchronous MPC protocol to achieve tradeoffs between termination and security (correctness and privacy). While many asynchronous MPC protocols (e.g. [PCR09, CP15, Cho20, Coh16, HNP05]) can be adapted to the two-threshold setting, we choose to adapt the protocol in [Coh16] for simplicity.

The protocol in [Coh16] achieves all guarantees simultaneously for the corruption threshold $\frac{1}{3}n$. At a high level, the idea of this protocol is to use a threshold fully homomorphic encryption scheme (TFHE) with threshold $k = \frac{1}{3}n$ and let parties distribute encryption shares of their inputs to each other. Then, parties agree on a common set of at least $\frac{2}{3}n$ parties, whose inputs will be taken into account during the function evaluation. In this step, n Byzantine Agreement protocols are run. Parties can then locally evaluate the function which is to be computed on their respective input shares by carrying out the corresponding (homomorphic) arithmetic operations on these shares. After this local computation has succeeded, parties pool their shares of the computation's result to decrypt the final output of the protocol. We modify the thresholds in this protocol in the following manner. Instead of setting $k = \frac{1}{3}n$, we set $k = \frac{3}{4}n$. Intuitively, assuming a perfect Byzantine Agreement (BA) functionality, this modification has the effect that the adversary needs to corrupt $\frac{3}{4}n$ parties to break privacy, but can prevent the protocol from terminating by withholding decryption shares whenever it corrupts more than $\frac{1}{4}n$ parties. However, one can see that if one realizes the BA functionality using

a traditional protocol with validity and consistency thresholds $\frac{1}{3}n$, the overall statement will only have security $\frac{1}{3}n$.

We show how to improve the security threshold T of the protocol by using, as a sub-component, an asynchronous BA protocol which trades liveness for consistency without sacrificing validity. Our protocol inherits the thresholds of the improved BA protocol, achieving any $T < n - 2t$, where t is the termination threshold.

8.1.2 Synchronous Protocols over an Asynchronous Network

We argue that it is not trivial to enhance a synchronous MPC protocol to achieve responsiveness. Two ways to execute a synchronous protocol over a network with unknown delay δ are as follows:

Time-Out Based. Perhaps the easiest approach to execute a synchronous protocol over this network is to model each round using Δ clock ticks, where Δ is a known upper bound on the network delay. In this case, the output is obtained at a time which depends on Δ. Note that Δ has to be set high enough to accommodate any conditions, and such that any honest party has enough time to perform its local computation; if an honest party is slightly later than Δ in any round, it will be considered corrupted throughout the whole computation. In realistic settings where δ is hard to predict, we will have that $\Delta \gg \delta$. Hence, any synchronous protocol (even constant-round) is slow.

Notification Based. A well-known approach (see e.g. [KLR06]) to "speed up" a synchronous protocol is to let the parties simulate a synchronized clock in an event-based fashion over an asynchronous network. More concretely, the idea is that each party broadcasts a notification once it finishes a particular round i and only advances to round $i + 1$ upon receiving a notification for round i from all parties. It is not hard to see that this approach does not achieve the responsiveness guarantees we aim for. To this end, observe that a **single** corrupted party P_j can make all parties wait Δ clock ticks in each round, simply by not sending a notification in this particular round. Note that parties cannot infer that P_i is corrupted, unless they wait for Δ clock ticks, because δ is unknown. Hence, unless there are *no corruptions*, an approach along these lines

can not ensure responsiveness. In contrast, our protocol guarantees that parties obtain fast outputs as long as there are up to t corruptions.

8.1.3 Related Work

Despite being a very natural direction of research, compilers for achieving tradeoffs between asynchronous and synchronous protocol have only begun to be studied in relatively recent works.

Pass and Shi study a hybrid type of state-machine replication (SMR) protocol in [PS17a] which confirms transactions at an asynchronous speed and works in the model of *mildly adaptive* malicious corruptions; such corruptions take a short time to take effect and as such model a slightly weaker adversary than one that is fully adaptive. Subsequently, Pass and Shi show a general paradigm for SMR protocols with optimistic confirmation of transactions called *Thunderella* [PS18]. In their work, they show how to achieve optimistic transaction confirmation (at asynchronous network speed) as long as the majority of some designated committee and a party called the 'accelerator' are honest and faithfully notarize transactions for confirmation. If the committee or the accelerator become corrupted, the protocol uses a synchronous SMR protocol to recover and eventually switch back to the asynchronous path of the protocol. Their protocol achieves safety and liveness against a fully adaptive adversary, but can easily be kept on the slow, synchronous path forever in this case. Subsequently, Loss and Moran [LM18] showed how to obtain compilers for the simpler case of BA that achieve tradeoffs between responsiveness and safety against a fully adaptive adversary.

The work by Guo et al. [GPS19] introduced a model which weakens classical synchrony. There, the adversary can interrupt the communication between certain sets of parties, as long as in each round there is a (possibly different) connected component with an honest majority of the nodes. Although their focus is not on responsive protocols, the authors include an MPC responsive protocol, based on threshold FHE for the case of full-security as extended security. Our protocols differ from theirs in various aspects: 1) In contrast to their protocol, our approach is conceptually simpler and allows to plug-in any asynchronous and synchronous protocol in a black-box manner and automatically inherit the thresholds for each of the guarantees, and the assumptions from each of the protocols. For example, we can plug-in a synchronous protocol with

full security and unanimous abort, and obtain the corresponding guarantees; one could further consider other types of guarantees, or design MPC protocols from different types of assumptions which would all be inherited automatically from our compiler; 2) We phrase all our results in the UC framework and capture in a very general fashion the guarantees that the protocol provides as part of the ideal functionality. This leads to some differences, e.g. our ideal functionality allows to capture responsiveness guarantees; also allows to take into account in the computation the inputs from all parties in some cases.

Further Related Work. Best-of-both worlds compilers for distributed protocols (in particular MPC protocols) come in many flavours and we are only able to list an incomplete summary of related work. Goldreich and Petrank [GP90] give a black-box compiler for Byzantine agreement which focuses on achieving protocols which have expected constant round termination, but in the worst case terminate after a fixed number of rounds. Kursawe [Kur00] gives a protocol for Byzantine agreement that has an optimistic *synchronous path* which achieves Byzantine agreement if every party behaves honestly and the network is well-behaved. If the synchronous path fails, then parties fall back to an asynchronous path which is robust to network partitions. However, the overall protocol tolerates only $\frac{1}{3}n$ corrupted parties in order to still achieve safety and liveness. A recent line of works [BKL19, BKL20, BLL20] studied protocols resilient to t_2 corruptions when run in a synchronous network and also to t_1 corruptions if the network is asynchronous, for $0 < t_1 < \frac{1}{3}n \leq t_2 < \frac{1}{2}n$. A line of works [BHN10, CPR17, PR18] consider the setting where parties have a few synchronous rounds before switching to fully asynchronous computation. Here, one can achieve protocols with better security guarantees than purely asynchronous ones. Finally, the line of works [FHHW03, IKLP06, Kat07, FHW04, HLM13] consider different thresholds to achieve more fine-grained security guarantees.

Worth mentioning, are the works of [IKLP06, Kat07], which consider MPC protocols with full security up for an honest majority t, and security with abort for a dishonest majority T. Our protocols achieve results in this direction as well, except that our threshold t includes responsiveness as well. Note that the impossibility of [Kat07], where it is shown that $T + t \geq n$ is impossible does not apply to our work, since we consider a *weaker* trade-off $T + 2t < n$. Moreover, the fact that our threshold t for

full security case includes responsiveness as well is essential to prove that the bound $T + 2t < n$ is tight.

8.2 Preliminaries

Threshold Encryption Scheme. We assume the existence of a secure public-key encryption scheme which enables threshold decryption. (See Section 2.2)

Digital Signature Scheme. We assume the existence of a digital signature scheme unforgeable against adaptively chosen message attacks. Given a signing key sk and a verification key vk, let $\mathsf{Sign}_{\mathsf{sk}}$ and $\mathsf{Ver}_{\mathsf{vk}}$ the signing and verification functions. We write $\sigma = \mathsf{Sign}_{\mathsf{sk}}(m)$ meaning using sk, sign a plaintext m to obtain a signature σ. Moreover, we write $\mathsf{Ver}_{\mathsf{vk}}(m, \sigma) = 1$ to indicate that σ is a valid signature on m.

8.3 Model

8.3.1 Adversary

We consider a static adversary, who can corrupt up to f parties at the onset of the execution and make them deviate from the protocol arbitrarily. The adversary is also computationally bounded.

8.3.2 Real and Ideal World

We consider a real world where parties have access to a synchronized global clock functionality $\mathcal{G}_{\text{CLK}}$, and a network functionality $\mathcal{N}^{\delta}$ of pairwise authenticated channels with an unknown upper bound on the message delay δ, as defined in Section 6.1.4.

We introduce ideal functionality $\mathcal{F}_{\text{HYB}}^{\mathtt{fs}}$ (resp. $\mathcal{F}_{\text{HYB}}^{\mathtt{ua}}$) which allows to capture the guarantees that asynchronous and synchronous protocols for secure function evaluation offer in a fine-grained manner. The functionality has access to the global functionality $\mathcal{G}_{\text{CLK}}$, and allows parties to evaluate a function f. The idea is that up to t corruptions, parties have full security and responsiveness. Moreover, in the case of $\mathcal{F}_{\text{HYB}}^{\mathtt{fs}}$, if up to $t \leq T < n/2$ parties are corrupted, full security is guaranteed, i.e. all

honest parties obtain the correct and identical output, and the inputs from honest parties remain secret. The functionality $\mathcal{F}_{\text{HYB}}^{\mathtt{ua}}$ is the same, except that it guarantees security with unanimous abort up to $t \leq T < n$ corruptions instead of full security, i.e., honest parties obtain the correct output or unanimously obtain $\perp$.

The number of inputs that the function is guaranteed to take into account and the time at which it provides output depends the number of corruptions. The time-out divides the execution into two phases: an asynchronous and a synchronous phase.

- If there are up to t corruptions, parties are guaranteed to obtain an output at time $\tau_{\mathtt{asynch}}$, which depends on δ. This fast output is identical to every party and is guaranteed to take into account at least $n - t$ inputs, i.e. can ignore the inputs from up to t honest parties.

- Otherwise, the parties are guaranteed to obtain the same output, but at a time which depends on Δ. More concretely, there are two latest points in time at which parties can obtain an output after the time-out occurs: $\tau_{\mathtt{OD}} < \tau_{\mathtt{OND}}$. Either all parties obtain the output by $\tau_{\mathtt{OD}}$, which is guaranteed to take into account $n - t$ inputs, or all parties obtain output at a later time $\tau_{\mathtt{OND}}$, which is guaranteed to take into account all inputs.

The adversary can in addition gain certain capabilities depending on the amount of corruption it performs. More technically, we introduce a tamper function Tamper, parametrized by a tuple of thresholds (t, T). This allows to naturally capture the different guarantees for the two corruption thresholds t and T. Basically, if the number of corruptions is greater than t, the adversary can prevent the parties to obtain fast outputs. And beyond T, no security guarantee is ensured, as the adversary learns the inputs from the honest parties and can choose the outputs as well.

Tamper Function. The ideal functionality is parameterized by a tamper function, which indicates the adversary's capabilities depending on the threshold. We consider two thresholds: T for full security, and t for responsiveness.

Definition 8.3.1. We define the ideal functionality with parameters (t, T) if it has the following tamper function $\mathsf{Tamper}^{\text{HYB}}_{t,T}$:

Function $\mathsf{Tamper}^{\text{HYB}}_{t,T}$

// Flags indicating violation of c correctness, p privacy, r responsiveness
$(c, p, r) = \mathsf{Tamper}^{\text{HYB}}_{t,T}$, where:

- $c = 1$, $p = 1$ if and only if $|\mathcal{P} \setminus \mathcal{H}| > T$.
- $r = 1$ if and only if $|\mathcal{P} \setminus \mathcal{H}| > t$

The ideal functionality has in addition a set of parameters. It contains a parameter $\tau_{\texttt{asynch}}$ which models the maximum output delay in the asynchronous phase, and parameters $\tau_{\texttt{OD}}$ and $\tau_{\texttt{OND}}$ which model the output delays for an output that takes into account $n - t$ inputs, or an output with all the inputs. One can think of $\tau_{\texttt{asynch}} = O(\delta)$, and $\tau_{\texttt{OD}} < \tau_{\texttt{OND}}$ are times which depend on Δ.

In addition, it keeps the following local variables:

- `FastOutput` indicates if the output contains $n - t$ inputs or all inputs.
- τ keeps the current time.
- $\tau_{\texttt{tout}}$ is the pre-defined time-out to switch between the two phases.
- `sync` is the phase being executed (asynchronous or synchronous).
- x_i, y_i the input and output for party P_i.
- w_i indicates if the adversary decided to not deliver output y_i in the asynchronous phase. The adversary can only use this capability if the number of corruptions is larger than t.
- $\mathcal{I}$ keeps the set of parties whose input are taken into account for the fast output.

Functionality $\mathcal{F}_{\text{HYB}}^{\text{fs}}$

The functionality keeps track of the current time τ.
The functionality is parametrized by δ, $\tau_{\texttt{asynch}}$, $\tau_{\texttt{OD}}$, $\tau_{\texttt{OND}}$, Tamper, $\tau_{\texttt{tout}}$ and the function to evaluate f.
The functionality stores variables $\texttt{FastOutput}$, τ, $\texttt{sync}$, x_i, y_i, w_i. These variables are initialized as $\texttt{FastOutput} = \texttt{false}$, $\tau = 0$, $\texttt{sync} = \texttt{false}$, $x_i = \perp$, and $y_i = w_i = \perp$.
It keeps $\mathcal{I} = \mathcal{H}$, where $\mathcal{H}$ is the set of honest parties, and a set $\mathcal{C} = \varnothing$.

Timeout/Clock :

Each time the functionality is activated, update τ accordingly.
If $\tau \geq \tau_{\texttt{tout}}$, set $\texttt{sync} = \texttt{true}$. If $\texttt{FastOutput} = \texttt{false}$, compute $y_1 = \cdots = y_n = f(x_1, \ldots, x_n)$.

Asynchronous Phase If $\texttt{sync} = \texttt{false}$ do the following:

- At the onset of the execution, output δ and $\tau_{\texttt{asynch}}$ to the adversary.
- On input $(\textsc{Input}, v_i, \text{sid})$ from party P_i:
  - If some party has received output, ignore this message. Otherwise, set $x_i = v_i$.
  - If $x_i \neq \perp$ for each $P_i \in \mathcal{I}$, set each output to $y_j = f(x'_1, \ldots, x'_n)$, where $x'_i = x_i$ for each $P_i \in \mathcal{I} \cup (\mathcal{P} \setminus \mathcal{H})$ and $x'_i = \perp$ otherwise.
  - Output $(\textsc{Input}, P_i, \text{sid})$ to the adversary.
- On input $(\textsc{GetOutput}, \text{sid})$ from P_i do the following:
  - If the output has not been set yet or is blocked, i.e., $y_i = \perp$ or $w_i = \texttt{aBlocked}$, ignore this message.
  - If $\tau \geq \tau_{\texttt{asynch}}$ output $(\textsc{output}, y_i, \text{sid})$ to P_i and set $\texttt{FastOutput} = \texttt{true}$.
  - Otherwise, output $(\textsc{output}, P_i, \text{sid})$ to the adversary.

Synchronous Phase If $\texttt{sync} = \texttt{true}$ do the following:

- On input $(\textsc{GetOutput}, \text{sid})$ from party P_i
  - If $\texttt{FastOutput} = \texttt{true}$ and $\tau \geq \tau_{\texttt{tout}} + \tau_{\texttt{OD}}$, it outputs $(\textsc{Output}, y_i, \text{sid})$ to P_i.
  - If $\texttt{FastOutput} = \texttt{false}$ and $\tau \geq \tau_{\texttt{tout}} + \tau_{\texttt{OND}}$, it outputs $(\textsc{Output}, y_i, \text{sid})$ to P_i.

Adversary

Upon each party corruption, update $(c, p, r) = \mathsf{Tamper}^{\text{HYB}}_{t,T}$.

// Core Set and Delivery of Outputs

1: Upon receiving a message $(\text{NO-INPUT}, \mathcal{P}', \text{sid})$ from the adversary, if $\texttt{sync} = \texttt{false}$, $\mathcal{P}'$ is a subset of $\mathcal{P}$ of size $|\mathcal{P}'| \leq t_r$ and $y_1 = \cdots = y_n = \perp$, set $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$.

2: On input $(\text{DELIVEROUTPUT}, i, \text{sid})$ from the adversary, if $y_i \neq \perp$ and $\texttt{sync} = \texttt{false}$, output $(\text{OUTPUT}, y_i, \text{sid})$ to P_i and set $\texttt{FastOutput} = \texttt{true}$.

// Adversary's capabilities

3: On input $(\text{TAMPEROUTPUT}, P_i, y_i', \text{sid})$ from the adversary, if $c = 1$, set $y_i = y_i'$.

4: If $p = 1$, output $(x_1, \ldots, x_n)$ to the adversary.

5: On input $(\text{BLOCKASYNCHOUTPUT}, P_i, \text{sid})$ from the adversary, if $r = 1$ and $\texttt{sync} = \texttt{false}$, set $w_i = \texttt{aBlocked}$.

In the version where $\mathcal{F}^{\texttt{ua}}_{\text{HYB}}$ provides security with unanimous abort and no fairness, the adversary can in addition choose to set the output to $\perp$ for all honest parties and learn the output $y_{\texttt{sync}}$, in the case $\texttt{FastOutput} = \texttt{false}$.

8.4 Compiler

In this section, we present a protocol which realizes the ideal functionality presented in the previous section. The protocol works with a setup $\mathcal{F}_{\text{SETUP}}$, where parties have access to a public-key infrastructure used to sign values, and keys for a threshold encryption scheme.

The protocol uses a number of sub-protocols:

- $\Pi_{\texttt{zk}}$ is a bilateral zero-knowledge protocol which allows a party to prove knowledge of a witness corresponding to a statement.
- $\Pi_{\texttt{aMPC}}$ is an asynchronous MPC protocol that provides full security up to t corruptions, and security without termination (correctness and privacy) up to $T \geq t$ corruptions.
- $\Pi^{\texttt{fs}}_{\texttt{sMPC}}$ (resp. $\Pi^{\texttt{ua}}_{\texttt{sMPC}}$) is a synchronous MPC protocol with full security (resp. security with unanimous abort) up to T corruptions.

- Π_{sBC} is a synchronous broadcast protocol secure up to T corruptions.

8.4.1 Key-Distribution Setup

The compiler works with a key distribution setup. The setup can be computed once for multiple instances of the protocol, without knowing the parties' inputs nor the function to evaluate.

As usual, we describe our compiler in a hybrid model where parties have access to an ideal functionality $\mathcal{F}_{\text{SETUP}}$. At a very high level, $\mathcal{F}_{\text{SETUP}}$ allows to distribute the keys for a threshold encryption scheme and a digital signature scheme. The threshold encryption scheme here does not need to be homomorphic. More concretely, it provides to each party P_i a global public key ek and a private key share dk_i. Moreover, it gives a PKI infrastructure. That is, it gives to each party P_i a signing key sk_i and the verification keys of all parties $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$.

We describe the two setups, PKI setup $\mathcal{F}_{\text{PKI}}$ and threshold encryption setup $\mathcal{F}_{\text{TE}}$ independently. The setup of the protocol consists of includes both functionalities $\mathcal{F}_{\text{SETUP}} = [\mathcal{F}_{\text{PKI}}, \mathcal{F}_{\text{TE}}]$.

Digital Signature Setup. The protocol assumes a signature setup. That is, each party P_i has a pair secret key and verification key $(\mathsf{sk}_i, \mathsf{vk}_i)$, where vk_i is known to all parties.

Threshold Encryption Setup. The protocol assumes also a threshold encryption setup, which allows each party to access a global public key ek and a private key share dk_i.

8.4.2 Zero-Knowledge

The protocol Π_{zk} is a bilateral zero-knowledge protocol which allows a party to prove knowledge of a witness corresponding to a statement. The protocol must be UC-secure, meaning that it has to UC-realize the ZK functionality (described in Section D.1 for completeness). As shown in [DDO+01], such a protocol exists in the $\mathcal{F}_{\text{CRS}}$-hybrid model for any relation. For this protocol, we need *proofs of correct decryption*, where the relation is parametrized by a threshold encryption scheme. The statement consists of ek, a ciphertext c, and a decryption share d. The witness is a decryption key share dk_i such that $d = \mathsf{Dec}_{\mathsf{dk}_i}(c)$.

8.4.3 Synchronous MPC

Classical synchronous MPC protocols [RB89, GMW87, BMR90, CDN01], for $\Pi^{\mathtt{fs}}_{\mathtt{sMPC}}$ can be proven to UC-realize an ideal MPC functionality $\mathcal{F}^{\mathtt{fs}}_{\mathrm{SYNC}}$ (described in Section D.1 for completeness) up to $T < n/2$ corruptions, which allows a set of n parties to evaluate a specific function f. For the case of unanimous abort, where the adversary is allowed to set the output $\perp$, one can instantiate $\Pi^{\mathtt{ua}}_{\mathtt{sMPC}}$ for any $T < n$ [FGH+02, GL02].

8.4.4 Synchronous Byzantine Broadcast

A Byzantine broadcast primitive allows a party P_s, called the sender, to consistently distribute a message among a set of parties $\mathcal{P}$.

Definition 8.4.1. Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender P_s initially holds an input v, and parties terminate upon generating output. Π is a T-secure broadcast protocol if the following conditions hold up to T corruptions:

- Validity: If the sender P_s is honest, every honest party outputs the sender's message v.
- Consistency: All honest parties output the same message.

The classical result of Dolev-Strong [DS83] shows that synchronous broadcast protocol $\Pi_{\mathtt{sBC}}$ can be achieved for any $T < n$, assuming a public-key infrastructure. The protocol UC-realizes the synchronous broadcast functionality $\mathcal{F}_{\mathrm{sBC}}$ (which is a synchronous MPC functionality, where the output is the sender's input) for our setting with static corruptions [GKKZ11a, KMTZ13].

8.4.5 Asynchronous MPC

In this section we formally define what it means for a protocol $\Pi_{\mathtt{aMPC}}$ to achieve full security up to t corruptions and security without termination (correctness and privacy) up to $T \geq t$ corruptions. In Section 8.5.2 we show how to achieve such a protocol.

In a nutshell, the idea is that the protocol realizes an ideal MPC functionality which is parametrized with the two thresholds (t, T). If the adversary corrupts up to t parties, all honest parties obtain all the

security guarantees as a conventional asynchronous MPC functionality. If the adversary corrupts $t \leq f \leq T$ parties, it is allowed to block any party from obtaining output; however, those parties that obtain output, are ensured to obtain the correct output, and privacy is still guaranteed. Finally, if the adversary corrupts $f > T$ parties, no guarantees remain: the adversary learns the inputs from all honest parties and can choose the outputs to be anything.

To model formally an asynchronous MPC functionality, we borrow ideas from [KMTZ13, CGHZ16]. In traditional asynchronous protocols, the parties are guaranteed to *eventually* receive output, meaning that the adversary can delay the output of honest parties in an arbitrary but finite manner. The reason for this is that the assumed network guarantees eventual delivery. One can make the simple observation that if the network has an unknown upper bound δ, then the adversary can delay the outputs of honest parties up to time $\tau_{\texttt{asynch}} = \tau(\delta)$, which is a function of δ. The guarantee obtained in an asynchronous MPC with eventual delivery (e.g. as in [CGHZ16]) is a special case of our functionality, namely when $\tau_{\texttt{asynch}} = \infty$. We describe it for the case where $\tau_{\texttt{asynch}}$ is a fixed time, but one can model $\tau_{\texttt{asynch}}$ to be probabilistic as well.

It is known that asynchronous protocols cannot achieve simultaneously fast termination (at a time which depends on δ) and input completeness. This is because δ is unknown and hence it is impossible to distinguish between an honest slow party and an actively corrupted party. If fast termination must be ensured even when up to t parties are corrupted, the parties can only wait for $n - t$ inputs. Since the adversary is able to schedule the delivery of messages from honest parties, it can also typically choose exactly a set of parties $\mathcal{P}' \subseteq \mathcal{P}$, $|\mathcal{P}'| \leq t$, whose input is not considered. Therefore, the ideal functionality also allows the simulator to choose this set. As in [CGHZ16], and similar to the network functionality $\mathcal{N}^{\delta}$, we use a "fetch-based" mode functionality and allow the simulator to specify a delay on the delivery to every party.

Functionality $\mathcal{F}_{\text{ASYNC}}$

$\mathcal{F}_{\text{ASYNC}}$ is connected to a global clock functionality $\mathcal{G}_{\text{CLK}}$. It is parameterized by a set $\mathcal{P}$ of n parties, a function f, a tamper function $\mathsf{Tamper}_{t,T}$, a delay

δ, and a maximum delay $\tau_{\texttt{asynch}}$. It initializes the variables $x_i = y_i = \bot$, $\tau_{\texttt{in}} = \bot$ and $\tau_i = 0$ for each party $P_i \in \mathcal{P}$ and the variable $\mathcal{I} = \mathcal{H}$, where $\mathcal{H}$ is the set of honest parties.
Upon receiving input from any party or the adversary, it queries $\mathcal{F}_{\text{CLOCK}}$ for the current time and updates τ accordingly.

Party P_i:

1: On input $(\text{INPUT}, v_i, \text{sid})$ from party P_i:
- If some party has received output, ignore this message. Otherwise, set $x_i = v_i$.
- If $x_i \neq \bot$ for each $P_i \in \mathcal{I}$, set each output to $y_j = f(x'_1, \ldots, x'_n)$, where $x'_i = x_i$ for each $P_i \in \mathcal{I} \cup (\mathcal{P} \setminus \mathcal{H})$ and $x'_i = \bot$ otherwise. Set $\tau_{\texttt{in}} = \tau$.
- Output $(\text{INPUT}, P_i, \text{sid})$ to the adversary.

2: On input $(\text{GETOUTPUT}, \text{sid})$ from P_i, if the output is not set or is blocked, i.e., $y_i \in \{\bot, \top\}$, ignore the message. Otherwise, if the current time is larger than the time set by the adversary, $\tau \geq \tau_i$, output $(\text{OUTPUT}, y_i, \text{sid})$ to P_i.

Adversary:

1: Upon receiving a message $(\text{NO-INPUT}, \mathcal{P}', \text{sid})$ from the adversary, if $\mathcal{P}'$ is a subset of $\mathcal{P}$ of size $|\mathcal{P}'| \leq t$ and $y_1 = \cdots = y_n = \bot$, set $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$.
2: On input $(\text{SETOUTPUTTIME}, P_i, \tau', \text{sid})$ from the adversary, if $\tau_{\texttt{in}} \neq \bot$ and $\tau' < \tau_{\texttt{in}} + \tau_{\texttt{asynch}}$, set $\tau_i = \tau'$.

Upon each party corruption, update $(c, p, l) = \mathsf{Tamper}^{\text{ASYNCH}}_{t,T}$.

1: On input $(\text{TAMPEROUTPUT}, P_i, y'_i, \text{sid})$ from the adversary, if $c = 1$, set $y_i = y'_i$.
2: If $p = 1$, output $(x_1, \ldots, x_n)$ to the adversary.
3: On input $(\text{BLOCKOUTPUT}, P_i, \text{sid})$ from the adversary, if $l = 1$, set $y_i = \top$.

Similar to $\mathcal{F}_{\text{HYB}}$, we parametrize the functionality by a tamper function to capture the guarantees depending on the set of corrupted parties. The $\mathcal{F}_{\text{ASYNC}}$ functionality has the tamper function $\mathsf{Tamper}^{\text{ASYNCH}}_{t,T}$, where the adversary can tamper with the output value and learn the inputs if the number of corruptions is larger than T, and is allowed to block the delivery of the outputs if the number of corruptions is larger than t.

Definition 8.4.2. We define an asynchronous MPC functionality with full security t and security without termination T, if it has the tamper

function $\mathsf{Tamper}_{t,T}^{\text{ASYNCH}}$:

Function $\mathsf{Tamper}_{t,T}^{\text{ASYNCH}}$

$(c, p, l) = \mathsf{Tamper}_{t,T}^{\text{ASYNCH}}$, where:
- $c = 1$, $p = 1$ if and only if $|\mathcal{P} \setminus \mathcal{H}| > T$.
- $l = 1$ if and only if $|\mathcal{P} \setminus \mathcal{H}| > t$.

8.4.6 Protocol Compiler

The protocol has two phases: an asynchronous phase and a synchronous phase, separated by a pre-defined timeout. The timeout is set large enough (using Δ and the number of asynchronous rounds) so that the asynchronous phase should have supposedly terminated if there were not too many corruptions.

During the asynchronous phase, parties may obtain an output $y_{\mathtt{asynch}}$. We need to ensure (1) that if an honest party obtains an output $y_{\mathtt{asynch}}$ during the asynchronous phase, then every other honest party obtains this output as well; and (2) that the adversary does not learn two outputs. We remark that even if the function to evaluate is the same, the output obtained from the synchronous MPC protocol $\Pi_{\mathtt{sMPC}}$ is not necessarily $y_{\mathtt{asynch}}$. This is because in an asynchronous protocol $\Pi_{\mathtt{aMPC}}$, up to t inputs from honest parties can be ignored. This is the reason why we require that $\Pi_{\mathtt{aMPC}}$ evaluates the function $f' = \mathsf{Enc}_{\mathsf{ek}}(f)$. During the synchronous phase, parties agree on whether they execute the synchronous protocol $\Pi_{\mathtt{sMPC}}$. The parties will invoke $\Pi_{\mathtt{sMPC}}$ only if it is guaranteed that the adversary did not obtain $y_{\mathtt{asynch}}$. Also, if the parties do not invoke $\Pi_{\mathtt{sMPC}}$, it is guaranteed that they can jointly decrypt the output $y_{\mathtt{asynch}}$.

Asynchronous Phase. In this phase, parties optimistically execute $\Pi_{\mathtt{aMPC}}$. When a party P_i obtains as output a ciphertext $c = [y]$ from $\Pi_{\mathtt{aMPC}}$, it sends a signature of c and collects a list L of $n - t$ signatures on the same c. Once such list L is collected, it runs a robust threshold decryption protocol. For that, P_i computes a decryption share $d_i = \mathsf{Dec}_{\mathsf{dk}_i}(c)$, and proves using $\Pi_{\mathtt{zk}}$ to each P_j that d_i is a correct decryption share of c. Upon receiving d_i and a correct proof of decryption share for c from $n-t$ parties, compute and output $y_i = \mathsf{Rec}(\{d_j\})$.

Synchronous Phase. After the timeout, parties execute a synchronous

broadcast protocol to send a pair list-ciphertext (c, L), where L contains at least $n-t$ signatures on c, if such a list was collected during the asynchronous phase. If a party receives via broadcast any valid L, then it sends its decryption share d_i and runs the same robust threshold decryption protocol as above. Otherwise, parties execute the synchronous MPC $\Pi_{\texttt{sMPC}}$.

Observe that if an honest party collects a list L of $n-t$ signatures on a ciphertext $[y]$ during the asynchronous phase, it broadcasts the pair $([y], L)$ during the synchronous phase. Then, every honest party obtains at least a valid pair $([y], L)$ after the broadcast round finishes. By a standard quorum argument, if there are up to $T < n - 2t$ corruptions, there cannot be two signature lists of size $n-t$ on different values. Given that honest parties only sign the correct output ciphertext $[y_{\texttt{asynch}}]$ from $\Pi_{\texttt{aMPC}}$, this is the only value that can gather a list of signatures. Hence, all parties are instructed to run the robust threshold decryption protocol, and if there are up to t corruptions, every honest party is guaranteed to receive enough decryption shares to obtain the output $y_{\texttt{asynch}}$. On the other hand, if no honest party obtained such a pair during the asynchronous phase, it is guaranteed that the adversary did not learn $y_{\texttt{asynch}}$, since no honest party sent its decryption share. However, it might be that the adversary collected a valid $([y_{\texttt{asynch}}], L')$. The adversary can then decide whether to broadcast a valid pair. If it does, every party will hold this pair and everyone outputs $y_{\texttt{asynch}}$ as before. And if it does not, no honest party holds a valid pair after the broadcast round, and every party can safely run the synchronous MPC protocol $\Pi_{\texttt{sMPC}}$.

We remark that it is not enough that upon the timeout parties simply *send* $([y], L)$, because the parties need to have agreement on whether or not to invoke $\Pi_{\texttt{sMPC}}$. It can happen that the adversary is the only one who collected $([y], L)$.

Protocol $\Pi^{\Delta}_{\texttt{mpc}}(P_i)$

The party stores the current time τ, a flag $\texttt{sync} = \texttt{false}$ and a variable $\tau_{\texttt{sync}} = \perp$. Let $\tau_{\texttt{tout}} = T_{\texttt{asynch}}(\Delta) + T_{\texttt{zk}}(\Delta) + \Delta$ be a known upper bound on the time to execute the asynchronous phase, composed of protocols $\Pi_{\texttt{aMPC}}$, $\Pi_{\texttt{zk}}$ and a network transmission message. Also, let $T_{zk}(\Delta)$ denote an upper bound on the time to execute $\Pi_{\texttt{zk}}$.

Clock / Timeout Each time the party is activated do the following:

1: Query $\mathcal{G}_{\text{CLK}}$ for the current time and updates τ accordingly.
2: If $\tau \geq \tau_{\texttt{tout}}$, set $\texttt{sync} = \texttt{true}$ and $\tau_{\texttt{sync}} = \tau$.

Setup:

1: If activated for the first time input $(\text{GETKEYS}, \text{sid})$ to $\mathcal{F}_{\text{SETUP}}$. We denote the public key ek, a $(n-t, n)$-share dk_i of the corresponding secret key dk, the signing key sk and the verification key vk.

Asynchronous Phase: If $\texttt{sync} = \texttt{false}$ handle the following commands.

- On input $(\text{INPUT}, x_i, \text{sid})$ (and following activations) do
  1: Execute $\Pi_{\texttt{aMPC}}$ with input x_i and wait until an output c is received.
  2: Send $(c, \mathsf{Sign}(c, \mathsf{sk}))$ to every other party using $\mathcal{N}$.
  3: Receive signatures and values via $\mathcal{N}$ until you received $n-t$ signatures $L = (\sigma_1, \ldots, \sigma_l)$ on a value c.
  4: Send (c, L) to every party using $\mathcal{N}$.
  5: Receive message lists (c, L'). For each such list send (c, L') to every party using $\mathcal{N}$.
  6: Once done with the above, compute $d_i = \mathsf{Dec}_{\mathsf{dk}_i}(c)$, and prove, using ZK, to each P_j, that d_i is a correct decryption share of c.
  7: Upon receiving $n-t$ correct decryption shares for c, compute and output $y = \mathsf{Rec}(\{d_j\})$.
- At every clock tick, if it is not possible to progress with the list above, send (CLOCKREADY) to $\mathcal{G}_{\text{CLK}}$.

Synchronous Phase: If $\texttt{sync} = \texttt{true}$ and $\tau \geq \tau_{\texttt{sync}}$, stop all previous steps and do the following commands.

- On input (CLOCKREADY) do:
  1: Send (CLOCKREADY) to $\mathcal{G}_{\text{CLK}}$.
  2: **if** $\tau \geq \tau_{\texttt{sync}}$ **then**
  3: Use $\Pi_{\texttt{sBC}}$ to broadcast (c, L), for each pair (c, L) received during the Asynchronous Phase.
  4: Wait until $\Pi_{\texttt{sBC}}$ terminated. If a pair (c, L) was received as output, compute $d_i = \mathsf{Dec}_{\mathsf{dk}_i}(c)$, and prove, using ZK, to each P_j, that d_i is a correct decryption share of c. Otherwise, if no pair (c, L) was received, run the synchronous MPC protocol $\Pi^{\texttt{fs}}_{\texttt{sMPC}}$ with input x_i.
- If there was an output (c', L') from $\Pi_{\texttt{sBC}}$, wait for $T_{zk}(\Delta)$ clock ticks. After that, if $n-t$ correct decryption shares d_j are received from $\mathcal{N}$,

compute and reconstruct the value $y = \mathsf{Rec}(\{d_j\})$ from c, and output y. Otherwise, if there was no output (c, L') from $\Pi_{\mathtt{sBC}}$, output the output received from $\Pi^{\mathtt{fs}}_{\mathtt{sMPC}}$.

Let $T_{zk}(\delta)$, $T_{\mathtt{sync}}(\Delta)$, $T_{\mathtt{BC}}(\Delta)$, $T_{\mathtt{asynch}}(\delta)$ be the corresponding time to execute the protocols $\Pi_{\mathtt{zk}}$, $\Pi_{\mathtt{sMPC}}$, $\Pi_{\mathtt{sBC}}$ and $\Pi_{\mathtt{aMPC}}$, respectively. We state the following theorem, and the proof is formally described in Section D.2. The communication complexity is inherited from the corresponding sub-protocols.

Theorem 8.4.3. *Assuming PKI and CRS, for any $\Delta \geq \delta$, $\Pi^{\Delta}_{\mathtt{mpc}}$ realizes $\mathcal{F}^{\mathtt{fs}}_{HYB}$ with full security with responsiveness t and full security $\min\{T, n - 2t\}$. The maximum delay of the asynchronous phase is $\tau_{\mathtt{asynch}} = T_{\mathtt{asynch}}(\delta) + T_{\mathtt{zk}}(\delta) + \delta$, and of the synchronous phase is $\tau_{\mathtt{OD}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{zk}}(\Delta)$ for a fast output with $n - t$ inputs, and otherwise is $\tau_{\mathtt{OND}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{sync}}(\Delta)$ for an output with all the inputs.*

By replacing the invocation of $\Pi^{\mathtt{fs}}_{\mathtt{sMPC}}$ to $\Pi^{\mathtt{ua}}_{\mathtt{sMPC}}$, one realizes $\mathcal{F}^{\mathtt{ua}}_{\mathrm{HYB}}$ for the same parameters. Let $\Pi^{\Delta}_{\mathtt{hyb\text{-}ua}}$ denote the same protocol as $\Pi^{\Delta}_{\mathtt{mpc}}$, except that the invocation of $\Pi^{\mathtt{fs}}_{\mathtt{sMPC}}$ is replaced by $\Pi^{\mathtt{ua}}_{\mathtt{sMPC}}$.

Theorem 8.4.4. *Assuming PKI and CRS, for any $\Delta \geq \delta$, $\Pi^{\Delta}_{\mathtt{hyb\text{-}ua}}$ realizes $\mathcal{F}^{\mathtt{ua}}_{HYB}$ with full security with responsiveness t and security with unanimous abort $\min\{T, n - 2t\}$. The maximum delay of the asynchronous phase is $\tau_{\mathtt{asynch}} = T_{\mathtt{asynch}}(\delta) + T_{\mathtt{zk}}(\delta) + \delta$, and of the synchronous phase is $\tau_{\mathtt{OD}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{zk}}(\Delta)$ for a fast output with $n - t$ inputs, and otherwise is $\tau_{\mathtt{OND}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{sync}}(\Delta)$ for an output with all the inputs.*

8.5 Asynchronous Protocols

In this section, we show how to obtain $\Pi_{\mathtt{aMPC}}$ with full security with responsiveness up to t corruptions and security (correctness and privacy) up to T corruptions, for any $t < \frac{n}{3}$ and any $T < n - 2t$.

Technical Remark. In our model, parties have access to a synchronized clock. The asynchronous protocols do not read the clock, but in our model they need to specify at which point the parties send a (ClockReady) message to $\mathcal{G}_{\mathrm{CLK}}$, so that the clock advances. Observe that we do not model time within a single asynchronous round (between fetching and

sending messages), or computation time. Hence, in an asynchronous protocol, at every activation, each party P_i *fetches* the messages from the assumed functionalities, and then checks whether it has any message available that it can send. If so, it sends the corresponding message. Otherwise, it sends a (ClockReady) message to $\mathcal{G}_{\text{CLK}}$.

8.5.1 Asynchronous Byzantine Agreement

The goal of Byzantine agreement is to allow a set of parties to agree on a common value (see Definition 7.3.3).

The first step is to obtain an asynchronous Byzantine Agreement protocol BA with higher consistency threshold. In Section D.3, we formally prove security of such a protocol BA in the UC framework for any validity t_v, consistency t_c and termination t_l, such that $t_l \leq t_v < \frac{n}{3}$ and $t_c + 2t_l < n$.

The general idea is to trade termination by consistency, while keeping validity. The protocol is quite simple. First, each party P_i runs with input x_i a regular Byzantine agreement protocol secure up to a single threshold $t' = t_v < n/3$. Once an output x is obtained from the BA, it computes a signature $\sigma = \mathsf{Sign}(x, \mathsf{sk})$ and sends it to every other party. Once $n - t_l$ signatures on a value x' are collected, the party sends the list containing the signatures along with the value x' to every other party, and terminates with output x'. Since there cannot be two lists of $n - t_l$ signatures on different values if there are up to $t_c < n - 2t_l$ corruptions, this prevents parties to output different values if there are up to $t_c < n - 2t_l$ corruptions. On the other hand, termination is reduced to t_l. One can also verify that validity is inherited from the regular BA protocol: if every honest party starts with input x, no honest party signs any other value $x' \neq x$, and hence there cannot be a list of $n - t_l$ signatures on x', given that $t_l \leq t_v$.

Lemma 8.5.1. *There is a Byzantine agreement protocol* BA *with validity, consistency and termination parameters* (t_v, t_c, t_l)*, for any* $t_l < \frac{n}{3}$*,* $t_l \leq t_v < \frac{n}{3}$ *and* $t_c < n - 2t_l$*, assuming a PKI infrastructure setup* $\mathcal{F}_{PKI}$*. The expected maximum delay for the output is* $\tau_{aba} = O(\delta)$*.*

8.5.2 Two-Threshold Asynchronous MPC

In order to realize $\mathcal{F}_{\text{ASYNC}}$ with full security up to t and security with no termination (correctness and privacy) up to T, where $t < \frac{n}{3}$ and

$T + 2t < n$, we follow the ideas from [Coh16, HNP05, HNP08], and replace the single-threshold asynchronous BA protocol for the one that we obtained in Section 8.5.1 with increased consistency $t_c < n - 2t_l$.

The protocol works with a threshold FHE setup, similar to [Coh16], which we model with the functionality $\mathcal{F}_{\text{SETUP}}^{\texttt{FHE}}$, which is the same as $\mathcal{F}_{\text{SETUP}}$ from Section 8.4.1, except that the threshold encryption scheme is fully-homomorphic. For completeness, we review the definition of a FHE scheme in Section 2.2.3.

The protocol uses in addition a number of sub-protocols:

- $\texttt{BA}$ is a Byzantine agreement protocol with liveness threshold $t_l = t < n/3$, validity $t \leq t_v < n/3$ and consistency $t_c = T < n - 2t$.
- $\Pi_{\texttt{zk}}$ is a bilateral zero-knowledge protocol, similar to the one in Section 8.4.

Very roughly, the protocol asks each party P_i to encrypt its input x_i and distribute it to all parties. Then, parties homomorphically evaluate the function over the encrypted inputs to obtain an encrypted output, and jointly decrypt the output. Of course, the protocol does not work like that. In order to achieve robustness, we need that every party proves in zero-knowledge the correctness of essentially every value provided during the protocol execution.

We are interested in ZK proofs for two relations, parametrized by a threshold encryption scheme with public encryption key ek:

1. *Proof of Plaintext Knowledge:* The statement consists of ek, and a ciphertext c. The witness consists of a plaintext m and randomness r such that $c = \mathsf{Enc}_{\mathsf{ek}}(m; r)$.
2. *Proof of Correct Decryption:* The statement consists of ek, a ciphertext c, and a decryption share d. The witness consists of a decryption key share dk_i, such that $d = \mathsf{Dec}_{\mathsf{dk}_i}(c)$.

The protocol proceeds in three phases: the input stage, the computation and threshold-decryption stage, and the termination stage.

Input Stage. The goal of the input stage is to define an encrypted input for each party. In order to ensure that the inputs are independent, the parties are required to perform a proof of plaintext knowledge of their

ciphertext. It is known that input completeness and guaranteed termination cannot be simultaneously achieved in asynchronous networks, since one cannot distinguish between an honest slow party and an actively corrupted party. Given that we only guarantee termination up to t corruptions, we can take into account $n - t$ input providers.

The input stage is as follows: each party P_i encrypts its input to obtain a ciphertext c_i. It then constructs a certificate π_i that P_i knows the plaintext of c_i and that c_i is the only input of P_i, using bilateral zero-knowledge proofs and signatures. It then sends (c_i, π_i) to every other party, and constructs a *certificate of distribution* $\texttt{dist}_i$, which works as a non-interactive proof that (c_i, π_i) was distributed to at least $n-t$ parties. This certificate is sent to every party.

After P_i collects $n - t$ certificates of distribution, it knows that at least $n - t$ parties have proved knowledge of the plaintext of their input ciphertext and distributed the ciphertext correctly to $n - t$ parties. If the number of corruptions is smaller than $n-t$, this implies that each of the $n - t$ parties have proved knowledge of the plaintext of their input ciphertext and also have distributed the ciphertext to at least 1 honest party. At this point, if each party is instructed to echo the certified inputs they saw, then every honest party will end up holding the $n - t$ certified inputs. To determine who they are, the parties compute a common set of input providers. For that, n asynchronous Byzantine Agreement protocols are run, each one to decide whether a party's input will be taken into account. To ensure that the size of the common set is at least $n - t$, each party P_i inputs 1 to the BAs of those parties for which it saw a certified input. It then waits until there are $n - t$ ones from the BAs before inputting any 0.

Protocol $\Pi^{\texttt{input}}_{\texttt{aMPC}}(P_i)$

The protocol keeps sets S_i and D_i, initially empty. Let x_i be the input for P_i.

Setup:

1: If activated for the first time input (GetKeys, sid) to $\mathcal{F}^{\texttt{FHE}}_{\text{Setup}}$. We denote the public key ek, a $(n-t, n)$-share dk_i of the corresponding secret key dk, the signing key sk and the verification key vk.

Plaintext Knowledge and Distribution:

1: Compute $c_i = \mathsf{Enc}_{\mathsf{ek}}(x_i)$.
2: Prove to each P_j knowledge of the plaintext of c_i, using Π_{zk}.
3: Upon receiving a correct proof of plaintext knowledge for a ciphertext c_j from P_j, send $\sigma_i^{\mathtt{popk}} = \mathsf{Sign}_{\mathsf{sk}_i}(c_j)$ to P_j.
4: Upon receiving $n-t$ signatures $\{\sigma_j^{\mathtt{popk}}\}$, compute $\pi_i = \{\sigma_j^{\mathtt{popk}}\}$ and send (c_i, π_i) to all parties.
5: Upon receiving a message (c_j, π_j) from P_j, send $\sigma_i^{\mathtt{dist}} = \mathsf{Sign}_{\mathsf{sk}_i}((c_j, \pi_j))$ to P_j. Add $(j, (c_j, \pi_j))$ to S_i.
6: Upon receiving $n-t$ signatures $\{\sigma_j^{\mathtt{dist}}\}$, compute $\mathtt{dist}_i = \{\sigma_j^{\mathtt{dist}}\}$ and send $((c_i, \pi_i), \mathtt{dist}_i)$ to all parties.
7: Upon receiving $((c_j, \pi_j), \mathtt{dist}_j)$ from P_j, add j to D_i.

Select Input Providers: Once $|D_i| > n - t$, stop the above rules and proceed as follows:

1: Send S_i to every party.
2: Once $n-t$ sets $\{S_j\}$ are collected, let $R = \bigcup_j S_j$ and enter n asynchronous Byzantine agreement protocols $\mathtt{BA}$ with inputs $v_1, \ldots, v_n \in \{0,1\}$, where $v_j = 1$ if $\exists (j, (c_j, \pi_j)) \in R$. Keep adding possibly new received sets to R.
3: Wait until there are at least $n-t$ outputs which are one. Then, input 0 for the BAs which do not have input yet.
4: Let $w_1, \ldots, w_n$ be the outputs of the BAs.
5: Let $\mathtt{CoreSet} := \{j | w_j = 1\}$.
6: For each $j \in \mathtt{CoreSet}$ with $(j, (c_j, \pi_j)) \in R$, send $(j, (c_j, \pi_j))$ to all parties. Wait until each tuple $(j, (c_j, \pi_j)), j \in \mathtt{CoreSet}$ is received.

Computation and Threshold-Decryption Stage. After input stage, parties have agreed on a common subset $\mathtt{CoreSet}$ of size at least $n-t$ parties, and each party holds the $n-t$ ciphertexts corresponding to the encryption of the input from each party in $\mathtt{CoreSet}$. In the computation stage, the parties homomorphically evaluate the function, resulting on the ciphertext c encrypting the output. In the threshold-decryption stage, each party P_i computes the decryption share $d_i = \mathsf{Dec}_{\mathsf{dk}_i}(c)$, and proves in zero-knowledge simultaneously towards all parties that the decryption share is correct. Once $n-t$ correct decryption shares on the same ciphertext are collected, P_i reconstructs the output y_i.

Protocol $\Pi^{\mathtt{comp}}_{\mathtt{aMPC}}(P_i)$

Start once $\Pi^{\mathtt{input}}_{\mathtt{aMPC}}(P_i)$ is completed. Let `CoreSet` be the resulting set of at least $n-t$ parties, and let the input ciphertexts be c_j, for each $j \in \mathtt{CoreSet}$.

Function Evaluation:

1: For each $j \notin \mathtt{CoreSet}$, assume a default valid ciphertext c_j for P_j.
2: Locally compute the homomorphic evaluation of the function $c = f_{\mathsf{ek}}(c_1, \ldots, c_n)$.

Threshold Decryption:

1: Compute a decryption share $d_i = \mathsf{Dec}_{\mathsf{dk}_i}(c)$.
2: Prove, using $\Pi_{\mathtt{zk}}$, to each P_j that d_i is a correct decryption share of c.
3: Upon receiving a correct proof of decryption share for a ciphertext c' and decryption share d_j from P_j, send $\sigma_i^{\mathtt{pocs}} = \mathsf{Sign}_{\mathsf{sk}_i}((d_j, c'))$ to P_j.
4: Upon receiving $n-t$ signatures $\{\sigma_j^{\mathtt{pocs}}\}$ on the same pair (d_i, c'), compute $\mathtt{ProofShare}_i = \{\sigma_j^{\mathtt{pocs}}\}$ and send $((d_i, c'), \mathtt{ProofShare}_i)$ to all parties.
5: Upon receiving $n-t$ valid pairs $((d_j, c'), \mathtt{ProofShare}_j)$ for the same c', compute the output $y_i = \mathsf{Rec}(\{d_j\})$.

Termination Stage. The termination stage ensures that all honest parties terminate with the same output. This stage is essentially a Bracha broadcast [BT85] of the output value. The idea is that each party P_i votes for one output y_i and continuously collects outputs votes. More concretely, P_i sends y_i to every other party. If P_i receives $n-2t$ votes on the same value y, it knows that y is the correct output (because at least an honest party obtained the value y as output if the security threshold $T < n-2t$ is satisfied). Hence, if no output was computed yet, it sets $y_i = y$ as its output and sends y_i to every other party. Observe that if the security threshold is not satisfied, the adversary can tamper the outputs, but so can the simulator. Once $n-t$ votes on the same value y are collected, terminate with output y. If a party receives $n-t$ votes on y, and termination should be guaranteed ($f \leq t$), there are $n-2t$ honest parties that voted for y, and hence every honest party which did not output will at some point collect $n-2t$ votes on y, and hence will also vote for y. Since each honest party which terminated voted for y and each honest party which did not terminated voted for y as well, this means that all honest parties which did not terminate will receive $n-t$ votes for y.

Protocol $\Pi^{\texttt{term}}_{\texttt{aMPC}}(P_i)$

During the overall protocol, execute this protocol concurrently.

Waiting for Output:

1: Wait until the output c is computed from $\Pi^{\texttt{comp}}_{\texttt{aMPC}}(P_i)$.

Adopt Output:

1: Wait until receiving $n - 2t$ votes for the same value y.
2: Adopt y as output, and send y to every other party.

Termination:

1: Wait until receiving $n - t$ votes for the same value y.
2: Terminate.

Let us denote $\Pi_{\texttt{aMPC}}$ the protocol that executes concurrently the protocols $\Pi^{\texttt{input}}_{\texttt{aMPC}}$, $\Pi^{\texttt{comp}}_{\texttt{aMPC}}$ and $\Pi^{\texttt{term}}_{\texttt{aMPC}}$. Each party, at every activation, tries to progress with any of the subprotocols. If they cannot, they output (ClockReady) to $\mathcal{G}_{\text{CLK}}$ so that the clock advances. In Section D.4, we prove the following theorem.

Theorem 8.5.2. *The protocol* $\Pi_{\texttt{aMPC}}$ *uses* $\mathcal{F}^{\texttt{FHE}}_{\text{SETUP}}$ *as setup and realizes* $\mathcal{F}_{\text{ASYNC}}$ *on any function* f *on the inputs, with full security up to* t *corruptions and security without termination up to* T*, for any* $t < n/3$ *and* $T + 2t < n$*. The total maximum delay for the honest parties to obtain output is* $\tau_{\texttt{asynch}} = \tau_{\texttt{aba}}(\delta) + 2\tau_{\texttt{zk}}(\delta) + 9\delta$*.*

8.6 Impossibility Results

In this section we argue that the obtained trade-offs are optimal. We prove that any MPC protocol that achieves full security with responsiveness up to t corruptions, and extended security with unanimous abort up to T corruptions needs to satisfy $T + 2t < n$. Since full security is stronger than security with unanimous abort, these bounds also hold for the case where the extended security is full security.

Lemma 8.6.1. *Let* t*,* T *be such that* $T + 2t \geq n$*. There is no MPC protocol* Π *that achieves full security with responsiveness up to* t *corruptions, and extended security with unanimous abort up to* $T \geq t$ *corruptions.*

Proof. Let δ be the unknown delay upper bound. Moreover, let $\delta' \ll \delta$

be such that the time to execute Π when messages are scheduled within δ' is $\tau(\delta') < \delta$.

Assume without loss of generality that $3t = n$. We prove impossibility for the case where the function to be computed is the majority function. Consider three sets S_0, S_1 and S, where $|S_0| = |S_1| = t$ and $|S| = T$.

First, consider an execution where parties in S_0 and S are honest and have input 0, and parties in S_1 are corrupted and crash. Moreover, the adversary *instantly* delivers the messages between S_0 and S (within δ'). Since full security with responsiveness is guaranteed, parties in S_0 output 0 at time $\tau(\delta')$. Similarly, in an execution where parties in S_1 and S are honest and have input 1, the parties in S_1 output at time $\tau(\delta')$.

Now, consider an execution where S is corrupted, and the parties in S_0 and S_1 have inputs 0 and 1 respectively. The corrupted parties in S emulate an honest protocol execution with input $b \in \{0, 1\}$ with the parties in S_b. Moreover, the adversary delays δ the messages between S_0 and S_1. A party in S_0 (resp. S_1) cannot distinguish between the two executions, because it outputs at time $\tau(\delta') < \delta$, and hence outputs 0 (resp. 1).

However, since T parties are corrupted, the protocol provides security with unanimous abort meaning that in the ideal world all honest parties output the same value (which may be $\perp$).

This contradicts the fact that Π achieves full security with responsiveness up to t corruptions and unanimous abort up to T corruptions. □

In addition, classical bounds in synchronous MPC with full security, show that full security for dishonest majority $T \geq n/2$ is impossible [Cle86]. As a consequence, MPC with extended full security is impossible for dishonest majority.

8.7 Conclusions

We summarize all our results. Using the compiler from Section 8.4 and the following instantiations:

- A bilateral zero-knowledge protocol like in [DDO+01], which uses CRS.

- A synchronous MPC with full security (resp. unanimous abort) for $T < n/2$ (resp. $T < n$), using a protocol such as [RB89] (resp. [FGH+02, GL02]).
- A synchronous broadcast protocol for $T < n$ such as [DS83] from PKI.
- An asynchronous MPC with full security up to $t < n/3$ and security without termination up to $T < n - 2t$, as described in Section 8.5.2, based on PKI and threshold FHE (achievable from CRS [AJL+12]).

We obtain the following corollaries, where $T_{\mathtt{sync}}(\Delta)$ and $T_{\mathtt{BC}}(\Delta)$ are the running times for the synchronous MPC protocol and the synchronous broadcast:

Corollary 8.7.1. *There exists a protocol parametrized by $\Delta \geq \delta$, which realizes $\mathcal{F}_{HYB}^{\mathtt{fs}}$ on any function f, with full security with responsiveness t and full security T for any $t < \frac{n}{3}$ and $T < \min\{n/2, n - 2t\}$, in the $(\mathcal{G}_{CLK}, \mathcal{N}^{\delta}, \mathcal{F}_{PKI}, \mathcal{F}_{CRS})$-hybrid world. The expected maximum delay of the asynchronous phase is $\tau_{\mathtt{asynch}} = O(\delta)$, and the maximum delay of the synchronous phase is $\tau_{\mathtt{OD}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{zk}}(\Delta)$ if an output was delivered in the asynchronous phase, and otherwise is $\tau_{\mathtt{OND}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{sync}}(\Delta)$.*

For $t_r = \frac{n}{4}$, we obtain $\mathcal{F}_{\mathrm{HYB}}^{\mathtt{fs}}$ with correctness with privacy for any $t_s < \frac{n}{2}$.

Corollary 8.7.2. *There exists a protocol parametrized by $\Delta \geq \delta$, which realizes $\mathcal{F}_{HYB}^{\mathtt{ua}}$ on any function f, with full security with responsiveness t and full security T for any $t < \frac{n}{3}$ and $T < n - 2t$, in the hybrid world $(\mathcal{G}_{CLK}, \mathcal{N}^{\delta}, \mathcal{F}_{PKI}, \mathcal{F}_{CRS})$. The expected maximum delay of the asynchronous phase is $\tau_{\mathtt{asynch}} = O(\delta)$, and the maximum delay of the synchronous phase is $\tau_{\mathtt{OD}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{zk}}(\Delta)$ if an output was delivered in the asynchronous phase, and otherwise is $\tau_{\mathtt{OND}} = T_{\mathtt{BC}}(\Delta) + T_{\mathtt{sync}}(\Delta)$.*

Appendix D

Details of Chapter 8

D.1 UC Zero-Knowledge and Synchronous MPC

For completeness, we formally describe the UC functionalities for zero-knowledge [CF01] and synchronous MPC [KMTZ13].

D.1.1 Zero-Knowledge

We formally describe the UC ZK functionality, which allows a prover to prove knowledge of a certain witness w for a statement x satisfying a relation R.

Functionality ZK

ZK is connected to a global clock functionality $\mathcal{G}_{\text{CLK}}$. It is parameterized by a prover P, verifier V, a relation R, and a delay time $\tau_{\mathtt{zk}}$. It also stores the current time τ and keeps a buffer `buffer` of messages containing the proofs that is initially empty.
Each time the functionality is activated, it first queries $\mathcal{G}_{\text{CLK}}$ for the current time and updates τ accordingly.

Zero-Knowledge Proof:

1: On input (x, w) from P, if $R(x, w) = 1$, create a new identifier sid and record the tuple $(\tau, \tau+1, (x, w), \text{sid})$.Then sends (x, sid) to the adversary.
2: On input $(\text{GETPROOF}, \text{sid})$ from V, for each tuple $(T_{\texttt{init}}, T_{\texttt{end}}, (x, w), \text{sid})$ such that $T_{\texttt{end}} \leq \tau$, remove it from `buffer` and output (x, sid) to V.
3: On input $(\text{DELAY}, T, \text{sid})$ from the adversary, if there is a tuple $(T_{\texttt{init}}, T_{\texttt{end}}, (x, w), \text{sid})$ in `buffer` and $T_{\texttt{end}} + T \leq T_{\texttt{init}} + \tau_{\texttt{zk}}$, then set $T_{\texttt{end}} = T_{\texttt{end}} + T$ and return (DELAY-OK) to the adversary. Otherwise, ignore the message.

D.1.2 Synchronous MPC

We describe the ideal functionality $\mathcal{F}_{\text{SYNC}}^{\texttt{fs}}$ for full security and $\mathcal{F}_{\text{SYNC}}^{\texttt{ua}}$. The functionality is connected to a global clock $\mathcal{G}_{\text{CLK}}$ and is parametrized by the delay time $\tau_{\texttt{sync}}$ for which the honest parties obtain the output. For simplicity, we model the synchronous functionality with deterministic termination, but one can extend this to probabilistic termination using the frameworks presented in [CCGZ16, CCGZ17].

Tamper Function for Synchronous SFE. The function $\mathsf{Tamper}_T^{\text{SYNCH}}$ models the adversary's capabilities for a SFE functionality secure up to a single threshold T. The adversary can tamper with the output and learn the inputs from honest parties if and only if the number of corruptions is larger than T.

Definition D.1.1. We define a synchronous SFE functionality secure up to T corruptions if it has the following tamper function $\mathsf{Tamper}_T^{\text{SYNCH}}$:

Function $\mathsf{Tamper}_T^{\text{SYNCH}}$

$(c, p) = \mathsf{Tamper}_T^{\text{SYNCH}}$, where:
- $c = 1$, $p = 1$ if and only if $|\mathcal{P} \setminus \mathcal{H}| > T$.

Functionality $\mathcal{F}_{\text{SYNC}}^{\texttt{fs}}$

$\mathcal{F}_{\text{SYNC}}$ is connected to a global clock $\mathcal{G}_{\text{CLK}}$. $\mathcal{F}_{\text{SYNC}}$ is parameterized by a set $\mathcal{P}$

of n parties, a function f and a tamper function $\mathsf{Tamper}_T^{\text{SYNCH}}$, and a delay time at which the parties obtain output $\tau_{\mathtt{sync}}$. Additionally, it initializes $\tau = 0$ and, for each party P_i, $x_i = y_i = \perp$. It keeps the set of honest parties $\mathcal{H}$.
Upon receiving input from any party or the adversary, it queries $\mathcal{F}_{\text{CLOCK}}$ for the current time and updates τ accordingly.

Party:

1: On input $(\text{INPUT}, v_i, \text{sid})$ from each party $P_i \in \mathcal{H}$ at a fixed time τ':
   - If $x_i = \perp$, it sets $x_i = v_i$.
   - Set $\tau_{\mathtt{out}} = \tau' + \tau_{\mathtt{sync}}$.
2: If for each party $P_i \in \mathcal{H}$ $x_i \neq \perp$, set each $y_i = f(x_1, \ldots, x_n)$.
3: On input $(\text{GETOUTPUT}, \text{sid})$ from honest party P_i or the adversary (for corrupted P_i), if $\tau \geq \tau_{\mathtt{out}}$, it outputs $(\text{OUTPUT}, y_i, \text{sid})$ to P_i.

Adversary: Upon party corruption, set $(c, p) = \mathsf{Tamper}_T^{\text{SYNCH}}((x_1, \ldots, x_n), \mathcal{H})$.

1: On input $(\text{TAMPEROUTPUT}, P_i, y_i', \text{sid})$ from the adversary, if $c = 1$, set $y_i = y_i'$.
2: If $p = 1$, output $(x_1, \ldots, x_n)$ to the adversary.
3: On input $(\text{INPUT}, v_i, \text{sid})$ from the adversary on behalf of P_i, set $x_i = v_i$.

In the version where $\mathcal{F}_{\text{SYNC}}^{\mathtt{ua}}$ provides security with unanimous abort, the adversary can in addition choose to set the output to $\perp$ for all parties after learning the output.

D.2 Proof of Theorem 8.4.3

In this section, we show the proof of the Theorem 8.4.3, stated in Section 8.4.

Completeness. We first show that the protocol is complete. That is, if there are no corruptions, no environment can distinguish the real world from the ideal world. To this end, we need to argue that the output the parties obtain in both worlds are exactly the same. Observe that even if the adversary does not corrupt any party, it can still delay messages.

Given that the time-out occurs after $\tau_{\mathtt{asynch}} = T_{\mathtt{asynch}}(\delta) + T_{zk}(\delta) + \delta$ clock ticks and there are no corruptions, every honest party obtains

output during the asynchronous phase. More concretely, each honest party obtains an output $[y_{\texttt{asynch}}]$ from $\Pi_{\texttt{aMPC}}$ and manages to collect a list L of $n-t$ signatures on this ciphertext during the asynchronous phase, decrypts $[y_{\texttt{asynch}}]$ and obtains the output $y_{\texttt{asynch}}$.

Soundness. To argue soundness, we first describe the simulator. The simulator $\mathcal{S}_{hyb}$ has to simulate the view of the dishonest parties during the protocol execution.

Algorithm $\mathcal{S}_{hyb}$

Clock / Timeout At every activation, the simulator does the following:

1: Query $\mathcal{G}_{\text{CLK}}$ for the current time and updates τ accordingly.
2: If $\tau \geq \tau_{\texttt{tout}}$, set $\texttt{sync} = \texttt{true}$, $\tau_{\texttt{sync}} = \tau$.

Network Messages:

The simulator prepares a set $\texttt{buffer} = \varnothing$ to simulate the messages that are sent to corrupted parties throughout the simulation (recall the variable $\texttt{buffer}$ in $\mathcal{N}$). More concretely, it does the following:

1: On input δ from $\mathcal{F}_{\text{HYB}}$, output δ to the adversary.
2: On input $(\text{FETCHMESSAGES}, i)$ from P_i, for each message tuple $(T_{\texttt{init}}, T_{\texttt{end}}, P_k, P_i, m, \texttt{id}_m)$ from $\texttt{buffer}$ where $T_{\texttt{end}} \leq \tau$, output (k, m) to P_i.
3: On input $(\text{DELAY}, D, \texttt{id})$ from the adversary, if there exists a tuple $(T_{\texttt{init}}, T_{\texttt{end}}, P_i, P_j, m, \texttt{id})$ in $\texttt{buffer}$ and $T_{\texttt{end}} + D \leq T_{\texttt{init}} + \delta$, then set $T_{\texttt{end}} = T_{\texttt{end}} + D$ and return (DELAY-OK) to the adversary. Otherwise, ignore the message.

Setup:

1: The simulator generates the keys at the beginning of the execution. That is, it computes $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Keygen}_{(n-t,n)}(1^\kappa)$, where $\mathsf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$, and $(\mathsf{vk}_j, \mathsf{sk}_j) \leftarrow \mathsf{SigGen}(1^\kappa)$ for each party P_j. Then, it records the tuple $(\text{sid}, \mathsf{ek}, \mathsf{dk}, \mathsf{vk}, \mathsf{sk})$, where $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ and $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
2: On input $(\text{GETKEYS}, \text{sid})$ from a corrupted party P_i, send output $(\text{sid}, \mathsf{ek}, \mathsf{dk}_i, \mathsf{vk}, \mathsf{sk}_i)$ to P_i.

Asynchronous Phase:

It receives the time output $\tau_{\texttt{asynch}}$ from $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$. It keeps a variable τ_i for each party P_i.

// Internal emulation of $\Pi_{\texttt{aMPC}}$

1: Emulate the messages of the protocol $\Pi_{\texttt{aMPC}}$. If a corrupted party P_i is supposed to gets an output from the protocol, output $c_0 = [0]$, an encryption of 0.
// Internal emulation of $\mathcal{N}$.
2: As soon as $\tau_i = 0$ for an honest party P_i, input to `buffer`, the tuple $(\tau, \tau+1, i, j, [y], \mathsf{Sign}([y], \mathsf{sk}_i)), \texttt{id})$, for each party P_j and freshly generated `id`. Output $(\textsc{Sent}, i, j, ([y], \mathsf{Sign}([y], \mathsf{sk}_i))), \texttt{id})$ to the adversary.
3: As soon as the current time τ is such that there are $n - t$ tuples $(\tau_1, \tau_2, j, i, ([y], \mathsf{Sign}([y], \mathsf{sk}_j)))$ such that $\tau_2 \leq \tau$ for the same i in `buffer`, input to `buffer` the tuple $(\tau, \tau + 1, i, j, ([y], L'), \texttt{id})$, for each P_j, where L' contains the list of signatures.
// Internal emulation of $\Pi_{\texttt{zk}}$.
4: The simulator internally emulate the delays of $\Pi_{\texttt{zk}}$. Upon receiving an output y from $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$, it computes an encryption $[y]$ under the key ek. Then, it computes the decryption shares of the corrupted parties $d_i = \mathsf{DecShare}_{\mathsf{sk}_i}(c_0)$, and sets the decryption shares from honest parties such that $(d_1, \ldots, d_n)$ forms a secret sharing of the output value y.
5: Every time the adversary requests validity of the decryption share d_i from an honest party P_i, it responds with a confirmation of the validity of d_i.
6: Every time a corrupted P_i provides a proof of correct decryption (c', d'), check whether $c' = c_0$ and $d' = d_i$. If so, record that a correct proof of decryption was input by P_i.
// Delivery of honest parties' outputs.
7: On input $(\textsc{Output}, P_i, \text{sid})$ from $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$, where P_i is an honest party, if P_i obtained $n - t$ correct decryption shares, input $(\textsc{DeliverOutput}, P_i, \text{sid})$ to $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$.

Synchronous Phase:

// Internal emulation of $\Pi_{\texttt{sBC}}$
1: The simulator emulates the messages from the broadcast protocol. For each emulated honest party P_i that received a valid pair $([y], L)$ in the asynchronous phase, output $([y], L)$ to the adversary after $T_{\texttt{BC}}(\Delta)$ clock ticks.
2: On input a valid pair $([y], L)$ from the adversary, after $T_{\texttt{BC}}(\Delta)$ start the emulation of $\Pi_{\texttt{zk}}$.
// Internal emulation of $\Pi_{\texttt{zk}}$

3: Output the message (c_0, d_i) at the corresponding time, for each honest P_i. That is, keep a local delay u_i for each honest party, which can be updated on input $(\text{DELAY}, D, \texttt{id})$, and output the message if $\tau \geq \tau_{\texttt{sync}} + T_{\texttt{BC}}(\Delta) + u_i$.
// Internal emulation of $\Pi_{\texttt{sMPC}}$.
4: If no valid pair was received from the adversary, and no honest party received a valid pair in the asynchronous phase, emulate the messages of the synchronous MPC protocol.

Tamper Function:

1: On input $(\text{TAMPEROUTPUT}, P_i, y_i', \text{sid})$ from the adversary, forward the input to $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$.
2: On input $(x_1, \ldots, x_n)$ from $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$, output it to the adversary.
3: When the adversary blocks an output from the asynchronous MPC protocol in the real world, the simulator forwards the input $(\text{BLOCKASYNCHOUTPUT}, P_i, \text{sid})$ to $\mathcal{F}_{\text{HYB}}^{\texttt{fs}}$.

We need to prove that the real and ideal worlds are indistinguishable. First, we remark that the simulator emulates the network by keeping a variable `buffer` which stores the messages that are sent. If a corrupted party inputs a message to $\mathcal{N}$ in the real world, the simulator inputs the corresponding tuple to `buffer` exactly the same way as $\mathcal{N}$. Moreover, the simulator have to input to `buffer` all messages that are sent from honest parties to corrupted parties in the real world. One can see that such messages correspond to signatures on an encrypted output and lists of such signatures. All these messages can be simulated. Observe that the simulator uses an encryption of 0 instead of $[y]$ in all the messages above. By the security of the threshold encryption scheme, both messages are indistinguishable. We remark that the simulator has knowledge of all the keys from the parties, since it simulates the setup functionality $\mathcal{F}_{\text{SETUP}}$.

Now we analyze each phase individually.

Setup Phase. It is straightforward to see that the messages that the adversary sees during the setup phase are identical in both worlds. This is because the simulator executes the key generation algorithms for both the threshold encryption and the digital signature scheme as the functionality $\mathcal{F}_{\text{SETUP}}$ in the ideal world.

Asynchronous Phase. We argue that the view of the adversary is indistinguishable in both worlds.

Internal emulation of $\Pi_{\texttt{aMPC}}$. The simulator keeps a delay variable τ_i for each party P_i, which it sets the same way as the adversary. When $\tau_i = 0$, a corrupted party P_i gets the encryption $[y]$ in the real world. In the ideal world, the simulator outputs an encryption of 0, $c_0 = [0]$, when $\tau_i = 0$ as well.
Internal emulation of $\mathcal{N}$. In the real world, the corrupted parties obtain two types of messages after obtaining the ciphertext $[y]$: signatures on $[y]$ and lists of signatures. Once an honest party obtains $[y]$ from the asynchronous MPC protocol, it inputs to $\mathcal{N}$ a signature of $[y]$ towards every party. Then, when $n - t$ signatures are collected, the honest party inputs the list to $\mathcal{N}$ towards every party.

The simulator maintains a variable `buffer` which stores the messages that are sent via the network. It then inputs signatures of $[0]$ on behalf of each honest party P_i to `buffer`, towards every party (in particular, towards corrupted parties), and at the corresponding time. Once $n - t$ signatures are collected with destination P_i, the simulator emulates internally the protocol of P_i, and inputs to `buffer` the corresponding list, towards every party.
Internal emulation of $\Pi_{\texttt{zk}}$. The simulator keeps a delay variable u_i for each party P_i, which it sets the same way as the adversary. When the delay is met, a corrupted party P_i gets a proof of correct decryption $([y], d_i)$, where $d_i = \mathsf{DecShare}_{\mathsf{sk}_i}([y])$ from $\Pi_{\texttt{zk}}$ in the real world. In the ideal world, the simulator outputs a pair (c_0, d_i), where $d_i = \mathsf{DecShare}_{\mathsf{sk}_i}(c_0)$, where the decryption shares from honest parties are set such that they reconstruct the value y.
Delivery of honest parties' outputs. The simulator has the power to deliver the outputs of honest parties in the ideal world. Hence, it delivers the outputs at the corresponding time. Namely, when the honest party has the output ciphertext $[y]$ and collects $n - t$ decryption shares in the real world.

Synchronous Phase. We argue again that the view of the adversary is exactly the same in both worlds.
Internal emulation of $\Pi_{\texttt{sBC}}$. In the real world, the parties broadcast all valid pairs $([y], L)$ that were received in the Asynchronous phase. This behavior is emulated by the simulator as follows: the simulator keeps track of the honest parties that obtained a valid pair $([y], L)$ during the asynchronous phase. The simulator then internally emulates $\Pi_{\texttt{sBC}}$ and

outputs the valid pairs $([y], L)$ at the end of the broadcast round, after $T_{\mathtt{BC}}$ clock ticks. Also, if the adversary inputs a valid pair $([y], L)$ during the broadcast round, it also outputs the valid pair $([y], L)$ to each party at the corresponding time.

Internal emulation of $\Pi_{\mathtt{zk}}$. After the round of synchronous broadcasts terminated, if a valid pair $([y], L)$ was received, then in the real world the honest parties send the decryption shares along with proofs of correct decryption using $\Pi_{\mathtt{zk}}$. In the ideal world, the simulator the internal emulation of $\Pi_{\mathtt{zk}}$ is similar to the one during the asynchronous phase.

Internal emulation of $\Pi_{\mathtt{sMPC}}$. If no valid pair was received, in the real world the parties execute $\Pi_{\mathtt{sMPC}}$, whose behavior is directly emulated by the $\mathcal{F}_{\mathrm{HYB}}^{\mathtt{fs}}$ functionality in the ideal world. That is, the simulator forwards the output from $\mathcal{F}_{\mathrm{HYB}}^{\mathtt{fs}}$ to the adversary.

All that is left to do is to argue about the messages the adversary obtains from breaking the correctness, privacy and termination thresholds.

Full Security. In the real world, if the adversary corrupts more than T parties, it can set the output of the asynchronous protocol $\Pi_{\mathtt{aMPC}}$ to any output y, and it can also obtain the inputs from the honest parties. In this case, the simulator learns the inputs from the honest parties as well and can set the output correspondingly.

Similarly, if the adversary corrupts more than $n - 2t$ parties, it can forge a list of signatures on any value and choose the output, potentially violating security. But in this case the simulator can also set the output of $\mathcal{F}_{\mathrm{HYB}}^{\mathtt{fs}}$ in the ideal world and learn the inputs from $\mathcal{F}_{\mathrm{HYB}}^{\mathtt{fs}}$, since the full security threshold is $\min(T, n - 2t)$.

Termination. We remark that even if the responsiveness bound t of is violated, all the adversary can do in the real world is to prevent a party to obtain an output from $\Pi_{\mathtt{aMPC}}$. Hence, responsiveness is lost and the simulator will block the output from the asynchronous phase.

One can prove similarly the theorem with the variant where the hybrid offers unanimous abort (Theorem 8.4.4). The proof is exactly the same as in Theorem 8.4.3, except that the emulation of the synchronous MPC protocol is according to the messages of the protocol $\Pi_{\mathtt{sMPC}}^{\mathtt{ua}}$ that gives security with unanimous abort instead of full security.

D.3 ABA with Increased Consistency

D.3.1 Ideal Functionality

We introduce the asynchronous functionality for Byzantine Agreement, $\mathcal{F}_{\text{ABA}}$. The asynchronous Byzantine Agreement functionality $\mathcal{F}_{\text{ABA}}$ can be seen as a instantiation of the asynchronous MPC functionality $\mathcal{F}_{\text{ASYNC}}$ introduced in Section 8.4.5 with a specific function and tamper function. The function $f^{\mathcal{F}_{\text{ABA}}}$ to evaluate is defined as follows: If the honest parties in the core set have preagreement on an input value x, the output value is also x. Otherwise, the output value is the same for every honest party, but is defined by the adversary.

We define the functionality $\mathcal{F}_{\text{ABA}}$ to be an asynchronous MPC functionality $\mathcal{F}_{\text{ASYNC}}$ evaluating the function $f^{\mathcal{F}_{\text{ABA}}}$, and parametrized by the tamper function $\mathsf{Tamper}^{\text{BA}}_{t_v,t_c,t_l}$ defined below.

Definition D.3.1. We say that a Byzantine Agreement functionality has validity, consistency and termination parameters $T = (t_v, t_c, t_l)$ if it has the following tamper function $\mathsf{Tamper}^{\text{BA}}_{t_v,t_c,t_l}$:

Function $\mathsf{Tamper}^{\text{BA}}_{t_v,t_c,t_l}(x_1, ..., x_n, \mathcal{H})$

$(c, p, d) = \mathsf{Tamper}^{\text{BA}}_{t_v,t_c,t_l}(x_1, ..., x_n, \mathcal{H})$, where:

- $c = 0$ if and only if $|\mathcal{P} \setminus \mathcal{H}| \leq t_v$ and there exists x such that for all $P_i \in \mathcal{H} : x_i = x$, or $|\mathcal{P} \setminus \mathcal{H}| < t_c$.
- $p = 1$.
- $l = 1$ if and only if $|\mathcal{P} \setminus \mathcal{H}| \geq t_l$.

D.3.2 Protocol Description

In this section we show how to increase the consistency of an ABA protocol by sacrificing liveness.

In the following, we describe a protocol which operates with PKI setup $\mathcal{F}_{\text{PKI}}$ and uses a secure ABA protocol $\texttt{BA}$ with parameters (t_v, t_c, t_l) as primitive. It then realizes a binary asynchronous Byzantine Agreement functionality with the same validity $t'_v = t_v$ and termination $t'_l = t_l$, but with consistency $t'_c < n - 2t_l$.

The protocol is quite simple. First, each party P_i run with input x_i the protocol $\texttt{BA}'$, and once an output x is obtained, it computes a signature $\sigma = \mathsf{Sign}(x, \mathsf{sk})$ and sends it to every other party. Once $n - t_l$ signatures on

a value x' are collected, the party sends the list containing the signatures along with the value x' to every other party, and terminates with output x'. The idea is that there cannot be two lists of $n - t_l$ signatures on different values if there are up to $t_c < n - 2t_l$ corruptions.

Protocol $\Pi^{\mathtt{con}}_{\mathtt{aBA}}(P_i)$

Setup:

1: Input (GETDSSKEYS, sid) to $\mathcal{F}_{\text{PKI}}$. Let the signing key be sk and the corresponding verification key vk.

Asynchronous Phase: Upon every activation, progress with the following list of instructions. If not possible, output (CLOCKREADY) to $\mathcal{G}_{\text{CLK}}$.

1: On input x_i, execute $\mathtt{BA}$ on input x_i. Let x denote the output.
2: Compute the signature $\sigma = \mathsf{Sign}(x, \mathsf{sk})$.
3: Input $(\text{SEND}, i, j, (x, \sigma))$, for each party P_j, to $\mathcal{N}$.
4: Upon receiving $\ell \geq n - t$ valid messages of the form (x', σ) from $\mathcal{N}$, let $L = (x', \sigma_1, \ldots, \sigma_l)$ be the list containing these ℓ signatures on x'. Input (SEND, i, j, L), for each party P_j, to $\mathcal{N}$, and terminate with output x'.

Let $\tau_{\mathtt{aba}}(\delta)$ denote the running time of $\mathtt{BA}'$ that has validity, consistency and termination parameters (t_v, t_c, t_l), $t_l \leq \frac{n}{3}$.

Lemma D.3.2. *The protocol* $\Pi^{\mathtt{con}}_{\mathtt{aBA}}$ *operates with PKI setup* $\mathcal{F}_{\text{PKI}}$*, and is a secure ABA protocol with validity, consistency and termination parameters* (t_v, t'_c, t_l)*, for any* $t'_c < n - 2t_l$*. The maximum delay for the output is* $\tau_{\mathtt{con}} = \tau_{\mathtt{aba}}(\delta) + \delta$*.*

Proof. **Completeness.** We first argue that if the adversary does not corrupt any party, the real world and the ideal world are indistinguishable. The output is the same in both worlds. If every party has the same input b, in the real world, $\mathtt{BA}'$ outputs b, and then each party signs b and collects $n-t_l$ signatures on b. This implies that the parties terminate with output b, which is the value that is output in the ideal world as well. The same happens if the parties do not hold the same input. In this case, in the real world, each party obtains the input x_1, signs this value, collects $n-t_l$ signatures and terminates with output x_1. This is also the output of the ideal world.

Soundness. We start describing the simulator. The job of the simulator $\mathcal{S}_{con}$ is to simulate the view of the adversary during the protocol execution. For readability, let us denote the ideal world Byzantine agreement functionality with improved consistency $\mathcal{F}_{\text{ABA}}$.

On a very high level, the simulator simulates internally the messages that the real world functionalities $\mathcal{N}$, $\mathcal{F}_{\text{SETUP}}$ and the protocol $\texttt{BA}'$ output to the adversary. In order to simulate the messages that the adversary obtains from the asynchronous network $\mathcal{N}$, the simulator simply keeps the variable `buffer` as in $\mathcal{N}$, which records the messages sent via $\mathcal{N}$ in the real world, with the delays of the messages. It also records the delays that the adversary inputs, and only delivers the messages when the corresponding party fetches the messages and the delay of the message is 0. To simulate the messages from $\mathcal{F}_{\text{SETUP}}$, the simulator executes the DSS key generation algorithm at the onset of the execution, and outputs the signing keys of the corrupted parties and all the verification keys to the adversary. Finally, to simulate the messages from $\texttt{BA}'$, the simulator waits for the adversary to define a *core set* $\mathcal{I}$ (which by default is the set of honest parties), and after all parties in $\mathcal{I}$ provide his input bit, the simulator computes the output as in $\texttt{BA}'$: if there is preagreement on a value x, that is the output, and otherwise, the output corresponds to the input of the corrupted party with lowest index.

Algorithm $\mathcal{S}_{con}$

Network Messages:

The simulator prepares a set `buffer` $= \varnothing$ to simulate the messages that are sent to corrupted parties throughout the simulation (recall the variable `buffer` in $\mathcal{N}$). More concretely, it does the following:

1: On input $(\text{FETCHMESSAGES}, i)$ from P_i, for each message tuple $(0, P_k, P_i, m, \texttt{id}_m)$ in `buffer`, output (k, m) to P_i.
2: On input $(\text{DELAY } \mathcal{N}, T, \texttt{id})$ from the adversary, if there exists a tuple $(D, P_i, P_j, m, \texttt{id})$ in `buffer` then set $D = D + T$ and return (DELAY-OK) to the adversary. Otherwise, ignore the message.

Setup:

1: The simulator generates the keys at the beginning of the execution. That is, it computes $(\mathsf{vk}_j, \mathsf{sk}_j) \leftarrow \mathsf{SigGen}(1^\kappa)$ for each party P_j. Then, it records the tuple $(\text{sid}, \mathsf{vk}, \mathsf{sk})$, where $\mathsf{vk} = (\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ and $\mathsf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$.
2: On input $(\text{GETKEYS}, \text{sid})$ from a corrupted party P_i, send $(\text{sid}, \mathsf{vk}, \mathsf{sk}_i)$ to P_i.

Main:

1: On input $(\text{NO-INPUT}, \mathcal{P}', \text{sid})$ from the adversary, set a variable $\mathcal{I} = \mathcal{H} \setminus \mathcal{P}'$, and forward $(\text{NO-INPUT}, \mathcal{P}', \text{sid})$ to $\mathcal{F}_{\text{aBA}}$.
2: Upon receiving the input b_i from honest party P_i or the adversary on behalf of a party, set $x_i^{BA} = b_i$. Moreover, if it is from the adversary, forward x_i^{BA} to $\mathcal{F}_{\text{aBA}}$.
3: On input $(\text{OUTPUT}, x, \text{sid})$ from $\mathcal{F}_{\text{aBA}}$, output $(\text{OUTPUT}, x, \text{sid})$ to the adversary.
4: Emulate the messages of the sub-protocol `BA` by keeping the delays of each honest party.
5: As soon as P_i obtains output from `BA`, input to `buffer`, on behalf of P_i, the tuple $(\tau, \tau+1, i, j, (x, \mathsf{Sign}(x, \mathsf{sk}_i)), \mathtt{id})$ for each corrupted party P_j and freshly generated `id`. Output $(\text{SENT}, i, j, (x, \mathsf{Sign}(x, \mathsf{sk}_i)), \mathtt{id})$ to the adversary.
6: Once there are $n - t_l$ tuples of the form $(\tau_1, \tau_2, j, i, (x', \mathsf{Sign}(x', \mathsf{sk}_j)))$ have been delivered from `buffer` to a fixed honest party P_i, input, for each j, to `buffer` the tuple $(\tau, \tau+1, i, j, L, \mathtt{id})$, where L contains the list of signatures on the value x'. Output $(\text{SENT}, i, j, L, \mathtt{id})$ to the adversary.
7: Keep track of the delays so that the parties receive the output at the same time as in the real world.

Tamper Function:

1: On input $(\text{TAMPEROUTPUT}, P_i, y_i', \text{sid})$, where P_i is honest, from the adversary, forward the input to $\mathcal{F}_{\text{aBA}}$.
2: On input $(x_1, \ldots, x_n)$ from $\mathcal{F}_{\text{aBA}}$, output it to the adversary.
3: On input $(\text{BLOCKOUTPUT}, P_i, \text{sid})$, where P_i is honest, from the adversary, forward the input to $\mathcal{F}_{\text{aBA}}$.

In order to prove that the real world and the ideal world are indistinguishable, we divide cases depending on the adversary's capabilities.

If the validity threshold is satisfied, i.e. $|\mathcal{P} \setminus \mathcal{H}| \leq t_v$ and the parties in the core-set have the same input, or the consistency threshold is satisfied, i.e. $|\mathcal{P} \setminus \mathcal{H}| \leq t_c$, then `BA` ensures that the output at Step 1 is consistent

among the honest parties. Let us denote this value x. In this case, if $|\mathcal{P} \setminus \mathcal{H}| \leq t_l$, then every honest party eventually receives a list of $n - T_L$ signatures on x. In the ideal world, the output is x as well. Otherwise, if $|\mathcal{P} \setminus \mathcal{H}| > t_l$, some honest parties may not receive a list of $n - t_l$ signatures on x, and hence they do not receive any output. For these honest parties, the simulator blocks the output value of these parties.

On the other hand, if it is not the case that $|\mathcal{P} \setminus \mathcal{H}| \leq t_v$ where the parties in the core-set have the same input, nor the consistency threshold is satisfied, i.e. $|\mathcal{P} \setminus \mathcal{H}| > t_c$, then it is not guaranteed that the output after Step 1. (from $\mathcal{F}_{\text{ABA}}$) is consistent. However, we still need that if $|\mathcal{P} \setminus \mathcal{H}| \leq t'_c < n - 2t_l$, all final outputs are consistent. That is the case, because there cannot be two lists of signatures of size at least $n - t_l$ on different values. Assume towards contradiction, that there are such two lists. Observe that any two lists of size $n - t_l$, intersect in at least $n - 2t_l$ parties. Since $|\mathcal{P} \setminus \mathcal{H}| \leq t'_c < n - 2t_l$, there must be at least one honest party in this intersection. But honest parties do not send signatures on different values.

Moreover, let us remark that in the real world, the parties only send messages in Step 2 via the network, and in Step 1 via the protocol $\texttt{BA}'$. This means, since the adversary can only delay each network message by up to δ clock ticks, and the output from $\mathcal{F}_{\text{ABA}}$ up to $\tau_{\texttt{aba}}(\delta)$ clock ticks, then the maximum delay for the output is $\tau_{\texttt{con}} = \tau_{\texttt{aba}}(\delta) + \delta$. Hence, it is enough that the simulator has the power to delay the output up to $\tau_{\texttt{con}}$ clock ticks.

□

If we assume an asynchronous Byzantine Agreement $\texttt{BA}'$ which runs concurrently in expected constant time as in [BEY03], with validity, consistency and termination for any $t < \frac{n}{3}$ corruptions, Corollary 8.5.1 follows.

D.4 Proof of Theorem 8.5.3

In this section, we prove the theorem stated in Section 8.5.2.

Completeness. We first show that the protocol is complete. It is easy to see that, if there are no corruptions, no environment can distinguish the real world from the ideal world. First, observe that the output that is

evaluated in both worlds is the same, since the simulator sets the core set containing the same parties as in the real world. Moreover, the simulator delivers the outputs of honest parties at the time at which the honest parties obtain the output and terminate in the real execution.

One can readily verify, that in the protocol, the parties send messages in 9 steps, performs calls to $\Pi_{\mathtt{zk}}$ in two steps, and executes in parallel n BAs during the input provider selection. Hence, the protocol takes at most $\tau_{\mathtt{aba}}(\delta) + 2\tau_{\mathtt{zk}}(\delta) + 9\delta$ clock ticks to execute.

Soundness. At a very high level, the consistency property of BA can affect both correctness and privacy of the overall SFE. Moreover, it is important that the validity of BA is higher than the termination threshold t_l. Otherwise, when parties wait for the input ciphertexts from each $j \in \mathtt{CoreSet}$, it might be that no party has this input ciphertext and the protocol does not terminate. Given that $t_v \geq t_l$, then in the region of thresholds where there are up to t_l corruptions, validity is guaranteed to hold and hence in the input phase parties are guaranteed to collect all tuples $(j, (c_j, \pi_j))$ such that $j \in \mathtt{CoreSet}$. Let us now describe the simulator.

Algorithm $\mathcal{S}_{\text{MPC}}$

Network Messages:

The simulator prepares a set $\mathtt{buffer} = \varnothing$ to simulate the messages that are sent to corrupted parties throughout the simulation (recall the variable buffer in $\mathcal{N}$). More concretely, it does the following:

1: Let δ be the network delay, received from $\mathcal{F}_{\text{ASYNC}}$
2: On input (FETCHMESSAGES, i) from P_i, for each message tuple $(0, P_k, P_i, m, \mathtt{id}_m)$ in buffer, output (k, m) to P_i.
3: On input (DELAY $\mathcal{N}, T, \mathtt{id}$) from the adversary, if there exists a tuple $(D, P_i, P_j, m, \mathtt{id})$ in buffer and $T \leq \delta$, then set $D = D + T$ and return (DELAY-OK) to the adversary. Otherwise, ignore the message.

Setup:

1: The simulator generates the keys at the beginning of the execution. That is, it computes and records $(\mathsf{ek}, \mathsf{dk}) \leftarrow \mathsf{Keygen}_{(n-t_l,n)}(1^\kappa)$, where $\mathsf{dk} = (\mathsf{dk}_1, \ldots, \mathsf{dk}_n)$.
2: On input (GETKEYS, sid) from a corrupted party P_i, output $(\text{sid}, \mathsf{ek}, \mathsf{dk}_i)$ to P_i.

Input Stage:

// Plaintext Knowledge and Distribution.

1: Set $c_i = \mathsf{Enc}_{\mathsf{ek}}(0)$, for each honest party P_i.
2: The simulator keeps track of the delays the adversary sets for the outputs from $\Pi_{\mathtt{zk}}$. Then, when the adversary requests the output of P_i from $\Pi_{\mathtt{zk}}$ at the corresponding time, the simulator responds with a confirmation of the validity of the ciphertext c_i.
3: On input $\sigma_j^{\mathtt{popk}}$ from corrupted party P_j to P_i, input the tuple $(\tau, \tau + 1, P_j, P_i, \sigma_j^{\mathtt{popk}}, \mathtt{id})$ to `buffer`.
4: When a corrupted party P_i inputs $((\mathsf{ek}, c_i), (x_i, r_i))$ to prove plaintext knowledge of c_i to a party P_j, the simulator checks that $c_i = \mathsf{Enc}_{\mathsf{ek}}(x_i, r_i)$. If so, it inputs $(\tau, \tau + 1, P_j, P_i, \sigma_j^{\mathtt{popk}}, \mathtt{id})$ to `buffer`.
5: As soon as there are $n - t$ tuples $(\tau_1, \tau_2, P_j, P_i, \sigma_j^{\mathtt{popk}}, \mathtt{id})$ for different P_j, such that $\tau \geq \tau_2$ in `buffer`, then compute $\pi_i = \{\sigma_j^{\mathtt{popk}}\}$ and input $(\tau, \tau + 1, i, j, (c_i, \pi_i), \mathtt{id})$ for each P_j.
6: On input (c_i, π_i) from a corrupted party P_i to P_j, the simulator inputs $(\tau, \tau + 1, P_i, P_j, (c_i, \pi_i), \mathtt{id})$ to `buffer`.
7: As soon as there is a tuple $(\tau_1, \tau_2, P_j, P_i, (c_j, \pi_j), \mathtt{id})$, such that $\tau \geq \tau_2$ in `buffer`, input a signature to `buffer`. That is, input $(\tau, \tau+1, i, j, \sigma_i^{\mathtt{dist}}, \mathtt{id})$ to `buffer`.
8: As soon as there are $n-t$ tuples $(\tau_1, \tau_2, P_j, P_i, \sigma_j^{\mathtt{dist}}, \mathtt{id})$ for different P_j, such that $\tau \geq \tau_2$ in `buffer`, then start simulating the input provider selection.

// Input Providers.

9: For each party P_i, keep track of the parties which successfully proved plaintext knowledge to P_i. We denote that set S_i.
10: The simulator inputs to `buffer` each set S_i towards every party. That is, input $(\tau, \tau + 1, i, j, S_i, \mathtt{id})$ to `buffer`, for each P_j.
11: Once an emulated honest party P_i received $n - t$ such sets, emulate for that party the execution of the BAs. That is, input a 1 to P_j's BA, if P_j is in one of the received sets. Take into account all the commands tampering the outputs or blocking the outputs of the BAs that come from the adversary, and change the output accordingly.
12: Wait until there are $n - t$ ones as outputs from the BAs. Then, input 0 to the remaining BAs.

13: Define $\texttt{CoreSet}^i$ as the set of parties such that the emulated BA for that party outputted 1. Observe that if the adversary corrupted more than $n-2t$, the consistency of the BAs is not satisfied, since $t_c < n-2t$, and hence the core sets can be different.
14: The simulator emulates each party P_i, by inputting the pairs (c_j, π_j) that it collected in the $n-t$ sets S_j, to $\texttt{buffer}$.

Computation and Threshold Stage:

// Setting the Core Set.
1: Once the simulator computes $\texttt{CoreSet}^i$ from the previous Stage, do the following: if the core sets are consistent, it sends to $\mathcal{F}_{\text{ASYNC}}$ the input values x_i from each corrupted party, and also inputs $(\text{NO-INPUT}, \mathcal{P} \setminus \texttt{CoreSet}, \texttt{id})$ to $\mathcal{F}_{\text{ASYNC}}$. It obtains the output y. Otherwise, input any of the core sets $\texttt{CoreSet}^i$ to $\mathcal{F}_{\text{ASYNC}}$. Then, obtain the inputs from honest parties (if the core set are not consistent, $f \geq n-2t$, the simulator is allowed to obtain the inputs since privacy is not satisfied).
// Computation.
2: For each honest party P_i, the simulator internally computes the evaluated ciphertext $c^i = f_{\mathsf{ek}}(c_1, \ldots, c_{|\texttt{CoreSet}^i|})$, based on the ciphertext from the input providers.
// Threshold Decryption.
3: The simulator computes the decryption share $d_i = \mathsf{DecShare}_{\mathsf{dk}_i}(c^i)$ for each corrupted party P_i, and sets the decryption shares from honest parties such that $(d_1, \ldots, d_n)$ forms a secret sharing of the output value y, if the core sets are consistent. Otherwise, for each honest P_i it can evaluate the function on the inputs in $\texttt{CoreSet}^i$ to obtain y_i, encrypt it, and set the decryption share exactly as in the real world. In this case, the simulator also fixes the output of P_i to y_i.
4: Each time the adversary requests validity of the decryption share d_i from an honest party P_i, the simulator responds with a confirmation of the validity of d_i.
5: As soon as the adversary inputs a decryption share d_i for ciphertext c', the simulator checks the validity of the decryption share, and if it is valid, inputs to $\texttt{buffer}$ a signature on (d_i, c').
6: Once an emulated honest party P_i received $n-t$ signatures on the same pair (d_i, c'), it computes a proof that the decryption share d_i for c' is correct $\texttt{ProofShare}_i = \{\sigma_j^{\texttt{pocs}}\}$. It inputs to $\texttt{buffer}$ the tuple $((d_i, c'), \texttt{ProofShare}_i)$ to every party.

7: When an honest party receives $n - t$ tuples of the form $((d_i, c'), \texttt{ProofShare}_i)$ with the same c', it sets his output bit to y.

Termination Stage:

1: The simulator keeps track of the votes that each party performs. That is, if an emulated honest party P_i received an output y in the previous stage, it inputs y to `buffer`, towards every other party.
2: As soon as an emulated honest party receives $n - 2t$ votes on y, if the party P_i did not vote yet, it sets its output to y, and inputs y to `buffer`, towards every other party.
3: As soon as an emulated honest party receives $n - t$ votes on y, the simulator delivers the party's output in the ideal world.

We define a series of hybrids to argue that no environment can distinguish between the real world and the ideal world.

Hybrids and security proof.

Hybrid 1**.** This corresponds to the real world execution. Here, the simulator knows the inputs and keys of all honest parties.

Hybrid 2**.** We modify the real-world execution in the computation stage. Here, when a corrupted party requests a proof of decryption share from an honest party, the simulator simply gives a valid response without checking the witness from the honest party.

Hybrid 3**.** This is similar to Hybrid 2, but in the computation of the decryption shares is different. In this case, the simulator obtains the output y from $\mathcal{F}_{\text{ASYNC}}$, computes the decryption shares of corrupted parties, and then adjusts the decryption shares of honest parties such that the decryption shares $(d_1, \ldots, d_n)$ form a secret sharing of the output value y. That is, here the simulator does not need to know the secret key share of honest parties to compute the decryption shares. If there are more than $n - 2t$ corrupted parties, privacy is broken, so the simulator obtains the inputs from the honest parties and computes the decryption shares as in the previous hybrid.

Hybrid 4**.** We modify the previous hybrid in the Input Stage. Here, when a corrupted party requests a proof of plaintext knowledge from an honest party, the simulator simply gives a valid response without checking the witness from the honest party.

Hybrid 5**.** We modify the previous hybrid in the Input Stage. Here, the honest parties, instead of sending an encryption of the actual input, they send an encryption of 0.

Hybrid 6**.** This corresponds to the ideal world execution.

In order to prove that no environment can distinguish between the real world and the ideal world, we prove that no environment can distinguish between any two consecutive hybrids.

Claim 1**.** No efficient environment can distinguish between Hybrid 1 and Hybrid 2.

Proof of claim. This follows trivially, since the honest parties always have a valid witness in $\Pi_{\mathtt{zk}}$. ◊

Claim 2**.** No efficient environment can distinguish between Hybrid 2 and Hybrid 3.

Proof of claim. This follows from properties of a secret sharing scheme and the security of the threshold encryption scheme. Given that the threshold is $n - t$, any number corrupted decryption shares below $n - t$ does not reveal anything about the output y. Moreover, one can find shares for honest parties such that $(d_1, \ldots, d_n)$ is a sharing of y. Above $n - t$ corruptions, the simulator obtains the inputs from honest parties, and hence both hybrids are trivially indistinguishable. ◊

Claim 3**.** No efficient environment can distinguish between Hybrid 3 and Hybrid 4.

Proof of claim. This follows trivially, since the honest parties always have a valid witness in $\Pi_{\mathtt{zk}}$. ◊

Claim 4**.** No efficient environment can distinguish between Hybrid 4 and Hybrid 5.

Proof of claim. This follows from the semantic security of the encryption scheme. ◊

Claim 5**.** No efficient environment can distinguish between Hybrid 5 and Hybrid 6.

Proof of claim. This follows, because the simulator in the ideal world and the simulator in Hybrid 5 emulate internally the joint behavior of the ideal assumed functionalities, exactly the same way. ◊

We conclude that the real world and the ideal world are indistinguishable.

Chapter 9

Topology-Hiding Computation

9.1 Introduction

9.1.1 Topology-Hiding Computation

Secure communication over an insecure network is one of the fundamental goals of cryptography. The security goal can be to hide different aspects of the communication, ranging from the content (secrecy), the participants' identity (anonymity), the existence of communication (steganography), to hiding the topology of the underlying network in case it is not complete.

Incomplete networks arise in many contexts, such as the Internet of Things (IoT) or ad-hoc vehicular networks. Hiding the topology can, for example, be important because the position of a node within the network depends on the node's location. This could in information about the node's identity or other confidential parameters. The goal is that parties, and even colluding sets of parties, can not learn anything about the network, except their immediate neighbors.

Incomplete networks have been studied in the context of communication security, referred to as secure message transmission (e.g.[DDWY90]), where the goal is to enable communication between any pair of entities, despite an incomplete communication graph. Also, anonymous commu-

nication has been studied extensively (see, e.g. [Cha81, RC88, SGR97]). Here, the goal is to hide the identity of the sender and receiver in a message transmission. A classical technique to achieve anonymity is the so-called mix-net technique, introduced by Chaum [Cha81]. Here, *mix* servers are used as proxies which shuffle messages sent between peers to disable an eavesdropper from following a message's path. The onion routing technique [SGR97, RC88] is perhaps the most known instantiation of the mix-technique. Another anonymity technique known as *Dining Cryptographers networks*, in short DC-nets, was introduced in [Cha88] (see also [Bd90, GJ04]). However, none of these approaches can be used to hide the network topology. In fact, message transmission protocols assume (for their execution) that the network graph is public knowledge.

The problem of *topology-hiding communication* was introduced by Moran et al. [MOR15]. The authors propose a broadcast protocol in the cryptographic setting, which does not reveal any additional information about the network topology to an adversary who can access the internal state of any number of passively corrupted parties (that is, they consider the semi-honest setting). This allows to achieve topology-hiding MPC using standard techniques to transform broadcast channels into secure point-to-point channels. At a very high level, [MOR15] uses a series of nested multi-party computations, in which each node is emulated by a secure computation of its neighbor. This emulation then extends to the entire graph recursively. In [HMTZ16], the authors improve this result and provide a construction that makes only black-box use of encryption and where the security is based on the DDH assumption. However, both results are feasible only for graphs with logarithmic diameter. Topology hiding communication for certain classes of graphs with large diameter was described in [AM17]. This result was finally extended to allow for arbitrary (connected) graphs in [ALM17a].

A natural next step is to extend these results to settings with more powerful adversaries. Unfortunately, even a protocol in the setting with fail-corruptions (in addition to passive corruptions) turns out to be difficult to achieve. In fact, as shown already in [MOR15], some leakage in the fail-stop setting is inherent. It is therefore no surprise that all previous protocols (secure against passive corruptions) leak information about the network topology if the adversary can crash parties. The core problem is that crashes can interrupt the communication flow of the protocol at any point and at any time. If not properly dealt with by the

protocol, those outages cause shock waves of miscommunication, which allows the adversary to probe the network topology.

A first step in this direction was recently achieved in [BBMM18] where a protocol for topology-hiding communication secure against a fail-stop adversary is given. However, the resilience against crashes comes at a hefty price; the protocol requires that parties have access to secure hardware modules which are initialized with correlated, pre-shared keys. Their protocol provides security with abort and the leakage is arbitrarily small.

In the information-theoretic setting, the main result is negative [HJ07]: any MPC protocol in the information-theoretic setting inherently leaks information about the network graph. They also show that if the routing table is leaked, one can construct an MPC protocol which leaks no additional information. However, the work in [BBC+19] shows that for specific types of graphs, and assuming a bound on the corruption threshold, one can achieve information-theoretic THC.

Finally, in [BBC+20], the authors investigate the minimal assumptions required for topology-hiding broadcast and anonymous broadcast.

9.1.2 Comparison to Previous Work

In [ALM17a] the authors present a broadcast protocol for the semi-honest setting based on random walks. This broadcast protocol is then compiled into a full topology-hiding computation protocol. However, the random walk protocol fails spectacularly in the presence of fail-stop adversaries, leaking a lot of information about the structure of the graph. Every time a node aborts, any number of walks get cut, meaning that they no longer carry any information. When this happens, adversarial nodes get to see which walks fail along which edges, and can get a good idea of where the aborting nodes are in the graph.

We also note that, while we use ideas from [BBMM18], which achieves the desired result in a trusted-hardware model, we cannot simply use their protocol and substitute the secure hardware box for a standard primitive. In particular, they use the fact that each node can maintain an encrypted "image" of the entire graph by combining information from all neighbors, and use that information to decide whether to give output or abort. This appears to require both some form of obfuscation and a trusted setup, whereas our protocol uses neither.

9.1.3 Contributions

We propose the first topology-hiding MPC protocol secure against passive and fail-stop adversaries (with arbitrarily small leakage) that is based on standard assumptions. Our protocol does not require setup, and its security can be based on either the DDH, QR or LWE assumptions. A comparison of our results to previous works in topology-hiding communication is found in Table 9.1.

Theorem 9.1.1 (informal)**.** *If DDH, QR or LWE is hard, then for any MPC functionality $\mathcal{F}$, there exists a topology-hiding protocol realizing $\mathcal{F}$ for any network graph G leaking at most an arbitrarily small fraction p of a bit, which is secure against an adversary that does any number of static passive corruptions and adaptive crashes. The round and communication complexity is polynomial in the security parameter κ and $1/p$.*

Adversary	Graph	Hardness Asm.	Model	Reference
semi-honest	log diam.	Trapdoor Perm.	Standard	[MOR15]
	log diam.	DDH	Standard	[HMTZ16]
	cycles, trees, log circum.	DDH	Standard	[AM17]
	arbitrary	DDH or QR	Standard	[ALM17a]
fail-stop	arbitrary	OWF	Trusted Hardware	[BBMM18]
semi-malicious & fail-stop	arbitrary	DDH or QR or LWE	Standard	[This work]

Table 9.1: Adversarial model and security assumptions of existing topology-hiding broadcast protocols. The table also shows the class of graphs for which the protocols have polynomial communication complexity in the security parameter and the number of parties.

Our topology-hiding MPC protocol is obtained by compiling a MPC protocol from a topology-hiding broadcast protocol leaking at most a fraction p of a bit. We note that although it is well known that without leakage any functionality can be implemented on top of secure communication, this statement cannot be directly lifted to the setting with leakage. In essence, if a communication protocol is used multiple times, it leaks multiple bits. However, we show that our broadcast protocol, leaking

at most a fraction p of a bit, can be executed sequentially and in parallel, such that the result leaks also at most the same fraction p. As a consequence, any protocol can be compiled into one that hides topology and known results on implementing any multiparty computation can be lifted to the topology hiding setting. However, this incurs a multiplicative overhead in the round complexity.

We then present a topology hiding protocol to evaluate any poly-time function using FHE whose round complexity will amount to that of a single broadcast execution. To do that, we first define an enhanced encryption scheme, which we call *Deeply Fully-Homomorphic Public-Key Encryption* (DFH-PKE), with similar properties as the PKCR scheme presented in [AM17, ALM17a] and provide an instantiation of DFH-PKE under FHE. Next, we show how to obtain a protocol using DFH-PKE to evaluate any poly-time function in a topology hiding manner.

We also explore another natural extension of semi-honest corruption, the so-called *semi-malicious* setting. As for passive corruption, the adversary selects a set of parties and gets access to their internal state. But in addition, the adversary can also set their randomness during the protocol execution. This models the setting where a party uses an untrusted source of randomness which could be under the control of the adversary. This scenario is of interest as tampered randomness sources have caused many security breaches in the past [HDWH12, CNE+14]. In this chapter, we propose a general compiler that enhances the security of protocols that tolerate passive corruption with crashes to semi-malicious corruption with crashes.

9.2 Preliminaries

9.2.1 Notation

For a public-key pk and a message m, we denote the encryption of m under pk by $[m]_{\mathsf{pk}}$. Furthermore, for k messages $m_1, \ldots, m_k$, we denote by $[m_1, \ldots, m_k]_{\mathsf{pk}}$ a vector, containing the k encryptions of messages m_i under the same key pk.

For an algorithm $\mathsf{A}(\cdot)$, we write $\mathsf{A}(\cdot\,; U^*)$ whenever the randomness used in $\mathsf{A}(\cdot)$ should be made explicit and comes from a uniform distribution. By $\approx_c$ we denote that two distribution ensembles are computation-

ally indistinguishable.

9.2.2 Model of Topology-Hiding Communication

Adversary. Most of our results concern an adversary, who can *statically passively corrupt* an arbitrary set of parties $\mathcal{Z}^p$, with $\left|\mathcal{Z}^p\right| < n$. Passively corrupted parties follow the protocol instructions (this includes the generation of randomness), but the adversary can access their internal state during the protocol.

A *semi-malicious* corruption (see, e.g., [AJL+12]) is a stronger variant of a passive corruption. Again, we assume that the adversary selects any set of semi-malicious parties $\mathcal{Z}^s$ with $\left|\mathcal{Z}^s\right| < n$ before the protocol execution. These parties follow the protocol instructions, but the adversary can access their internal state and can additionally choose their randomness.

A *fail-stop* adversary can adaptively crash parties. After crashing a party, it stops sending messages. Note that crashed parties are not necessarily corrupted. In particular, the adversary has no access to the internal state of a crashed party unless it is in the set of corrupted parties. This type of fail-stop adversary is stronger and more general than the one used in [BBMM18], where only passively corrupted parties can be crashed. In particular, in our model the adversary does not necessarily learn the neighbors of crashed parties, whereas in [BBMM18] they are revealed to it by definition.

Communication Model. We state our results in the UC framework. We consider a synchronous communication network. Following the approach in [MOR15], to model the restricted communication network we define the $\mathcal{N}$-hybrid model. The $\mathcal{N}$ functionality takes as input a description of the graph network from a special "graph party" P_{graph} and then returns to each party P_i a description of its neighborhood. After that, the functionality acts as an "ideal channel" that allows parties to communicate with their neighbors according to the graph network.

Similarly to [BBMM18], we change the $\mathcal{N}$ functionality from [MOR15] to deal with a fail-stop adversary.

Functionality $\mathcal{N}$

The functionality keeps the following variables: the set of crashed parties $\mathcal{C}$ and the graph G. Initially, $\mathcal{C} = \varnothing$ and $G = (\varnothing, \varnothing)$.

Initialization Step:

1: The party P_{graph} sends graph G' to $\mathcal{N}$. $\mathcal{N}$ sets $G = G'$.
2: $\mathcal{N}$ sends to each party P_i its neighborhood $\mathbf{N}_G(P_i)$.

Communication Step:

1: If the adversary crashes party P_i, then $\mathcal{N}$ sets $\mathcal{C} = \mathcal{C} \cup \{P_i\}$.
2: If a party P_i sends the command $(\textsc{Send}, j, m)$, where $P_j \in \mathbf{N}_G(P_i)$ and m is the message to P_j, to $\mathcal{N}$ and $P_i \notin \mathcal{C}$, then $\mathcal{N}$ outputs (i, m) to party P_j.

Observe that since $\mathcal{N}$ gives local information about the network graph to all corrupted parties, any ideal-world adversary should also have access to this information. For this reason, similar to [MOR15], we use in the ideal-world the functionality $\mathcal{F}_{\text{INFO}}$, which contains only the Initialization Step of $\mathcal{N}$.

To model leakage we extend $\mathcal{F}_{\text{INFO}}$ by a leakage phase, where the adversary can query a (possibly probabilistic) leakage function $\mathcal{L}$ once. The inputs to $\mathcal{L}$ include the network graph, the set of crashed parties and arbitrary input from the adversary.

We say that a protocol leaks one bit of information if the leakage function $\mathcal{L}$ outputs one bit. We also consider the notion of leaking a fraction p of a bit. This is modeled by having $\mathcal{L}$ output the bit only with probability p (otherwise, $\mathcal{L}$ outputs a special symbol $\perp$). Here our model differs from the one in [BBMM18], where in case of the fractional leakage, $\mathcal{L}$ always gives the output, but the simulator is restricted to query its oracle with probability p over its randomness. As noted there, the formulation we use is stronger. We denote by $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ the information functionality with leakage function $\mathcal{L}$.

Functionality $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$

The functionality keeps the following variables: the set of crashed parties $\mathcal{C}$

and the graph G. Initially, $\mathcal{C} = \varnothing$ and $G = (\varnothing, \varnothing)$.

Initialization Step:

1: The party P_{graph} sends graph $G' = (V, E)$ to $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$. $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ sets $G = G'$.
2: $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ sends to each party P_i its neighborhood $\mathbf{N}_G(P_i)$.

Leakage Step:

1: If the adversary crashes party P_i, then $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ sets $\mathcal{C} = \mathcal{C} \cup \{P_i\}$.
2: If the adversary sends the command (LEAK, q) to $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ for the first time, then $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ outputs $\mathcal{L}(q, \mathcal{C}, G)$ to the adversary.

Security Model. Our protocols provide security with abort. In particular, the adversary can choose some parties, who do not receive the output (while the others still do). That is, no guaranteed output delivery and no fairness is provided. Moreover, the adversary sees the output before the honest parties and can later decide which of them should receive it.

Technically, we model such ability in the UC framework as follows: First, the ideal world adversary receives from the ideal functionality the outputs of the corrupted parties. Then, it inputs to the functionality an *abort vector* containing a list of parties who do not receive the output.

Definition 9.2.1. We say that a protocol Π topology-hidingly realizes a functionality $\mathcal{F}$ with $\mathcal{L}$-leakage, in the presence of an adversary who can statically passive corrupt and adaptively crash any number of parties, if it UC-realizes $(\mathcal{F}_{\text{INFO}}^{\mathcal{L}} \parallel \mathcal{F})$ in the $\mathcal{N}$-hybrid model.

9.2.3 Background

Graphs and Random Walks. In an undirected graph $G = (V, E)$ we denote by $\mathbf{N}_G(P_i)$ the neighborhood of $P_i \in V$. The k-neighborhood of a party $P_i \in V$ is the set of all parties in V within distance k to P_i.

In our work we use the following lemma from [ALM17a]. It states that in an undirected connected graph G, the probability that a random walk of length $8|V|^3\tau$ covers G is at least $1 - \frac{1}{2^\tau}$.

Lemma 9.2.2 ([ALM17a])**.** *Let $G = (V, E)$ be an undirected connected graph. Further let $\mathcal{W}(u, \tau)$ be a random variable whose value is the set of*

nodes covered by a random walk starting from u and taking $8|V|^3\tau$ steps. We have

$$\Pr_{\mathcal{W}}[\mathcal{W}(u,\tau) = V] \geq 1 - \frac{1}{2^\tau}.$$

PKCR Encryption. As in [ALM17a], our protocols require a public key encryption scheme with additional properties, called *Privately Key Commutative and Rerandomizable encryption*. We assume that the message space is bits. Then, a PKCR encryption scheme should be: privately key commutative and homomorphic with respect to the OR operation[1]. We formally define these properties below.

Let $\mathcal{PK}$, $\mathcal{SK}$ and $\mathcal{C}$ denote the public key, secret key and ciphertext spaces. As any public key encryption scheme, a PKCR scheme contains the algorithms $\mathsf{Keygen} : \{0,1\}^* \to \mathcal{PK} \times \mathcal{SK}$, $\mathsf{Enc} : \{0,1\} \times \mathcal{PK} \to \mathcal{C}$ and $\mathsf{Dec} : \mathcal{C} \times \mathcal{SK} \to \{0,1\}$ for key generation, encryption and decryption respectively (where Keygen takes as input the security parameter).

Privately Key-Commutative. We require $\mathcal{PK}$ to form a commutative group under the operation $\circledast$. So, given any $\mathsf{pk}_1, \mathsf{pk}_2 \in \mathcal{PK}$, we can efficiently compute $\mathsf{pk}_3 = \mathsf{pk}_1 \circledast \mathsf{pk}_2 \in \mathcal{PK}$ and for every pk, there exists an inverse denoted pk^{-1}.

This group must interact well with ciphertexts; there exists a pair of efficiently computable algorithms $\mathsf{AddLayer} : \mathcal{C} \times \mathcal{SK} \to \mathcal{C}$ and $\mathsf{DelLayer} : \mathcal{C} \times \mathcal{SK} \to \mathcal{C}$ such that

- For every public key pair $\mathsf{pk}_1, \mathsf{pk}_2 \in \mathcal{PK}$ with corresponding secret keys sk_1 and sk_2, message $m \in \mathcal{M}$, and ciphertext $c = [m]_{\mathsf{pk}_1}$,

$$\mathsf{AddLayer}(c, \mathsf{sk}_2) = [m]_{\mathsf{pk}_1 \circledast \mathsf{pk}_2}.$$

- For every public key pair $\mathsf{pk}_1, \mathsf{pk}_2 \in \mathcal{PK}$ with corresponding secret keys sk_1 and sk_2, message $m \in \mathcal{M}$, and ciphertext $c = [m]_{\mathsf{pk}_1}$,

$$\mathsf{DelLayer}(c, \mathsf{sk}_2) = [m]_{\mathsf{pk}_1 \circledast \mathsf{pk}_2^{-1}}.$$

[1]PKCR encryption was introduced in [AM17, ALM17a], where it had three additional properties: key commutativity, homomorphism and rerandomization, hence, it was called Privately Key Commutative and *Rerandomizable* encryption. However, rerandomization is actually implied by the strengthened notion of homomorphism. Therefore, we decided to not include the property, but keep the name.

Notice that we need the secret key to perform these operations, hence the property is called *privately* key-commutative.

OR-Homomorphic. We also require the encryption scheme to be OR-homomorphic, but in such a way that parties cannot tell how many 1's or 0's were OR'd (or who OR'd them). We need an efficiently-evaluatable homomorphic-OR algorithm, $\mathsf{HomOR} : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$, to satisfy the following: for every two messages $m, m' \in \{0,1\}$ and every two ciphertexts $c, c' \in \mathcal{C}$ such that $\mathsf{Dec}(c, \mathsf{sk}) = m$ and $\mathsf{Dec}(c, \mathsf{sk}) = m'$,

$$\{(m, m', c, c', \mathsf{pk}, \mathsf{Enc}(m \vee m', \mathsf{pk}; U^*))\}$$
$$\approx_c$$
$$\{(m, m', c, c', \mathsf{pk}, \mathsf{HomOR}(c, c', \mathsf{pk}; U^*))\}$$

Note that this is a stronger definition for homomorphism than usual; usually we only require correctness, not computational indistinguishability.

In [HMTZ16], [AM17] and [ALM17a], the authors discuss how to get this kind of homomorphic OR under the DDH assumption, and later [ALM17b] show how to get it with the QR assumption. For more details on other kinds of homomorphic cryptosystems that can be compiled into OR-homomorphic cryptosystems, see [ALM17b].

Random Walk Approach [ALM17a]. Our protocol builds upon the protocol from [ALM17a]. We give a high level overview. To achieve broadcast, the protocol computes the OR. Every party has an input bit: the sender inputs the broadcast bit and all other parties use 0 as input bit. Computing the OR of all those bits is thus equivalent to broadcasting the sender's message.

First, let us explain a simplified version of the protocol that is unfortunately not sound, but gets the basic principal across. Each node encrypts its bit under a public key and forwards it to a random neighbor. The neighbor OR's its own bit, adds a fresh public key layer, and it forwards the ciphertext to a randomly chosen neighbor. Eventually, after about $O(\kappa n^3)$ steps, the random walk of every message visits every node in the graph, and therefore, every message will contain the OR of all bits in the network. Now we start the backwards phase, reversing the walk and peeling off layers of encryption.

This scheme is not sound because seeing where the random walks are coming from reveals information about the graph! So, we need to disguise

that information. We will do so by using correlated random walks, and will have a walk running down each direction of each edge at each step (so 2× number of edges number of walks total). The walks are correlated, but still random. This way, at each step, each node just sees encrypted messages all under new and different keys from each of its neighbors. So, intuitively, there is no way for a node to tell anything about where a walk came from.

9.3 Topology-Hiding Broadcast

In this section we present a protocol, which securely realizes the broadcast functionality BC (with abort) in the $\mathcal{N}$-hybrid world and leaks at most an arbitrarily small (but not negligible) fraction of a bit. If no crashes occur, the protocol does not leak any information. The protocol is secure against an adversary that (a) controls an arbitrary static set of passively corrupted parties and (b) adaptively crashes any number of parties. Security can be based either on the DDH, the QR or the LWE assumption. To build intuition we first present the simple protocol variant which leaks at most one bit.

Functionality BC

When a party P_i sends a bit $b \in \{0,1\}$ to the functionality BC, then BC sends b to each party $P_j \in \mathcal{P}$.

9.3.1 Protocol Leaking One Bit

We first introduce the broadcast protocol variant $\mathsf{BC\text{-}OB}$ which leaks at most one-bit. The protocol is divided into n consecutive phases, where, in each phase, the parties execute a modification of the random-walk protocol from [ALM17a]. More specifically, we introduce the following modifications:

Single Output Party: There will be n phases. In each phase only one party, P_o, gets the output. Moreover, it learns the output from exactly one of the random walks it starts.

To implement this, in the respective phase all parties except P_o start their random walks with encryptions of 1 instead of their input bits. This ensures that the outputs they get from the random walks will always be 1. We call these walks *dummy* since the contain no information. Party P_o, on the other hand, starts exactly one random walk with its actual input bit (the other walks it starts with encryptions of 1). This ensures (in case no party crashes) that P_o actually learns the broadcast bit.

Happiness Indicator: Every party P_i holds an *unhappy-bit* u_i. Initially, every P_i is happy, i.e., $u_i = 0$. If a neighbor of P_i crashes, then in the next phase P_i becomes unhappy and sets $u_i = 1$. The idea is that an unhappy party makes all phases following the crash become dummy.

This is implemented by having the parties send along the random walk, instead of a single bit, an encrypted tuple $[b, u]_{\mathsf{pk}}$. The bit u is the OR of the unhappy-bits of the parties in the walk, while b is the OR of their input bits and their unhappy-bits. In other words, a party P_i on the walk homomorphically ORs $b_i \vee u_i$ to b and u_i to u.

Intuitively, if all parties on the walk were happy at the time of adding their bits, b will actually contain the OR of their input bits and u will be set to 0. On the other hand, if any party was unhappy, b will always be set to 1, and $u = 1$ will indicate an abort.

Intuitively, the adversary learns a bit of information only if it manages to break the one random walk which P_o started with its input bit (all other walks contain the tuple $[1, 1]$). Moreover, if it crashes a party, then all phases following the one with the crash abort, hence, they do not leak any information. More formally, parties execute, in each phase, protocol $\mathsf{RandomWalkPhase}$. This protocol takes as global inputs the length $\mathtt{T}$ of the random walk and the P_o which should get output. Additionally, each party P_i has input (d_i, b_i, u_i) where d_i is its number of neighbors, u_i is its unhappy-bit, and b_i is its input bit.

Protocol $\mathsf{RandomWalkPhase}(\mathtt{T}, P_o, (d_i, b_i, u_i)_{P_i \in \mathcal{P}})$

Initialization Stage:

1: Each party P_i generates $\mathtt{T} \cdot d_i$ keypairs $(\mathsf{pk}_{i\to j}^{(r)}, \mathsf{sk}_{i\to j}^{(r)}) \leftarrow \mathsf{Keygen}(1^\kappa)$ where $r \in \{1, \ldots, \mathtt{T}\}$ and $j \in \{1, \ldots, d_i\}$.
2: Each party P_i generates $\mathtt{T} - 1$ random permutations on d_i elements $\left\{\pi_i^{(2)}, \ldots, \pi_i^{(\mathtt{T})}\right\}$
3: For each party P_i, if any of P_i's neighbors crashed in any phase before the current one, then P_i becomes unhappy, i.e., sets $u_i = 1$.

Aggregate Stage: Each party P_i does the following:

1: **if** P_i is the recipient P_o **then**
2: Party P_i sends to the first neighbor the ciphertext $[b_i \vee u_i, u_i]_{\mathsf{pk}_{i\to 1}^{(1)}}$ and the public key $\mathsf{pk}_{i\to 1}^{(1)}$, and to any other neighbor P_j it sends ciphertext $[1,1]_{\mathsf{pk}_{i\to j}^{(1)}}$ and the public key $\mathsf{pk}_{i\to j}^{(1)}$.
3: **else**
4: Party P_i sends to each neighbor P_j ciphertext $[1,1]_{\mathsf{pk}_{i\to j}^{(1)}}$ and the key $\mathsf{pk}_{i\to j}^{(1)}$.
5:
// Add layer while ORing own input bit
6: **for** any round r from 2 to $\mathtt{T}$ **do**
7: For each neighbor P_j of P_i, do the following (let $k = \pi_i^{(r)}(j)$):
8: **if** P_i did not receive a message from P_j **then**
9: Party P_i sends ciphertext $[1,1]_{\mathsf{pk}_{i\to k}^{(r)}}$ and key $\mathsf{pk}_{i\to k}^{(r)}$ to neighbor P_k.
10: **else** // AddLayer and HomOR are applied component-wise
11: Let $\mathsf{c}_{j\to i}^{(r-1)}$ and $\overline{\mathsf{pk}}_{j\to i}^{(r-1)}$ be the ciphertext and the public key P_i received from P_j. Party P_i computes $\overline{\mathsf{pk}}_{i\to k}^{(r)} = \overline{\mathsf{pk}}_{j\to i}^{(r-1)} \circledast \mathsf{pk}_{i\to k}^{(r)}$ and
$\hat{\mathsf{c}}_{i\to k}^{(r)} \leftarrow \mathsf{AddLayer}\left(\mathsf{c}_{j\to i}^{(r-1)}, \mathsf{sk}_{i\to k}^{(r)}\right)$.
12: P_i computes $[b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{i\to k}^{(r)}}$ and
$\mathsf{c}_{i\to k}^{(r)} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{i\to k}^{(r)}}, \hat{\mathsf{c}}_{i\to k}^{(r)}, \overline{\mathsf{pk}}_{i\to k}^{(r)}\right)$.
13: Party P_i sends ciphertext $\mathsf{c}_{i\to k}^{(r)}$ and public key $\overline{\mathsf{pk}}_{i\to k}^{(r)}$ to neighbor P_k.

Decrypt Stage: Each party P_i does the following:

1: For each neighbor P_j of P_i, if P_i did not receive a message from P_j at round T of the Aggregate Stage, then it sends ciphertext $\mathsf{e}_{i\to j}^{(\mathtt{T})} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}$ to P_j. Otherwise, P_i sends to P_j $\mathsf{e}_{i\to j}^{(\mathtt{T})} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}, \mathsf{c}_{j\to i}^{(\mathtt{T})}, \overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}\right)$.
2: **for** any round r from $\mathtt{T}$ to 2 **do**
3: For each neighbor P_k of P_i:
4: **if** P_i did not receive a message from P_k **then**
5: Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(r-1)}}$ to neighbor P_j, where $k = \pi_i^{(r)}(j)$.
6: **else**
7: Denote by $\mathsf{e}_{k\to i}^{(r)}$ the ciphertext P_i received from P_k, where $k = \pi_i^{(r)}(j)$. Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = \mathsf{DelLayer}\left(\mathsf{e}_{k\to i}^{(r)}, \mathsf{sk}_{i\to k}^{(r)}\right)$ to neighbor P_j.
8: If P_i is the recipient P_o, then it computes $(b,u) = \mathsf{Decrypt}(\mathsf{e}_{1\to i}^{(1)}, \mathsf{sk}_{i\to 1}^{(1)})$ and **outputs** (b,u,u_i). Otherwise, it **outputs** $(1,0,u_i)$.

The actual protocol BC-OB consists of n consecutive runs of the random walk phase protocol RandomWalkPhase.

Protocol BC-OB$(\mathtt{T}, (d_i,b_i)_{P_i\in\mathcal{P}})$

Each party P_i keeps bits b_i^{out}, u_i^{out} and u_i, and sets $u_i = 0$.
for o from 1 to n **do**
Parties jointly execute
$\left((b_i^{tmp}, v_i^{tmp}, u_i^{tmp})_{P_i\in\mathcal{P}}\right) = \mathsf{RandomWalkPhase}(\mathtt{T}, P_o, (d_i,b_i,u_i)_{P_i\in\mathcal{P}})$.
Each party P_i sets $u_i = u_i^{tmp}$.
Party P_o sets $b_o^{out} = b_o^{tmp}$, $u_o^{out} = v_o^{tmp}$.
For each party P_i, **if** $u_i^{out} = 0$ **then** party P_i **outputs** b_i^{out}.

The protocol BC-OB leaks information about the topology of the graph during the execution of RandomWalkPhase, in which the first crash occurs. (Every execution before the first crash proceeds almost exactly as the protocol in [ALM17a] and in every execution afterwards all values are blinded by the unhappy-bit u.) We model the leaked information by a query to the leakage function $\mathcal{L}_{OB}$. The function outputs only one bit

and, since the functionality $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$ allows for only one query to the leakage function, the protocol leaks overall one bit of information.

The inputs passed to $\mathcal{L}_{OB}$ are: the graph G and the set $\mathcal{C}$ of crashed parties, passed to the function by $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$, and a triple $(F, P_s, \mathtt{T}')$, passed by the simulator. The idea is that the simulator needs to know whether the walk carrying the output succeeded or not, and this depends on the graph G. More precisely, the set F contains a list of pairs (P_f, r), where r is the number of rounds in the execution of RandomWalkPhase, at which P_f crashed. $\mathcal{L}_{OB}$ tells the simulator whether any of the crashes in F disconnected a freshly generated random walk of length $\mathtt{T}'$, starting at given party P_s.

Function $\mathcal{L}_{OB}((F, P_s, \mathtt{T}'), \mathcal{C}, G)$

if for any $(P_f, r) \in F$, $P_f \notin \mathcal{C}$ **then** Return 0.
else
Generate in G a random walk of length $\mathtt{T}'$ starting at P_s.
Return 1 if for any $(P_f, r) \in F$ removing party P_f after r rounds disconnects the walk and 0 otherwise.

We prove the following theorem in Section E.1.1.

Theorem 9.3.1. *For κ security parameter and $\mathtt{T} = 8n^3(\log(n) + \kappa)$ protocol BC-OB$(\mathtt{T}, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{OB}} ||\mathsf{BC}$ (with abort) in the $\mathcal{N}$ hybrid-world, where the leakage function $\mathcal{L}_{OB}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

9.3.2 Protocol Leaking a Fraction of a Bit

We now show how to go from BC-OB to the actual broadcast protocol BC-FB$_p$ which leaks only a fraction p of a bit. The leakage parameter p can be arbitrarily small. However, the complexity of the protocol is proportional to $1/p$. As a consequence, $1/p$ must be polynomial and p cannot be negligible.

The idea is to leverage the fact that the adversary can gain information in only one execution of RandomWalkPhase. Imagine that a single execution succeeds only with a small probability p, and otherwise the output bit b is 1. Moreover, assume that during RandomWalkPhase the

adversary does not learn whether it will fail until it can decrypt the output.

We can now, for each phase, repeat RandomWalkPhase ρ times, so that with overwhelming probability one of the repetitions does not fail. A party P_o can then compute its output as the AND of outputs from all repetitions (or abort if any repetition aborted). On the other hand, the adversary can choose only one execution of RandomWalkPhase, in which it learns one bit of information (all subsequent repetitions will abort). Moreover, it must choose it before it knows whether the execution succeeds. Hence, the adversary learns one bit of information only with probability p.

What is left is to modify RandomWalkPhase, so that it succeeds only with probability p, and so that the adversary does not know whether it will succeed. We only change the Aggregate Stage. Instead of an encrypted tuple $[b, u]$, the parties send along the walk $\lfloor 1/p \rfloor + 1$ encrypted bits $[b^1, \ldots, b^{\lfloor 1/p \rfloor}, u]$, where u again is the OR of the unhappy-bits, and every b^k is a copy the bit b in RandomWalkPhase, with some caveats. For each phase o, and for every party $P_i \neq P_o$, all b^k are copies of b in the walk and they all contain 1. For P_o, only one of the bits, b^k, contains the OR, while the rest is initially set to 1.

During the Aggregate Stage, the parties process every ciphertext corresponding to a bit b^k the same way they processed the encryption of b in the RandomWalkPhase. Then, before sending the ciphertexts to the next party on the walk, the encryptions of the bits b^k are randomly shuffled. (This way, as long as the walk traverses an honest party, the adversary does not know which of the ciphertexts contain dummy values.) At the end of the Aggregate Stage (after $\mathtt{T}$ rounds), the last party chooses uniformly at random one of the $\lfloor 1/p \rfloor$ ciphertexts and uses it, together with the encryption of the unhappy-bit, to execute the Decrypt Stage as in RandomWalkPhase. The information leaked by BC-FB$_p$ is modeled by the following function $\mathcal{L}_{FB_p}$.

Function $\mathcal{L}_{FB_p}((F, P_s, \mathtt{T}'), \mathcal{C}, G)$

Let $p' = 1/\lfloor 1/p \rfloor$. With probability p', return $\mathcal{L}_{OB}((F, P_s, \mathtt{T}'), \mathcal{C}, G)$ and with probability $1 - p'$ return $\perp$.

A description of the modified protocol ProbabilisticRandomWalkPhase$_p$

and a proof of the following theorem can be found in Section E.1.2.

Theorem 9.3.2. *Let κ be the security parameter. For $\tau = \log(n) + \kappa$, $T = 8n^3\tau$, and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, protocol $\mathsf{BC\text{-}FB}_p(T, \rho, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes $\mathcal{F}_{INFO}^{\mathcal{L}_{FB_p}} || \mathsf{BC}$ in the $\mathcal{N}$ hybrid-world with abort, where the leakage function $\mathcal{L}_{FB_p}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

9.4 From Broadcast to Topology-Hiding Computation

We showed how to get topology-hiding broadcasts. To get additional functionality (e.g. for compiling MPC protocols), we have to be able to compose these broadcasts. When there is no leakage, this is straightforward: we can run as many broadcasts in parallel or in sequence as we want and they will not affect each other. However, if we consider a broadcast secure in the fail-stop model that leaks at most 1 bit, composing t of these broadcasts could lead to leaking t bits.

The first step towards implementing any functionality in a topology-hiding way is to modify our broadcast protocol to a topology-hiding all-to-all multibit broadcast, without aggregating leakage. Then, we show how to sequentially compose such broadcasts, again without adding leakage. Finally, one can use standard techniques to compile MPC protocols from broadcast. In the following, we give a high level overview of each step. A detailed description of the transformations can be found in Section E.2.

All-to-all Multibit Broadcast. The first observation is that a modification of $\mathsf{BC\text{-}FB}_p$ allows one party to broadcast multiple bits. Instead of sending a single bit b during the random-walk protocol, each party sends a vector $\vec{b}$ of bits encrypted separately under the same key. That is, in each round of the Aggregate Phase, each party sends a vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$.

We can extend this protocol to all-to-all multibit broadcast, where each party P_i broadcasts a message $(b_1, \ldots, b_k)$, as follows. Each of the vectors $\vec{b_i}$ in $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$ contains nk bits, and P_i uses the bits from $n(i-1)$ to ni to communicate its message. That is, in the Aggregate

Stage, *every* P_i homomorphically OR's $\vec{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$ with the received encrypted vectors.

Sequential execution. All-to-all broadcasts can be composed sequentially by preserving the state of unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort.

Topology-Hiding computation. With the above statements, we conclude that any MPC protocol can be compiled into one that leaks only a fraction p of a bit in total. This is achieved using a public key infrastructure, where in the first round the parties use the topology hiding all-to-all broadcast to send each public key to every other party, and then each round of the MPC protocol is simulated with an all-to-all multibit topology-hiding broadcast. As a corollary, any functionality $\mathcal{F}$ can be implemented by a topology-hiding protocol leaking any fraction p of a bit.

9.5 Efficient Topology-Hiding Computation with FHE

One thing to note is that compiling MPC from broadcast is rather expensive, especially in the fail-stop model; we need a broadcast for every round. However, we will show that an FHE scheme with additive overhead can be used to evaluate any polynomial-time function f in a topology-hiding manner. Additive overhead applies to ciphertext versus plaintext sizes and to error with respect to all homomorphic operations if necessary. We will employ an altered random walk protocol, and the total number of rounds in this protocol will amount to that of a single broadcast. We remark that FHE with additive overhead can be obtained from subexponential iO and subexponentially secure OWFs (probabilistic iO), as shown in [CLTV15].

9.5.1 Deeply-Fully-Homomorphic Public-Key Encryption

In the altered random walk protocol, the PKCR scheme is replaced by a deeply-fully-homomorphic PKE scheme (DFH-PKE). Similarly to PKCR,

a DFH-PKE scheme is a public-key encryption scheme enhanced by algorithms for adding and deleting layers. However, we do not require that public keys form a group, and we allow the ciphertexts and public keys on different levels (that is, for which a layer has been added a different number of times) to be distinguishable. Moreover, DFH-PKE offers full homomorphism.

This is captured by three additional algorithms: $\mathsf{AddLayer}_r$, $\mathsf{DelLayer}_r$, and HomOp_r, operating on ciphertexts with r layers of encryption (we will call such ciphertexts level-r ciphertexts). A level-r ciphertext is encrypted under a level-r public key (each level can have different key space).

Adding a layer requires a new secret key sk. The algorithm $\mathsf{AddLayer}_r$ takes as input a vector of level-r ciphertexts $[\![\vec{m}]\!]_{\mathbf{pk}}$ encrypted under a level-r public key, the corresponding level-r public key $\mathbf{pk}$, and a new secret key sk. It outputs a vector of level-$(r+1)$ ciphertexts and the level-$(r+1)$ public key, under which it is encrypted. Deleting a layer is the opposite of adding a layer.

With HomOp_r, one can compute any function on a vector of encrypted messages. It takes a vector of level-r ciphertexts encrypted under a level-r public key, the corresponding level-r public key $\mathbf{pk}$ and a function from a permitted set $\mathcal{F}$ of functions. It outputs a level-r ciphertext that contains the output of the function applied to the encrypted messages.

Intuitively, a DFH-PKE scheme is secure if one can simulate any level-r ciphertext without knowing the history of adding and deleting layers. This is captured by the existence of an algorithm $\mathsf{Leveled\text{-}Encrypt}_r$, which takes as input a plain message and a level-r public key, and outputs a level-r ciphertext. We require that for any level-r encryption of a message $\vec{m}$, the output of $\mathsf{AddLayer}_r$ on that ciphertext is indistinguishable from the output of $\mathsf{Leveled\text{-}Encrypt}_{r+1}$ on $\vec{m}$ and a (possibly different) level-$(r+1)$ public key. An analogous property is required for $\mathsf{DelLayer}_r$. We will also require that the output of HomOp_r is indistinguishable from a level-r encryption of the output of the functions applied to the messages. We refer to Section E.3 for a formal definition of a DFH-PKE scheme and to Section E.3.1 for an instantiation from FHE.

Remark. If we relax DFH-PKE and only require homomorphic evaluation of OR, then this relaxation is implied by any OR-homomorphic PKCR scheme (in PKCR, additionally, all levels of key and ciphertext spaces are the same, and the public key space forms a group). Such

OR-homomorphic DFH-PKE would be sufficient to prove the security of the protocols BC-OB and BC-FB$_p$. However, for simplicity and clarity, we decided to describe our protocols BC-OB and BC-FB$_p$ from a OR-homomorphic PKCR scheme.

9.5.2 Topology-Hiding Computation from DFH-PKE

To evaluate any function f, we modify the topology-hiding broadcast protocol (with PKCR replaced by DFH-PKE) in the following way. During the Aggregate Stage, instead of one bit for the OR of all inputs, the parties send a vector of encrypted inputs. At each round, each party homomorphically adds its input together with its id to the vector. The last party on the walk homomorphically evaluates f on the encrypted inputs, and (homomorphically) selects the output of the party who receives it in the current phase. The Decrypt Stage is started with this encrypted result.

Note that we still need a way to make a random walk dummy (this was achieved in BC-OB and BC-FB$_p$ by starting it with a 1). Here, we will have an additional input bit for the party who starts a walk. In case this bit is set, when homomorphically evaluating f, we (homomorphically) replace the output of f by a special symbol. We refer to Section E.4 for a detailed description of the protocol and a proof of the following theorem.

Theorem 9.5.1. *For security parameter κ, $\tau = \log(n) + \kappa$, $\mathcal{T} = 8n^3\tau$, and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, the protocol DFH-THC topology-hidingly evaluates any poly-time function f, $\mathcal{F}_{INFO}^{\mathcal{L}_{FB_p}} || f$ in the $\mathcal{N}$ hybrid-world.*

9.6 Security Against Semi-malicious Adversaries

In this section, we show how to generically compile our protocols to provide in addition security against a semi-malicious adversary. The transformed protocol proceeds in two phases: Randomness Generation and Deterministic Execution. In the first phase, we generate the random tapes for all parties and in the second phase we execute the given protocol with parties using the pre-generated random tapes. The tapes are

generated in such a way that the tape of each party P_i is the sum of random values generated from each party. Hence, as long as one party is honest, the generated tape is random.

Randomness Generation. The goal of the first phase is to generate for each party P_i a uniform random value r_i, which can then be used as randomness tape of P_i in the phase of Deterministic Execution.[2]

Protocol GenerateRandomness

1: Each party P_i generates $n+1$ uniform random values $s_i^{(0)}, s_i^{(1)}, \ldots, s_i^{(n)}$ and sets $r_i^{(0)} := s_i^{(0)}$.
2: **for** any round r from 1 to n **do**
3: Each party P_i sends $r_i^{(r-1)}$ to all its neighbors.
4: Each party P_i computes $r_i^{(r)}$ as the sum of all values received from its (non-crashed) neighbors in the current round and the value $s_i^{(k)}$.
5: Each party P_i outputs $r_i := r_i^{(n)}$.

Lemma 9.6.1. *Let G' be the network graph without the parties which crashed during the execution of* GenerateRandomness*. Any party P_i whose connected component in G' contains at least one honest party will output a uniform value r_i. The output of any honest party is not known to the adversary. The protocol* GenerateRandomness *does not leak any information about the network-graph (even if crashes occur).*

Proof. First observe that all randomness is chosen at the beginning of the first round. The rest of the protocol is completely deterministic. This implies that the adversary has to choose the randomness of corrupted parties independently of the randomness chosen by honest parties.

If party P_i at the end of the protocol execution is in a connected component with honest party P_j, the output r_i is a sum which contains at least one of the values $s_j^{(r)}$ from P_j. That summand is independent of the rest of the summands and uniform random. Thus, r_i is uniform random as well.

Any honest party will (in the last round) compute its output as a

[2]To improve overall communication complexity of the protocol the values generated in the first phase could be used as local seeds for a PRG which is then used to generate the actual random tapes.

sum which contains a locally generated truly random value, which is not known to the adversary. Thus, the output is also not known to the adversary.

Finally, observe that the message pattern seen by a party is determined by its neighborhood. Moreover, the messages received by corrupted parties from honest parties are uniform random values. This implies, that the view of the adversary in this protocol can be easily simulated given the neighborhood of corrupted parties. Thus, the protocol does not leak any information about the network topology. □

Transformation to Semi-malicious Security. In the second phase of Deterministic Execution, the parties execute the protocol secure against passive and fail-stop corruptions, but instead of generating fresh randomness during the protocol execution, they use the random tape generated in the first phase.

Protocol EnhanceProtocol(Π)

1: The parties execute GenerateRandomness to generate random tapes.
2: If a party witnessed a crash in GenerateRandomness, it pretends that it witnessed this crash in the first round of the protocol Π.
3: The parties execute Π, using the generated randomness tapes, instead of generating randomness on the fly.

Theorem 9.6.2. *Let $\mathcal{F}$ be an MPC functionality and let* Π *be a protocol that topology-hidingly realizes $\mathcal{F}$ in the presence of static passive corruptions and adaptive crashes. Then, the protocol* EnhanceProtocol(Π) *topology-hidingly realizes $\mathcal{F}$ in the presence of static semi-malicious corruption and adaptive crashes. The leakage stays the same.*

Proof. (sketch) The randomness generation protocol GenerateRandomness used in the first phase is secure against a semi-malicious fail-stopping adversary. Lemma 9.6.1 implies that the random tape of any semi-malicious party that can interact with honest parties is truly uniform random. Moreover, the adversary has no information on the random tapes of honest parties. This implies that the capability of the adversary in the execution of the actual protocol in the second phase (which for fixed random tapes is deterministic) is the same as for an semi-honest fail-stopping ad-

versary. This implies that the leakage of $\mathsf{EnhanceProtocol}(\Pi)$ is the same as for Π as the randomness generation protocol does not leak information (even if crashes occur).

□

As a corollary of Theorems 9.3.2 and 9.6.2, we obtain that any MPC functionality can be realized in a topology-hiding manner secure against an adversary that does any number of static semi-malicious corruptions and adaptive crashes, leaking at most an arbitrary small fraction of information about the topology.

Appendix E

Details of Chapter 9

E.1 Topology-Hiding Broadcast

This section contains supplementary material for Section 9.3.

E.1.1 Protocol Leaking One Bit

In this section we prove Theorem 9.3.1 from Section 9.3.1.

Theorem 9.3.1. *For κ security parameter and $\mathtt{T} = 8n^3(\log(n) + \kappa)$ protocol $\mathsf{BC\text{-}OB}(\mathtt{T}, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes $\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{OB}} || \mathsf{BC}$ (with abort) in the $\mathcal{N}$ hybrid-world, where the leakage function $\mathcal{L}_{OB}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

Proof. **Completeness.** We first show that the protocol is complete. To this end, we need to ensure that the probability that all parties get the correct output is overwhelming in κ. That is, the probability that all non-dummy random walks (of length $\mathtt{T} = 8n^3(\log(n) + \kappa)$) reach all nodes is overwhelming.

By Lemma 9.2.2, a walk of length $8n^3\tau$ does not reach all nodes with probability at most $\frac{1}{2^\tau}$. Then, using the union bound, we obtain that the probability that there is a party whose walk does not reach all nodes is at most $\frac{n}{2^\tau}$. Hence, all n walks (one for each party) reach all nodes with

probability at least $1 - \frac{n}{2^\tau}$. If we want this value to be overwhelming, e.g. $1 - \frac{1}{2^\kappa}$, we can set $\tau := \kappa + \log(n)$.

Soundness. We now need to show that no environment can distinguish between the real world and the simulated world, when given access to the adversarially-corrupted parties. We first describe on a high level the simulator $\mathcal{S}_{OB}$ and argue that it simulates the real execution.

In essence, the task of $\mathcal{S}_{OB}$ is to simulate the messages sent by honest parties to passively corrupted parties. Consider a corrupted party P_c and its honest neighbor P_h. The messages sent from P_h to P_c during the Aggregate Stage are ciphertexts, to which P_h added a layer, and corresponding public keys. Since P_h is honest, the adversary does not know the secret keys corresponding to the sent public keys. Hence, $\mathcal{S}_{OB}$ can simply replace them with encryptions of a pair $(1, 1)$ under a freshly generated public key. The group structure of keys in PKCR guarantees that a fresh key has the same distribution as the composed key (after executing $\mathsf{AddLayer}$). Semantic security implies that the encrypted message can be replaced by $(1, 1)$.

Consider now the Decrypt Stage at round r. Let $\mathsf{pk}_{c\to h}^{(r)}$ be the public key sent by P_c to P_h in the Aggregate Stage (note that this is not the key discussed above; there we argued about keys sent in the opposite direction). $\mathcal{S}_{OB}$ will send to P_c a fresh encryption under $\mathsf{pk}_{c\to h}^{(r)}$. We now specify what it encrypts.

Note that the only interesting case is when the party P_o receiving output is corrupted and when we are in the round r in which the (only one) random walk carrying the output enters an area of corrupted parties, containing P_o (that is, when the walk with output contains from P_h all the way to P_o only corrupted parties). In this one message in round r the adversary learns the output of P_o. All other messages are simply encryptions of $(1, 1)$.

For this one meaningful message, we consider three cases. If any party crashed in a phase preceding the current one, $\mathcal{S}_{OB}$ sends an encryption of $(1, 1)$ (as in the real world the walk is made dummy by an unhappy party). If no crashes occurred up to this point (round r in given phase), $\mathcal{S}_{OB}$ encrypts the output received from BC. If a crash happened in the given phase, $\mathcal{S}_{OB}$ queries the leakage oracle $\mathcal{L}_{OB}$, which essentially executes the protocol and tells whether the output or $(1, 1)$ should be sent.

Simulator. Below, we present the pseudocode of the simulator. The es-

sential part of it is the algorithm PhaseSimulation, which is also illustrated in Figure E.1.

Simulator $\mathcal{S}_{OB}$

1. $\mathcal{S}_{OB}$ corrupts passively $\mathcal{Z}^p$.
2. $\mathcal{S}_{OB}$ sends inputs for all parties in $\mathcal{Z}^p$ to BC and receives the output bit b^{out}.
3. For each $P_i \in \mathcal{Z}^p$, $\mathcal{S}_{OB}$ receives $\mathbf{N}_G(P_i)$ from $\mathcal{F}_{\text{INFO}}^{\mathcal{L}}$.
4. Throughout the simulation, if $\mathcal{A}$ crashes a party P_f, so does $\mathcal{S}_{OB}$.
5. Now $\mathcal{S}_{OB}$ has to simulate the view of all parties in $\mathcal{Z}^p$.

   In every phase in which P_o should get the output, first of all the Initialization Stage is executed among the parties in $\mathcal{Z}^p$ and the T key pairs are generated for every $P_i \in \mathcal{Z}^p$. Moreover, for every $P_i \in \mathcal{Z}^p$ the permutations $\pi_i^{(r)}$ are generated, defining those parts of all random walks, which pass through parties in $\mathcal{Z}^p$.

   The messages sent by parties in $\mathcal{Z}^p$ are generated by executing the protocol RandomWalkPhase. The messages sent by correct parties $P_i \notin \mathcal{Z}^p$ are generated by executing PhaseSimulation(P_o, P_i), described below.
6. $\mathcal{S}_{OB}$ sends to BC the abort vector (in particular, the vector contains all parties P_o who should receive their outputs in phases following the first crash and, depending on the output of $\mathcal{L}_{OB}$, the party who should receive its output in the phase with first crash).

Algorithm PhaseSimulation(P_o, P_i)

If $P_o \in \mathcal{Z}^p$, let w denote the random walk generated in the Initialization Stage (at the beginning of the simulation of this phase), which starts at P_o and carries the output bit. Let ℓ denote the number of parties in $\mathcal{Z}^p$ on w before the first correct party. If $P_o \notin \mathcal{Z}^p$, w and ℓ are not defined.
For every $P_j \in \mathcal{Z}^p \cap \mathbf{N}_G(P_i)$, let $\mathsf{pk}_{j\to i}^{(r)}$ denote the public key generated in the Initialization Stage by P_j for P_i and for round r.

Initialization Stage

1: For every neighbor $P_j \in \mathcal{Z}^p$ of the correct P_i, $\mathcal{S}_{OB}$ generates $\mathtt{T}$ key pairs $(\mathsf{pk}^{(1)}_{i\to j}, \mathsf{sk}^{(1)}_{i\to j}), \ldots, (\mathsf{pk}^{(\mathtt{T})}_{i\to j}, \mathsf{sk}^{(\mathtt{T})}_{i\to j})$.

Aggregate Stage

1: In round r, for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{OB}$ sends $([1,1]_{\mathsf{pk}^{(r)}_{i\to j}}, \mathsf{pk}^{(r)}_{i\to j})$ to P_j.

Decrypt Stage

1: **if** $\mathcal{A}$ crashed any party in any phase before the current one or $P_o \notin \mathcal{Z}^p$ **then**
2: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{OB}$ sends $[1,1]_{\mathsf{pk}^{(r)}_{j\to i}}$ to P_j.
3: **else**
4: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{OB}$ sends $[1,1]_{\mathsf{pk}^{(r)}_{j\to i}}$ to P_j unless the following three conditions hold: (a) P_i is the first party not in $\mathcal{Z}^p$ on w, (b) P_j is the last party in $\mathcal{Z}^p$ on w, and (c) $r = 2\mathtt{T} - \ell$.
5: If the three conditions hold (in particular $r = 2\mathtt{T} - \ell$), $\mathcal{S}_{OB}$ does the following. If $\mathcal{A}$ did not crash any party in a previous round, $\mathcal{S}_{OB}$ sends $[b^{out}, 0]_{\mathsf{pk}^{(r)}_{j\to i}}$ to party P_j.
6: Otherwise, let F denote the set of pairs $(P_f, s - \ell + 1)$ such that $\mathcal{A}$ crashed P_f in round s. $\mathcal{S}_{OB}$ queries $\mathcal{F}^{\mathcal{L}_{OB}}_{\text{INFO}}$ for the leakage on input $(F, P_i, \mathtt{T} - \ell)$. If the returned value is 1, it sends $[1,1]_{\mathsf{pk}^{(r)}_{j\to i}}$ to P_j. Otherwise it sends $[b^{out}, 0]_{\mathsf{pk}^{(r)}_{j\to i}}$ to party P_j.

We prove that no environment can tell whether it is interacting with $\mathcal{N}$ and the adversary in the real world or with $\mathcal{F}^{\mathcal{L}}_{\text{INFO}}$ and the simulator in the ideal world.

Hybrids and security proof.

Hybrid 1. $\mathcal{S}_1$ simulates the real world exactly. This means, $\mathcal{S}$ has information on the entire topology of the graph, each party's input, and can simulate identically the real world.

Hybrid 2. $\mathcal{S}_2$ replaces the real keys with the simulated public keys, but still knows everything about the graph as in the first hybrid.

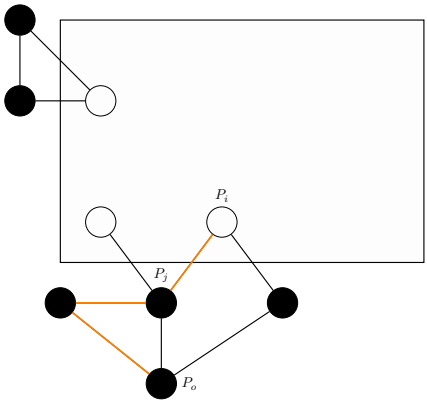

Figure E.1: An example of the algorithm executed by the simulator $\mathcal{S}_{OB}$. The filled circles are the corrupted parties. The red line represents the random walk generated by $\mathcal{S}_{OB}$ in Step 5, in this case of length $\ell = 3$. $\mathcal{S}_{OB}$ simulates the Decrypt Stage by sending fresh encryptions of $(1, 1)$ at every round from every honest party to each of its corrupted neighbors, except in round $2\mathtt{T} - 3$ from P_i to P_j. If no crash occurred up to that point, $\mathcal{S}_{OB}$ sends encryption of $(b^{out}, 0)$. Otherwise, it queries the leakage oracle about the walk of length $\mathtt{T} - 3$, starting at P_i.

More formally, in each random walk phase and for each party $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ where $\mathbf{N}_G(P_i) \cap \mathcal{Z}^p \neq \varnothing$, $\mathcal{S}_2$ generates T key pairs $(\mathsf{pk}^{(1)}_{i\to j}, \mathsf{sk}^{(1)}_{i\to j}), \ldots, (\mathsf{pk}^{(\mathtt{T})}_{i\to j}, \mathsf{sk}^{(\mathtt{T})}_{i\to j})$ for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$. In each round r of the corresponding Aggregate Stage and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_2$ does the following. P_i receives ciphertext $[b, u]_{\mathsf{pk}^{(r)}_{*\to i}}$ and the public key $\mathsf{pk}^{(r)}_{*\to i}$ destined for P_j. Instead of adding a layer and homomorphically OR'ing the bit b_i, $\mathcal{S}_2$ computes $(b', u') = (b \vee b_i \vee u_i, u \vee u_i)$, and sends $[b', u']_{\mathsf{pk}^{(r)}_{i\to j}}$ to P_j. In other words, it sends the same message as $\mathcal{S}_1$ but encrypted with a fresh public key. In the corresponding Decrypt Stage, P_i will get back a ciphertext from P_j encrypted under this exact fresh public key.

Hybrid 3. $\mathcal{S}_3$ now simulates the ideal functionality during the Aggre-

gate Stage. It does so by sending encryptions of $(1,1)$ instead of the actual messages and unhappy bits. More formally, in each round r of the Aggregate Stage and for all parties $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ and $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_3$ sends $[1,1]_{\mathsf{pk}_{i\to j}^{(r)}}$ instead of the ciphertext $[b,u]_{\mathsf{pk}_{i\to j}^{(r)}}$ sent by $\mathcal{S}_2$.

Hybrid 4. $\mathcal{S}_4$ does the same as $\mathcal{S}_{OB}$ during the Decrypt Stage for all steps except for round $2\mathtt{T}-\ell$ of the first random walk phase in which the adversary crashes a party. This corresponds to the original description of the simulator except for the 'Otherwise' condition of Step 6 in the Decrypt Stage.

Hybrid 5. $\mathcal{S}_5$ is the actual simulator $\mathcal{S}_{OB}$.

In order to prove that no environment can distinguish between the real world and the ideal world, we prove that no environment can distinguish between any two consecutive hybrids when given access to the adversarially-corrupted nodes.

Claim 1. *No efficient distinguisher D can distinguish between Hybrid 1 and Hybrid 2.*

Proof of claim. The two hybrids only differ in the computation of the public keys that are used to encrypt messages in the Aggregate Stage from any honest party $P_i \in \mathcal{P}\setminus\mathcal{Z}^p$ to any dishonest neighbor $P_j \in \mathbf{N}_G(P_i)\cap\mathcal{Z}^p$.

In Hybrid 1, party P_i sends to P_j an encryption under a fresh public key in the first round. In the following rounds, the encryption is sent either under a product key $\overline{\mathsf{pk}}_{i\to j}^{(r)} = \overline{\mathsf{pk}}_{k\to i}^{(r-1)} \circledast \mathsf{pk}_{i\to j}^{(r)}$ or under a fresh public key (if P_i is unhappy). Note that $\overline{\mathsf{pk}}_{k\to i}^{(r-1)}$ is the key P_i received from a neighbor P_k in the previous round.

In Hybrid 2, party P_i sends to P_j an encryption under a fresh public key $\mathsf{pk}_{i\to j}^{(r)}$ in every round.

The distribution of the product key used in Hybrid 1 is the same as the distribution of a freshly generated public-key. This is due to the (fresh) $\mathsf{pk}_{i\to j}^{(r)}$ key which randomizes the product key. Therefore, no distinguisher can distinguish between Hybrid 1 and Hybrid 2. ◊

Claim 2. *No efficient distinguisher D can distinguish between Hybrid 2 and Hybrid 3.*

Proof of claim. The two hybrids differ only in the content of the encrypted messages that are sent in the Aggregate Stage from any honest party $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ to any dishonest neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$.

In Hybrid 2, party P_i sends to P_j in the first round an encryption of $(b_i \vee u_i, u_i)$. In the following rounds, P_i sends to P_j either an encryption of $(b \vee b_i \vee u_i, u \vee u_i)$, if message (b, u) is received from neighbor $\pi_i^{-1}(j)$, or an encryption of $(1, 1)$ if no message is received.

In Hybrid 3, all encryptions that are sent from party P_i to party P_j are replaced by encryptions of $(1, 1)$.

Since the simulator chooses a key independent of any key chosen by parties in $\mathcal{Z}^p$ in each round, the key is unknown to the adversary. Hence, the semantic security of the encryption scheme guarantees that the distinguisher cannot distinguish between both encryptions. ◊

Claim 3. *No efficient distinguisher D can distinguish between Hybrid 3 and Hybrid 4.*

Proof of claim. The only difference between the two hybrids is in the Decrypt Stage. We differentiate two cases:

- A phase where the adversary did not crash any party in this or any previous phase. In this case, the simulator $\mathcal{S}_3$ sends an encryption of (b_W, u_W), where $b_W = \bigvee_{P_j \in W} b_j$ is the OR of all input bits in the walk and $u_W = 0$, since no crash occurred. $\mathcal{S}_4$ sends an encryption of $(b^{out}, 0)$, where $b^{out} = \bigvee_{P_i \in \mathcal{P}} b_i$. Since the graph is connected, $b^{out} = b_W$ with overwhelming probability, as proven in Corollary 9.2.2. Also, the encryption in Hybrid 4 is done with a fresh public key which is indistinguishable with the encryption done in Hybrid 3 by OR'ing many times in the graph, as shown in Claim 2.1 in [ALM17a].

- A phase where the adversary crashed a party in a previous phase or any round different than $2\mathtt{T} - \ell$ of the first phase where the adversary crashes a party. In Hybrid 4 the parties send an encryption of $(1, 1)$. This is also the case in Hybrid 3, because even if a crashed party disconnected the graph, each connected component contains a neighbor of a crashed party. Moreover, in Hybrid 4, the messages are encrypted with a fresh public key, and in Hybrid 3, the encryptions are obtained by the homomorphic OR operation.

Both encryptions are indistinguishable, as shown in in Claim 2.1 in [ALM17a].

$\Diamond$

Claim 4. *No efficient distinguisher D can distinguish between Hybrid 4 and Hybrid 5.*

Proof of claim. The only difference between the two hybrids is in the Decrypt Stage, at round $2\mathtt{T} - \ell$ of the first phase where the adversary crashes.

Let F be the set of pairs (P_f, r) such that $\mathcal{A}$ crashed P_f at round r of the phase. In Hybrid 4, a walk W of length $\mathtt{T}$ is generated from party P_o. Let W_1 be the region of W from P_o to the first not passively corrupted party and let W_2 be the rest of the walk. Then, the adversary's view at this step is the encryption of $(1, 1)$ if one of the crashed parties breaks W_2, and otherwise an encryption of $(b_W, 0)$. In both cases, the message is encrypted under a public key for which the adversary knows the secret key.

In Hybrid 5, a walk W_1' is generated from P_o of length $\ell \leq \mathtt{T}$ ending at the first not passively corrupted party P_i. Then, the simulator queries the leakage function on input $(F, P_i, \mathtt{T} - \ell)$, which generates a walk W_2' of length $\mathtt{T} - \ell$ from P_i, and checks whether W_2' is broken by any party in F. If W_2' is broken, P_i sends an encryption of $(1, 1)$, and otherwise an encryption of $(b_W, 0)$. Since the walk W' defined as W_1' followed by W_2' follows the same distribution as W, $b_W = b^{out}$ with overwhelming probability, and the encryption with a fresh public key which is indistinguishable with the encryption done by OR'ing many times in the graph, then it is impossible to distinguish between Hybrid 4 and Hybrid 5.

$\Diamond$

This concludes the proof of soundness. $\square$

E.1.2 Protocol Leaking a Fraction of a Bit

We give a description of $\mathsf{ProbabilisticRandomWalkPhase}_p$ which is the random-walk phase protocol for $\mathsf{BC\text{-}FB}_p$, from Section 9.3.2. Note that this protocol should be repeated ρ times in the actual protocol. The boxes indicate the parts that differ from RandomWalkPhase (cf. Section 9.3.1).

Protocol $\mathsf{ProbabilisticRandomWalkPhase}_p(\mathtt{T}, P_o, (d_i, b_i, u_i)_{P_i \in \mathcal{P}})$

Initialization Stage:

1: Each party P_i generates $\mathtt{T} \cdot d_i$ keypairs $(\mathsf{pk}_{i\to j}^{(r)}, \mathsf{sk}_{i\to j}^{(r)}) \leftarrow \mathsf{Keygen}(1^\kappa)$ where $r \in \{1, \ldots, \mathtt{T}\}$ and $j \in \{1, \ldots, d_i\}$.

2: Each party P_i generates $\mathtt{T} - 1$ random permutations on d_i elements $\left\{\pi_i^{(2)}, \ldots, \pi_i^{(\mathtt{T})}\right\}$

3: For each party P_i, if any of P_i's neighbors crashed in any phase before the current one, then P_i becomes unhappy, i.e., sets $u_i = 1$.

Aggregate Stage: Each party P_i does the following:

1: **if** P_i is the recipient P_o **then**

2: Party P_i sends to the first neighbor the public key $\mathsf{pk}_{i\to 1}^{(1)}$ and the ciphertext $\boxed{[b_i \vee u_i, 1, \ldots, 1, u_i]_{\mathsf{pk}_{i\to 1}^{(1)}}}$ ($\lfloor 1/p \rfloor - 1$ ciphertexts contain 1), and to any other neighbor P_j it sends $\boxed{[1, \ldots, 1, 1]_{\mathsf{pk}_{i\to j}^{(1)}}}$ and the public key $\mathsf{pk}_{i\to j}^{(1)}$.

3: **else**

4: Party P_i sends to each neighbor P_j ciphertext $\boxed{[1, \ldots, 1, 1]_{\mathsf{pk}_{i\to j}^{(1)}}}$ and the public key $\mathsf{pk}_{i\to j}^{(1)}$.

5: **for** any round r from 2 to $\mathtt{T}$ **do**

6: For each neighbor P_j of P_i, do the following (let $k = \pi_i^{(r)}(j)$):

7: **if** P_i did not receive a message from P_j **then**

8: Party P_i sends $\boxed{[1, \ldots, 1, 1]_{\mathsf{pk}_{i\to k}^{(r)}}}$ and $\mathsf{pk}_{i\to k}^{(r)}$ to neighbor P_k.

9: **else**

10: Let $\mathsf{c}_{j\to i}^{(r-1)}$ and $\overline{\mathsf{pk}}_{j\to i}^{(r-1)}$ be the ciphertext and the public key P_i received from P_j. Party P_i computes $\overline{\mathsf{pk}}_{i\to k}^{(r)} = \overline{\mathsf{pk}}_{j\to i}^{(r-1)} \circledast \mathsf{pk}_{i\to k}^{(r)}$ and $\hat{\mathsf{c}}_{i\to k}^{(r)} \leftarrow \mathsf{AddLayer}\left(\mathsf{c}_{j\to i}^{(r-1)}, \mathsf{pk}_{i\to k}^{(r)}\right)$.

11: Party P_i computes $\boxed{[b_i \vee u_i, \ldots, b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{i\to k}^{(r)}}}$ and $\boxed{\mathsf{c}_{i\to k}^{(r)} = \mathsf{HomOR}\left([b_i \vee u_i, \ldots, b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{i\to k}^{(r)}}, \hat{\mathsf{c}}_{i\to k}^{(r)}\right)}$.

12: Party P_i sends ciphertext $\mathsf{c}_{i\to k}^{(r)}$ and public key $\overline{\mathsf{pk}}_{i\to k}^{(r)}$ to neighbor P_k.

Decrypt Stage: Each party P_i does the following:

1: For each neighbor P_j of P_i:
2: **if** P_i did not get a message from P_j at round T of the Aggregate Stage **then**
3: Party P_i sends ciphertext $\mathsf{e}_{i\to j}^{(\mathtt{T})} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}$ to P_j.
4: **else**
5: Party P_i chooses uniformly at random one of the first $\lfloor 1/p \rfloor$ ciphertexts in $\mathsf{c}_{j\to i}^{(\mathtt{T})}$. Let $\bar{\mathsf{c}}_{j\to i}^{(\mathtt{T})}$ denote the tuple containing the chosen ciphertext and the last element of $\mathsf{c}_{j\to i}^{(\mathtt{T})}$ (the encryption of the unhappy bit). Party P_i computes and sends $\mathsf{e}_{i\to j}^{(\mathtt{T})} = \mathsf{HomOR}\left([b_i \vee u_i, u_i]_{\overline{\mathsf{pk}}_{j\to i}^{(\mathtt{T})}}, \bar{\mathsf{c}}_{j\to i}^{(\mathtt{T})}\right)$ to P_j.
6: **for** any round r from $\mathtt{T}$ to 2 **do**
7: For each neighbor P_k of P_i:
8: **if** P_i did not receive a message from P_k **then**
9: Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = [1,1]_{\overline{\mathsf{pk}}_{j\to i}^{(r-1)}}$ to neighbor P_j, where $k = \pi_i^{(r)}(j)$.
10: **else**
11: Denote by $\mathsf{e}_{k\to i}^{(r)}$ the ciphertext P_i received from P_k, where $k = \pi_i^{(r)}(j)$. Party P_i sends $\mathsf{e}_{i\to j}^{(r-1)} = \mathsf{DelLayer}\left(\mathsf{e}_{k\to i}^{(r)}, \mathsf{sk}_{i\to k}^{(r)}\right)$ to neighbor P_j.
12: If P_i is the recipient P_o, then it computes $(b,u) = \mathsf{Decrypt}(\mathsf{e}_{1\to i}^{(1)}, \mathsf{sk}_{i\to 1}^{(1)})$ and **outputs** (b,u,u_i). Otherwise, it **outputs** $(1,0,u_i)$.

Security Proof of the Protocol Leaking a Fraction of a Bit.

In this section we prove Theorem 9.3.2 from Section 9.3.2.

Theorem 9.3.2. *Let κ be the security parameter. For $\tau = \log(n) + \kappa$, $T = 8n^3\tau$ and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, the protocol $\mathsf{BC\text{-}FB}_p(T, \rho, (d_i, b_i)_{P_i \in \mathcal{P}})$ topology-hidingly realizes the functionalities $\mathcal{F}_{INFO}^{\mathcal{L}_{FB_p}} || \mathsf{BC}$ (with abort) in the $\mathcal{N}$ hybrid-world, where the leakage function $\mathcal{L}_{FB_p}$ is the one defined as above. If no crashes occur, then there is no abort and there is no leakage.*

Proof. **Completeness.** We first show that the protocol is complete. That is, that if the adversary does not crash any party, then every party

gets the correct output (the OR of all input bits) with overwhelming probability. More specifically, we show that if no crashes occur, then after ρ repetitions of a phase, the party P_o outputs the correct value with probability at least $1 - 2^{-(\kappa+\log(n))}$. The overall completeness follows from the union bound: the probability that all n parties output the correct value is at least $1 - 2^{-\kappa}$.

Notice that if the output of any of the ρ repetitions intended for P_o is correct, then the overall output of P_o is correct. A given repetition can only give an incorrect output when either the random walk does not reach all parties, which happens with probability at most $2^{-\tau}$, or when the repetition fails, which happens with probability $1 - p'$. Hence, the probability that a repetition gives the incorrect result is at most $1 - p' + 2^{-\tau}$. The probability that all repetitions are incorrect is then at most $(1-p'+2^{-\tau})^{\rho} \leq 2^{-(\kappa+\log(n))}$ (the inequality holds for $0 \leq p'-2^{-\tau} \leq 1$).

Soundness. We show that no environment can distinguish between the real world and the simulated world, when given access to the adversarially-corrupted nodes. The simulator $\mathcal{S}_{FB}$ for $\mathsf{BC\text{-}FB}_p$ is a modification of $\mathcal{S}_{OB}$. Here we only sketch the changes and argue why $\mathcal{S}_{FB}$ simulates the real world.

In each of the ρ repetitions of a phase, $\mathcal{S}_{FB}$ executes a protocol very similar to the one for $\mathcal{S}_{OB}$. In the Aggregate Stage, $\mathcal{S}_{FB}$ proceeds almost identically to $\mathcal{S}_{OB}$ (except that it sends encryptions of vectors $(1, \ldots, 1)$ instead of only two values). In the Decrypt Stage the only difference between $\mathcal{S}_{FB}$ and $\mathcal{S}_{OB}$ is in computing the output for the party P_o (as already discussed in the proof of Theorem 9.3.1, $\mathcal{S}_{FB}$ does this only when P_o is corrupted and the walk carrying the output enters an area of corrupted parties). In the case when there were no crashes before or during given repetition of a phase, $\mathcal{S}_{OB}$ would simply send the encrypted output. On the other hand, $\mathcal{S}_{FB}$ samples a value from the Bernoulli distribution with parameter p and sends the encrypted output only with probability p, while with probability $1 - p$ it sends the encryption of $(1, 0)$. Otherwise, the simulation is the same as for $\mathcal{S}_{OB}$.

It can be easily seen that $\mathcal{S}_{FB}$ simulates the real world in the Aggregate Stage and in the Decrypt Stage in every message other than the one encrypting the output. But even this message comes from the same distribution as the corresponding message sent in the real world. This

is because in the real world, if the walk was not broken by a crash, this message contains the output with probability p. The simulator encrypts the output also with probability p in the two possible cases: when there was no crash ($\mathcal{S}_{FB}$ samples from the Bernoulli distribution) and when there was a crash but the walk was not broken ($\mathcal{L}_{FB}$ is defined in this way).

Simulator. The simulator $\mathcal{S}_{FB}$ proceeds almost identically to the simulator $\mathcal{S}_{OB}$ given in the proof of Theorem 9.3.1 (cf. Section E.1.1). We only change the algorithm PhaseSimulation to ProbabilisticPhaseSimulation and execute it ρ times instead of only once.

Algorithm ProbabilisticPhaseSimulation(P_o, P_i)

If $P_o \in \mathcal{Z}^p$, let w denote the random walk generated in the Initialization Stage (at the beginning of the simulation of this phase), which starts at P_o and carries the output bit. Let ℓ denote the number of parties in $\mathcal{Z}^p$ on w before the first correct party. If $P_o \notin \mathcal{Z}^p$, w and ℓ are not defined.
For every $P_j \in \mathcal{Z}^p \cap \mathbf{N}_G(P_i)$, let $\mathsf{pk}_{j\to i}^{(r)}$ denote the public key generated in the Initialization Stage by P_j for P_i and for round r.

Initialization Stage

1: For every neighbor $P_j \in \mathcal{Z}^p$ of the correct P_i, $\mathcal{S}_{FB}$ generates $\mathtt{T}$ key pairs $(\mathsf{pk}_{i\to j}^{(1)},\mathsf{sk}_{i\to j}^{(1)}),\ldots,(\mathsf{pk}_{i\to j}^{(\mathtt{T})},\mathsf{sk}_{i\to j}^{(\mathtt{T})})$.

Aggregate Stage

1: In round r, for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends the tuple $([1,\ldots,1]_{\mathsf{pk}_{i\to j}^{(r)}}, \mathsf{pk}_{i\to j}^{(r)})$ (with $\lfloor 1/p \rfloor + 1$ ones) to P_j.

Decrypt Stage

1: **if** $P_o \notin \mathcal{Z}^p$ or $\mathcal{A}$ crashed any party in any phase before the current one or in any repetition of the current phase **then**
2: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends $[1,1]_{\mathsf{pk}_{j\to i}^{(r)}}$ to P_j.
3: **else**

4: In every round r and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_{FB}$ sends $\boxed{[1,1]_{\mathsf{pk}_{j\to i}^{(r)}}}$ to P_j unless the following three conditions hold: (a) P_i is the first party not in $\mathcal{Z}^p$ on w, (b) P_j is the last party in $\mathcal{Z}^p$ on w, and (c) $r = 2\mathtt{T} - \ell$.

5: If the three conditions hold (in particular $r = 2\mathtt{T} - \ell$), $\mathcal{S}_{FB}$ does the following. If $\mathcal{A}$ did not crash any party in a previous round,

6: $\mathcal{S}_{FB}$ samples a value x from the Bernoulli distribution with parameter p'. If $x = 1$ (with probability p'), $\mathcal{S}_{FB}$ sends to P_j the ciphertext $[b_{out}, 0]_{pk_{j\to i}^{(r)}}$ and otherwise it sends $[1,0]_{pk_{j\to i}^{(r)}}$.

7: Otherwise, let F denote the set of pairs $(P_f, s-\ell+1)$ such that $\mathcal{A}$ crashed P_f in round s. $\mathcal{S}_{FB}$ queries $\boxed{\mathcal{F}_{\text{INFO}}^{\mathcal{L}_{FB_p}}}$ for the leakage on input $(F, P_i, \mathtt{T} - \ell)$. If the returned value is 1, it sends $[1,1]_{\mathsf{pk}_{j\to i}^{(r)}}$ to P_j. Otherwise it sends $[b^{out}, 0]_{\mathsf{pk}_{j\to i}^{(r)}}$ to party P_j.

Hybrids and security proof. We consider similar steps as the hybrids from Paragraph E.1.1.

Hybrid 1. $\mathcal{S}_1$ simulates the real world exactly. This means, $\mathcal{S}_1$ has information on the entire topology of the graph, each party's input, and can simulate identically the real world.

Hybrid 2. $\mathcal{S}_2$ replaces the real keys with the simulated public keys, but still knows everything about the graph as in the first hybrid.

More formally, in each subphase of each random walk phase and for each party $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ where $\mathbf{N}_G(P_i) \cap \mathcal{Z}^p \neq \varnothing$, $\mathcal{S}_2$ generates T key pairs $(\mathsf{pk}_{i\to j}^{(1)}, \mathsf{sk}_{i\to j}^{(1)}), \ldots, (\mathsf{pk}_{i\to j}^{(\mathtt{T})}, \mathsf{sk}_{i\to j}^{(\mathtt{T})})$ for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$. Let $\alpha \coloneqq \lfloor \frac{1}{p} \rfloor$. In each round r of the corresponding Aggregate Stage and for every neighbor $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_2$ does the following: P_i receives ciphertext $[b_1, \ldots, b_\alpha, u]_{\mathsf{pk}_{*\to i}^{(r)}}$ and the public key $\mathsf{pk}_{*\to i}^{(r)}$ destined for P_j. Instead of adding a layer and homomorphically OR'ing the bit b_i, $\mathcal{S}_2$ computes $(b'_1, \ldots, b'_\alpha, u') = (b_1 \vee b_i \vee u_i, \cdots, b_\alpha \vee b_i \vee u_i, u \vee u_i)$, and sends $[b'_{\sigma(1)}, \cdots, b'_{\sigma(\alpha)}, u']_{\mathsf{pk}_{i\to j}^{(r)}}$ to P_j, where σ is a random permutation on α elements. In other words, it sends the same message as $\mathcal{S}_1$ but encrypted with a fresh

public key. In the corresponding Decrypt Stage, P_i will get back a ciphertext from P_j encrypted under this exact fresh public key.

Hybrid 3. $\mathcal{S}_3$ now simulates the ideal functionality during the Aggregate Stage. It does so by sending encryptions of $(1, \ldots, 1)$ instead of the actual messages and unhappy bits. More formally, let $\alpha := \lfloor \frac{1}{p} \rfloor$. In each round r of a subphase of a random walk phase and for all parties $P_i \in \mathcal{P} \setminus \mathcal{Z}^p$ and $P_j \in \mathbf{N}_G(P_i) \cap \mathcal{Z}^p$, $\mathcal{S}_3$ sends $[1, 1, \ldots, 1]_{\mathsf{pk}_{i \to j}^{(r)}}$ instead of the ciphertext $[b_1, \ldots, b_\alpha, u]_{\mathsf{pk}_{i \to j}^{(r)}}$ sent by $\mathcal{S}_2$.

Hybrid 4. $\mathcal{S}_4$ does the same as $\mathcal{S}_{FB}$ during the Decrypt Stage for all phases and subphases except for the first subphase of a random walk phase in which the adversary crashes a party.

Hybrid 5. $\mathcal{S}_5$ is the actual simulator $\mathcal{S}_{FB}$.

The proofs that no efficient distinguisher D can distinguish between Hybrid 1, Hybrid 2 and Hybrid 3 are similar to the Claim 1 and Claim 2. Hence, we prove indistinguishability between Hybrid 3, Hybrid 4 and Hybrid 5.

Claim 5. *No efficient distinguisher D can distinguish between Hybrid 3 and Hybrid 4.*

Proof of claim. The only difference between the two hybrids is in the Decrypt Stage. We differentiate three cases:

- A subphase l of a phase k where the adversary did not crash any party in this phase, any previous subphase, or any previous phase. In this case, $\mathcal{S}_3$ sends with probability p an encryption of (b_W, u_W), where $b_W = \bigvee_{u \in W} b_u$ is the OR of all input bits in the walk and $u_W = 0$ (since no crash occurs), and with probability $1 - p$ an encryption of $(1, 0)$. On the other hand, $\mathcal{S}_4$ samples r from a Bernoulli distribution with parameter p, and if $r = 1$, it sends an encryption of $(b_{out}, 0)$, where $b_{out} = \bigvee_{i \in [n]} b_i$, and if $r = 0$ it sends an encryption of $(1, 0)$. Since the graph is connected, $b_{out} = b_W$ with overwhelming probability, as proven in Corollary 9.2.2. Also, the encryption in Hybrid 4 is done with a fresh public key which is indistinguishable with the encryption done in Hybrid 3 by OR'ing many times in the graph, as shown in Claim 2.1. in [ALM17a].

- A subphase l of a phase k where the adversary crashed a party in a previous subphase or a previous phase.

  In Hybrid 3 the parties send encryptions of $(1,1)$. This is also the case in Hybrid 4, because even if a crashed party disconnected the graph, each connected component contains a neighbor of a crashed party. Moreover, in Hybrid 4, the messages are encrypted with a fresh public key, and in Hybrid 3, the encryptions are obtained by the homomorphic OR operation. Both encryptions are indistinguishable, as shown in Claim 2.1. in [ALM17a].

$\Diamond$

Claim 6. *No efficient distinguisher D can distinguish between Hybrid 4 and Hybrid 5.*

Proof of claim. The only difference between the two hybrids is in the Decrypt Stage of the first subphase of a phase where the adversary crashes.

Let F be the set of pairs (P_f, r) such that $\mathcal{A}$ crashed P_f at round r of the phase. In Hybrid 4, a walk W of length $\mathtt{T}$ is generated from party P_o. Let W_1 be the region of W from P_o to the first not passively corrupted party and let W_2 be the rest of the walk. Then, the adversary's view at this step is the encryption of $(1,1)$ if one of the crashed parties breaks W_2 or if the walk became dummy (which happens with probability $1-p$, since the ciphertexts are permuted randomly and only one ciphertext out of $\frac{1}{p}$ contains b_W). Otherwise, the adversary's view is an encryption of $(b_W, 0)$. In both cases, the message is encrypted under a public key for which the adversary knows the secret key.

In Hybrid 5, a walk W_1' is generated from P_o of length $\ell \leq \mathtt{T}$ ending at the first not passively corrupted party P_i. Then, the simulator queries the leakage function on input $(F, P_i, \mathtt{T} - \ell)$. Then, with probability p it generates a walk W_2' of length $\mathtt{T} - \ell$ from P_i, and checks whether W_2' is broken by any party in F. If W_2' is broken, P_i sends an encryption of $(1,1)$, and otherwise an encryption of $(b_W, 0)$. Since the walk W' defined as W_1' followed by W_2' follows the same distribution as W, $b_W = b^{out}$ with overwhelming probability, and the encryption with a fresh public key which is indistinguishable with the encryption done by OR'ing many times in the graph, then it is impossible to distinguish between Hybrid 4 and Hybrid 5. $\Diamond$

This concludes the proof of soundness. □

E.2 From Broadcast to Topology-Hiding Computation

This section contains supplementary material for Section 9.4.

Naive composition of broadcast. We first argue that composing t broadcasts with one bit leakage can in general leak t bits.

Given black-box access to a fail-stop secure topology-hiding broadcast with a leakage function, the naive thing to do to compose broadcasts is run both broadcasts, either in parallel or sequentially. So, consider composing two broadcasts together, first in parallel. Each protocol is running independently, and so if there is an abort, the simulator will need to query the leakage function twice, unless we can make the specific claim that the leakage function will output a correlated bit for independent instances given the same abort (note that our construction does not have this property).

If we run the protocols sequentially, we'll need to make a similar claim. If we are simulating this composition and there is both an abort in the first broadcast and the second, then we definitely need to query the leakage function for the first abort. Then, unless we can make specific claims about how we could start a broadcast protocol *after* there has already been an abort, we will need to query the leakage oracle again.

E.2.1 All-to-all Multibit Broadcast

We show how to edit the protocol $\mathsf{BC\text{-}FB}_p$ to implement all-to-all multibit broadcasts, meaning we can broadcast k multibit messages from k not-necessarily distinct parties in a single broadcast. The edited protocol leaks a fraction p of a bit in total. While this transformation is not essential to compile MPC protocols to topology-hiding ones, it will cut down the round complexity by a factor of n times the size of a message.

First observe that $\mathsf{BC\text{-}FB}_p$ actually works also to broadcast multiple bits. Instead of sending a single bit during the random-walk protocol, it is enough that parties send vectors of ciphertexts. That is, in each round parties send a vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$.

Now we show how to achieve an all-to-all broadcast. Assume each party P_i wants to broadcast some k-bit message, $(b_1, \ldots, b_k)$. We consider a vector of length nk, where each of the n parties is assigned to k slots for k bits of its message. Each of the vectors $\vec{b_i}$ in the vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$ described above will be of this form. P_i will use the slots from $n(i-1)$ to ni to communicate its message. This means that P_i will have as input vector $\vec{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$. Then, in the Aggregate Stage, the parties will input their input message into their corresponding slots (by homomorphically OR'ing the received vector with its input message). At the end of the protocol, each party will receive the output containing the broadcast message of each party P_j in the slots $n(j-1)$ to nj.

Lemma E.2.1. *Protocol $\mathsf{BC\text{-}FB}_p$ can be edited to an all-to-all multi-bit broadcast $\mathsf{MultibitBC}_p$, which is secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit. The round complexity of $\mathsf{MultibitBC}_p$ is the same as for $\mathsf{BC\text{-}FB}_p$.*

Proof. This involves the following transformation of protocol $\mathsf{BC\text{-}FB}_p$. Note that $\mathsf{BC\text{-}FB}_p$ is already multibit; during the random-walk protocol, parties send around vectors of ciphertexts: $[\vec{b}, u] := [b_1, \ldots, b_\ell, u]$. In the transformed protocol we will substitute each ciphertext encrypting a bit b_i with a vector of ciphertexts of length m, containing encryptions of a vector of bits $\vec{b_i}$. That is, we now think of parties sending a vector of vectors $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$. Technically, we "flatten" these vectors, that is, the parties will send vectors of length $m\ell + 1$ of ciphertexts.

Let us now explain the transformation. For an all-to-all broadcast, each party, P_i, wants to broadcast some k-bit message, $(b_1, \ldots, b_k)$. Consider a vector of ciphertexts of length nk, where each of the n parties is assigned to k slots for k bits of its message. Each of the vectors $\vec{b_i}$ in the vector $[\vec{b_1}, \ldots, \vec{b_\ell}, u]$ described above will be of this form. P_i will use the slots from $n(i-1)$ to ni to communicate its message.

We now have a look at the Aggregate Stage in the transformed protocol $\mathsf{MultibitBC}_p$.

- Every party P_i who wants to send the k bit message $(b_1, \ldots, b_k)$ prepares its input vector $\vec{b_i} = (0, \ldots, 0, b_1, \ldots, b_k, 0, \ldots, 0)$ by placing the bits $b_1, \ldots, b_k$ in positions from $n(i-1)$ to ni.

- At the beginning of the Aggregate Stage, the recipient P_o with the input vector $\vec{b_o}$ sends the ciphertext $[\vec{b_o} \vee u_o, \vec{1}, \ldots, \vec{1}, u_o]_{\mathsf{pk}^{(1)}_{i\to 1}}$ to its first neighbor. All other ciphertexts to all other neighbors j are just $[\vec{1}, \ldots, \vec{1}, 1]_{\mathsf{pk}^{(1)}_{i\to j}}$ [1].

  Every other party P_i starts the protocol with sending the ciphertext tuple $[\vec{1}, \ldots, \vec{1}, 1]_{\mathsf{pk}^{(1)}_{i\to j}}$ to every neighbor j.

- Upon receiving a ciphertext $[\vec{b}_1, \ldots, \vec{b}_\ell, u]_{pk^{(t)}_{j\to i}}$ at round r from a neighbor j, party P_i takes its input vector $\vec{b_i}$ and homomorphically OR's the vector $(\vec{b_i} \vee u_i, \ldots, \vec{b_i} \vee u_i, u_i)$ containing ℓ copies of the vector $\vec{b_i} \vee u_i$ to the ciphertext. The result is sent along the walk.

The rest of the protocol $\mathsf{MultibitBC}_p$ proceeds analogously to $\mathsf{BC\text{-}FB}_p$.

A quick check of correctness tells us that when a message is not made unhappy, and starts with 0's in the appropriate places, every party's broadcast message eventually gets OR'd in a different spot in the message vector, and so every party will get that broadcast.

A quick check of soundness tells us that the simulator works just as before: it simulates with the encrypted output (all nk bits) when there was no abort, and with a query to the leakage function if there was one. □

E.2.2 Sequential Execution Without Aggregated Leakage

We show how to construct a protocol, which implements any number of sequential executions of the protocol $\mathsf{MultibitBC}_p$, while preserving the leakage of a fraction p of a bit in total. The construction makes non-black-box use of the unhappy bits used in $\mathsf{MultibitBC}_p$. The idea is simply to preserve the state of the unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort.

Lemma E.2.2. *There exists a protocol, which implements any number k of executions of the protocol $\mathsf{MultibitBC}_p$, is secure against an adversary,*

[1] We are abusing notation: $\vec{b_o} \vee u_o$ means that we OR u_i with every coordinate in $\vec{b}$.

who statically passively corrupts and adaptively crashes any number of parties and leaks at most a fraction p of a bit in total. The complexity of the constructed protocol is k times the complexity of $\mathsf{MultibitBC}_p$.

Proof. The construction makes non-black-box use of the unhappy bits used in $\mathsf{MultibitBC}_p$. The idea is simply to preserve the state of the unhappy bits between sequential executions. That is, once some party sees a crash, it will cause all subsequent executions to abort.

Correctness and complexity of the above construction are trivial, since it simply executes the protocol $\mathsf{MultibitBC}_p$ k times.

We now claim that any leakage happens only in the one execution of protocol $\mathsf{MultibitBC}_p$, in which the first crash occurs. Once we show this, it is easy to see that the constructed protocol executing $\mathsf{MultibitBC}_p$ k times leaks at most a fraction p of a bit.

By Theorem 9.3.2, any execution without crashes causes no leakage (it an be easily simulated as in the setting with only passive corruptions and no fail-stop adversary). Further, assume that any party P_c crashes before $\mathsf{BC\text{-}FB}_p$ starts. Let $\mathbf{N}_G(a)$ be all of P_a's neighbors; all of them will have their unhappy bit set to 1. Because of the correctness of the random-walk protocol embedded within $\mathsf{BC\text{-}FB}_p$, the random walk will hit every node in the connected component, and so is guaranteed to visit a node in $\mathbf{N}_G(a)$. Therefore, every walk will become a dummy walk, which is easily simulated.

□

Remark. We note that the above technique to sequentially execute protocols which leak p bits and are secure *with abort* can be applied to a more general class of protocols (in particular, not only to our topology-hiding broadcast). The idea is that if a protocol satisfies the property that any abort before it begins implies that the protocol does not leak any information, then it can be executed sequentially leaking at most p bits.

E.2.3 Topology-Hiding Computation

We are now ready to compile any MPC protocol (secure against an adversary, who statically passively corrupts and adaptively crashes any number of parties) into one that is topology-hiding and leaks at most a fraction p of a bit.

To do this, it is enough to do a standard transformation using public key infrastructure. Let Π_{MPC} be a protocol that runs in M rounds. First, the parties use one all-to-all multi-bit topology-hiding broadcast protocol to send each public key to every other party. Then, each round of Π_{MPC} is simulated: the parties run n all-to-all multi-bit topology hiding broadcasts simultaneously to send the messages sent in that round encrypted under the corresponding public keys. After the broadcasts, each party can use their secret key to decrypt their corresponding messages.

Theorem E.2.3. *Assume PKCR exists. Then, we can compile any MPC protocol Π_{MPC} that runs in M rounds into a topology-hiding protocol with leakage function $\mathcal{L}_{FB_p}$, that runs in $MR+1$ rounds, where R is the round complexity of $\mathsf{BC\text{-}FB}_p$.* [2]

Proof. Recall the generic transformation for taking UC-secure topology-hiding broadcast and compiling it into UC-secure topology-hiding MPC using a public key infrastructure. Every MPC protocol with M rounds, Π_{MPC}, has at each round each party sending possibly different messages to every other party. This is a total of $O(n^2)$ messages sent at each round, but we can simulate this with n separate multi-bit broadcasts.

To transform Π_{MPC} into a topology-hiding protocol in the fail-stop model, given a multi-bit topology-hiding broadcast, we do the following:

- Setup phase. The parties use one multi-bit topology-hiding broadcast to give their public key to every other party.

- Each round of Π_{MPC}. For each party P_i that needs to send a message of k bits to party P_j, P_i encrypts that message under P_j's public key. Then, each party P_i broadcasts the $n-1$ messages it would send in that round of Π_{MPC}, one for each $j \neq i$, encrypted under the appropriate public keys. That is, P_i is the source for one multi-bit broadcast. All these multi-bit broadcasts are simultaneously executed via an all-to-all multi-bit broadcast, where each party broadcast a message of size $(n-1)k$ times.
  After the broadcasts, each node can use their secret key to decrypt the messages that were for them and continue with the protocol.

[2] In particular, the complexity of $\mathsf{BC\text{-}FB}_p$ is $n \cdot \rho \cdot 2\mathtt{T}$, where κ is the security parameter, $\tau = \log(n) + \kappa$, $\mathtt{T} = 8n^3\tau$ is the length of a walk and $\rho = \tau/(p' - 2^{-\tau})$ is the number of repetitions of a phase (with $p' = 1/\lfloor 1/p \rfloor$).

- At the end of the protocol, each party now has the output it would have received from running Π_{MPC}, and can compute its respective output.

First, this is a correct construction. We will prove this by inducting on the rounds of Π_{MPC}. To start, all nodes have all information they would have had at the beginning of Π_{MPC} as well as public keys for all other parties and their own secret key. Assume that the graph has just simulated round $r-1$ of Π_{MPC} and each party has the information it would have had at the end of round $r-1$ of Π_{MPC} (as well as the public keys etc). At the end of the r'th simulated round, each party P_i gets encryptions of messages sent from every other party P_j encrypted under P_i's public key. These messages were all computed correctly according to Π_{MPC} because all other parties had the required information by the inductive hypothesis. P_i can then decrypt those messages to get the information it needs to run the next round. So, by the end of simulating all rounds of Π_{MPC}, each party has the information it needs to complete the protocol and get its respective output.

Security of this construction (and, in particular, the fact that it only leaks a fraction p of a bit) follows directly from Lemma E.2.1 and Lemma E.2.2.

□

We can now conclude that any MPC functionality can be implemented by a topology-hiding protocol. Since PKCR is implied by either DDH, QR or LWE, we get the following theorem as a corollary.

Theorem 9.1.1. *If DDH, QR or LWE is hard, then any MPC functionality $\mathcal{F}$ can be realized by a topology-hiding protocol which is secure against an adversary that does any number of static passive corruptions and adaptive crashes, leaking an arbitrarily small fraction p of a bit. The round and communication complexity is polynomial in κ and $1/p$.*

Proof. Because every poly-time computable functionality $\mathcal{F}$ has an MPC protocol [CLOS02], we get that Theorem E.2.3 implies that we can get topology-hiding computation. The round and communication complexity is implied by Theorem E.2.3 and the complexity of $\mathsf{MultibitBC}_p$.

□

E.3 Deeply Fully-Homomorphic Public-Key Encryption

We present the formal definition of a deeply fully-homomorphic public-key encryption from Section 9.5.1.

Our protocol requires a PKE scheme $\mathcal{E}$ where (a) one can add and remove layers of encryption, while (b) one can homomorphically compute any function on encrypted bits (independent of the number of layers). This will be captured by three additional algorithms: $\mathsf{AddLayer}_r$, $\mathsf{DelLayer}_r$, and HomOp_r, operating on ciphertexts with r layers of encryption (we will call such ciphertexts level-r ciphertexts). A level-r ciphertext is encrypted under a level-r public key (we assume that each level can have different key space).

Definition E.3.1. A *deeply fully-homomorphic public-key encryption* (DFH-PKE) scheme is a PKE scheme with three additional algorithms $\mathsf{AddLayer}_r$, $\mathsf{DelLayer}_r$, and HomOp_r. We define additional public-key spaces $\mathcal{PK}_r$ and ciphertext spaces $\mathcal{C}_r$, for public keys and ciphertexts on level r. We require that $\mathcal{PK}_1 = \mathcal{PK}$ and $\mathcal{C}_1 = \mathcal{C}$. Let $\mathcal{F}$ be the family of efficiently computable functions.

- The algorithm $\mathsf{AddLayer}_r : \mathcal{C}_r^* \times \mathcal{PK}_r \times \mathcal{SK} \to \mathcal{C}_{r+1}^* \times \mathcal{PK}_{r+1}$ takes as input a level-r ciphertext $[\![m]\!]_{\mathbf{pk}}$, the corresponding level-r public key $\mathbf{pk}$, and a new secret key sk. It outputs a level-$(r+1)$ ciphertext and the level-$(r+1)$ public key, under which it is encrypted.
- The algorithm $\mathsf{DelLayer}_r : \mathcal{C}_{r+1}^* \times \mathcal{PK}_{r+1} \times \mathcal{SK} \to \mathcal{C}_r^* \times \mathcal{PK}_r$ deletes a layer from a level-$(r+1)$ ciphertext.
- The algorithm $\mathsf{HomOp}_r : \mathcal{C}_r^* \times \mathcal{PK}_r \times \mathcal{F} \to \mathcal{C}_r$ takes as input some k level-r ciphertexts encrypted under the same level-r public key, the corresponding public key, and a k-ary function f. It outputs a level-r ciphertext that contains f of the encrypted messages.

For convenience, it will be easy to describe the security of our enhanced encryption scheme with the help of an algorithm $\mathsf{Leveled\text{-}Encrypt}_r$, which takes as input a vector of plain messages and a level-r public key, and outputs a vector of level-r ciphertexts[3].

[3]This algorithm can be obtained by keeping an encryption of 0 and 1 as part of the leveled public key and rerandomizing the ciphertext using HomOp_r.

Definition E.3.2. For a DFH-PKE scheme, we additionally define the algorithm $\mathsf{Leveled\text{-}Encrypt}_r : \mathcal{M}^* \times \mathcal{PK}_r \to \mathcal{C}_r^* \times \mathcal{PK}_r$ that outputs the level-r encryptions of the messages $\vec{m}$ and the corresponding level-r public key.

Intuitively, we will require that one cannot obtain any information on the underlying layers of encryption from the output of $\mathsf{AddLayer}_r$ ($\mathsf{DelLayer}_r$). That is, that the output of $\mathsf{AddLayer}_r$ ($\mathsf{DelLayer}_r$) is indistinguishable from a level-$(r+1)$ (level-r) encryption of the message. We will also require that the output of HomOp_r is indistinguishable from a level-r encryption of the output of the functions applied to the messages.

Definition E.3.3. We require that a DFH-PKE scheme satisfies the following properties:

Aggregate Soundness. For every r, every vector of messages $\vec{m}$ and every efficiently computable pair of level-r public keys $\mathbf{pk_1}$ and $\mathbf{pk_2}$,

$$\{\mathsf{AddLayer}_r([\![\vec{m}]\!]_{\mathbf{pk_1}}, \mathbf{pk_1}, \mathsf{sk}; U^*) : (\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(1^\kappa; U^*)\}$$

$$\approx_c$$

$$\left\{(\mathsf{Leveled\text{-}Encrypt}_{r+1}(\vec{m}, \mathbf{pk_2'}; U^*), \mathbf{pk_2'}) : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(1^\kappa; U^*), \\ ([\![0]\!]_{\mathbf{pk_2'}}, \mathbf{pk_2'}) \leftarrow \mathsf{AddLayer}_r([\![0]\!]_{\mathbf{pk_2}}, \mathbf{pk_2}, \mathsf{sk}; U^*)\end{array}\right\}$$

Decrypt Soundness. For every r, every vector $\vec{m}$ and every efficiently computable level-r public key $\mathbf{pk_1}$,

$$\left\{\mathsf{DelLayer}_r([\![\vec{m}]\!]_{\mathbf{pk}}, \mathbf{pk}, \mathsf{sk}; U^*) : \begin{array}{c}(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{Keygen}(1^\kappa; U^*), \\ ([\![0]\!]_{\mathbf{pk}}, \mathbf{pk}) \leftarrow \mathsf{AddLayer}_r([\![0]\!]_{\mathbf{pk_1}}, \mathbf{pk_1}, \mathsf{sk}; U^*)\end{array}\right\}$$

$$\approx_c$$

$$\{(\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk_1}; U^*), \mathbf{pk_1})\}$$

Full-Homomorphism. For every vector of messages $\vec{m} \in \mathcal{M}^*$, every level-$r$ public key $\mathbf{pk}$, every vector of ciphertexts $\vec{c} \in \mathcal{C}^*$ and every function $f \in \mathcal{F}$ where $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk}) = \vec{c}$,

$$\{(\vec{m}, \vec{c}, \mathbf{pk}, f, \mathsf{Leveled\text{-}Encrypt}_r(f(\vec{m}), \mathbf{pk}; U^*))\}$$

$$\approx_c$$

$$\{(\vec{m}, \vec{c}, \mathbf{pk}, f, \mathsf{HomOp}_r(\vec{c}, \mathbf{pk}, f; U^*))\}$$

Note that $\mathsf{AddLayer}_r$ and $\mathsf{DelLayer}_r$ produce both the level-r encrypted messages and the level-r public key. In the case where we only need the public key, we will just call the procedure $\mathsf{AddLayer}_r([\![0]\!]_{\mathbf{pk}}, \mathbf{pk}, \mathsf{sk})$, since the encrypted message does not matter for producing a new public key — the same applies for $\mathsf{DelLayer}_r$.

Also note that one can create a level-r public key generating r level-1 key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Keygen}(1^\kappa)$ and using $\mathsf{AddLayer}$ to add the public keys one by one. Furthermore, with all secret keys $(\mathsf{sk}_1, \ldots, \mathsf{sk}_r)$ used in the creation of some level-r public key $\mathbf{pk}$, we can define a combined level-r secret key $\mathbf{sk} = (\mathsf{sk}_1, \ldots, \mathsf{sk}_r)$, which we can use to decrypt a level-r ciphertext by calling $\mathsf{DelLayer}$ r times.

E.3.1 Instantiation of DFH-PKE from FHE

We show how to instantiate DFH-PKE from FHE. As required from the DFH-PKE scheme, the level-1 public key space and ciphertext space are the FHE public key space and FHE ciphertext space respectively, i.e., $\mathcal{PK}_1 = \mathcal{PK}$ and $\mathcal{C}_1 = \mathcal{C}$. For $r > 1$, a level-r public key and ciphertext spaces are $\mathcal{PK}_r = \mathcal{PK} \times \mathcal{C}$ and $\mathcal{C}_r = \mathcal{C}$, respectively.

Notation. We denote by $\mathrm{FHE}.\mathsf{Enc}(m, \mathsf{pk})$ the FHE encryption algorithm that takes message m and encrypts under public key pk. In the same way, the FHE decryption algorithm is denoted by $\mathrm{FHE}.\mathsf{Dec}$. The FHE evaluation algorithm is defined as

$$\mathrm{FHE}.\mathsf{HomOp}([m_1, \ldots, m_n]_{\mathsf{pk}}, \mathsf{pk}, f) \coloneqq [f(m_1, \ldots, m_n)]_{\mathsf{pk}}.$$

It gets as input a vector of encrypted messages under pk, the public key pk and the function to evaluate, and it returns the output of f applied to the messages.

In the following we define the algorithms to add and delete layers of encryption. Both algorithms receive as inputs a vectors of ciphertexts, a leveled public key and a secret key.

Algorithm $\mathsf{AddLayer}_r((\mathsf{c}_1, \ldots, \mathsf{c}_n), \mathbf{pk}, \mathsf{sk})$

Let pk be the corresponding public key of sk.
$\mathsf{c}'_i \leftarrow \mathrm{FHE}.\mathsf{Enc}(\mathsf{c}_i, \mathsf{pk})$.
$\mathbf{pk}' \leftarrow (\mathsf{pk}, \mathrm{FHE}.\mathsf{Enc}(\mathbf{pk}, \mathsf{pk}))$.

return $((\mathsf{c}'_1, \ldots, \mathsf{c}'_n), \mathbf{pk}')$.

Algorithm $\mathsf{DelLayer}_r((\mathsf{c}'_1, \ldots, \mathsf{c}'_n), \mathbf{pk}', \mathsf{sk})$

Parse $\mathbf{pk}' = (\mathsf{pk}, [\mathbf{pk}]_{\mathsf{pk}})$.
$\mathbf{pk} \leftarrow \mathrm{FHE.Dec}([\mathbf{pk}]_{\mathsf{pk}}, \mathsf{sk})$.
$\mathsf{c}_i \leftarrow \mathrm{FHE.Dec}(\mathsf{c}'_i, \mathsf{sk})$.
return $((\mathsf{c}_1, \ldots, \mathsf{c}_n), \mathbf{pk})$.

Notice the recursive nature of leveling; to make notation less cumbersome, let $\mathbf{pk}_r = (\mathsf{pk}_r, [\mathsf{pk}_{r-1}, [\ldots [\mathsf{pk}_1]_{\mathsf{pk}_2} \ldots]_{\mathsf{pk}_{r-1}}]_{\mathsf{pk}_r})$, and $[\![m]\!]_{\mathbf{pk}_r}$ denotes the leveled ciphertext, i.e., $[\![m]\!]_{\mathbf{pk}_r} = [[\ldots [m]_{\mathsf{pk}_1} \ldots]_{\mathsf{pk}_{r-1}}]_{\mathsf{pk}_r}$. Hence, it is easy to see that the two algorithms above accomplish the following:

$$\mathsf{AddLayer}_r([\![\vec{m}]\!]_{\mathbf{pk}_r}, \mathbf{pk}_r, \mathsf{sk}_{r+1}) = ([\![\vec{m}]\!]_{\mathbf{pk}_{r+1}}, \mathbf{pk}_{r+1})$$

and

$$\mathsf{DelLayer}_r([\![\vec{m}]\!]_{\mathbf{pk}_{r+1}}, \mathbf{pk}_{r+1}, \mathsf{sk}_r) = ([\![\vec{m}]\!]_{\mathbf{pk}_r}, \mathbf{pk}_r)$$

In the following, we show how to apply any function f on any vector of level-r ciphertexts. It is clear that if the ciphertexts are level-1 ciphertexts, we can apply f using FHE directly. If the ciphertexts are level-r ciphertexts for $r > 1$, we FHE evaluate the ciphertexts and public key with a recursive function call on the previous level. More concretely, we use the following recursive algorithm to apply f to any vector of level-r ciphertexts:

Algorithm $\mathsf{HomOp}_r((\mathsf{c}_1, \ldots, \mathsf{c}_n), \mathbf{pk}, f)$

if $r = 1$ **then**
  Parse $\mathbf{pk} = \mathsf{pk}$.
  return $\mathrm{FHE.HomOp}((\mathsf{c}_1, \ldots, \mathsf{c}_n), \mathsf{pk}, f)$.
Parse $\mathbf{pk} = (\mathsf{pk}, [\mathbf{pk}_{r-1}]_{\mathsf{pk}})$, $\mathsf{c}_i = [\mathsf{c}'_i]_{\mathsf{pk}}$.
Let $f'(\cdot, \cdot) := \mathsf{HomOp}_{r-1}(\cdot, \cdot, f)$.
return $\mathrm{FHE.HomOp}(([(\mathsf{c}'_1, \ldots, \mathsf{c}'_n)]_{\mathsf{pk}}, [\mathbf{pk}_{r-1}]_{\mathsf{pk}}), \mathsf{pk}, f')$.

Lemma E.3.4. *For any r, algorithm HomOp_r is correct on leveled ciphertexts.*

Proof. We want to show that for a vector of level-r ciphertexts $\vec{c} = [\![\vec{m}]\!]_{\mathbf{pk}}$, $\mathsf{HomOp}_r(c, \mathbf{pk}, f) = [\![f(\vec{m})]\!]_{\mathbf{pk}}$. We will prove this via induction on r.

For the base case, consider $r = 1$. Here we go into the if statement, and the algorithm returns $\text{FHE.}\mathsf{HomOp}([\vec{m}]_{\mathsf{pk}}, \mathsf{pk}, f) = [f(\vec{m})]_{\mathsf{pk}}$ by the correctness of the FHE scheme.

Now, assume that $\mathsf{HomOp}_{r-1}([\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}, \mathbf{pk}_{r-1}, f) = [\![f(\vec{m})]\!]_{\mathbf{pk}_{r-1}}$ for all messages $\vec{m}$ encrypted under $r-1$ levels of keys. Calling HomOp_r on $[\![\vec{m}]\!]_{\mathbf{pk}_r}$ results in returning

$$\begin{aligned}
&\text{FHE.}\mathsf{HomOp}(([\![\vec{m}]\!]_{\mathbf{pk}_r}, [\mathbf{pk}_{r-1}]_{\mathsf{pk}_r}), \mathsf{pk}_r, \mathsf{HomOp}_{r-1}(\cdot,\cdot,f)) \\
&= [\mathsf{HomOp}_{r-1}([\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}, \mathbf{pk}_{r-1}, f)]_{\mathsf{pk}_r} \\
&= [\![f(\vec{m})]\!]_{\mathbf{pk}_r}
\end{aligned}$$

by correctness of the FHE homomorphic evaluation. □

We are also able to encrypt in a leveled way by exploiting the fully-homomorphic properties of the scheme, using the FHE.HomOp algorithm to apply encryption.

Algorithm $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk})$

if $r = 1$ **then**
  Parse $\mathbf{pk} = \mathsf{pk}$
  return $(\text{FHE.}\mathsf{Enc}(m_i, \mathsf{pk}))_i$
Parse $\mathbf{pk} = (\mathsf{pk}, [\mathbf{pk}_{r-1}]_{\mathsf{pk}})$.
Let $[\vec{m}]_{\mathsf{pk}} = (\text{FHE.}\mathsf{Enc}(m_i, \mathsf{pk}))_i$.
return $\text{FHE.}\mathsf{HomOp}(([\vec{m}]_{\mathsf{pk}}, [\mathbf{pk}_{r-1}]_{\mathsf{pk}}), \mathsf{pk}, \mathsf{Leveled\text{-}Encrypt}_{r-1})$

Finally, we need to prove that adding a fresh layer is equivalent to looking like a fresh random encryption.

Lemma E.3.5. $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk}_r) = [\![\vec{m}]\!]_{\mathbf{pk}_r}$.

Proof. We will prove this by induction on r. For $r = 1$, it follows from the base case that

$$\mathsf{Leveled\text{-}Encrypt}_1(\vec{m}, \mathbf{pk}_1) = \text{FHE.}\mathsf{Enc}(\vec{m}, \mathsf{pk}_1) = [\![\vec{m}]\!]_{\mathbf{pk}_1}.$$

Now, assume that for $r-1$, $\mathsf{Leveled\text{-}Encrypt}_{r-1}(\vec{m}, \mathbf{pk}_{r-1}) = [\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}$. This means that when we call $\mathsf{Leveled\text{-}Encrypt}_r(\vec{m}, \mathbf{pk}_r)$, we return

$$\mathrm{FHE.HomOp}([\vec{m}]_{\mathsf{pk}_r}, [\mathbf{pk}_{r-1}]_{\mathsf{pk}_r}, \mathsf{pk}_r, \mathsf{Leveled\text{-}Encrypt}_{r-1}) = [[\![\vec{m}]\!]_{\mathbf{pk}_{r-1}}]_{pk_r} = [\![\vec{m}]\!]_{\mathbf{pk}_r},$$

as desired. □

Lemma E.3.6. *The instantiation of DFH-PKE from FHE presented above satisfies the properties Aggregate Soundness, Decrypt Soundness and Full-Homomorphism, presented in Definition E.3.3.*

Proof. **Aggregate Soundness.** The algorithm $\mathsf{AddLayer}$ returns a pair $([\![\vec{m}]\!]_{\mathbf{pk}}, \mathbf{pk}')$, where $\vec{m}$ is a vector, and $\mathbf{pk}' = (\mathsf{pk}, \mathrm{FHE.Enc}(\mathbf{pk}, \mathsf{pk}))$ is a pair containing a fresh public key pk and an encryption of a level-r key $\mathbf{pk}$ under the fresh public key pk. Observe that this is exactly a level-$(r+1)$ key.

The tuple $(\mathsf{Leveled\text{-}Encrypt}_{r+1}(\vec{m}, \mathbf{pk_1}; U^*), \mathbf{pk_1})$, where $\mathbf{pk_1}$ is a level-$(r+1)$ public key obtained from adding a fresh layer to a level-r public key, has the same distribution: the first part of both tuples contain fresh FHE encryptions of level-r ciphertexts, and the second part is a level-$(r+1)$ public key.

Decrypt Soundness. This property is trivially achieved given the correctness of the FHE decryption algorithm and $\mathsf{Leveled\text{-}Encrypt}_r$.

Full-Homomorphism. The $\mathsf{Leveled\text{-}Encrypt}_r$ algorithm returns a level-r encryption of $f(\vec{m})$ which is the result of applying FHE homomorphic operations on a level-r ciphertext. The algorithm HomOp_r also returns a level-r ciphertext output by the FHE homomorphic operation. □

E.4 THC from DFH-PKE

In this section, we present a detailed description of protocol $\mathsf{DFH\text{-}THC}$ from Section 9.5.2.

We will use DFH-PKE to alter the $\mathsf{RandomWalkPhase}$ protocol (and by extension we can alter $\mathsf{ProbabilisticRandomWalkPhase}_p$). Then, executing protocols $\mathsf{BC\text{-}OB}$ and $\mathsf{BC\text{-}FB}_p$ that leak one bit and a fraction of a bit

respectively will be able to evaluate any poly-time function instead, while still leaking the same amount of information as a broadcast using these random walk protocols. The concept is simple. During the Aggregate Stage, parties will add a leveled encryption of their input and identifying information to a vector of ciphertexts, while adding a layer — we will not need sequential id's if each party knows where their input should go in the function. Then, at the end of the Aggregate Stage, nodes homomorphically evaluate f', which is the composition of a parsing function, to get one of each input in the right place, and f, to evaluate the function on the parsed inputs. The result is a leveled ciphertext of the output of f. This ciphertext is un-layered in the Decrypt Stage so that by the end, the relevant parties get the output.

For completeness, we give a detailed description of the modified protocol RandomWalkPhase leaking one bit, denoted DFH-RandomWalkPhase:

Initialization Stage. Each party P_i has its own input bit b_i and unhappiness bit u_i. Each party P_i knows the function f on n variables that the graph wants to compute, and generates $\mathtt{T} \cdot d_i$ keypairs and $\mathtt{T}-1$ permutations on d_i elements (d_i is the number of neighbors for party i). P_i also generates a unique ID (or uses a given sequential or other ID) p_i. If party P_i witnessed an abort from the last phase, it becomes unhappy, setting its unhappy bit $u_i = 1$.

Aggregate Stage. *Round 1.* Each party P_i sends to each neighbor P_j a vector of level-1 ciphertexts under $\mathsf{pk}_{i\to j}^{(1)}$ containing the input bit b_i, id p_i, unhappy bit u_i and a bit v_i indicating whether the walk is dummy or not.

If P_i is the party that gets the output in that phase, i.e., $P_i = P_o$, then it sends to the first neighbor an encryption of b_i, p_i, u_i and a bit $v_i = 0$ indicating that the walk should not be dummy. To all other neighbors, $v_i = 1$. In the case where $P_i \neq P_o$, $v_i = 1$ as well.

Round $r \in [2, T]$. Let $k = \pi_i^{(r)}(j)$. Upon receiving a vector of level-$(r-1)$ ciphertexts from P_j. Party P_i uses $\mathsf{sk}_{i\to k}^{(r)}$ to add a fresh layer with AddLayer to the vector of ciphertexts. The function AddLayer will return the vector $\vec{c}$ of level-r ciphertexts with the corresponding level-r public key **pk**. Then, P_i will encrypt its own input, id and unhappybit via Leveled-Encrypt under **pk** and appends these

ciphertexts to $\vec{c}$. It then sends to P_k the level-r public key and all the level-r ciphertexts.

If no vector of ciphertexts was received from P_j (i.e. P_j aborted), P_i generates a fresh level-r public key **pk** and secret key **sk**. It then generates a vector of level-r ciphertexts containing the bit 1 using Leveled-Encrypt under **pk**. The size of this vector corresponds to the size of the vector containing the dummy bit, r input bits, r ids, and r unhappy bits.

Evaluation. We are now at the last step in the walk. If P_i received an encrypted vector of level-T ciphertexts from P_j, it evaluates the vector using HomOp_T on the function f' which does the following: if the dummy bit is 1 or any unhappy bit set to 1, the function evaluates to $\perp$. Otherwise, it arranges the inputs by ids and evaluates f on the arranged inputs. That is, it evaluates $f \circ \mathsf{parse}$, where $\mathsf{parse}((m_{i_1}, p_{i_1}), \ldots, (m_{i_T}, p_{i_T})) = (m_1, \ldots, m_n)$. More concretely, for the vector of ciphertexts $\vec{c}$ and level-T public key **pk** received from P_j, P_i evaluates $\hat{c} \leftarrow \mathsf{HomOp}(\vec{c}, \mathbf{pk}, f')$, and sends $\hat{c}$ to P_j.

If P_i did not receive a message from P_j, or u_i has been set to 1, P_i sends a ciphertext containing $\perp$: it generates a fresh level-T public key **pk** and secret key **sk**, and uses Leveled-Encrypt under **pk** to send to P_j a level-T ciphertext containing $\perp$.

Decrypt Stage. *Round* $r \in [T, 2]$ If P_i receives a level-r ciphertext c from P_j, party P_i will delete a layer using the secret key $\mathsf{sk}_{i\to j}^{(r)}$ that was used to add a layer of encryption at round r of the Aggregate Stage. Otherwise, it uses Leveled-Encrypt to encrypt the message $\perp$ under the level-$(r-1)$ public key that was received in round r during the Aggregate Stage.

Output. If P_i is the party that gets the output in that phase, i.e., $P_i = P_o$ and it receives a level-1 ciphertext c from its first neighbour, P_i computes the output message using Decrypt using the secret key $\mathsf{sk}_{i\to 1}^{(1)}$. In any other case, P_i outputs $\perp$. P_i also outputs its unhappy bit u_i.

Now, DFH-THC runs the protocol DFH-RandomWalkPhase n times, similarly to BC-OB.

Protocol DFH-THC$(\mathtt{T}, (d_i, \mathsf{input}_i)_{P_i \in \mathcal{P}}, f)$

Each party P_i sets $\mathsf{output}_i = \mathbf{1}$ and $u_i = 0$.
for o from 1 to n **do**
  Parties jointly execute $\left((\mathsf{input}_i^{temp}, u_i^{temp})_{P_i \in \mathcal{P}}\right) =$
    DFH-RandomWalkPhase$(\mathtt{T}, P_o, (d_i, \mathsf{input}_i, u_i)_{P_i \in \mathcal{P}}, f)$.
  Party P_o sets $\mathsf{output}_o = \mathsf{input}_o^{temp}$.
  Each party P_i sets $u_i = u_i^{temp} \vee u_i$.
Each party P_i **outputs** output_i if $\mathsf{output}_i \neq \perp$.

Theorem 9.5.1. *For security parameter κ, $\tau = \log(n) + \kappa$, $T = 8n^3\tau$, and $\rho = \tau/(p' - 2^{-\tau})$, where $p' = 1/\lfloor 1/p \rfloor$, the protocol DFH-THC topology-hidingly evaluates any poly-time function f, $\mathcal{F}_{INFO}^{\mathcal{L}_{FB_p}} || f$ in the $\mathcal{N}$ hybrid-world.*

Sketch. This proof will look almost exactly like the proof of Theorem 9.3.2. The simulator and its use of the leakage oracle will behave in nearly the same manner as before.

- During the Aggregate Stage, the simulator sends leveled encryptions of 1 of the appropriate size with the appropriate number of layers.
- During the Decrypt Stage, the simulator sends the output encrypted with the appropriate leveled keys.

Because Leveled-Encrypt$_r$ is able to produce a distribution of ciphertexts that looks identical to AddLayer$_r$, and by semantic security of the FHE scheme, no party can tell what other public keys were used except the most recently added one, the simulated ciphertexts and public keys are computationally indistinguishable from those in the real walk. It is also worth pointing out that as long as the FHE scheme only incurs additive blowup in error and size, and $T = \mathsf{poly}(\kappa)$, the ciphertexts being passed around are only $\mathsf{poly}(\kappa)$ in size. □

Part III

Efficient Byzantine Agreement

Chapter 10

Expand-and-Extract: A New Design for Round-Efficient Byzantine Agreement

10.1 Introduction

In the problem of Byzantine Agreement (BA), a set of n parties want to agree on a common output y by running a distributed protocol. The protocol must remain secure even when up to some t out of the n parties are corrupted by arbitrarily deviating from the protocol. First formalized in the seminal work of Lamport et al. [LSP82], BA is one of the most fundamental problems in cryptography and distributed computing.

A key efficiency metric for high-performance BA protocols is their *round efficiency*: *how many coordinated rounds of communication are needed to reach agreement?* As proven by Dolev and Strong [DS83], no *deterministic* BA protocol can run in less than $t + 1$ rounds. As first demonstrated by Ben-Or [Ben83], and Rabin [Rab83], the lower bound of Dolev and Strong does not apply to *randomized* protocols. Feldman and Micali [FM97] (FM) gave the first expected-constant-round protocol

with optimal resilience ($t < n/3$) and unconditional security. Expected-constant-round protocols are important from both a theoretical and practical perspective. However, as proven by Dwork and Moses [DM90], and Moses and Tuttle [MT88], they inherently cannot achieve *simultaneous termination*, i.e., that all parties terminate the protocol during the same communication round. This can make such protocols unwieldy when used as building blocks in larger protocol contexts—as was for instance nicely exposed by Lindell et al. [LLR02] or Cohen et al. [CCGZ16]. Hence, an important implication of the FM paradigm also lies in its use to build protocols which achieve simultaneous termination, yet circumvent the lower bound of Dolev and Strong: it immediately yields a protocol with error probability at most 2^{-k} and simultaneous termination in $O(k)$ rounds. Several protocols have subsequently improved over the round and communication complexity of the original FM protocol, both with respect to unconditional and computational security. In this work, we revisit and refine the FM approach to obtain fixed-round protocols with improved round complexity.

Refining the FM paradigm: expand and extract. Informally speaking, FM achieves BA from a weaker type of agreement called *graded consensus* (GC). In GC, parties output a *grade* g along with their output y, where g indicates the party's confidence in whether or not agreement on y has been achieved. FM gives a GC protocol with grades $\{0, 1, 2\}$ (grade range 3). To achieve BA, instances of GC are alternated with a coin toss until a certain termination condition is met. Our main contribution is to present a generalized view of the FM paradigm which we refer to as *expand and extract*.

Whereas FM uses a GC protocol with a grade range of 3, we give a notion of GC called *Proxcensus* that allows for an arbitrary grade range of k, and demonstrate protocols for k-grade Proxcensus for different corruption thresholds—the 'expand' step.

For the 'extract' step, we apply a different randomization method (as opposed to the FM coin) that increases the per-iteration probability of meeting the FM termination condition, based on the fact that we allow for a higher grade range for GC.

Put together, we obtain improvements for randomized protocols in various models, but, to avoid too many case distinctions, we focus on the most interesting cases in this thesis. These cases assume a setup for

threshold signatures among the parties. Protocol security is proven in the random-oracle model.

Concretely, we give the following protocols for expand and extract:

- For $t < n/3$, we demonstrate a (perfectly secure) protocol for k-grade Proxcensus requiring $O(\log k)$ communication rounds (for the 'expand' step). We then show how to achieve BA (with overwhelming probability) from a single instance of Proxcensus (with a grade range exponential in the security parameter) and a (single) multivalued coin toss (via the 'extract' step)—in contrast to the traditional FM approach in which *several instances* of GC (and coin tosses) are iterated. This yields a binary BA protocol involving $\kappa+1$ rounds in order to achieve a target error probability of $2^{-\kappa}$. The best known fixed-round binary BA protocol [MV17] for $t < n/3$ requires 2κ rounds to achieve the same target error probability. Both protocols can be extended to any finite input domain at the expense of 2 additional communication rounds.

- For $t < n/2$, we demonstrate (computationally secure) protocols for k-grade Proxcensus for the 'expand' step. One requires about $k/2$, and another (roughly) about $\sqrt{k}$ (yet much more complex than the $k/2$-round version) communication rounds. Then we show how to achieve BA from 3-grade Proxcensus, applying the same 'extract' step as above. Note, however, that for this case, we also have to reiterate this process as in the original FM protocol. This yields a binary BA protocol involving $3\kappa/2$ rounds to achieve a target error probability of $2^{-\kappa}$ while the best known previous protocol [MV17] for $t < n/2$ required 2κ rounds. Both protocols can be extended to any finite input domain at the expense of 3 additional communication rounds.

For completeness, as this may be of independent interest, in the appendix, we also give protocols for k-grade Proxcast (the respective single-sender version) computationally secure against $t < n$ requiring $k-1$ communication rounds—improving over the M-gradecast protocol by Garay et al. [GKKO07]—which easily adapts to the player-replaceable setting in [CM16] for $t < n/2$.

10.1.1 More on previous work

Protocols use different constructions for iterated randomization by use of a distributed-coin subprotocol with different security guarantees. To give a direct comparison between the basic BA constructions of prior art, we interpret all of them as using an idealized coin (zero error and no bias) requiring 1 round of communication and $O(n^2)$ message complexity in the number of parties. Such coin protocols exist (except for negligible error) under standard cryptographic assumptions and a random oracle [CKS05, LJY14, LM18].

It should be noted that in order to achieve the claimed round complexity improvements, our BA constructions require a coin that is ideal (except for negligible error). Exploring improvements in settings where the coin may fail with substantial probability is left for future work.

Chen and Micali [CM16] proposed an expected-9-rounds BA protocol for a binary input domain achieving computational security for $t < n/3$. Constant-round protocols with computational security for $t < n/2$ were given in [FG03, KK06] (where the former, in contrast to the latter, relies on specific computational-hardness assumptions). Micali and Vaikuntanathan [MV17] gave a $(2\kappa + 3)$-round computationally secure protocol with guaranteed termination and error probability $\varepsilon = 2^{-\kappa}$ (in an ideal model with unforgeable signatures and a random oracle). Recently, Abraham et al. [ADD$^+$19] gave an expected 16-round protocol for BA computationally secure against $t < n/2$ with expected message complexity $\Theta(n^2)$ improving over the prior expected 29-round protocol in [KK06] with expected message complexity $\Theta(n^3)$. Note that their gain in message complexity is due to the use of threshold signatures (in contrast to the protocol in [KK06]) which requires special setup assumptions. The construction of [ADD$^+$19] is based on the PBFT paradigm in [CL99] (who gave a partially synchronous protocol for state-machine replication safe against $t < n/3$ with guaranteed liveness in case of no corruptions). PBFT is a deterministic iteration-based protocol where, in contrast to the compact 2-round graded-consensus in the FM paradigm, each iteration consists of a non-uniform sequence of 5 communication rounds. The solution of [ADD$^+$19] is essentially achieved by applying the election of a random leader for each iteration, and by applying threshold signatures.

The concept of graded broadcast (and graded consensus) was generalized to grades larger than 2 in [CFF$^+$05], called *proxcast* – achieving

stronger security properties with growing grade ranges. Our BA protocols make use of this generalization.

In [GKKO07], a solution for a subclass of proxcast (called gradecast with multiple grades therein) for $t < n$ was given. We generalize their construction to general proxcast (by example of its consensus variant we call *Proxcensus*).

10.2 Model and Preliminaries

10.2.1 Communication and adversary model

We consider a synchronous communication network with authenticated point-to-point channels. We describe the protocols as proceeding in a series of *rounds*. A message sent by an honest party P_i at the beginning of a round is guaranteed to be delivered by the end of that round. We consider an adversary who can corrupt up to t parties in a malicious (a.k.a. Byzantine) way. That is, a corrupted party may deviate from the protocol arbitrarily.

We consider a *strongly rushing, adaptive adversary* who can corrupt parties at any given point of the protocol execution. In every round of the protocol, it can observe the messages that the honest parties sent before choosing its own messages for that round. It has the following additional capability: when it observes that an honest party P sends a message m during some round i, it can immediately corrupt that party and replace m with a message m' of its choice (in particular, it can decide to drop m).

10.2.2 Cryptographic primitives

Threshold signatures. We use a t-out-of-n threshold signature scheme consisting of a tuple $(\mathsf{DistKeyGen}, \mathsf{SignShare}, \mathsf{VerShare}, \mathsf{Ver}, \mathsf{Combine})$ of algorithms that behave as follows.

- Protocol $\mathsf{DistKeyGen}$ is an interactive key generation protocol among n parties. Parties take as input common public parameters, the security parameter κ, and the integers t and n, where $1 \leq t \leq n$. At the end of the protocol, all parties have as output the same

public key pk, and each P_i holds a secret key sk_i. Note that all existing protocols for key generation assume the existence of either a broadcast channel or a trusted dealer.

- Given a message m and a secret key sk_i, one can use the function $\mathsf{SignShare}_{\mathsf{sk}_i}(\cdot) := \mathsf{SignShare}(\mathsf{sk}_i, \cdot) : \{0,1\}^* \to \{0,1\}^\kappa$ to compute a signature share $\sigma_i = \mathsf{SignShare}_{\mathsf{sk}_i}(m)$.

- Given the public key pk, one can use the function $\mathsf{VerShare}^i_{\mathsf{pk}}(\cdot,\cdot) := \mathsf{VerShare}(\mathsf{pk}, i, \cdot, \cdot) : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}$ to verify the validity of a signature share σ_i; we say that a signature share σ_i for a message m is valid if and only if $\mathsf{VerShare}(\mathsf{pk}, \sigma_i, m) = 1$.

- Given a set S of valid signature shares from $t+1$ distinct parties, it is possible to compute a signature $\Sigma = \mathsf{Combine}(\{\sigma\}_{\sigma \in S})$.

- Given the public key pk, one can use the function $\mathsf{Ver}_{\mathsf{pk}}(\cdot,\cdot) := \mathsf{Ver}(\mathsf{pk}, \cdot, \cdot) : \{0,1\}^* \times \{0,1\}^\kappa \to \{0,1\}$ to verify the validity of a signature Σ; we say that a signature Σ for a message m is valid if and only if $\mathsf{Ver}(\mathsf{pk}, \Sigma, m) = 1$.

As is common in the literature, we treat (threshold) signatures as idealized objects: we require that for any given threshold t, signatures remain perfectly unforgeable for a message m, given t signature shares on m. In reality, one can instantiate the scheme accordingly using any scheme which is unforgeable under chosen-message attack and use a standard hybrid argument to achieve security against a computationally-bounded adversary. Moreover, we assume that all parties start the protocol after the setup phase has been completed, i.e., they agree on a public key pk and all hold secret keys with the properties described above.

Coin-Flip. We assume the availability of an atomic primitive $\mathsf{CoinFlip}$ which, on input k, returns a uniform value Coin_k (uniform in some range depending on the protocol of choice). Moreover, the value of Coin_k remains uniform from the view of the adversary until the first honest party has queried $\mathsf{CoinFlip}$ on input k. Such a primitive can easily be constructed from a threshold signature scheme with threshold $t+1$ and unique signatures per message and public key and assuming random oracle, such as the ones in [LJY14]. To obtain a uniform value on input k, parties simply sign the value k and send their signature share to all

parties. Parties can then hash the reconstructed signature on the value k into a suitable domain to obtain a random value. Unforgeability of the scheme ensures that until at least one honest party sends its share, the value of Coin_k remains uniform from the adversary's view. Uniqueness ensures that all honest parties obtain the same coin.

10.2.3 Byzantine Agreement and Proxcensus

We first recall the definition of Byzantine agreement (BA).

Definition 10.2.1. A protocol among parties $\mathcal{P}$ where every party P_i inputs a value $x_i \in \mathcal{D}$ from some finite domain $\mathcal{D}$, and, upon termination, every party $P_i \in \mathcal{P}$ outputs a value $y_i \in \mathcal{D}$, achieves *Byzantine agreement* iff the following conditions hold with overwhelming probability in κ:

Validity. If all honest parties P_i input the same value $x_i = x$ then they all output $y_i = x$.

Consistency. Any two honest parties P_i and P_j compute the same output, $y_i = y_j$.

Termination. All honest parties eventually terminate the protocol.

Degraded versions of BA that require weaker conditions than Definition 10.2.1 were introduced in [Dol82, FM97] as building-blocks for more powerful protocols. These versions can both be seen as accompanying the output value with an additional grade value from a small domain to express the degree of agreement achieved after the protocol execution. In [CFF+05], this approach was generalized to arbitrary finite domains along the following lines (yielding their definition of *Proxcast*, the single-sender version of the following definition):

Definition 10.2.2. Let $s \in \mathbb{N}$ and $G = \lfloor \frac{s-1}{2} \rfloor$. A protocol among parties $\mathcal{P}$ where every party $P_i \in \mathcal{P}$ inputs a value $x_i \in \mathcal{D}$ from some finite domain $\mathcal{D}$, and, upon termination, every party $P_i \in \mathcal{P}$ outputs a value $y_i \in \mathcal{D}$ and a grade $g_i \in [0, G]$, achieves *s-slot Proxcensus*, or Prox_s iff the following conditions hold:

Validity. If all honest parties input the same value $x \in \mathcal{D}$ then every honest party P_i outputs $y_i = x$ and $g_i = G$.

Consistency. For any two honest parties P_i and P_j:

- $|g_i - g_j| \leq 1$.
- If s is odd: $\min(g_i, g_j) \geq 1 \quad \Rightarrow \quad y_i = y_j$.
- If s is even: $\max(g_i, g_j) \geq 1 \quad \Rightarrow \quad y_i = y_j$.

Termination. All honest parties eventually terminate the protocol.

Note that, for even (odd) s, a grade of at least 1 (2) implies agreement detection with respect to the value y.

Prox_s (see Fig. 10.1) can be seen as a functionality wherein all parties output on one of s sequential slots such that all honest parties end up in two adjacent slots (brace (a) in Figure 10.1), and all honest parties decide on the maximally graded slot for their input value in case of pre-agreement (brace (b) in Figure 10.1 in case of pre-agreement on z').

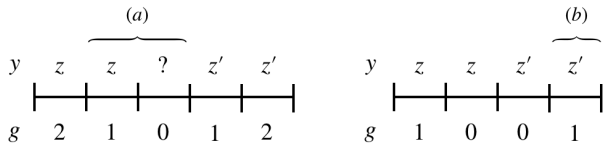

Figure 10.1: Proxcensus for odd s (left) and even s (right).

In particular, the well-known special cases mentioned above are Prox_3 known as *Crusader Agreement* [Dol82] and Prox_5 known as *graded consensus* [FM97] (originally defined in its single-sender broadcast version).

10.3 A Generalized Iteration Paradigm for Byzantine Agreement

10.3.1 Revisiting the Feldman-Micali Construction

We give a high-level review of their fixed-round protocol variant with respect to a binary input domain; assuming GC and the common coin as black boxes. Note that, in contrast of GC (Prox_5) for the expected-round case, Prox_3 is sufficient as a substitute for GC in the fixed-round case.

Each party P_i starts every iteration with their initial input value $x_i \in \{0,1\}$, and at the end of the iteration, overwrites x_i for use during the next iteration. Each protocol iteration consists of one execution of Prox_3 on the inputs x_i, yielding outputs $y_i, g_i \in \{0,1\}$; followed by a common coin $c \in \{0,1\}$. At the end of the iteration, a party P_i overwrites their input $x_i := y_i$ if $g_i = 1$ (keep the output of Prox_3), and $x := c$ if $g_i = 0$ (adopt the value of the coin). At the end of the k protocol iterations, each party P_i outputs x_i.

Validity: If all honest parties start an iteration (the protocol) with the same bit $x_i \equiv b$ then, by the validity of Prox_3, they all hold $x_i = y_i \equiv b$ and $g_i \equiv 1$ at the end of the iteration (the protocol).

Consistency: An iteration where an honest party outputs $g_i = 1$ and the coin yields $c = y_i$ (or all honest parties output $g_i = 0$) puts the honest parties into agreement. This event happens with probability at least $1/2$. Thus, by the validity argument above, the protocol fails to achieve BA with a probability of at most 2^{-k}.

10.3.2 Generalization

We propose a generalization of the Feldman-Micali iteration paradigm and show how it can be applied to achieve faster Byzantine agreement protocols. A generalized iteration with input x is composed of three components:

> **Expansion.** Execution of an s-slot Proxcensus: $(z, g) \leftarrow \mathsf{Prox}_s(x)$
>
> **Coin-Flip.** Execution of a multivalued coin-flip: $\mathsf{Coin} \leftarrow \mathsf{CoinFlip}$
>
> **Extraction.** A function f that takes as input (z, g, Coin) and outputs a value y, which is the output of the iteration.

Using this approach, we are able to substantially increase the probability of agreement per iteration round. For simplicity, we focus on protocols using ideal coins, meaning that the coin-flip component is an ideal 1-round multivalued coin-toss, which returns a uniform value with probability 1. Such a coin can be instantiated (tolerating a negligible failure probability) using unique threshold signatures in the random-oracle model [LJY14]. However, our techniques can similarly be applied with other coins.

In the following, we show two binary fixed-round Byzantine agreement protocols achieving a target error of $2^{-\kappa}$. The first tolerates $t < n/3$ corruptions, runs in $\kappa + 1$ rounds and uses a *single* coin-flip. The second protocol tolerates $t < n/2$ and runs in $\frac{3}{2}\kappa$ rounds. The protocols can be made multivalued with an additional cost of 2 (resp. 3) rounds for the case where $t < n/3$ (resp. $t < n/2$) [TC84].

10.3.3 Expansion

We show the two Proxcensus protocols used in the BA protocols. First, we show a Proxcensus that tolerates up to $t < n/3$ corruptions and achieves $2^r + 1$ slots in r rounds. Second, we show a protocol for $t < n/2$ that achieves $2r - 1$ slots in r rounds. Further protocols for Proxcensus up to $t < n/2$ (with quadratic number of slots w.r.t the number of rounds) and for proxcast up to $t < n$ (with linear number of slots) are shown in Section F.1 and F.2 for completeness.

Proxcensus for $t < n/3$

We show an expansion technique with unconditional security, which allows to expand a Proxcensus with s slots to a Proxcensus with $2s - 1$ slots in one additional round. Applying the expansion technique iteratively, we obtain a Proxcensus protocol with exponential slots in the number of rounds. The general idea is to run the Proxcensus protocol with s slots, $\mathsf{Prox_s}$, and echo the result. We know that all honest parties lie in two consecutive slots out of the s slots after $\mathsf{Prox_s}$. This implies that, after the echo, there will be $n - t$ values within two consecutive slots s_0 and s_1. Then, two consecutive slots accumulating $n - t$ votes constitute two consecutive slots in the new range, where the particular new slot is determined by which slot had $n - 2t$ values (in case of a tie, the upper slot is chosen). The parties then output the highest possible slot. See Figure 10.2 for an illustrative example.

Protocol $\mathsf{Prox}_{2s-1}(P_i)$

Let $G = \lfloor \frac{s-1}{2} \rfloor$ and $b = (s \mod 2)$. We describe the protocol from the point of view of party P_i with input x.

Run $\mathsf{Prox}_s(x)$. Let (z, h) denote the output value.
Send (z, h) to all parties. Denote as (z_j, h_j) the message received from party P_j.
Output Determination:
$y_i \coloneqq 0; g_i \coloneqq 0$
$S_0 \coloneqq \{j \colon h_j = 0\}$
$S_{z,g} \coloneqq \{j \colon z_j = z \ \wedge \ h_j = g\}$
if $b = 1 \wedge \exists z \colon |S_0 \cup S_{z,1}| \geq n - t \wedge |S_{z,1}| \geq n - 2t$ **then**
    Set $y_i \coloneqq z, g_i \coloneqq 1$
for $g = b$ to $G - 1$ **do**
    if $\exists z \colon |S_{z,g} \cup S_{z,g+1}| \geq n - t \wedge |S_{z,g+1}| \geq n - 2t$ **then**
        Set $y_i \coloneqq z, g_i \coloneqq 2g + 2 - b$
    else if $\exists z \colon |S_{z,g} \cup S_{z,g+1}| \geq n - t \wedge |S_{z,g}| \geq n - 2t$ **then**
        Set $y_i \coloneqq z, g_i \coloneqq 2g + 1 - b$
if $\exists z \colon |S_{z,G}| \geq n - t$ **then**
    Set $y_i \coloneqq z, g_i \coloneqq 2G + 1 - b = \lfloor \frac{2s-1}{2} \rfloor$
Output (y_i, g_i)

Lemma 10.3.1. *Let $s \geq 2$. Protocol Prox_{2s-1} satisfies validity.*

Proof. Suppose that all honest parties start with input v. Then, every honest party obtains (v, G) as output of Prox_s and send it to every party. As a result, every honest party has $|S_{v,G}| \geq n - t$ and sets $y_i = v$ and the maximal grade $g_i = 2G + 1 - b = \lfloor \frac{2s-1}{2} \rfloor$. □

Lemma 10.3.2. *Let $s \geq 2$. Protocol Prox_{2s-1} satisfies consistency.*

Proof. Let P_i be an honest party that outputs y_i with the maximal grade g_i among all honest parties.

We prove that $|g_i - g_j| \leq 1$. Consider the case where $g_i > 1$, as otherwise it is trivial. We divide three cases for P_i:

- $\exists z \colon |S_{z,g} \cup S_{z,g+1}| \geq n - t \wedge |S_{z,g}| \geq n - 2t$, where $g_i = 2g + 1 - b$. If $S_{z,g+1}$ contains an honest party, since g_i is maximal, all honest parties sent (z, g) or $(z, g + 1)$ after Prox_s. In this case, any honest P_j has $g_j = g_i$ because g_i is maximal. If all parties in $S_{z,g+1}$ are corrupted, this implies that there are at least $n - 2t$ honest parties in $S_{z,g}$ and all honest parties are in $S_{z,g-1} \cup S_{z,g}$. Hence, any honest P_j has $|S_{z,g-1} \cup S_{z,g}| \geq n - t \wedge |S_{z,g}| \geq n - 2t$, so $g_j \geq 2(g - 1) + 2 - b = 2g - b \geq g_i - 1$.

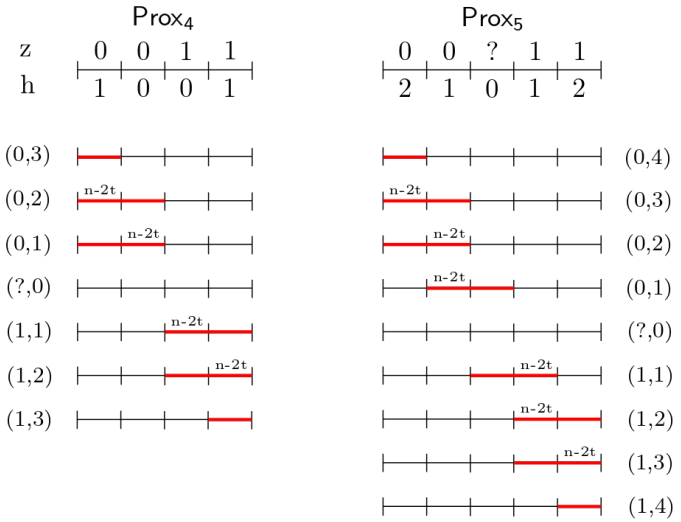

Figure 10.2: Example of the Proxcensus expansion starting from Prox_4 and Prox_5, respectively, for binary values. The expanded Proxcensus have 7 and 9 slots respectively. Each row indicates the condition to achieve each output written on the side. The red line indicates the region where it is required that $n - t$ echoed pairs are received, and the upper text "$n - 2t$" indicates that $n - 2t$ on that specific slot.

- $\exists z \colon |S_{z,g} \cup S_{z,g+1}| \geq n - t \wedge |S_{z,g+1}| \geq n - 2t$, where $g_i = 2g + 2 - b$. Here, any honest P_j also received $|S_{z,g} \cup S_{z,g+1}| \geq n - t$ as g_i is maximal, and either $|S_{z,g}| \geq n - 2t$ or $|S_{z,g+1}| \geq n - 2t$. Hence, $g_j \geq 2g + 1 - b = g_i - 1$.
- $\exists z \colon |S_{z,G}| \geq n - t$. Here, any honest P_j received $|S_{z,G-1} \cup S_{z,G}| \geq n - t$, as honest parties obtain adjacent grades from Prox_s, and moreover $|S_{z,G}| \geq n - 2t$. Hence, $g_j \geq 2(G-1) + 2 - b = 2G - b \geq g_i - 1$.

We prove that $\min(g_i, g_j) \geq 1 \Rightarrow y_i = y_j$. Note that the above argument also covers this statement if $\max(g_i, g_j) > 1$. Hence, we consider the case where $g_i = g_j = 1$. Towards a contradiction, assume that $y_i \neq y_j$. We divide two cases, depending on b:

$b = 1$**:** In this case, from P_i we have that : $|S_0 \cup S_{y_i,1}| \geq n - t \wedge |S_{y_i,1}| \geq n - 2t$. This means that there is an honest party P that had $(y_i, 1)$ as output of $\mathsf{Prox_s}$. Symmetrically for P_j, there is also an honest party P' that had $(y_j, 1)$ as output of $\mathsf{Prox_s}$. This immediately contradicts the consistency of $\mathsf{Prox_s}$.

$b = 0$**:** In this case, from P_i we have $|S_{y_i,0} \cup S_{y_i,1}| \geq n - t \wedge |S_{y_i,0}| \geq n-2t > t$. Symmetrically, from P_j we have $|S_{y_j,0} \cup S_{y_j,1}| \geq n-t \wedge |S_{y_j,0}| \geq n - 2t > t$. Since honest parties lie in adjacent slots, we know that all honest parties lie in $S_{y_i,0} \cup S_{y_j,0}$. Moreover, we know that there are $n-2t$ honest parties in $S_{y_i,0}$ and another $n - 2t$ honest parties in $S_{y_j,0}$. This leads to a contradiction, since $S_{y_i,0} \cup S_{y_i,1}$ contains $n - t$ parties and in addition $S_{y_j,0}$ contains $n-2t$ additional parties, which amount to a total of $n - t + n - 2t = 2n - 3t > n$ parties. □

As a result, we obtain a Proxcensus protocol for the case of $t < n/3$ corruptions that, given $\mathsf{Prox_s}$, in r additional rounds it achieves $\mathsf{Prox}_{2^r(\mathsf{s}-1)+1}$. Interpreting the parties' input configuration as the base case Prox_2 (setting $g_i \equiv 0$), we obtain the following corollary.

Corollary 10.3.3. *Let $t < n/3$. For any $r \geq 0$, Prox_{2^r+1} achieves Proxcensus with $2^r + 1$ slots and perfect security. The protocol runs in r rounds and has $O(rn^2)$ message complexity.*

Proxcensus for $t < n/2$

We introduce a Proxcensus protocol that runs in r rounds and achieves $2r-1$ slots. The protocol is similar to the proxcast in Section F.1 (adapted to the agreement case): each party signs its input and sends it to all parties. Then, each party tries to collect a threshold signature on a value, and upon receiving such a threshold signature, it forwards it to all parties. However, parties now send in addition an extra message ω at the beginning of round two indicating whether a threshold signature Σ was reconstructed in round one. At the end of round two, if $n - t$ such ω are received, one computes a threshold signature Ω that proves that there was an honest party which reconstructed Σ. By propagating Ω, we are able to increase the number of slots to $2r - 1$. The way to determine the output and grade is different: a party P_i sets its output to (y, g) if it has a threshold signature Σ on y at round $r - g$, does not have any

threshold signature on any $y' \neq y$ by round $g+1$, and obtained the proof Ω at round $r-g+1$. See Table 10.1 for an example.

	$1\|0$	$?\|0$	$?\|?$	$0\|?$	$0\|1$
Ω	$1\|0$	$1\|0$	$?\|?$	$0\|1$	$0\|1$ Ω
	$1\|0$	Ω $1\|?$	$?\|?$	$?\|1$ Ω	$0\|1$
(v,g):	(0,2)	(0,1)	($\perp$,0)	(1,1)	(1,2)

Table 10.1: Conditions for each slot in $\mathsf{Prox_5}$ for binary values. Row i indicates the condition to be satisfied at the end of round i. $b_0|b_1$, $b_0, b_1 \in \{0,1\}$, indicates whether a threshold signature Σ on 0 or 1 was received, and ? indicates that anything could happen. Ω indicates that a proof Ω was received.

Protocol $\mathtt{Prox}_{2r-1}(P)$

Setup: Parties make use of a unique $(n-t)$-out-n threshold signature scheme.

Party P starts with input v. Let $\mathcal{I}_k \subset [n]$ denote the parties that send correctly formed messages m in round k, i.e., where m is of the form $\{(x, \Sigma) | \mathtt{Verify}(\mathsf{pk}, \Sigma, x) = 1\}$. Initialize $\Omega^2 \coloneqq \perp$.

Round 1:
$\sigma \leftarrow \mathsf{SignShare}(sk, v)$.
Send (v, σ) to all parties. Denote as (v_i^1, σ_i^1) the message received from party P_i.
Set $S^1 \coloneqq \{(v, \Sigma) | \exists k_1, ..., k_{n-t} \colon \Sigma^1 \leftarrow \mathsf{Combine}(\sigma_{k_1}^1, \ldots, \sigma_{k_{n-t}}^1) \wedge \mathsf{Ver}(pk, \Sigma^1, v) = 1\}$.
if $S_i^1 = \{(v, \Sigma)\}$ **then**
  $\omega \leftarrow \mathsf{SignShare}(sk, v)$
Round 2:
Send S^1 and ω to all parties. Denote as S_i^1, ω_i the values received from party P_i
Set $S^2 \coloneqq \bigcup_{i \in \mathcal{I}_2} S_i^1$
if $\exists k_1, \ldots, k_{n-t} \colon \forall j \colon \mathsf{VerShare}(pk, \omega_{k_j}, v) = 1$ **then**
  $\Omega^2 \leftarrow \mathsf{Combine}(\omega_{k_1}, \ldots, \omega_{k_{n-t}})$
Rounds $j = 3$ **to** s:
Send S^{j-1}, Ω^{j-1} to all parties. Denote as $S_i^{j-1}, \Omega_i^{j-1}$ the values received from party P_i

Set $S^j \coloneqq \bigcup_{i \in \mathcal{I}_j} S_i^{j-1}$; $\Omega^j \coloneqq \bigcup_{i \in \mathcal{I}_j} \Omega_i^{j-1}$
Output Determination:
$y \coloneqq 0; g \coloneqq 0$
for $j = 1$ to $s - 1$ **do**
    if $\exists z, j \colon (z, \cdot) \in S^{s-j} \wedge \exists \Omega \in \Omega^{s-j+1} \colon \texttt{Verify}(pk, \Omega, z) = 1 \wedge \forall z' \neq z \colon (z', \cdot) \notin S^{j+1}$ **then**
        Set $y \coloneqq z, g \coloneqq j$
Output (y, g)

Lemma 10.3.4. *Let $t < n/2$ and $r \geq 3$. Assuming unique threshold signatures, $\texttt{Prox}_{2r-1}$ achieves a $(2r-1)$-slot Proxcensus in r rounds and $O(rn^2)$ message complexity.*

We prove validity and consistency in the following lemmas.

Lemma 10.3.5. *Let $r \geq 3$. Protocol $\texttt{Prox}_{2r-1}$ satisfies validity.*

Proof. Suppose that all honest parties start with input v. Observe that, there is no threshold signature computed on any value $v' \neq v$. In the first round, all honest parties send (v, σ^0) and so all honest parties hold $S^1 = \{(v, \Sigma)\}$ after the first round. Now, all honest parties compute a signature share ω on v and send it to all parties, together with S^1 in the second round. Therefore, honest parties will all hold $S^2 = \{(v, \Sigma)\}$ in round 2 and moreover are able to compute a threshold signature Ω^2 in that round. In each following round $j = 3$ to r, honest parties all send S^{j-1} and Ω, and so $S^j = \{(v, \Sigma)\}$. Therefore, all honest parties hold a threshold signature Ω^2 on v (that was computed in round two) and for all honest parties, $S^1 = S^2 = \cdots = S^r = \{(v, \cdot)\}$. Thus, all honest parties output v, as required. □

Lemma 10.3.6. *Let $r \geq 3$. Protocol $\texttt{Prox}_{2r-1}$ satisfies consistency.*

Proof. Let P_i be the honest party that outputs y_i the maximal grade g_i among all honest parties.

We prove that $|g_i - g_j| \leq 1$. Consider the case where $g_i > 1$, as otherwise the statement is trivial. From P_i, we know:

- $(y_i, \cdot) \in S^{r-g_i}$. This implies that any honest P_j has $(y_i, \cdot) \in S^{r-g_i+1}$.

- There is $\Omega \in \Omega^{r-g_i+1}$ such that $\texttt{Verify}(pk, \Omega, y_i) = 1$. This implies that any honest P_j received Ω and hence has stored $\Omega \in \Omega^{r-g_i+2}$.
- $\forall z' \neq y_i \colon (z', \cdot) \notin S^{g_i+1}$. This implies that any honest P_j has $\forall z' \neq y_i \colon (z', \cdot) \notin S^{g_i}$.

With the above facts, we see that any P_j has grade $g_j \geq g_i - 1$. Now we prove that $\min(g_i, g_j) \geq 1 \Rightarrow y_i = y_j$. Toward contradiction, assume that $y_i \neq y_j$. Since $g_i \geq 1$ (resp. $g_j \geq 1$), P_i (resp. P_j) obtained a threshold signature Ω on y_i (resp. y_j) at round $r - g_i + 1$ (resp. $r - g_j + 1$). This implies that there must be an honest party P that has sent a signature share ω in round 2. This implies, that for P, $y_i \in S^1$, which implies that also y_i in S^2 for P_j, which contradicts the requirement that $\forall z' \neq y_j \colon (z', \cdot) \notin S^2$.

□

10.3.4 Extraction

The extraction function can be interpreted pictorially as a *cut* that splits the slots in Proxcensus at the position indicated by the coin into two sides. If a party is placed at a position on the right (resp. left) side of the coin, it will decide on the output value 1 (resp. 0) (see Figure 10.3).

More formally, let s be the number of slots in Proxcensus, $G \coloneqq \lfloor \frac{s-1}{2} \rfloor$ be the maximal grade, and $r \coloneqq (s \mod 2)$ be the remainder modulo 2 of s. The extraction function takes as input a binary value $b \in \{0, 1\}$, a grade $g \in [0, G]$ and a coin value $c \in [1, s]$, and it outputs a binary value $f(b, g, c) \in \{0, 1\}$, defined as follows:

$$f(b, g, c) = \begin{cases} 1, & \text{if } (b = 1 \wedge c \leq g + G + 1 - r) \vee (b = 0 \wedge c \leq G - g) \\ 0, & \text{otherwise} \end{cases}$$

10.3.5 Efficient Fixed κ-Round Byzantine Agreement

We put the pieces together and show an efficient binary BA protocol. Using standard techniques [TC84], one can achieve a multivalued Byzantine agreement protocol with an additional cost of 2 (resp. 3) rounds when $t < n/3$ (resp. $t < n/2$).

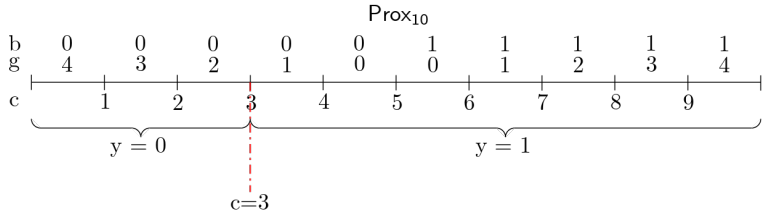

Figure 10.3: Extraction function applied to Prox_{10}. The coin takes values from $[1, 9]$. If the slot (value $b \in \{0, 1\}$ and grade $g \in \{0, 4\}$) lies in the left (resp. right) side of the coin value c, the function $f(b, g, c)$ outputs the value $y = 0$ (resp. $y = 1$).

The idea is to run the expansion component with one of the Proxcensus protocols Prox_s from Section 10.3.3, followed by a $(s-1)$-valued coin-flip, and the extraction function described in Section 10.3.4.

It is not hard to see that our approach allows to have an error per-iteration which is inversely proportional to the number of slots in Proxcensus. More precisely, since honest parties lie in two adjacent slots after the invocation of Prox_s, there is only one possible coin value (out of $s-1$) that lead to parties having different inputs.

The protocol is described from the point of view of party P_i and for a general Proxcensus protocol with s slots.

Protocol $\Pi^s_{\mathtt{iter}}(P_i)$

Let $G \coloneqq \lfloor \frac{s-1}{2} \rfloor$, and s be a positive number. Let f be the extraction function from Section 10.3.4. Let b denote the input bit.

1: $(b', g) \leftarrow \mathsf{Prox}_s(b)$
2: $c \leftarrow \mathsf{CoinFlip}$ // CoinFlip returns a uniform value in $[1, s-1]$
3: Output $f(b', g, c)$

Theorem 10.3.7. *Let $t < n$. $\Pi^s_{\mathtt{iter}}$ achieves binary Byzantine Agreement against an adaptive, strongly rushing adversary with probability $1 - \frac{1}{s-1}$. The protocol makes a single invocation to Prox_s and to a $(s-1)$-valued ideal Coin-Flip protocol.*

Proof. **Validity.** If all honest parties P_i input the same value b, then

$b' = b$ and $g = G$ by validity of Prox_s. Parties then output b because $c \in [1, s-1]$. More concretely, if $b' = 1$, then $c \leq 2G + 1 - j = s - 1$, so all parties output 1. And if $b' = 0$, then all parties output 0 because $c > 0$.

Consistency. Consistency of Prox_s guarantees that any two honest parties P_i and P_j lie on two consecutive slots. Parties only output different bits if the coin fails or the coin splits the two slots into different sides. Conversely, if the coin does not split the parties into different sides (which happens with probability $\frac{1}{s-1}$), then parties reach agreement. More concretely, we consider the following cases where honest parties lie on two consecutive slots (if all honest parties lie on the same slot, agreement is reached regardless of the coin value):

- s even: If there are honest parties that obtain $(0, 0)$ and $(1, 0)$, respectively, as output of Prox_s, then parties output different bits only if $c = G + 1$. Otherwise, assume that there are honest parties that output $(b', g-1)$ and (b', g), $1 \leq g \leq G$, respectively. If $b' = 1$ (resp. $b' = 0$), then parties output different bits only if $c = g+G+1$ (resp. $c = G - g + 1$).

- s odd: We only need to consider the case where honest parties output $(b', g-1)$ and (b', g), $1 \leq g \leq G$, respectively, since we assume that not all honest parties lie on the same slot. If $b' = 1$ (resp. $b' = 0$), then parties output different bits only if $c = g + G$ (resp. $c = G - g + 1$).

Termination. Obvious. □

We obtain the following corollary:

Corollary 10.3.8. *Assuming unique threshold signatures and a 1-round ideal Coin-Flip protocol, there are protocols that achieve binary Byzantine Agreement with probability $1 - 2^{-\kappa}$ secure against a strongly rushing adaptive adversary corrupting up to t parties, achieving the following:*

- *For $t < n/3$, it runs in $\kappa + 1$ rounds and has $O(\kappa n^2)$ message complexity. The protocol makes a single multivalued coin-flip invocation.*

- *For $t < n/2$, it runs in $\frac{3}{2}\kappa$ rounds, and has $O(\kappa n^2)$ message complexity.*

Proof. **Case $t < n/3$:** The statement follows from Theorem 10.3.7 and the use of a 1-round 1-fair ideal Coin-Flip protocol, setting $s = 2^\kappa + 1$, and using the Proxcensus protocol that achieves s slots in κ rounds and $O(n^2)$ message complexity from Corollary 10.3.3.

Case $t < n/2$: Security follows from Theorem 10.3.7 and the use of a 1-round 1-fair ideal Coin-Flip protocol, setting the number of slots to $s = 5$ and running the protocol $\Pi^s_{\texttt{iter}}$ sequentially $\frac{\kappa}{2}$ times. Each invocation to $\Pi^s_{\texttt{iter}}$ takes 3 rounds, where we run the 3-round Prox_5 protocol from Section 10.3.3, and the coin-flip in parallel to the third round of Prox_5. Note that after round 2 of Prox_5, the slot-pair where the honest parties lie is already fixed. The probability of not reaching agreement in each invocation to $\Pi^s_{\texttt{iter}}$ is $\frac{1}{4} = 2^{-2}$. Running the protocol sequentially $\frac{\kappa}{2}$ times therefore allows to achieve agreement except with probability $2^{-2 \cdot \frac{\kappa}{2}} = 2^{-\kappa}$. The total number of rounds is $\frac{3}{2}\kappa$. The claim on message complexity is inherited from the message complexity of the Proxcensus sub-protocol from Section 10.3.3. □

Efficiency Comparison with Micali and Vaikuntanathan [MV17]. We give a brief comparison with the most efficient, fixed-round protocol that we are aware of in the $n/3$ and $n/2$ regime. To the best of our knowledge, this is the protocol by Micali and Vaikuntanathan (MV) [MV17]. The binary version of their protocol requires 2κ rounds to achieve a termination error of $2^{-\kappa}$, and incurs a communication complexity of $O(\kappa n^3)$ complexity, even assuming threshold signatures.

Our protocol for $t < n/3$ requires only $\kappa+1$ rounds, with $O(\kappa n^2)$ message complexity. This means that we achieve the same error probability within roughly half the number of rounds, and save a factor of n in the message complexity.

Our protocol for $t < n/2$ regime requires $\frac{3}{2}\kappa$ rounds and $O(\kappa n^2)$ message complexity, which gives an improvement of about 25% in the round complexity, and a factor of n in the message complexity with respect to MV.

Both our protocols and MV can be extended to arbitrary finite domains with an additional cost of 2 (resp. 3) rounds when $t < n/3$ (resp. $t < n/2$) by applying the construction of Turpin and Coan [TC84].

Finally, in context of MV and the Turpin-Coan construction, we observe an additional advantage of carefully adjusting the slot range of Proxcensus. In their original model (standard signatures, player replaceability), the communication complexity of the MV protocol (for $t < n/2$) can be reduced by a factor of n by substituting their 3-round $\{0, 1, 2\}$-gradecast protocol by 3-round Prox_s^4, the single-sender version of Prox_4—see Appendix F.1.

Appendix F

Details of Chapter 10

F.1 Efficient Generic Proxcast for $t < n$

In [GKKO07], under the notion *M-gradecast*, it was demonstrated how to achieve s-round s-slot Proxcast for *odd* s secure against $t < n$. We extend their result to achieving $(s-1)$-round s-slot Proxcast for general $s \geq 2$, secure against $t < n$, using essentially the same construction.

Definition F.1.1. Let $s \in \mathbb{N}$ and $G \stackrel{\triangle}{=} \lfloor \frac{s-1}{2} \rfloor$. A protocol among parties $\mathcal{P}$ where a distinguished dealer (or sender) $P_d \in \mathcal{P}$ inputs a value $x_d \in \mathcal{D}$ from some finite domain $\mathcal{D}$, and, upon termination, every party $P_i \in \mathcal{P}$ outputs a value $y_i \in \mathcal{D}$ and a grade $g_i \in [0, G]$, achieves *s-slot proxcast, or* Prox_s^d (or Prox_s, in generic use) if and only if the following conditions hold:

Validity. If the dealer P_d is honest then every honest party P_i outputs $y_i = x_d$ and $g_j = G$.

Consistency. For any two honest parties P_i and P_j:

- $|g_i - g_j| \leq 1$.
- $\min(g_i, g_j) \geq 1 \quad \Rightarrow \quad y_i = y_j$.
- if $s = 2k$ $(k \in \mathbb{N})$ and $g_i > 0$ then $y_i = y_j$.

Proxcensus

Let $s = 2k + b$, $b \in \{0, 1\}$. The protocol is similar to Dolev-Strong broadcast with the difference that parties do not add their signatures. In the first round, the dealer signs his input and sends the signed message to every other player. For the next $k - 1$ rounds, the parties collect all validating message/signature pairs originating from the dealer. If, during any one of these rounds, a "new" valid message/signature pair is received then this pair is sent to all parties (but only up to the second time as the existence of two contradicting signed messages by the dealer is sufficient to detect the dealer's misbehavior). At the end, a player accepts a message with grade $g \in [0, G]$ if, at the end of any $2g + 1 - b$ consecutive rounds, the same unique message/signature pair from the dealer was seen; and on grade $g = 0$, otherwise.

Protocol $\Pi_{\texttt{Proxcast}}(P_d, P_i)$

Setup: Parties know the dealer's public key pk, and the dealer has the secret key sk as well.
Let $G := \lfloor \frac{s-1}{2} \rfloor$, and $s := 2k + b$ for $b \in \{0, 1\}$.
The dealer P_d starts with input x.

1: **Round** 1: Dealer P_d sends (x, σ), $\sigma = \mathsf{Sign}_{\mathsf{sk}}(x)$, to all P_j. Each party P_i sets $S_i^1 = \{(z, \sigma) \mid \mathsf{Ver}_{\mathsf{pk}}(z, \sigma) = 1\}$.
2: **for** $r = 2$ to $s - 1$ **do**
3: **Round** r: Party P_i sends S_i^{r-1}. Receive S_j^{r-1} from each party P_j, and let $S_i^r = \bigcup_j S_j^{r-1}$.
4: $y_i := 0$; $g_i := 0$;
5: **for** $g = 0$ to G **do**
6: **if** $\exists z, r : S_i^r = \cdots = S_i^{r+2g-b} = \{(z, \sigma)\}$ **then**
7: Set $y_i := z$, $g_i = g$.
8: P_i outputs (y_i, g_i).

Lemma F.1.2. *Let $t < n$. Assuming that the dealer has a public-key setup, $\Pi_{\texttt{Proxcast}}$ achieves a s-slot Proxcensus in $s - 1$ rounds and $O(sn^2)$ message complexity.*

Proof. **Validity.** If the dealer is honest, then each honest party P_i collects the same set $S_i^r = \{(x, \sigma)\}$ at the end of every round r.

Consistency. The cases $s \leq 3$ are trivial — thus consider $s > 3$.

- $|g_i - g_j| \leq 1$: Consider a party P_i with a maximal grade $g_i > 1$ among all honest parties (the case $g_i \leq 1$ is trivial). This means, there are $L = 2g+1-b > 2$ consecutive rounds such that $S_i^r = \cdots = S_i^{r+2g-b} = \{(z, \sigma)\}$. We claim that every honest party P_j sees at least $L-2 > 0$ such rounds, namely rounds $r+1, \ldots, r+2g-b-1$:
  - As P_i sees a unique (z, σ)-pair at round $r + 2g - b$, P_j cannot have seen a conflicting pair in any round before as, otherwise, he would have sent it to P_i.
  - As P_i sees an (z, σ)-pair at round r, P_j sees it at round $r + 1$ as P_i sent it to P_j.
- $\min(g_i, g_j) \geq 1 \quad \Rightarrow \quad y_i = y_j$: Assume that $g_i > 0$ for an honest party P_i. This implies a sequence of at least two rounds such that $S_i^r = S_i^{r+1} = \{(z, \sigma)\}$. As $S_j^r \subseteq S_i^{r+1}$, and the sets grow monotonically, it follows that there is no round r' such that $S_j^{r'} = \{(z', \sigma')\}$ with $z' \neq z$. Hence, $y_j = y_i$ or $g_j = 0$.
- If $s = 2k$ ($k \in \mathbb{N}$) and $g_i > 0$ then $y_i = y_j$: Assume an honest party P_i with $g_i = 1$ implying that there are $L = 2g + 1 - b = 2g + 1 \geq 3$ consecutive rounds such that $S_i^r = \cdots = S_i^{r+2g-b} = \{(z, \sigma)\}$. Thus, an honest party P_j sees at least one such round, and $y_j = y_i$ due to the monotone growth of the sets.

□

A player-replaceable variant for $t < n/2$. The above proxcast protocol for $t < n$ relies on the fact that a player seeing a signature relays it during the next round in order to make it public. With player replacement, this is not guaranteed anymore since the participating player set is now different during every round. However, this can be compensated for by lowering the threshold to $t < n/2$, and strengthening the grade-determination condition

$$\exists z, r : S_i^r = \cdots = S_i^{r+2g-b} = \{(z, \sigma)\}$$

with the additional requirement that each such S_i^r ($r > 1$) must have been forwarded by at least $n-t$ parties during round r; implicitly guaranteeing the global forwarding of such a signature already during the same round as at least one of these $n - t$ forwarding parties must be honest.

F.2 Quadratic Proxcensus for $t < n/2$

We introduce an improved version (for large r) of Proxcensus that runs in r rounds and achieves $3+(r-3)(r-2)$ slots. The protocol develops on the ideas from the previous Proxcensus protocols in Section 10.3.3, with some changes: instead of forwarding a signature only after the first round and propagating it, parties repeatedly create and send an additional signature ω_j at each round $j > 1$ indicating whether a threshold signature was reconstructed in the previous round. More precisely, the protocol proceeds as follows. Each party P_i sends a signature share at round 1 on their input value. If P_i collects $n-t$ signature shares on the same value v, P_i forms a threshold signature Ω_1 for v at the end of round 1. At round 2, if Ω_1 was formed only for v, P_i echoes Ω_1 and also sends a signature share ω_2 indicating that Ω_1 was formed only for v. If $n-t$ such ω_2 are received at the end of round 2, P_i computes a threshold signature Ω_2. In general, P_i sends (resp. echoes) each formed (resp. received) threshold signature, and in addition sends a signature share ω_j for v at round j if P_i formed a threshold signature Ω_{j-1} for v at the end of round $j-1$, and was not able to form any threshold signature Ω_k, $k \in [1, j-1]$, for any value $v' \neq v$.

By propagating all these additional signatures ω_j, we are able to increase the number of slots to $3+(r-3)(r-2)$, for $r \geq 3$. At the end of the protocol, P_i determines the output and grade checking a sequence of condition predicates. P_i evaluates a sequence of predicates, each indicating whether P_i received a certain threshold signature at a specific round. We denote $\mathsf{Condition}_{y,g,j}$ the predicate checking that a certain threshold signature needs to be formed or received at round j to output a value y with grade g. Moreover, we denote $\mathsf{Condition}_{y,g}$ the set of all conditions that need to be satisfied to output value y with grade g, over all rounds.

The condition predicates are defined inductively, starting from the highest grade (see Table F.1 for a concrete example):

- Let $G = 1 + \frac{(j-3)(j-2)}{2}$. Then, $\mathsf{Condition}_{y,G,j}$ indicates whether P_i formed the threshold signature Ω_j for value y at round j.

- $\mathsf{Condition}_{y,g,j}$, $0 < g < G$, is inductively derived as follows: P_i formed or received a threshold signature Σ for value y by the end of round j, where $\Sigma = \Omega_{j-1}$ if there is a predicate $\mathsf{Condition}_{y,g+1,j'}$,

$j' > r$, indicating that Ω_j is obtained for value y by round j'. Otherwise, Σ is the threshold signature that is obtained according to $\mathsf{Condition}_{y,g+1,j-1}$.

- $\mathsf{Condition}_{y,0,j}$ is always true.

Intuitively, if an honest party satisfies $\mathsf{Condition}_{y,g}$, $g \geq 1$, then every honest party satisfies $\mathsf{Condition}_{y,g-1}$ for two reasons: 1) honest parties forward each threshold signature that they receive or form, and 2) the existence of a threshold signature Ω_j, $j > 1$, implies that an honest party P_k sent ω_j at the beginning of round j, meaning that P_k obtained Ω_{j-1} at the end of round $j-1$. This P_k therefore sent Ω_{j-1} at the beginning of round j, and every honest party received Ω_{j-1} by the end of round j.

Moreover, the conditions are designed such that any $\mathsf{Condition}_{y,g}$, $g \geq 1$, requires that the threshold signature Ω_3 is obtained in some round. This guarantees that the conditions $\mathsf{Condition}_{y,1}$ and $\mathsf{Condition}_{y',1}$ are mutually disjoint, for $y \neq y'$. To see this, suppose P_i outputs $(y, 1)$, and thereby received Ω_3 at the last round. Note that this condition implies that there is an honest P_k that obtained Ω_2 for y and did not receive Ω_1 for any other value y' by round 2. This implies that no honest party received Ω_1 for y' by round 1, and therefore no honest party can output $(y', 1)$.

	Ω_1	?	?	?	?	?	?	?	?	?	?	?	?	?	Ω_1
	Ω_2	Ω_1	Ω_1	Ω_1	Ω_1	Ω_1	Ω_1	?	Ω_1	Ω_1	Ω_1	Ω_1	Ω_1	Ω_1	Ω_2
	Ω_3	Ω_2	Ω_2	Ω_2	Ω_2	Ω_2	Ω_2	?	Ω_2	Ω_2	Ω_2	Ω_2	Ω_2	Ω_2	Ω_3
	Ω_4	Ω_3	Ω_3	Ω_3	Ω_3	Ω_2	Ω_2	?	Ω_2	Ω_2	Ω_3	Ω_3	Ω_3	Ω_3	Ω_4
	Ω_5	Ω_4	Ω_4	Ω_3	Ω_3	Ω_3	Ω_2	?	Ω_2	Ω_3	Ω_3	Ω_3	Ω_4	Ω_4	Ω_5
	Ω_6	Ω_5	Ω_4	Ω_4	Ω_3	Ω_3	Ω_3	?	Ω_3	Ω_3	Ω_3	Ω_4	Ω_4	Ω_5	Ω_6
(v,g):	(0,7)	(0,6)	(0,5)	(0,4)	(0,3)	(0,2)	(0,1)	($\perp$,0)	(1,1)	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)

Table F.1: Conditions for each slot in Prox_{15} for binary values. Row i indicates the condition to be satisfied at the end of round i. To output the pair (v, g), all conditions in the column for that pair need to be satisfied, where Ω_r at row i indicates that a threshold signature Ω_r was received at round i, and ? indicates that there is no requirement. Note that Ω_3 is required at some position for every grade $g > 0$.

Protocol $\texttt{Prox}_{3+(r-3)(r-2)}(P_i)$

Setup: Parties make use of a unique $(n-t)$-out-n threshold signature scheme.

Party P_i starts with input v.

1: **Round** 1:
2: $\sigma \leftarrow \mathsf{SignShare}(sk, v)$.
3: Send (v, σ) to all parties. Denote as (v^i, σ^i) the message received from party P_i.
4: Set $S^1 \coloneqq \{(v, \Sigma) | \exists k_1, ..., k_{n-t} \colon \Sigma \leftarrow \mathsf{Combine}(\sigma^{k_1}, \ldots, \sigma^{k_{n-t}}) \wedge \mathsf{Ver}(pk, \Sigma, v) = 1\}$.
5: Set $T \coloneqq \varnothing$;
6: **Rounds** $j = 2$ **to** r:
7: **if** $S^{j-1} = \{(v, \Omega_{j-1})\} \ \wedge \ \forall v' \neq v \ \forall \ell < j \ (v', \Omega_\ell) \notin T$ **then**
8: $\quad \omega_j \leftarrow \mathsf{SignShare}(sk, (v, j-1))$
9: Forward all new pairs of value and threshold signature received or formed in the previous round. Moreover, if a signature share ω_j was computed on a value v, also send (v, ω_j) to all parties. Denote as (v^i, ω_j^i) the message received from party P_i.
10: Set $S^j \coloneqq \{(v, \Sigma) | \exists k_1, ..., k_{n-t} \colon \Sigma \leftarrow \mathsf{Combine}(\omega_j^{k_1}, \ldots, \omega_j^{k_{n-t}}) \wedge \mathsf{Ver}(pk, \Sigma, v) = 1\}$.
11: Add to T all newly formed or received threshold signature schemes (with the corresponding value).
12: **Output Determination:**
13: Output (y, g) with the highest grade such that $\mathsf{Condition}_{y,g}$ is satisfied.

Lemma F.2.1. *Let $t < n/2$ and $r \geq 3$. Assuming unique threshold signatures, $\texttt{Prox}_{3+(r-3)(r-2)}$ achieves a $(3+(r-3)(r-2))$-slot Proxcensus in r rounds and $O(rn^2)$ message complexity.*

We prove validity and consistency in the following lemmas.

Lemma F.2.2. *Let $r \geq 3$. Protocol $\texttt{Prox}_{3+(r-3)(r-2)}$ satisfies validity.*

Proof. Suppose that all honest parties start with input v. Thus, all honest parties send a signature share on v in the first round, and so all honest parties hold a $S^1 = \{v, \Omega_1\}$ after the first round. Note that since no honest party ever signs a signature share on any other value $v' \neq v$, at each round $j \in [2, r]$, all honest parties compute a signature share ω_j on v and send it to all parties, and all honest parties compute a threshold

signature Ω_j by the end of round j. Thus, $\mathsf{Condition}_{y,G}$ is satisfied and all honest parties output v. □

Lemma F.2.3. *Let $r \geq 3$. Protocol $\mathtt{Prox}_{3+(\mathtt{r}-3)(\mathtt{r}-2)}$ satisfies consistency.*

Proof. We first prove that any two honest parties P_i and P_ℓ output grades g_i and g_ℓ with $|g_i - g_\ell| \leq 1$.

Let P_i be the honest party that outputs the maximal grade g_i among all honest parties. If $g_i \geq 1$, then trivially $|g_i - g_\ell| \leq 1$. Hence, suppose that P_i outputs (v, g_i), $g_i > 1$. This implies that P_i satisfies $\mathsf{Condition}_{y,g_i}$. We show that any honest party P_j satisfies $\mathsf{Condition}_{y,g_i-1}$. Let $j \in [2, r]$. We show that P_ℓ satisfies $\mathsf{Condition}_{y,g_i-1,j}$. There are two cases: 1) P_i obtained Ω_j at some round $j' > j$, then there is an honest party P_k that sent ω_j at the beginning of round j. This means that P_k obtained Ω_{j-1} at the end of round $j-1$, and therefore sent Ω_{j-1} at the beginning of round j, and every honest party received Ω_{j-1} by the end of round j; 2) P_i did not obtain such Ω_j, in which case every honest party satisfies $\mathsf{Condition}_{y,g_i-1,j}$ by the fact that P_i echoes all formed threshold signatures.

Now we prove that if $g_i \geq 1$ and $g_\ell \geq 1$, then the parties output the same value, i.e. $y_i = y_\ell$.

This follows from the fact that the conditions are designed in such a way that any condition $\mathsf{Condition}_{y,g}$, $g \geq 1$, requires that the threshold signature Ω_3 is obtained in some round for the corresponding value. That is, $\mathsf{Condition}_{y_i,g_i}$ (resp. $\mathsf{Condition}_{y_\ell,g_\ell}$) requires obtaining Ω_3 for value y_i (resp. y_ℓ). We show that both conditions cannot be simultaneously satisfied. From $\mathsf{Condition}_{y_i,g_i}$, we know that there is an honest P_k that obtained Ω_2 for y_i and did not receive Ω_1 for y_ℓ by round 2. This implies that no honest party received Ω_1 for y_ℓ by round 1, and therefore no honest party created a signature share ω_2 for $y_j\ell$. As a result, Ω_2 (and hence also Ω_3) cannot be computed for y_ℓ, and no honest party can satisfy $\mathsf{Condition}_{y_\ell,g_\ell}$. □

Chapter 11

Asynchronous Byzantine Agreement with Subquadratic Communication

11.1 Introduction

Byzantine agreement (BA) [LSP82] is a fundamental problem in distributed computing. In this context, n parties wish to agree on a common output even when f of those parties might be adaptively corrupted. Although BA is a well-studied problem, it has recently received increased attention due to its application to blockchain (aka state machine replication) protocols. Such applications typically involve a large number of parties, and it is therefore critical to understand how the communication complexity of BA scales with n. While protocols with adaptive security and $o(n^2)$ communication complexity have been obtained in both the synchronous [KS10] and partially synchronous [ACD+19] settings, there are currently no such solutions for the *asynchronous* model.[1] This leads us to ask:

[1] Tolerating $f < n/3$ *static* corruptions is easy; see Section 11.1.1.

Is it possible to design an asynchronous BA protocol with subquadratic communication complexity that tolerates $\Theta(n)$ adaptive corruptions?

We positively answer this question.

We show asynchronous BA protocols with (expected) subquadratic communication complexity that can tolerate adaptive corruption of any $f < (1-\epsilon)n/3$ of the parties, for arbitrary constant $\epsilon > 0$. (This corruption threshold is almost optimal, as it is known [Bra87] that asynchronous BA is impossible altogether for $f \geq n/3$, even assuming prior setup and static corruptions.) Our solutions rely on two building blocks, each of independent interest:

1. We show a BA protocol $\Pi_{\mathtt{BA}}$ tolerating f adaptive corruptions and having subquadratic communication complexity. This protocol assumes prior setup by a trusted dealer for each BA execution, but the size of the setup is independent of n.

2. We construct a secure-computation protocol Π_{MPC} tolerating up to f adaptive corruptions, and relying on a subquadratic BA protocol as a subroutine. For the special case of no-input functionalities, the number of BA executions depends only on the security parameter, and the communication complexity is subquadratic when the output length is independent of n.

We can combine these results to give an affirmative answer to the original question. Specifically, using a trusted dealer, we can achieve an *unbounded* number of BA executions with $o(n^2)$ communication per execution. The idea is as follows. Let L be the number of BA executions required by Π_{MPC} for computing a no-input functionality. The dealer provides the parties with the setup needed for $L+1$ executions of $\Pi_{\mathtt{BA}}$; the total size of this setup is linear in L but independent of n. Then, each time the parties wish to carry out Byzantine agreement, they will use one instance of their setup to run $\Pi_{\mathtt{BA}}$, and use the remaining L instances to refresh their initial setup by running Π_{MPC} to simulate the dealer. Since the size of the setup for $\Pi_{\mathtt{BA}}$ is independent of n, the total communication complexity is subquadratic in n.

Alternately, we can avoid a trusted dealer (though we do still need to assume a PKI) by having the parties run an arbitrary adaptively secure protocol to generate the initial setup. This protocol may not have

subquadratic communication complexity; however, once it is finished the parties can revert to the solution above which has subquadratic communication per BA execution. Overall, this gives BA with *amortized* subquadratic communication.

11.1.1 Related Work

The problem of BA was initially introduced by Lamport, Shostak and Pease [LSP82]. Without some form of setup, BA is impossible (even in a synchronous network) when $f \geq n/3$. Fischer, Lynch, and Patterson [FLP85] ruled out deterministic protocols for asynchronous BA even when $f = 1$. Starting with the work of Rabin [Rab83], randomized protocols for asynchronous BA have been studied in both the setup-free setting [CR93, MHR14] as well as the setting with a PKI and a trusted dealer [CKS00].

Dolev and Reischuk [DR85] show that any BA protocol achieving subquadratic communication complexity (even in the synchronous setting) must be randomized. BA with subquadratic communication complexity was first studied in the synchronous model by King et al., who gave setup-free *almost-everywhere* BA protocols with polylogarithmic communication complexity for the case of $f < (1 - \epsilon)n/3$ static corruptions [KSSV06] and BA with $O(n^{1.5})$ communication complexity for the same number of adaptive corruptions [KS10]. Subsequently, several works [Mic17, MV17, PS17b, ACD+19, GPS19] gave improved protocols with subquadratic communication complexity (in the synchronous model with an adaptive adversary) using the "player replaceability paradigm," which requires setup in the form of verifiable random functions.

Abraham et al. [ACD+19] show a BA protocol with adaptive security and subquadratic communication complexity in the *partially synchronous* model. They also give a version of the Dolev-Reischuk bound that rules out subquadratic BA (even with setup, and even in the synchronous communication model) against a strong adversary who is allowed to remove messages sent by honest parties from the network after those parties have been adaptively corrupted. One can obtain a similar lower bound adapting their ideas to the standard asynchronous model where honest parties' messages can be arbitrarily delayed, but cannot deleted once they are sent (see e.g.[BKLL20, Ram20]). We refer to the work of Garay et al. [GKKZ11b] for further discussion of these two models.

Cohen et al. [CKS20] show an adaptively-secure asynchronous BA pro-

tocol with $o(n^2)$ communication. However, they consider a non-standard asynchronous model in which the adversary cannot arbitrarily schedule delivery of messages. In particular, the adversary in their model cannot reorder messages sent by honest parties in the same protocol step. We work in the standard asynchronous model. On the other hand, our work requires stronger computational assumptions and a trusted dealer (unless we settle for amortized subquadratic communication complexity).

We remark for completeness that asynchronous BA with subquadratic communication complexity for a *static* adversary corrupting $f < n/3$ of the parties is trivial using a committee-based approach, assuming a trusted dealer. Roughly, the dealer chooses a random committee of $\Theta(\kappa)$ parties (where κ is a security parameter) who then run BA on behalf of everyone. Achieving subquadratic BA *without* any setup in the static-corruption model is an interesting open question.

Asynchronous secure multi-party computation (MPC) was first studied by Ben-Or, Canetti and Goldreich [BCG93]. Since then, improved protocols have been proposed with both unconditional [SR00, PSR02, PCR08] and computational [HNP05, HNP08, CP15, Coh16] security. All these protocols achieve optimal output quality, and incur a total communication complexity of at least $\Theta(n^3\kappa)$ assuming the output has length κ. Our MPC protocol gives a trade-off between the communication complexity and the output quality. In particular, we achieve subquadratic communication complexity when the desired output quality is sublinear (as in the case of no-input, randomized functions).

11.1.2 Overview

In Section 11.2 we discuss our model and recall some standard definitions. We show how to achieve asynchronous reliable consensus and reliable broadcast with subquadratic communication in Section 11.3. In Section 11.4 we present an asynchronous BA protocol with subquadratic communication complexity, assuming prior setup by a trusted dealer for each execution. In Section 11.5 we show a communication-efficient asynchronous protocol for secure multi-party computation (MPC). We describe how these components can be combined to give our main results in Section 11.6.

11.2 Preliminaries and Definitions

We denote the security parameter by κ, and assume $\kappa < n = \mathsf{poly}(\kappa)$. In all our protocols, we implicitly assume parties take 1^κ as input; in our definitions, we implicitly allow properties to fail with probability negligible in κ. We let PPT stand for probabilistic polynomial time. We use standard digital signatures, where a signature on a message m using secret key sk is computed as $\sigma \leftarrow \mathsf{Sign}_{\mathsf{sk}}(m)$; a signature is verified relative to public key pk by calling $\mathsf{Vrfy}_{\mathsf{pk}}(m, \sigma)$. For simplicity, we assume in our proofs that the adversary cannot forge valid signatures on behalf of honest parties. When replacing the signatures with real-world instantiations, our theorems follow except with an additive negligible failure probability.

Model. We consider a setting where n parties $P_1, \ldots, P_n$ run a distributed protocol over a network in which all parties are connected via pairwise authenticated channels. We work in the *asynchronous* model, meaning the adversary can arbitrarily schedule the delivery of all messages, so long as all messages are eventually delivered. We consider an *adaptive* adversary that can corrupt some bounded number f of the parties at any point during the execution of some protocol, and cause them to deviate arbitrarily from the protocol specification. However, we assume the "atomic send" model, which means that (1) if at some point in the protocol an honest party is instructed to send several messages (possibly to different parties) simultaneously, then the adversary can corrupt that party either before or after it sends all those messages, but not in the midst of sending those messages; and (2) once an honest party sends a message, that message is guaranteed to be delivered eventually even if that party is later corrupted. In addition, we assume secure erasure.

In many cases we assume an incorruptible dealer who can initialize the parties with setup information in advance of any protocol execution. Such setup may include both public information given to all parties, as well as private information given to specific parties; when we refer to the size of a setup, we include the total private information given to all parties but count the public information only once. A public key infrastructure (PKI) is one particular setup, in which all parties hold the same vector of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and each honest party P_i holds the honestly generated secret key sk_i corresponding to pk_i.

Byzantine agreement. We include here the standard definition of

Byzantine agreement. Definitions of other primitives are given in the relevant sections.

Definition 11.2.1. (Byzantine agreement) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input v_i and parties terminate upon generating output. Π is an f-secure Byzantine agreement protocol if the following hold when at most f parties are corrupted:

- **Validity:** if every honest party has the same input value v, then every honest party outputs v.
- **Consistency:** all honest parties output the same value.

11.3 Building Blocks

In this section we show asynchronous protocols with subquadratic communication for reliable consensus, reliable broadcast, graded consensus, and coin flipping.

11.3.1 Reliable Consensus

Reliable consensus is a weaker version of Byzantine agreement where termination is not required. The definition follows.

Definition 11.3.1. (Reliable consensus) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input v_i and parties terminate upon generating output. Π is an f-secure reliable consensus protocol if the following hold when at most f parties are corrupted:

- **Validity:** if every honest party has the same input value v, then every honest party outputs v.
- **Consistency:** either no honest party terminates, or all honest parties output the same value.

We show a reliable consensus protocol Π_{RC} with subquadratic communication. The protocol can be viewed as a variant of Bracha's reliable broadcast protocol [Bra87, BT85] for the case where every party has input. The protocol assumes prior setup initialized by a trusted dealer.

The trusted setup has expected size $O(\kappa^2)$ and takes the following form. First, the dealer selects two secret committees C_1, C_2 by independently placing each party in C_1 (resp., C_2) with probability κ/n. Then, for each party P_i in C_1 (resp., C_2), the dealer generates a public/private key pair $(\mathsf{pk}_{1,i}, \mathsf{sk}_{1,i})$ (resp., $(\mathsf{pk}_{2,i}, \mathsf{sk}_{2,i})$) for a digital signature scheme and gives the associated private key to P_i; the public keys (but not the identities of the members of the committees) are given to all parties.

The protocol itself is described below. It begins by having each party in C_1 send its signed input to all the parties. The parties in C_2 then send a signed `ready` message on a value v the first time they either (1) receive v from $\kappa - t$ parties in C_1 or (2) receive `ready` messages on v from $t + 1$ parties in C_2. All parties terminate upon receiving `ready` messages on the same value from $\kappa - t$ parties in C_2. Each committee has expected size $O(\kappa)$, and each member of a committee sends a single message to all parties; thus, $O(\kappa n)$ messages are sent (in expectation) during the protocol.

Security relies on the fact that an adversary cannot corrupt too many members of C_1 (resp., C_2) "until it is too late," except with negligible probability. For a static adversary this is immediate. For an adaptive adversary this follows from the fact that each member of a committee sends only a single message and erases its signing key after sending that message; thus, once the attacker learns that some party is in a committee, adaptively corrupting that party is useless.

Protocol $\Pi_{\mathsf{RC}}(P_i)$

Let ϵ be a constant parameter. We describe the protocol from the point of view of a party P_i with input v_i, assuming the setup described in the text. Set $t = (1 - \epsilon) \cdot \kappa/3$.

1: If $P_i \in C_1$: Compute $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}_{1,i}}(v_i)$, erase $\mathsf{sk}_{1,i}$, and send $(\mathtt{echo}, (i, v_i, \sigma_i))$ to all parties.

2: If $P_i \in C_2$: As long as no `ready` message has yet been sent, do: upon receiving $(\mathtt{echo}, (j, v, \sigma_j))$ with $\mathsf{Ver}_{\mathsf{pk}_{1,j}}(v, \sigma_j) = 1$ on the same value v from at least $\kappa - t$ distinct parties, or receiving $(\mathtt{ready}, (j, v, \sigma_j))$ with $\mathsf{Ver}_{\mathsf{pk}_{2,j}}(v, \sigma_j) = 1$ on the same value v from strictly more than t distinct parties, compute $\sigma_i \leftarrow \mathsf{Sign}_{\mathsf{sk}_{2,i}}(v)$, erase $\mathsf{sk}_{2,i}$, and send $(\mathtt{ready}, (i, v, \sigma_i))$ to all parties.

3: Upon receiving $(\texttt{ready}, (j, v, \sigma_j))$ with $\mathsf{Ver}_{\mathsf{pk}_{2,j}}(v, \sigma_j) = 1$ on the same value v from at least $\kappa - t$ distinct parties and, output v and terminate.

Theorem 11.3.2. *Let $0 < \epsilon < 1/3$ and $f \leq (1 - 2\epsilon) \cdot n/3$. Then Π_{RC} is an f-secure reliable consensus protocol with expected setup size $O(\kappa^2)$ and expected communication complexity $O((\kappa + \mathcal{I}) \cdot \kappa n)$, where $\mathcal{I}$ is the size of each party's input.*

Proof. Recall that $t = (1-\epsilon)\cdot\kappa/3$. Say a party is *1-honest* if it is in C_1 and is not corrupted when executing step 1 of the protocol, and *1-corrupted* if it is in C_1 but corrupted when executing step 1 of the protocol. Define 2-honest and 2-corrupted analogously. Lemma G.1.3 shows that with overwhelming probability C_1 (resp., C_2) contains fewer than $(1 + \epsilon) \cdot \kappa$ parties; there are more than $\kappa - t$ parties who are 1-honest (resp., 2-honest); and there are fewer than $t < \kappa - t$ parties who are 1-corrupted (resp., 2-corrupted). For the rest of the proof we assume these hold. We also use the fact that once a 1-honest (resp., 2-honest) party P sends a message, that message is the only such message that will be accepted by honest parties on behalf of P (even if P is adaptively corrupted after sending that message).

We first prove that Π_{RC} is f-valid. Assume all honest parties start with the same input v. Each of the parties that is 1-honest sends an `echo` message on v to all other parties, and so every honest party eventually receives valid `echo` messages on v from more than $\kappa - t$ distinct parties. Since there are fewer than $\kappa - t$ parties that are 1-corrupted, no honest party receives valid `echo` messages on $v' \neq v$ from $\kappa - t$ or more distinct parties. It follows that every 2-honest party sends a `ready` message on v to all other parties. A similar argument then shows that all honest parties output v and terminate.

Toward showing consistency, we first argue that if honest P_i, P_j send `ready` messages on v_i, v_j, respectively, then $v_i = v_j$. Assume this is not the case, and let P_i, P_j be the first honest parties to send `ready` messages on distinct values v_i, v_j. Then P_i (resp., P_j) must have received at least $\kappa - t$ valid `ready` messages on v_i (resp., v_j). But then at least

$$(\kappa - t) + (\kappa - t) = (1 + \epsilon) \cdot \kappa + t$$

valid `ready` messages were received by P_i, P_j overall. But this is impossible, since the maximum number of such messages is at most $|C_2|$ plus

the number of 2-corrupted parties (because 2-honest parties send at most one `ready` message), which is strictly less than $(1+\epsilon)\cdot\kappa + t$.

Now, assume an honest party P_i outputs v. Then P_i must have received valid `ready` messages on v from at least $\kappa - t$ distinct parties in C_2, more than $\kappa - 2t > t$ of whom are 2-honest. As a consequence, all 2-honest parties eventually receive valid `ready` messages on v from more than t parties, and so all 2-honest parties eventually send a `ready` message on v. Thus, all honest parties eventually receive valid `ready` messages on v from at least $\kappa - t$ parties, and so output v also. □

11.3.2 Reliable Broadcast

Reliable broadcast allows a sender to consistently distribute a message to a set of parties. In contrast to full-fledged broadcast (and by analogy to reliable consensus), reliable broadcast does not require termination.

Definition 11.3.3. (Reliable broadcast) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where a designated sender P^* initially holds input v^*, and parties terminate upon generating output. Π is an f-secure reliable broadcast protocol if the following hold when at most f parties are corrupted:

- **Validity:** if P^* is honest at the start of the protocol, then every honest party outputs v^*.
- **Consistency:** either no honest party terminates, or all honest parties output the same value.

It is easy to obtain a reliable broadcast protocol Π_{RBC} from reliable consensus: the sender P^* simply signs its message and sends it to all parties, who then run reliable consensus on what they received. In addition to the setup for the underlying reliable consensus protocol, Π_{RBC} assumes P^* has a public/private key pair $(\mathsf{pk}^*, \mathsf{sk}^*)$ with pk^* known to all other parties.

Protocol Π_{RBC}

Let ϵ be a constant parameter.

1: P^* does: compute $\sigma^* \leftarrow \mathsf{Sign}_{\mathsf{sk}^*}(v^*)$, erase sk^*, and send (v^*, σ^*) to all parties.
2: Upon receiving (v^*, σ^*) with $\mathsf{Ver}_{\mathsf{pk}^*}(v, \sigma) = 1$, input v to Π_{RC} (with parameter ϵ).
3: Upon receiving output v from Π_{RC}, output v and terminate.

Theorem 11.3.4. *Let $0 < \epsilon < 1/3$ and $f \leq (1-2\epsilon) \cdot n/3$. Then Π_{RBC} is an f-secure reliable broadcast protocol with expected setup size $O(\kappa^2)$ and expected communication complexity $O((\kappa + \mathcal{I}) \cdot \kappa n)$, where $\mathcal{I}$ is the size of the sender's input.*

Proof. Consistency follows from consistency of Π_{RC}. As for validity, if P^* is honest at the outset of the protocol then P^* sends (v^*, σ^*) to all parties in step 1; even if P^* is subsequently corrupted, that is the only valid message from P^* that other parties will receive. As a result, every honest party runs Π_{RC} using input v, and validity of Π_{RC} implies validity of Π_{RBC}. □

11.3.3 Graded Consensus

Graded consensus [FM88] can be viewed as a weaker form of consensus where parties output a grade along with a value, and agreement is required to hold only if some honest party outputs a grade of 1. Our definition does not require termination upon generating output.

Definition 11.3.5. (Graded consensus) Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input v_i and is supposed to output a value w_i along with a grade $g_i \in \{0, 1\}$. Π is an f-secure graded-consensus protocol if the following hold when at most f parties are corrupted:

- **Graded validity:** if every honest party has the same input value v, then every honest party outputs $(v, 1)$.
- **Graded consistency:** if some honest party outputs $(w, 1)$, then every honest party P_i outputs (w, g_i).

We formally describe a graded-consensus protocol Π_{GC} inspired by the graded consensus protocol of Canetti and Rabin [CR93], and prove the following theorem in Appendix G.2.

Theorem 11.3.6. *Let $0 < \epsilon < 1/3$ and $f \leq (1-2\epsilon) \cdot n/3$. Then Π_{GC} is an f-secure graded-consensus protocol with expected setup size $O(\kappa^3)$ and expected communication complexity $O((\kappa + \mathcal{I}) \cdot \kappa^2 n)$, where $\mathcal{I}$ is the size of each party's input.*

11.3.4 A Coin-Flip Protocol

We describe here a protocol that allows parties to generate a sequence of random bits (coins) $\mathsf{Coin}_1, \ldots, \mathsf{Coin}_T$ for a pre-determined parameter T. We denote the sub-protocol to generate the ith coin by $\mathsf{CoinFlip}(i)$. Roughly speaking, the protocol guarantees that (1) when all honest parties invoke $\mathsf{CoinFlip}(i)$, all honest parties output the same value Coin_i and (2) until the first honest party invokes $\mathsf{CoinFlip}(i)$, the value of Coin_i is uniform.

Our coin-flip protocol assumes setup provided by a trusted dealer that takes the following form: For each iteration $1, \ldots, T$, the dealer chooses uniform $\mathsf{Coin}_i \in \{0,1\}$; chooses a random subset E_i of the parties by including each party in E_i with probability κ/n; and then gives authenticated secret shares of Coin_i (using a perfectly secret $\lceil \kappa/3 \rceil$-out-of-$|E_i|$ secret-sharing scheme) to the members of E_i. (Authentication is done by having the dealer sign the shares.) Since each share (including the signature) has size $O(\kappa)$, the size of the setup is $O(\kappa^2 T)$.

The coin-flip protocol itself simply involves having the parties in the relevant subset send their shares to everyone else. The communication complexity is thus $O(\kappa^2 n)$ per iteration.

Lemma 11.3.7. *Let $0 < \epsilon < 1/3$ and $f \leq (1-2\epsilon) \cdot n/3$. Then as long as at most f parties are corrupted, $\mathsf{CoinFlip}(i)$ satisfies the following:*

1. *all honest parties obtain the same value Coin_i,*
2. *until the first honest party invokes $\mathsf{CoinFlip}(i)$, the value of Coin_i is uniform from the adversary's perspective.*

Proof. Lemma G.1.3 implies that, except with negligible probability, E_i contains more than $\lceil \kappa/3 \rceil$ honest parties and fewer than $(1-\epsilon) \cdot \kappa/3$ corrupted parties. The stated properties follow. □

11.4 Single-Shot BA

In this section we describe a BA protocol Π_{BA} with subquadratic communication complexity. (See Figure 11.4.) Π_{BA} assumes setup that is then used for a single execution of the protocol. The setup for Π_{BA} corresponds to the setup required for $O(\kappa)$ executions of graded consensus, $O(\kappa)$ iterations of the coin-flip sub-protocol, and a single execution of reliable consensus. Using the protocols from the previous section, Π_{BA} thus requires setup of size $O(\kappa^4)$ overall.

Following ideas by Mostéfaoui et al. [MHR14], our protocol consists of a sequence of $\Theta(\kappa)$ iterations, where each iteration invokes a graded-consensus subprotocol and a coin-flip subprotocol. In each iteration there is a constant probability that honest parties reach agreement; once agreement is reached, it cannot be undone in later iterations. The coin-flip protocol allows parties to adopt the value of a common coin if agreement has not yet been reached (or, at least, if parties are unaware that agreement has been reached). Reliable consensus is used so parties know when to terminate.

We prove security via a sequence of lemmas. Throughout the following, we fix some value $0 < \epsilon < 1/3$ and let $f \leq (1-2\epsilon)n/3$ be a bound on the number of corrupted parties.

Protocol $\Pi_{\mathsf{BA}}(P)$

Let ϵ be a constant parameter. We describe the protocol from the point of view of a party with input $v \in \{0,1\}$.

Protocol Loop Set $b := v$ and $\mathsf{ready} := \texttt{false}$.

1: **for** $k = 1$ to $\kappa + 1$ **do**
2: Run Π_{GC} on input b, and let (b, g) denote the output.
3: Invoke $\mathsf{CoinFlip}(k)$ to obtain Coin_k.
4: If $g = 0$ then set $b := \mathsf{Coin}_k$.
5: Run Π_{GC} on input b, and let (b, g) denote the output.
6: If $g = 1$ and $\mathsf{ready} = \texttt{false}$, then set $\mathsf{ready} := \texttt{true}$ and run Π_{RC} on input b.

Termination

1: If Π_{RC} terminates with output b', output b' and terminate.

Lemma 11.4.1. *If at most f parties are corrupted during an execution of Π_{BA}, then with all but negligible probability some honest party sets* $\mathsf{ready} = \mathtt{true}$ *within the first κ iterations.*

Proof. Consider an iteration k of Π_{BA} such that no honest party set $\mathsf{ready} = \mathtt{true}$ in any previous iteration. (This is trivially true in the first iteration). We begin by showing that some honest party sets $\mathsf{ready} = \mathtt{true}$ in that iteration with probability at least $1/2$. Consider two cases:

- If some honest party outputs $(b, 1)$ in the first execution of Π_{GC} during iteration k, then graded consistency of Π_{GC} guarantees that every other honest party outputs $(b, 1)$ or $(b, 0)$ in that execution. The value b is independent of Coin_k, because b is determined prior to the point when the first honest party invokes $\mathsf{CoinFlip}(i)$; thus, $\mathsf{Coin}_k = b$ with probability $1/2$. If that occurs, then all honest parties input b to the second execution of Π_{GC} and, by graded validity, every honest party outputs $(g, 1)$ in the second execution of Π_{GC} and sets $\mathsf{ready} = \mathtt{true}$.

- Say no honest party outputs grade 1 in the first execution of Π_{GC} during iteration k. Then all honest parties input Coin_k to the second execution of Π_{GC} and, by graded validity, every honest party outputs $(g, 1)$ in the second execution of Π_{GC} and sets $\mathsf{ready} = \mathtt{true}$.

Thus, in each iteration where no honest party has yet set $\mathsf{ready} = \mathtt{true}$, some honest party sets $\mathsf{ready} = \mathtt{true}$ in that iteration with probability at least $1/2$. We conclude that the probability that no honest party has set $\mathsf{ready} = \mathtt{true}$ after κ iterations is negligible. □

Lemma 11.4.2. *Assume at most f parties are corrupted during execution of Π_{BA}. If some honest party executes Π_{RC} using input b in iteration k, then (1) honest parties who execute Π_{GC} in any iteration $k' > k$ use input b, and (2) honest parties who execute Π_{RC} in any iteration $k' \geq k$ use input b.*

Proof. Consider the first iteration k in which some honest party P sets $\mathsf{ready} = \mathtt{true}$, and let b denote P's input to Π_{RC}. P must have received $(b, 1)$ from the second execution of Π_{GC} in iteration k. By graded consistency, all other honest parties must receive $(b, 0)$ or $(b, 1)$ from that execution of Π_{GC} as well. Thus, any honest parties who execute Π_{RC} in

iteration k use input b, and any honest parties who run[2] the first execution of Π_{GC} in iteration $k+1$ will use input b as well. Graded validity ensures that any honest party who receives output from that execution of Π_{GC} will receive $(b, 1)$, causing them to use input b to the next execution of Π_{GC} as well as Π_{RC} (if they execute those protocols), and so on. □

Lemma 11.4.3. *Assume at most f parties are corrupted during an execution of Π_{BA}. If some honest party sets* ready = `true` *within the first κ iterations and executes Π_{RC} using input b, then all honest parties terminate with output b.*

Proof. Let $k \leq \kappa$ be the first iteration in which some honest party sets ready = `true` and executes Π_{RC} using input b. By Lemma 11.4.2, any other honest party who executes Π_{RC} must also use input b, and furthermore all honest parties who execute Π_{GC} in any subsequent iteration use input b there as well. We now consider two cases:

- If no honest party terminates before all honest parties receive output from the second execution of Π_{GC} in iteration $k+1$, then graded validity of Π_{GC} ensures that all honest parties receive $(b, 1)$ as output from that execution, and thus all parties execute Π_{RC} using input b at this point if they have not done so already. Validity of Π_{RC} then ensures that all honest parties output b and terminate.
- If some honest party P has terminated before all honest parties receive output from the second execution of Π_{GC} in iteration $k+1$, validity of Π_{RC} implies that P must have output b. In that case, consistency of Π_{RC} guarantees that all parties will eventually output b and terminate.

This completes the proof. □

Theorem 11.4.4. *Let $0 < \epsilon < 1/3$ and $f \leq (1-2\epsilon) \cdot n/3$. Then Π_{BA} is an f-secure BA protocol with expected setup size $O(\kappa^4)$ and expected communication complexity $O(\kappa^4 n)$.*

Proof. By Lemma 11.4.1, with overwhelming probability some honest party sets ready = `true` within the first κ iterations and thus executes

[2]Note that some honest parties may terminate before others, and in particular it may be the case that not all honest parties run some execution of Π_{GC}.

Π_{RC} using some input b. It follows from Lemma 11.4.3 that all honest parties eventually output b and terminate. This proves consistency.

Assume all honest parties have the same input v. Unless some honest party terminates before all honest parties have concluded the first iteration, one can verify (using graded validity of Π_{GC}) that in the first iteration all honest parties output $(v, 1)$ from the first execution of Π_{GC}; use input v to the second execution of Π_{GC}; output $(v, 1)$ from the second execution of Π_{GC}; and execute Π_{RC} using input v. But the only way some honest party could terminate before all honest parties have concluded the first iteration is if that party executes Π_{RC} using input v. Either way, Lemma 11.4.3 shows that all honest parties will terminate with output v, proving validity. □

11.5 MPC with Subquadratic Communication

In this section we give a protocol for asynchronous secure multiparty computation (MPC). Our protocol uses a Byzantine agreement protocol as a subroutine; importantly, the number of executions of Byzantine agreement is independent of the number of parties as well as the output length, as long as the desired input quality is low enough. Our MPC protocol also relies on a sub-protocol for (a variant of the) *asynchronous common subset* problem; we give a definition, and a protocol with subquadratic communication complexity, in the next section.

11.5.1 Validated ACS with Subquadratic Communication

A protocol for the asynchronous common subset (ACS) problem [BKR94, Can96] allows n parties to agree on a subset of their initial inputs of some minimum size. We consider a *validated* version of ACS (VACS), where it is additionally ensured that all values in the output multiset satisfy a given predicate Q [CHP12, CKPS01].

Definition 11.5.1. Let Q be a predicate, and let Π be a protocol executed by parties $P_1, \ldots, P_n$, where each party outputs a multiset of size

at most n, and terminates upon generating output. Π is an **f-secure Q-validated ACS protocol with ℓ-output quality** if the following hold when at most f parties are corrupted and every honest party's input satisfies Q:

- **Q-Validity:** if an honest party outputs S, then each $v \in S$ satisfies $Q(v) = 1$.
- **Consistency:** every honest party outputs the same multiset.
- **ℓ-Output quality:** all honest parties output a multiset of size at least ℓ that contains inputs from at least $\ell - f$ parties who were honest at the start of the protocol.

Our VACS protocol $\Pi_{\mathsf{acs}}^{\ell,Q}$ is inspired by the protocol of Ben-Or et al. [BKR94]. During the setup phase, a secret committee C is chosen by independently placing each party in C with probability s/n, where $s = \frac{3}{2+\epsilon}\ell$ and ℓ is the desired output quality. Each party in the committee acts as a sender in a reliable-broadcast protocol, and then the parties run $|C|$ instances of Byzantine agreement to agree on the set of reliable-broadcast executions that terminated. The expected communication complexity and setup size for $\Pi_{\mathsf{acs}}^{\ell,Q}$ are thus (in expectation) a factor of $O(\ell)$ larger than those for reliable broadcast and Byzantine agreement.

Protocol $\Pi_{\mathsf{acs}}^{\ell,Q}(P)$

Let ϵ be a constant parameter. The protocol has ℓ-output quality and predicate Q. We describe the protocol from the point of view of a party P with input v. We assume prior setup in which a committee C is chosen (see text).

1: Execute $|C|$ instances of reliable broadcast, denoted $\mathtt{RBC}_1, \ldots, \mathtt{RBC}_{|C|}$. If P is the ith member of C, then P executes the ith instance of Π_{RBC} as the sender using input v.
2: On output v_i from $\mathtt{RBC}_i$ with $Q(v_i) = 1$, if P has not yet begun executing the ith instance $\mathtt{BA}_i$ of Byzantine agreement, then begin that execution using input 1.
3: When P has output 1 in ℓ instances of Byzantine agreement, then begin executing any other instances of Byzantine agreement that have not yet begun using input 0.

4: Once P has terminated in all instances of Byzantine agreement, let $\mathsf{CoreSet}$ be the indices of those instances that resulted in output 1. After receiving output v_i from $\mathtt{RBC}_i$ for all $i \in \mathsf{CoreSet}$, output the multiset $\{v_i\}_{i \in \mathsf{CoreSet}}$.

Using the protocols from the previous sections, we thus obtain:

Theorem 11.5.2. *Let $0 < \epsilon < 1/3$, $f \leq (1-2\epsilon) \cdot n/3$, and $\ell \leq (1+\epsilon/2) \cdot 2n/3$. Then $\Pi_{\mathsf{acs}}^{\ell,Q}$ is an f-secure Q-validated ACS protocol with ℓ-output quality. It has expected setup size $O(\ell\kappa^4)$ and expected communication complexity $O(\ell \cdot (\mathcal{I} + \kappa^3) \cdot \kappa n)$, where $\mathcal{I}$ is the size of each party's input, and uses $O(\ell)$ invocations of Byzantine agreement in expectation.*

Proof. Say v is in the multiset output by some honest party, where v was output by $\mathtt{RBC}_i$. $\mathtt{BA}_i$ must have resulted in output 1, which (by validity of $\mathtt{BA}$) can only occur if some honest party used input 1 when executing $\mathtt{BA}_i$. But then $Q(v) = 1$. This proves Q-validity of $\Pi_{\mathsf{acs}}^{\ell,Q}$.

By consistency of $\mathtt{BA}$, all honest parties agree on $\mathsf{CoreSet}$. If $i \in \mathsf{CoreSet}$, then $\mathtt{BA}_i$ must have resulted in output 1 which means that some honest party P must have used input 1 to $\mathtt{BA}_i$. (Validity or $\mathtt{BA}_i$ ensures that if all honest parties used input 0, the output of $\mathtt{BA}$ must be 0). But then P must have terminated in $\mathtt{RBC}_i$; consistency of $\mathtt{RBC}_i$ then implies that all honest parties eventually terminate $\mathtt{RBC}_i$ with the same output v_i. Consistency of $\Pi_{\mathsf{acs}}^{\ell,Q}$ follows.

Lemma G.1.3 shows that with overwhelming probability there are more than $\frac{2+\epsilon}{3} \cdot \frac{3}{2+\epsilon}\ell = \ell$ honest parties in C at step 1 of the protocol. Validity of $\mathtt{RBC}$ implies that in the corresponding instances of $\mathtt{RBC}$, all honest parties terminate with an output satisfying Q. If every honest party begins executing all the corresponding instances of $\mathtt{BA}$, those ℓ instances will all yield output 1. The only way all honest parties might not begin executing all those instances of $\mathtt{BA}$ is if some honest party outputs 1 in some (other) ℓ instances of $\mathtt{BA}$, but then consistency of $\mathtt{BA}$ implies that all honest parties output 1 in those same ℓ instances. We conclude that every honest party outputs 1 in at least ℓ instances of $\mathtt{BA}$, and so outputs a multiset S of size at least ℓ. Since each instance of $\mathtt{RBC}$ (and so each corrupted party) contributes at most one value to S, this proves ℓ-output quality. □

11.5.2 Secure Multiparty Computation

We construct an MPC protocol $\Pi^{\ell}_{\mathsf{MPC}}$ that offers a tradeoff between communication complexity and output quality, the number of inputs taken into the account for the computation. We say that the protocol has ℓ-output quality, if the computation takes into account at least ℓ inputs.[3] Our protocol has subquadratic communication complexity when the output quality and the output length of the functionality being computed are sublinear in the number of parties. A similar protocol that uses additive homomorphic encryption for the synchronous setting has recently been proposed in [GHK+21], following the player-replaceable paradigm where entities are only in charge of sending a single message (denoted as the You-Only-Speak-Once YOSO property).

We provide a high-level overview of our protocol next, with a full description afterwards.

Let $t = (1 - \epsilon) \cdot \kappa/3$. Our protocol assumes trusted setup as follows:

1. A random committee C is selected by including each party in C independently with probability κ/n. This is done in the usual way by giving each member of the committee a secret key for a signature scheme, and giving the corresponding public keys to all parties. In addition:

   (a) We assume a threshold fully homomorphic encryption scheme [AJL+12, BGG+18] $\mathsf{TFHE} = (\mathsf{KGen}, \mathsf{Enc}, \mathsf{DecShare}, \mathsf{Rec}, \mathsf{Eval})$ with non-interactive decryption whose secret key is shared in a t-out-of-$|C|$ manner among the parties in C. (We refer to Section G.3.1 for its security definition.)

   Specifically, we assume a TFHE public key ek is given to all parties, while a share dk_i of the corresponding secret key is given to the ith party in C.

   (b) The setup for $\Pi^{\ell}_{\mathsf{MPC}}$ includes setup for $|C|$ instances of Π_{RBC} (with the ith party in C the sender for the ith instance of Π_{RBC}), as well as one instance of Π_{RC}.

2. All parties are given a list of $|C|$ commitments to each of the TFHE shares dk_i; the randomness ω_i for the ith commitment is given to

[3] Usually asynchronous MPC protocols are described with optimal $n - f$ output quality.

the ith member of C.

3. All parties are given the TFHE encryption of a random κ-bit value r. We denote the resulting ciphertext by $c_{\mathsf{rand}} \leftarrow \mathsf{Enc}_{ek}(r)$.

4. Parties are given the setup for one instance of VACS protocol $\Pi^{\ell,Q}_{\mathsf{acs}}$. We further assume that each party in the committee that is chosen as part of the setup for that protocol is given a secret key for a signature scheme, and all parties are given the corresponding public keys.

5. All parties are given a common reference string (CRS) for a universally composable non-interactive zero-knowledge (UC-NIZK) proof [DDO+01] (see below).

The overall expected size of the setup is $O((\ell + \kappa) \cdot \mathsf{poly}(\kappa))$.

Fix a (possibly randomized) functionality g the parties wish to compute. We assume without loss of generality that g uses exactly κ random bits (one can always use a PRG to ensure this). To compute g, each party P_i begins by encrypting its input x_i using the TFHE scheme, and signing the result; it also computes an NIZK proof of correctness for the resulting ciphertext. The parties then use VACS (with ℓ-output quality) to agree on a set S containing at least ℓ of those ciphertexts. Following this, parties carry out a local computation in which they evaluate g homomorphically using the set of ciphertexts in S as the inputs and the ciphertext c_{rand} (included in the setup) as the randomness. This results in a ciphertext c^* containing the encrypted result, held by all parties. Parties in C enable decryption of c^* by using reliable broadcast to distribute shares of the decrypted value (along with a proof of correctness). Finally, the parties use reliable consensus to agree on when to terminate.

In the description above, we have omitted some details. In particular, the protocol ensures adaptive security by having parties erase certain information once it is no longer needed. This means, in particular, that we do not need to rely on equivocal TFHE [CsW19].

In our protocol, parties generate UC-NIZK proofs for different statements. (Note that UC-NIZK proofs are proofs of knowledge; they are also non-malleable.) In particular, we define the following languages, parameterized by values (given to all parties) contained in the setup:

1. $(i, c_i) \in L_1$ if there exist x_i, r_i such that $c_i = \mathsf{Enc}_{\mathsf{ek}}(x_i; r_i)$.

2. $(i, c^*, d_i) \in L_2$ if $d_i = \mathsf{DecShare}_{\mathsf{dk}_i}(c^*)$ and $\mathsf{com}_i = \mathsf{Com}(dk_i; \omega_i)$. (Here, com_i is the commitment to dk_i included in the setup.)

Protocol $\Pi^{\ell}_{\mathsf{MPC}}(P_i)$

Let ϵ be a constant parameter. The protocol has ℓ-output quality. and predicate Q. We describe the protocol from the point of view of a party P_i with input x_i, assuming the setup described in the text. Let $t = (1-\epsilon)\cdot\kappa/3$.

1: Compute $c_i \leftarrow \mathsf{Enc}_{\mathsf{ek}}(x_i)$ along with a UC-NIZK proof π_i that $(i, c_i) \in L_1$. Erase x_i and the randomness used to generate c_1 and π_i.
Execute $\Pi^{\ell,Q}_{\mathsf{acs}}$ using input $(i, \mathsf{Sign}_{\mathsf{sk}_i}(c_i), c_i, \pi_i)$, where $Q(i, \sigma, c, \pi) = 1$ iff $\mathsf{Vrfy}_{\mathsf{pk}_i}(c, \sigma) = 1$ and π is a correct proof for (i, c). Let S' denote the multiset output by $\Pi^{\ell,Q}_{\mathsf{acs}}$. Let $S \subseteq S'$ be the set obtained by including, for all i, only the lexicographically first tuple $(i, \star, \star, \star)$ in S'. Let $I = \{i | \exists\, (i, \star, \star, \star) \in S\}$.

2: Define the circuit $\mathcal{C}_g$ taking $|I| + 1$ inputs, where $\mathcal{C}_g(\{x_i\}_{i\in I}, r) = g(\{x_i\}_{i\in I}, \{\perp\}_{i\notin I}; r)$. Compute $c^* := \mathsf{Eval}_{\mathsf{ek}}(\mathcal{C}_g, \{c_i\}_{i\in I}, c_{\mathsf{rand}})$.
If $P_i \in C$, compute $d_i := \mathsf{DecShare}_{\mathsf{dk}_i}(c^*)$ and a UC-NIZK proof π'_i that $(i, c^*, d_i) \in L_2$. Erase dk_i, ω_i, and the randomness used to generate π'_i.
Execute $|C|$ instances of Π_{RBC}. If P_i is the ith member of C, it executes the ith instance of Π_{RBC} as the sender using input (i, d_i, π'_i).

3: Upon receiving t outputs $\{(j, d_j, \pi'_j)\}$ from the Π_{RBC} instances, with valid proofs and distinct j, compute $y_i := \mathsf{Rec}(\{d_j\})$ and execute Π_{RC} with input y_i. When Π_{RC} terminates with output y, output (y, I) and terminate.

We prove the following theorem in Appendix G.4.

Theorem 11.5.3. *Let $0 < \epsilon < 1/3$, $f \le (1-2\epsilon)\cdot n/3$, and $\ell \le (1+\epsilon/2)\cdot 2n/3$. Assuming appropriate security of the NIZK proofs and* TFHE*, protocol $\Pi^{\ell}_{\mathsf{MPC}}$ f-securely computes g with ℓ-output quality. $\Pi^{\ell}_{\mathsf{MPC}}$ requires setup of expected size $O((\ell+\kappa)\cdot\mathsf{poly}(\kappa))$, has expected communication complexity $O((\ell+\kappa)\cdot(\mathcal{I}+\mathcal{O})\cdot\mathsf{poly}(\kappa)\cdot n)$, where $\mathcal{I}$ is the size of each party's input and $\mathcal{O}$ is the size of the output, and invokes Byzantine agreement $O(\ell)$ times in expectation.*

11.6 Putting it All Together

The BA protocol Π_{BA} from Section 11.4 requires prior setup by a trusted dealer that can be used only for a *single* BA execution. Using multiple, independent instances of the setup it is, of course, possible to support any *bounded* number of BA executions. But a new idea is needed to support an *unbounded* number of executions.

In this section we discuss how to use the MPC protocol from Section 11.5 to achieve this goal. The key idea is to use that protocol to *refresh the setup* each time a BA execution is done. We first describe how to modify our MPC protocol to make it suitable for our setting, and then discuss how to put everything together to obtain the desired result.

11.6.1 Securely Simulating a Trusted Dealer

As just noted, the key idea is for the parties to use the MPC protocol from Section 11.5 to simulate a trusted dealer. In that case the parties are evaluating a no-input (randomized) functionality, and so do not need any output quality; let $\Pi_{\mathsf{MPC}} = \Pi^0_{\mathsf{MPC}}$. Importantly, Π_{MPC} has communication complexity subquadratic in n.

Using Π_{MPC} to simulate a dealer, however, requires us to address several technicalities. As described, Π_{MPC} evaluates a functionality for which all parties receive the *same* output. But simulating a dealer requires the parties to compute a functionality where parties receive *different* outputs. The standard approach for adapting MPC protocols to provide parties with different outputs does not work in our context: specifically, using symmetric-key encryption to encrypt the output of each party P_i using a key that P_i provides as part of its input does not work since Π_{MPC} has no output quality (and even $\Pi^{\ell}_{\mathsf{MPC}}$ only guarantees ℓ-output quality for $\ell < n$). Assuming a PKI, we can fix this by using public-key encryption instead (in the same way); this works since the public keys of the parties can be incorporated into the functionality being computed—since they are common knowledge—rather than being provided as inputs to the computation.

Even when using public-key encryption as just described, however, additional issues remain. Π_{MPC} has (expected) subquadratic communication complexity only when the output length $\mathcal{O}$ of the functionality being computed is sublinear in the number of parties. Even if the dealer

algorithm generates output whose length is independent of n, naively encrypting output for every party (encrypting a "null" value of the appropriate length for parties whose output is empty) would result in output of total length linear in n. Encrypting the output only for parties with non-empty output does not work either since, in general, this might reveal which parties get output, which in our case would defeat the purpose of the setup!

We can address this difficulty by using *anonymous public-key encryption* [BBDP01]. Roughly, an anonymous public-key encryption (APKE) scheme has the property that a ciphertext leaks no information about the public key pk used for encryption, except to the party holding the corresponding secret key sk (who is able to decrypt the ciphertext using that key). Using APKE to encrypt the output for each party who obtains non-empty output, and then randomly permuting the resulting ciphertexts, allows us to compute a functionality with sublinear output length while hiding which parties receive output. This incurs—at worst—an additional multiplicative factor of κ in the output length.

Summarizing, we can simulate an arbitrary dealer algorithm in the following way. View the output of the dealer algorithm as $\mathsf{pub}, \{(i, s_i)\}$, where pub represents the public output that all parties should learn, and each s_i is a private output that only P_i should learn. Assume the existence of a PKI, and let pk_i denote a public key for an APKE scheme, where the corresponding secret key is held by P_i. Then use Π_{MPC} to compute $\mathsf{pub}, \{\mathsf{Enc}_{\mathsf{pk}_i}(s_i)\}$, where the ciphertexts are randomly permuted. As long as the length of the dealer's output is independent of n, the output of this functionality is also independent of n.

11.6.2 Unbounded Byzantine Agreement with Subquadratic Communication

We now show how to use the ideas from the previous section to achieve an *unbounded* number of BA executions with subquadratic communication. We describe two solutions: one involving a trusted dealer who initializes the parties with a one-time setup, and another that does not require a dealer (but does assume a PKI) and achieves expected subquadratic communication in an amortized sense.

For the first solution, we assume a trusted dealer who initializes the parties with the setup for one instance of Π_{BA} and one instance of Π_{MPC}.

(We also assume a PKI, which could be provided by the dealer as well; however, when we refer to the setup for Π_{MPC} we do not include the PKI since it does not need to be refreshed.) Importantly, the setup for Π_{MPC} allows the parties to compute any no-input functionality; the size of the setup is fixed, independent of the size of the circuit for the functionality being computed or its output length. For an execution of Byzantine agreement, the parties run Π_{BA} using their inputs and then use Π_{MPC} to refresh their setup by simulating the dealer algorithm. (We stress that the parties refresh the setup for both Π_{BA} and Π_{MPC}.) The expected communication complexity per execution of Byzantine agreement is the sum of the communication complexities of Π_{BA} and Π_{MPC}. The former is subquadratic; the latter is subquadratic if we follow the approach described in the previous section. Thus, the parties can run an unbounded number of subquadratic BA executions while only involving a trusted dealer once.

Alternately, we can avoid a trusted dealer by having the parties simulate the dealer using an arbitrary adaptively secure MPC protocol. (We still assume a PKI.) The communication complexity of the initial MPC protocol may be arbitrarily high, but all subsequent BA executions will have subquadratic (expected) communication complexity as above. In this way we achieve an unbounded number of BA executions with amortized (expected) subquadratic communication complexity.

Appendix G

Details of Chapter 11

G.1 Concentration Inequalities

We briefly recall the following standard concentration bounds.

Lemma G.1.1. (Markov bound) *Let X be a non-negative random variable. Then for $a > 0$,*

$$\Pr[X \geq a] \leq \frac{E[X]}{a}.$$

Lemma G.1.2 (Chernoff bound)**.** *Let $X_1, ..., X_n$ be independent Bernoulli random variables with parameter p. Let $X := \sum_i X_i$, so $\mu := E[X] = p \cdot n$. Then, for $\delta \in [0, 1]$*

- $\Pr[X \leq (1-\delta) \cdot \mu] \leq e^{-\delta^2 \mu/2}$.
- $\Pr[X \geq (1+\delta) \cdot \mu] \leq e^{-\delta^2 \mu/(2+\delta)}$.

Let $\chi_{s,n}$ denote the distribution that samples a subset of the n parties, where each party is included independently with probability s/n. The following lemma will be useful in our analysis.

Lemma G.1.3. *Fix $s \leq n$ and $0 < \epsilon < 1/3$, and let $f \leq (1 - 2\epsilon) \cdot n/3$ be a bound on the number of corrupted parties. If $C \leftarrow \chi_{s,n}$, then:*

1. *C contains fewer than $(1+\epsilon) \cdot s$ parties except with probability $e^{-\frac{\epsilon^2 s}{2+\epsilon}}$.*

2. *C contains more than $(1+\epsilon/2)\cdot 2s/3$ honest parties except with probability at most $e^{-\epsilon^2 s/12\cdot(1+\epsilon)}$.*

3. *C contains fewer than $(1-\epsilon)\cdot s/3$ corrupted parties except with probability at most $e^{-\epsilon^2 s/(6-9\epsilon)}$.*

Proof. Let $H \subseteq [n]$ be the indices of the honest parties. Let X_j be the Bernoulli random variable indicating if $P_j \in C$, so $\Pr[X_j = 1] = s/n$. Define $Z_1 = \sum_j P_j$, $Z_2 := \sum_{j\in H} X_j$, and $Z_3 := \sum_{j\notin H} X_j$. Then:

1. Since $E[Z_1] = s$, setting $\delta = \epsilon$ in Lemma G.1.2 yields

$$\Pr\left[Z_1 \geq (1+\epsilon)\cdot s\right] \leq e^{-\epsilon^2 s/(2+\epsilon)}.$$

2. Since $E[Z_2] \geq (n-f)\cdot s/n \geq (1+\epsilon)\cdot 2s/3$, setting $\delta = \frac{\epsilon}{2+2\epsilon}$ in Lemma G.1.2 yields

$$\Pr\left[Z_2 \leq \frac{(1+\epsilon/2)\cdot 2s}{3}\right] \leq e^{-\epsilon^2 s/12\cdot(1+\epsilon)}.$$

3. Since $E[Z_3] \leq f\cdot s/n \leq (1-2\epsilon)\cdot s/3$, setting $\delta = \frac{\epsilon}{1-2\epsilon}$ in Lemma G.1.2 yields

$$\Pr\left[Z_3 \geq \frac{(1-\epsilon)\cdot s}{3}\right] \leq e^{-\epsilon^2 s/(6-9\epsilon)}.$$

□

G.2 Graded Consensus

We describe a graded-consensus protocol Π_{GC} in Figure G.2. The protocol is inspired by the graded consensus protocol of Canetti and Rabin [CR93]. Π_{GC} assumes setup that defines three secret committees C_1, C_2, C_3 by including each party independently in each committee with probability κ/n. Each party in a committee will act as a sender in a reliable-broadcast protocol `RBC`; independent setup is used for each of these. The graded-consensus protocol itself consists of three phases, where in phase i, each party in committee C_i uses `RBC` to send a phase-specific message to all parties. In the first phase, parties in C_1 reliably broadcast their input

values. All parties wait until $\kappa - t$ of these executions of reliable broadcast output a value (and terminate), and then set their $\mathsf{prepare}_2$ value to be the majority value among those outputs. In the second phase, parties in C_2 reliably broadcast their $\mathsf{prepare}_2$ values. All parties wait for $\kappa - t$ of these executions of reliable broadcast to output values consistent with the values from the first phase, and then set their $\mathsf{prepare}_3$ value to be the majority among such outputs. In the third phase, parties in C_3 reliably broadcast their $\mathsf{prepare}_3$ values. Parties wait for $\kappa - t$ of these executions of reliable broadcast to output values consistent with the received $\mathsf{prepare}_2$ values, and then decide on their output.

Since each set C_i has expected size $O(\kappa)$, the expected communication complexity and setup size for Π_{GC} are only a factor of κ larger than their corresponding values for $\mathtt{RBC}$. Instantiating $\mathtt{RBC}$ using Π_{RBC} gives the complexity bounds stated in Theorem 11.3.6.

Protocol Π_{GC}

We describe the protocol from the point of view of a party P_i with input $v_i \in \{0, 1\}$. We let $\mathtt{RBC}$ denote a reliable broadcast protocol.

Protocol Execution

1: Initialize $\hat{S}_1 = \hat{S}_2 = S_1 = S_2 = S_3 := \varnothing, b_1 := v, b_2 = b_3 := \perp$
2: If $P_i \in C_1$: participate in $\mathtt{RBC}_i$ as the sender with input $(\mathsf{prepare}_1, b_1)$. Participate in the remaining protocols $\mathtt{RBC}_j$, $j \neq i, j \in C_1$, as the receiver.
3: Upon receiving output $(\mathsf{prepare}_1, j, b_j)$ in $\mathtt{RBC}_j$, add (b_j, j) to S_1.
4: When $|S_1| = \kappa - t$, do: Set $\hat{S}_1 = S_1$ and set b_2 to the majority bit among values in $\hat{S}_1$. Participate in $\mathtt{RBC}_i$ as the sender with input $(\mathsf{prepare}_2, i, \hat{S}_1, b_2)$ if $P_i \in C_2$. Participate in the other protocols $\mathtt{RBC}_1, \ldots, \mathtt{RBC}_{|C_2|}$ as the receiver.
5: Upon receiving output $(\mathsf{prepare}_2, j, \hat{S}_{1,j}, b_j)$ in $\mathtt{RBC}_j$ do: if $\hat{S}_{1,j} \subseteq S_1$ and b_j is the majority bit among $\hat{S}_{1,j}$, add (b_j, j) to S_2.
6: When $|S_2| = \kappa - t$, do: Set $\hat{S}_2 = S_2$ and set b_3 to the majority bit among values in $\hat{S}_2$. Participate in $\mathtt{RBC}_i$ as the sender with input $(\mathsf{prepare}_3, i, \hat{S}_2, b_3)$ if $P_i \in C_3$. Participate in each protocol $\mathtt{RBC}_j$, $j \neq i, j \in C_3$, as the receiver.
7: Upon receiving output $(\mathsf{prepare}_3, j, \hat{S}_{2,j}, b_j)$ in $\mathtt{RBC}_j$ do: if $j \in C_3$, $\hat{S}_{2,j} \subseteq S_2$, and b_j is the majority bit among $\hat{S}_{2,j}$, add (b_j, j) to S_3.
8: When $|S_3| = \kappa - t$, execute the Output Determination step.

Output Determination

1: If there exists $b \in \{0,1\}$ s.t. for all $(b_j, j) \in \hat{S}_2$ it holds that $b_j = b$, then output $(b, 1)$.
2: Else if there exists $b \in \{0,1\}$ s.t. for all $(b_j, j) \in S_3$ it holds that $b_j = b$, then output $(b, 0)$.
3: Else output $(0, 0)$.

Lemma G.2.1. *Let P_i and P_j be honest parties, and denote as $S_{1,j}, S_{1,i}$ the respective sets S_1 of those parties in an execution of Π_{GC}. Then with overwhelming probability, eventually $S_{1,j} \subseteq S_{1,i}$.*

Proof. Suppose that $(b_\ell, \ell) \in S_{1,j}$. With overwhelming probability, this implies that P_j output $(\mathsf{prepare}_1, \ell, b_\ell)$ in $\mathtt{RBC}_\ell$ where P_ℓ in C_1. By the consistency property of $\mathtt{RBC}$, P_i either eventually outputs $(\mathsf{prepare}_1, \ell, b_\ell)$ in $\mathtt{RBC}_\ell$ and hence adds (b_ℓ, ℓ) to $S_{1,i}$ or terminates Π_{GC} (with overwhelming probability). Thus, every value in $S_{1,j}$ is eventually added to $S_{1,i}$ (and hence $S_{1,j} \subseteq S_{1,i}$), with overwhelming probability. □

Lemma G.2.2. *Let P_i and P_j be honest parties, and denote as sets $S_{2,j}, S_{2,i}$ the respective sets S_2 of those parties in an execution of Π_{GC}. Then with overwhelming probability, eventually $S_{2,j} \subseteq S_{2,i}$.*

Proof. Denote as $S_{1,j}, S_{1,i}$ the respective sets S_1 of parties P_i and P_j and suppose that $(b_\ell, \ell) \in S_{2,j}$. With overwhelming probability, this implies that P_j output $(\mathsf{prepare}_2, \ell, \hat{S}_{1,\ell}, b_\ell)$ in $\mathtt{RBC}_\ell$ where P_ℓ in C_2, $\hat{S}_{1,\ell} \subseteq S_{1,j}$, and b_ℓ is the majority bit among values in $\hat{S}_{1,\ell}$. By the consistency property of $\mathtt{RBC}$ and the previous lemma, P_i either eventually outputs $(\mathsf{prepare}_2, \ell, b_\ell)$ in $\mathtt{RBC}_\ell$ and $\hat{S}_{1,\ell} \subseteq S_{1,j} \subseteq S_{1,i}$ or or terminates Π_{GC} (with overwhelming probability). Once the former happens, P_i adds (b_ℓ, ℓ) to $S_{2,i}$. Thus, every value in $S_{2,j}$ is eventually added to $S_{2,i}$ (and hence $S_{2,j} \subseteq S_{2,i}$), with overwhelming probability. □

Lemma G.2.3. *With overwhelming probability, for every honest party P_i the sets S_1, S_2, and S_3 are each eventually of size $\kappa - t$.*

Proof. Let P_i be an honest party. We analyze the size of the sets in sequence.

S_1: By validity, P_i outputs in all $\mathtt{RBC}$ instances corresponding to honest parties in C_1 with overwhelming probability and adds a corresponding

tuple to S_1 as a result. Since by Lemma G.1.3, at least $\kappa - t$ parties in C_1 are honest, the claim for S_1 follows.

S_2: Since all honest parties S_1 sets eventually become of size $\kappa - t$, all honest parties P_j in C_2 eventually send a message $(\mathsf{prepare}_2, \ell, \hat{S}_{1,j}, b_j)$ in $\mathtt{RBC}_j$. By Lemma G.2.1, $\hat{S}_{1,j} \subseteq S_i$, with overwhelming probability, eventually. This implies that all checks for the instance $\mathtt{RBC}_j$ are satisfied in Step 5 with overwhelming probability at some point. By validity of RBC and Lemma G.1.3, P_i eventually adds at least $\kappa - t$ tuples of the form (b_j, j) to S_2 in this manner, with overwhelming probability.

S_3: The argument for S_3 is analogous to the previous one. □

Lemma G.2.4. *If all honest parties P_i in C_1 send $(\mathsf{prepare}_1, i, b)$ in $\mathtt{RBC}_i$, then no honest party adds a tuple $(1 - b, j)$ to S_2, with overwhelming probability.*

Proof. Assume toward a contradiction that all honest parties P_i in C_1 send $(\mathsf{prepare}_1, i, b)$ in $\mathtt{RBC}_i$ and there is an honest party P that adds a tuple $(1-b, j)$ to S_2. This implies that it received $(\mathsf{prepare}_2, j, \hat{S}_{1,j}, 1-b)$ in $\mathtt{RBC}_j$, where $j \in C_2$ and $1 - b$ is the majority value among values in $\hat{S}_{1,j}$, and $|\hat{S}_{1,j}| \geq \kappa - t$. By assumption, $\hat{S}_{1,j} \subseteq S_1$, and so P outputs in all instances of RBC that correspond to $\hat{S}_{1,j}$. By validity and since all honest parties in C_1 send $(\mathsf{prepare}_1, i, b)$ in $\mathtt{RBC}_i$, at least $\kappa - 2t > t$ of the tuples in $\hat{S}_{1,j}$ are of the form (b, j) and at most most t tuples in $\hat{S}_{1,j}$ are of the form $(1 - b, j)$, with overwhelming probability. This contradicts that b is the majority value among values in $\hat{S}_{1,j}$. □

Lemma G.2.5. *If an honest party P_i has $\hat{S}_2$ such that all $(b_j, j) \in \hat{S}_2$, $b_j = b$ for some $b \in \{0, 1\}$, then each honest party P_j has for all $(b_j, j) \in S_3$, $b_j = b$, with overwhelming probability.*

Proof. Let $\hat{S}_2$ be the set of P_i at Step 6 that contains consistently the same value, i.e., such that all $(b_j, j) \in \hat{S}_2$, $b_j = b$.

We argue that the set S_3 of P_j consistently contains b as well, since any set $\hat{S}_{2,k}$ that P_j accepts in Step 7 has the majority bit b.

Assume that there exists a set $\hat{S}_{2,k}$ that has a different majority bit than b. Then, the sets $\hat{S}_2$ of P_i and $\hat{S}_{2,k}$ both have size at least $\kappa - t$. Since there are at most t dishonest parties by Lemma G.1.3, and by validity of

the reliable broadcast, at least $\kappa - 2t$ values came from $\mathsf{prepare}_2$ messages from honest parties.

Since honest parties only send one prepare value, this implies that there are $2\kappa - 4t = 2\kappa/3 \cdot (1 + 2\epsilon)$ distinct honest parties, which is in contradiction with Lemma G.1.3. Hence, the majority bits in all accepted sets $\hat{S}_2^{2,k}$ is the same and the statement follows. □

Lemma G.2.6. Π_{GC} *satisfies graded validity.*

Proof. By Lemma G.2.3, every honest party accumulates a set S_3 of size $\kappa - t$ and hence outputs a value and a grade.

Assume all honest parties have input v. This implies that all honest parties P_i in C_1 send $(\mathsf{prepare}_1, i, v)$. By Lemma G.2.4, no honest parties add $(1 - v, j)$ to S_2. Hence, every honest party outputs $(v, 1)$. □

Lemma G.2.7. *All honest parties generate output in* Π_{GC}.

Proof. This follows from the fact that every honest party eventually accumulates S_3 with size $\kappa - t$ by Lemma G.2.3. □

Lemma G.2.8. Π_{GC} *satisfies graded consistency.*

Proof. Termination is argued in Lemma G.2.7. Let P_i and P_j be two honest parties. Assume that P_i outputs $(v, 1)$. Let $\hat{S}_2^i$ denote the set $\hat{S}_2$ that P_i accumulates in Step 6. Then $\hat{S}_2^i$ contains consistently the same bit. By Lemma G.2.2, P_j cannot output $(1 - v, 1)$. Moreover, by Lemma G.2.5, the set $\hat{S}_3^j$ consistently contains v, and hence P_j outputs $(v, 1)$ or $(v, 0)$. □

G.3 Additional Definitions

G.3.1 Threshold Fully Homomorphic Encryption

In our application, we require simulation security defined below. The definitions follow prior work [Gen09, vGHV10, BV11, AJL+12, BGG+18].

Definition G.3.1. We say a TFHE scheme is simulation secure if there is a probabilistic polynomial-time simulator Sim such that for any probabilistic polynomial-time adversary $\mathcal{A}$, the following experiments are computationally indistinguishable:

$\text{REAL}_{\mathcal{A},C}(1^\kappa, 1^t, 1^N)$:

1. Compute $(\mathsf{ek}, \{\mathsf{dk}_i\}_{i=1}^N) \leftarrow \mathsf{Keygen}(1^\kappa, 1^t, 1^N)$ and give ek to $\mathcal{A}$.
2. $\mathcal{A}$ adaptively chooses a subset $S \subset [N]$ with $|S| < t$ as well as messages $m_1, \ldots, m_n$ and a circuit C. In return, $\mathcal{A}$ is given $\{dk_i\}_{i\in S}$ and $\{c_i \leftarrow \mathsf{Enc}_{\mathsf{ek}}(m_i)\}_{i=1}^n$.
3. $\mathcal{A}$ outputs $\{(m_i', r_i')\}_{i\in S}$. Define $c_i' := \mathsf{Enc}_{\mathsf{ek}}(m_i'; r_i')$ for $i \in S$.
4. Let $c^* := \mathsf{Eval}_{\mathsf{ek}}(\{c_i\}_{i=1}^n, \{c_i'\}_{i\in S})$ and give $\{\mathsf{Dec}_{\mathsf{dk}_i}(c^*)\}_{i\notin S}$ to $\mathcal{A}$.

$\text{IDEAL}_{\mathcal{A},C}(1^\kappa, 1^t, 1^N)$:

1. Compute $\mathsf{ek} \leftarrow \mathsf{Sim}(1^\kappa, 1^t, 1^N)$ and give ek to $\mathcal{A}$.
2. $\mathcal{A}$ adaptively chooses a subset S of parties with $|S| < t$ as well as messages $m_1, \ldots, m_n$ and a circuit C. In return, $\mathsf{Sim}(1^n)$ is run to compute $\{dk_i\}_{i\in S}$ and $\{c_i\}_{i=1}^n$ that are given to $\mathcal{A}$.
3. $\mathcal{A}$ outputs $\{(m_i', r_i')\}_{i\in S}$.
4. Let $y = C(\{m_i\}_{i=1}^n, \{m_i'\}_{i\in S})$. Compute $\{d_i\}_{i\notin S} \leftarrow \mathsf{Sim}(y)$ and give the result to $\mathcal{A}$.

G.3.2 Anonymous Public-Key Encryption

We recall the definition of anonymous public-key encryption [BBDP01].

Definition G.3.2. A CPA-secure public-key encryption scheme $\mathcal{PE} = (\mathsf{Keygen}, \mathsf{Enc}, \mathsf{Dec})$ is $\mathsf{anonymous}$ if the following is negligible for any PPT adversary $\mathcal{A}$:

$$\left|\Pr\left[\begin{array}{c}(\mathsf{pk}_0, \mathsf{sk}_0) \leftarrow \mathsf{Keygen}(1^\kappa); (\mathsf{pk}_1, \mathsf{sk}_1) \leftarrow \mathsf{Keygen}(1^\kappa);\\ m \leftarrow \mathcal{A}(\mathsf{pk}_0, \mathsf{pk}_1); b \leftarrow \{0,1\}; c \leftarrow \mathsf{Enc}_{\mathsf{pk}_b}(m)\end{array} : A(c) = b\right] - \frac{1}{2}\right|.$$

G.4 Proof of Theorem 11.5.4

The claims regarding the communication complexity, size of the setup, and the number of invocations of Byzantine agreement follow because Π_{MPC}^ℓ runs a VACS protocol with ℓ-output quality and inputs of length

$O(\mathcal{I} \cdot \mathsf{poly}(\kappa))$ in step 2; $|C| = O(\kappa)$ (in expectation) reliable broadcast protocols on inputs of length $O(\mathcal{O} \cdot \mathsf{poly}(\kappa))$ in step 4; and a reliable consensus protocol on inputs of length $\mathcal{O}$ in step 5. To prove security of the protocol, we define a simulator $\mathcal{S}$ that works as follows:

Setup: $\mathcal{S}$ generates the setup honestly, with two exceptions:

- $\mathcal{S}$ uses the TFHE simulator to generate the TFHE public key, $t-1$ decryption keys $\{dk_i\}$, and $n+1$ ciphertexts $c_1, \ldots, c_n, c_{\mathsf{rand}}$.
- $\mathcal{S}$ generates the CRS for the UC-NIZK proof system using the corresponding simulator.

In particular, this defines a set C of the parties.

Corruptions: Whenever $\mathcal{A}$ corrupts a party, $\mathcal{S}$ gives $\mathcal{A}$ the state held by that party at that point in time. Let S denote the set of parties that $\mathcal{A}$ corrupts that are (1) in C and (2) corrupted before executing step 2 of the protocol. If $|S| \geq t$ the simulator aborts (call this event abort_1). Otherwise, when $\mathcal{A}$ corrupts the ith party in S, it is given dk_i as part of that party's state.

If $\mathcal{A}$ corrupts a party P_i before P_i has begun executing step 1 of the protocol, then $\mathcal{S}$ corrupts P_i in the ideal world to obtain P_i's input (which it then gives to $\mathcal{A}$ along with P_i's state). When $\mathcal{A}$ corrupts a party at any other point in the protocol, $\mathcal{S}$ delays its corruption of that party until after $\mathcal{S}$ sends $\mathsf{CoreSet}$ to the trusted party (as described below).

Steps 1–2: If P_i is uncorrupted when it is supposed to begin executing step 1 of the protocol, $\mathcal{S}$ begins executing $\Pi_{\mathsf{acs}}^{\ell,Q}$ on behalf of P_i, using c_i and a simulated proof π_i. Let S be the set output by any honest party following the execution of $\Pi_{\mathsf{acs}}^{\ell,Q}$, and define I as in step 1 of the protocol. $\mathcal{S}$ sets $\mathsf{CoreSet} := I$.

Let $\{c'_i\}_{i \in I}$ be the ciphertexts contained in the tuples in S. If $c'_i \neq c_i$ for some party P_i who was not corrupted by $\mathcal{A}$ when P_i began executing $\Pi_{\mathsf{acs}}^{\ell,Q}$, the simulator aborts (call this event abort_2). For each $i \in I$ corresponding to a party P_i who was corrupted by $\mathcal{A}$ when P_i began executing $\Pi_{\mathsf{acs}}^{\ell,Q}$, use the NIZK simulator and π_i to extract the plaintext x'_i corresponding to c'_i. Send $\mathsf{CoreSet}$ and $\{x'_i\}$ to the trusted party. Receive in return an output y.

Step 3–4: Run the TFHE simulator on input y to obtain $\{d_i\}_{i \in C \setminus S}$. For any party $P_i \in C$ that is not corrupted by step 3 of the protocol, $\mathcal{S}$ runs step 3 of the protocol on P_i's behalf using a simulated proof π'_i. Finally, for any party that is not corrupted by step 4 of the protocol, $\mathcal{S}$ runs step 5 on the protocol on that party's behalf using input y.

Note that abort_1 occurs with negligible probability by Lemma G.1.3, and abort_2 occurs with negligible probability by security of the VACS protocol and the signature scheme. Computational indistinguishability of the entire simulation follows straightforwardly.

Bibliography

[ACD+19] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 317–326. ACM, July / August 2019.

[ADD+19] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $o(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[AJL+12] Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In David Pointcheval and Thomas Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Heidelberg, April 2012.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, January 2017.

[ALM17a] Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. In Jonathan Katz and Hovav

Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 447–467. Springer, Heidelberg, August 2017.

[ALM17b] Adi Akavia, Rio LaVigne, and Tal Moran. Topology-hiding computation on all graphs. Cryptology ePrint Archive, Report 2017/296, 2017. `https://eprint.iacr.org/2017/296`.

[AM17] Adi Akavia and Tal Moran. Topology-hiding computation beyond logarithmic diameter. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part III*, volume 10212 of *LNCS*, pages 609–637. Springer, Heidelberg, April / May 2017.

[AMN+19] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. Cryptology ePrint Archive, Report 2019/270, 2019. `https://eprint.iacr.org/2019/270`.

[BBC+19] Marshall Ball, Elette Boyle, Ran Cohen, Tal Malkin, and Tal Moran. Is information-theoretic topology-hiding computation possible? In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 502–530. Springer, Heidelberg, December 2019.

[BBC+20] Marshall Ball, Elette Boyle, Ran Cohen, Lisa Kohl, Tal Malkin, Pierre Meyer, and Tal Moran. Topology-hiding communication from minimal assumptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 473–501. Springer, Heidelberg, November 2020.

[BBDP01] Mihir Bellare, Alexandra Boldyreva, Anand Desai, and David Pointcheval. Key-privacy in public-key encryption. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 566–582. Springer, Heidelberg, December 2001.

[BBMM18] Marshall Ball, Elette Boyle, Tal Malkin, and Tal Moran. Exploring the boundaries of topology-hiding computation.

In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 294–325. Springer, Heidelberg, April / May 2018.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61. ACM Press, May 1993.

[BCLM17] Christian Badertscher, Sandro Coretti, Chen-Da Liu-Zhang, and Ueli Maurer. Efficiency lower bounds for commit-and-prove constructions. In *IEEE International Symposium on Information Theory (ISIT)*, pages 1788–1792, 6 2017.

[Bd90] Jurjen N. Bos and Bert den Boer. Detection of disrupters in the DC protocol. In Jean-Jacques Quisquater and Joos Vandewalle, editors, *EUROCRYPT'89*, volume 434 of *LNCS*, pages 320–327. Springer, Heidelberg, April 1990.

[BDH+17] Brandon Broadnax, Nico Döttling, Gunnar Hartung, Jörn Müller-Quade, and Matthias Nagel. Concurrently composable security with shielded super-polynomial simulators. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 351–381. Springer, Heidelberg, April / May 2017.

[BDHK06] Michael Backes, Markus Dürmuth, Dennis Hofheinz, and Ralf Küsters. Conditional reactive simulatability. In *European Symposium on Research in Computer Security*, pages 424–443. Springer, 2006.

[Bea91] Donald Beaver. Secure multiparty protocols and zero-knowledge proof systems tolerating a faulty minority. *Journal of Cryptology*, 4(2):75–122, January 1991.

[Ben83] Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.

[BEY03] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[BGG+18] Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 565–596. Springer, Heidelberg, August 2018.

[BGP89] Piotr Berman, Juan A Garay, and Kenneth J Perry. Towards optimal distributed consensus. In /, pages 410–415. IEEE, 1989.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[BH93] Donald Beaver and Stuart Haber. Cryptographic protocols provably secure against dynamic adversaries. In Rainer A. Rueppel, editor, *EUROCRYPT'92*, volume 658 of *LNCS*, pages 307–323. Springer, Heidelberg, May 1993.

[BH07] Zuzana Beerliová-Trubíniová and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 376–392. Springer, Heidelberg, December 2007.

[BHMU05] Michael Backes, Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. On fairness in simulatability-based cryptographic systems. In *Proceedings of the 2005 ACM workshop on Formal methods in security engineering*, pages 13–22. ACM, 2005.

[BHN10] Zuzana Beerliová-Trubíniová, Martin Hirt, and Jesper Buus Nielsen. On the theoretical gap between synchronous and asynchronous MPC protocols. In Andréa W. Richa and Rachid Guerraoui, editors, *29th ACM PODC*, pages 211–218. ACM, July 2010.

[BIB89] Judit Bar-Ilan and Donald Beaver. Non-cryptographic fault-tolerant computing in constant number of rounds of inter-

action. In Piotr Rudnicki, editor, *8th ACM PODC*, pages 201–209. ACM, August 1989.

[BKL19] Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150. Springer, Heidelberg, December 2019.

[BKL20] Erica Blum, Jonathan Katz, and Julian Loss. Network-agnostic state machine replication. Cryptology ePrint Archive, Report 2020/142, 2020. `https://eprint.iacr.org/2020/142`.

[BKLL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Heidelberg, November 2020.

[BKR94] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In Jim Anderson and Sam Toueg, editors, *13th ACM PODC*, pages 183–192. ACM, August 1994.

[BLL20] Erica Blum, Chen-Da Liu-Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731. Springer, Heidelberg, August 2020.

[BLPV18] Fabrice Benhamouda, Huijia Lin, Antigoni Polychroniadou, and Muthuramakrishnan Venkitasubramaniam. Two-round adaptively secure multiparty computation from standard assumptions. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 175–205. Springer, Heidelberg, November 2018.

[BMR90] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[BMT18] Christian Badertscher, Ueli Maurer, and Björn Tackmann. On composable security for digital signatures. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 494–523. Springer, Heidelberg, March 2018.

[BPW07] Michael Backes, Birgit Pfitzmann, and Michael Waidner. The reactive simulatability (rsim) framework for asynchronous systems. *Information and Computation*, 205(12):1685–1720, 2007.

[Bra87] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[BT85] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[BV11] Zvika Brakerski and Vinod Vaikuntanathan. Efficient fully homomorphic encryption from (standard) LWE. In Rafail Ostrovsky, editor, *52nd FOCS*, pages 97–106. IEEE Computer Society Press, October 2011.

[Can96] Ran Canetti. *Studies in secure multiparty computation and applications*. PhD thesis, Citeseer, 1996.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[CCGZ16] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Matthew Robshaw and Jonathan

Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 240–269. Springer, Heidelberg, August 2016.

[CCGZ17] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Round-preserving parallel composition of probabilistic-termination cryptographic protocols. In Ioannis Chatzigiannakis, Piotr Indyk, Fabian Kuhn, and Anca Muscholl, editors, *ICALP 2017*, volume 80 of *LIPIcs*, pages 37:1–37:15. Schloss Dagstuhl, July 2017.

[CCL15] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew J. B. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 3–22. Springer, Heidelberg, August 2015.

[CDD+01] Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 262–279. Springer, Heidelberg, May 2001.

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper Buus Nielsen. Multiparty computation from threshold homomorphic encryption. In Birgit Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–299. Springer, Heidelberg, May 2001.

[CDPW07] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 61–85. Springer, Heidelberg, February 2007.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 19–40. Springer, Heidelberg, August 2001.

[CFF+05] Jeffrey Considine, Matthias Fitzi, Matthew K. Franklin, Leonid A. Levin, Ueli M. Maurer, and David Metcalf. Byzan-

tine agreement given partial broadcast. *Journal of Cryptology*, 18(3):191–217, July 2005.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In *28th ACM STOC*, pages 639–648. ACM Press, May 1996.

[CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multi-party computation based on one-way functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 998–1021. Springer, Heidelberg, December 2016.

[Cha81] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[Cha88] David Chaum. The dining cryptographers problem: Unconditional sender and recipient untraceability. *Journal of Cryptology*, 1(1):65–75, January 1988.

[CHL21] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous mpc with adaptive security. In *TCC*, 2021.

[Cho20] Ashish Choudhury. Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited. Cryptology ePrint Archive, Report 2020/906, 2020. `https://eprint.iacr.org/2020/906`.

[CHP12] Ashish Choudhury, Martin Hirt, and Arpita Patra. Unconditionally secure asynchronous multiparty computation with linear communication complexity. Cryptology ePrint Archive, Report 2012/517, 2012. `https://eprint.iacr.org/2012/517`.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, Heidelberg, August 2001.

[CKS00] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *19th ACM PODC*, pages 123–132. ACM, July 2000.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, July 2005.

[CKS20] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a COINcidence: Sub-quadratic asynchronous Byzantine agreement WHP, 2020. Available at `https://arxiv.org/abs/2002.06545`.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[CL17] Ran Cohen and Yehuda Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. *Journal of Cryptology*, 30(4):1157–1186, October 2017.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In *18th ACM STOC*, pages 364–369. ACM Press, May 1986.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[CLTV15] Ran Canetti, Huijia Lin, Stefano Tessaro, and Vinod Vaikuntanathan. Obfuscation of probabilistic circuits and applications. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 468–497. Springer, Heidelberg, March 2015.

[CM16] Jing Chen and Silvio Micali. Algorand. *arXiv preprint arXiv:1607.01341*, 2016.

[CNE+14] Stephen Checkoway, Ruben Niederhagen, Adam Everspaugh, Matthew Green, Tanja Lange, Thomas Ristenpart, Daniel J Bernstein, Jake Maskiewicz, Hovav Shacham, Matthew Fredrikson, et al. On the practical exploitability of dual ec in tls implementations. In *USENIX security symposium*, pages 319–335, 2014.

[Coh16] Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *PKC 2016, Part II*, volume 9615 of *LNCS*, pages 183–207. Springer, Heidelberg, March 2016.

[CP15] Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous MPC with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*, pages 1–10, 2015.

[CPR17] Ashish Choudhury, Arpita Patra, and Divya Ravi. Round and communication efficient unconditionally-secure MPC with t<n / 3 in partially synchronous network. In *ICITS 2017*, 2017.

[CPV17] Ran Canetti, Oxana Poburinnaya, and Muthuramakrishnan Venkitasubramaniam. Equivocating yao: constant-round adaptively secure multiparty computation in the plain model. In Hamed Hatami, Pierre McKenzie, and Valerie King, editors, *49th ACM STOC*, pages 497–509. ACM Press, June 2017.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51. ACM Press, May 1993.

[CsW19] Ran Cohen, abhi shelat, and Daniel Wichs. Adaptively secure MPC with sublinear communication complexity. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part II*, volume 11693 of *LNCS*, pages 30–60. Springer, Heidelberg, August 2019.

[Dam00] Ivan Damgård. Efficient concurrent zero-knowledge in the auxiliary string model. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 418–430. Springer, Heidelberg, May 2000.

[DDO+01] Alfredo De Santis, Giovanni Di Crescenzo, Rafail Ostrovsky, Giuseppe Persiano, and Amit Sahai. Robust non-interactive zero knowledge. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 566–598. Springer, Heidelberg, August 2001.

[DDWY90] Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. In *31st FOCS*, pages 36–45. IEEE Computer Society Press, October 1990.

[DHL21] Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient synchronous byzantine agreement and multiparty computation with asynchronous fallback. In *TCC*, 2021.

[DI05] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Heidelberg, August 2005.

[DJ01] Ivan Damgård and Mats Jurik. A generalisation, a simplification and some applications of Paillier's probabilistic public-key system. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 119–136. Springer, Heidelberg, February 2001.

[DKMR05] Anupam Datta, Ralf Küsters, John C. Mitchell, and Ajith Ramanathan. On the relationships between notions of simulation-based security. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 476–494. Springer, Heidelberg, February 2005.

[DKR15] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In Yevgeniy Dodis and Jes-

per Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 586–613. Springer, Heidelberg, March 2015.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a Byzantine environment: crash failures. *Information and Computation*, 88(2):156–186, 1990.

[DM00] Yevgeniy Dodis and Silvio Micali. Parallel reducibility for information-theoretically secure computation. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 74–92. Springer, Heidelberg, August 2000.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In Moti Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 581–596. Springer, Heidelberg, August 2002.

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 247–264. Springer, Heidelberg, August 2003.

[DNS98] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. In *30th ACM STOC*, pages 409–418. ACM Press, May 1998.

[Dol82] Danny Dolev. The byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM*, 32(1):191–204, 1985.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In Elizabeth

Borowsky and Sergio Rajsbaum, editors, *22nd ACM PODC*, pages 211–220. ACM, July 2003.

[FGH+02] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. Detectable Byzantine agreement secure against faulty majorities. In Aleta Ricciardi, editor, *21st ACM PODC*, pages 118–126. ACM, July 2002.

[FHHW03] Matthias Fitzi, Martin Hirt, Thomas Holenstein, and Jürg Wullschleger. Two-threshold broadcast and detectable multi-party computation. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 51–67. Springer, Heidelberg, May 2003.

[FHW04] Matthias Fitzi, Thomas Holenstein, and Jürg Wullschleger. Multi-party computation with hybrid security. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 419–438. Springer, Heidelberg, May 2004.

[FLL21] Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. Expand-and-extract: A new way of designing round-efficient byzantine agreement. In *PODC*, 2021.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[FM88] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

[FM97] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[GHK+21] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. Yoso:

You only speak once / secure mpc with stateless ephemeral roles. Cryptology ePrint Archive, Report 2021/210, 2021. https://eprint.iacr.org/2021/210.

[GHL20] Diana Ghinea, Martin Hirt, and Chen-Da Liu-Zhang. From partial to global asynchronous reliable broadcast. In *International Symposium on Distributed Computing — DISC 2020*, 2020.

[GJ04] Philippe Golle and Ari Juels. Dining cryptographers revisited. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 456–473. Springer, Heidelberg, May 2004.

[GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, October 2007.

[GKKZ11a] Juan A Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 179–186, 2011.

[GKKZ11b] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *30th ACM PODC*, pages 179–186. ACM, June 2011.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred J. Menezes and Scott A. Vanstone, editors, *CRYPTO'90*, volume 537 of *LNCS*, pages 77–93. Springer, Heidelberg, August 1991.

[GL02] Shafi Goldwasser and Yehuda Lindell. Secure computation without agreement. In *Proceedings of the 16th International Conference on Distributed Computing*, DISC '02, page 1732, Berlin, Heidelberg, 2002. Springer-Verlag.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[Gol02] Oded Goldreich. Concurrent zero-knowledge with timing, revisited. In *34th ACM STOC*, pages 332–340. ACM Press, May 2002.

[GP90] Oded Goldreich and Erez Petrank. The best of both worlds: Guaranteeing termination in fast randomized Byzantine agreement protocols. Technical report, Computer Science Department, Technion, 1990.

[GP15] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part II*, volume 9015 of *LNCS*, pages 614–637. Springer, Heidelberg, March 2015.

[GPS19] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 499–529. Springer, Heidelberg, August 2019.

[GRR98] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and fast-track multiparty computations with applications to threshold cryptography. In Brian A. Coan and Yehuda Afek, editors, *17th ACM PODC*, pages 101–111. ACM, June / July 1998.

[GS12] Sanjam Garg and Amit Sahai. Adaptively secure multi-party computation with dishonest majority. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 105–123. Springer, Heidelberg, August 2012.

[GWZ09] Juan A. Garay, Daniel Wichs, and Hong-Sheng Zhou. Somewhat non-committing encryption and efficient adaptively secure oblivious transfer. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 505–523. Springer, Heidelberg, August 2009.

[HDWH12] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *USENIX Security Symposium*, volume 8, page 1, 2012.

[HJ07] Markus Hinkelmann and Andreas Jakoby. Communications in unknown networks: Preserving the secret of topology. *Theoretical Computer Science*, 384(2-3):184–200, 2007.

[HKL20] Martin Hirt, Ard Kastrati, and Chen-Da Liu-Zhang. Multi-threshold asynchronous reliable broadcast and consensus. In *International Conference on Principles of Distributed Systems — OPODIS 2020*, 2020.

[HLM13] Martin Hirt, Christoph Lucas, and Ueli Maurer. A dynamic tradeoff between active and passive corruptions in secure multi-party computation. In Ran Canetti and Juan A. Garay, editors, *CRYPTO 2013, Part II*, volume 8043 of *LNCS*, pages 203–219. Springer, Heidelberg, August 2013.

[HLM21] Martin Hirt, Chen-Da Liu-Zhang, and Ueli Maurer. Adaptive security of multi-party protocols, revisited. In *TCC*, 2021.

[HM00] Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, January 2000.

[HM04] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. In *Proceedings of FCS*, pages 117–130. Citeseer, 2004.

[HMTZ16] Martin Hirt, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Network-hiding communication and applications to multi-party protocols. In Matthew Robshaw and Jonathan Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 335–365. Springer, Heidelberg, August 2016.

[HNP05] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 322–340. Springer, Heidelberg, May 2005.

[HNP08] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz, editors, *ICALP 2008, Part II*, volume 5126 of *LNCS*, pages 473–485. Springer, Heidelberg, July 2008.

[HUM13] Dennis Hofheinz, Dominique Unruh, and Jörn Müller-Quade. Polynomial runtime and composability. *Journal of Cryptology*, 26(3):375–441, July 2013.

[HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, Heidelberg, May / June 2010.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 483–500. Springer, Heidelberg, August 2006.

[IOZ14] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. Secure multi-party computation with identifiable abort. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Heidelberg, August 2014.

[JM20] Daniel Jost and Ueli Maurer. Overcoming impossibility results in composable security using interval-wise guarantees. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 33–62. Springer, Heidelberg, August 2020.

[Kat07] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In David S. Johnson and Uriel Feige, editors, *39th ACM STOC*, pages 11–20. ACM Press, June 2007.

[KK06] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Heidelberg, August 2006.

[KLP05] Yael Tauman Kalai, Yehuda Lindell, and Manoj Prabhakaran. Concurrent general composition of secure protocols in the timing model. In Harold N. Gabow and Ronald Fagin, editors, *37th ACM STOC*, pages 644–653. ACM Press, May 2005.

[KLR06] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 109–118. ACM Press, May 2006.

[KMTZ13] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Heidelberg, March 2013.

[KS10] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable Byzantine agreement with an adaptive adversary. In Andréa W. Richa and Rachid Guerraoui, editors, *29th ACM PODC*, pages 420–429. ACM, July 2010.

[KSSV06] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *17th SODA*, pages 990–999. ACM-SIAM, January 2006.

[KTR20] Ralf Küsters, Max Tuengerthal, and Daniel Rausch. The IITM model: a simple and expressive model for universal composability. *Journal of Cryptology*, 33(4):1461–1584, 2020.

[Kur00] Klaus Kursawe. *Optimistic asynchronous Byzantine agreement*. IBM Thomas J. Watson Research Division, 2000.

[Kur02] Klaus Kursawe. Optimistic Byzantine agreement. In *21st IEEE Symposium on Reliable Distributed Systems, 2002. Proceedings.*, pages 262–267. IEEE, 2002.

[KZZ16] Aggelos Kiayias, Hong-Sheng Zhou, and Vassilis Zikas. Fair and robust multi-party computation using a global transaction ledger. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 705–734. Springer, Heidelberg, May 2016.

[LJY14] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014.

[LLM+18] Rio LaVigne, Chen-Da Liu-Zhang, Ueli Maurer, Tal Moran, Marta Mularczyk, and Daniel Tschudi. Topology-hiding computation beyond semi-honest adversaries. In Amos Beimel and Stefan Dziembowski, editors, *TCC 2018, Part II*, volume 11240 of *LNCS*, pages 3–35. Springer, Heidelberg, November 2018.

[LLM+20a] Rio LaVigne, Chen-Da Liu-Zhang, Ueli Maurer, Tal Moran, Marta Mularczyk, and Daniel Tschudi. Topology-hiding computation for networks with unknown delays. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 215–245. Springer, Heidelberg, May 2020.

[LLM+20b] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and

asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 92–119. Springer, Heidelberg, December 2020.

[LLR02] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In Aleta Ricciardi, editor, *21st ACM PODC*, pages 203–212. ACM, July 2002.

[LM18] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018. `https://eprint.iacr.org/2018/235`.

[LM20a] David Lanzenberger and Ueli Maurer. Coupling of random systems. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 207–240. Springer, Heidelberg, November 2020.

[LM20b] Chen-Da Liu-Zhang and Ueli Maurer. Synchronous constructive cryptography. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part II*, volume 12551 of *LNCS*, pages 439–472. Springer, Heidelberg, November 2020.

[LMM20] Chen-Da Liu-Zhang, Varun Maram, and Ueli Maurer. On broadcast in generalized network and adversarial models. In *International Conference on Principles of Distributed Systems — OPODIS 2020*, 2020.

[LMRT17] Chen-Da Liu-Zhang, Ueli Maurer, Martin Raszyk, and Daniel Tschudi. Witness-hiding proofs of knowledge for cable locks. In *IEEE International Symposium on Information Theory (ISIT)*, pages 953–957, 6 2017.

[LP01] Anna Lysyanskaya and Chris Peikert. Adaptive security in the threshold setting: From cryptosystems to signature schemes. In Colin Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 331–350. Springer, Heidelberg, December 2001.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[LVC+16] Shengyun Liu, Paolo Viotti, Christian Cachin, Vivien Quéma, and Marko Vukolić. Xft: Practical fault tolerance beyond crashes. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 485–500, 2016.

[Mau02] Ueli Maurer. Indistinguishability of random systems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 110–132. Springer, 2002.

[Mau06] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[Mau11] Ueli Maurer. Constructive cryptography - a new paradigm for security definitions and proofs. In *In TOSCA*, pages 33,56, 2011.

[MHR14] Achour Mostéfaoui, Moumen Hamouma, and Michel Raynal. Signature-free asynchronous Byzantine consensus with $t < n/3$ and $O(n^2)$ messages. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 2–9. ACM, July 2014.

[Mic17] Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.

[MNR19] Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible Byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1041–1053, 2019.

[MOR15] Tal Moran, Ilan Orlov, and Silas Richelson. Topology-hiding computation. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *TCC 2015, Part I*, volume 9014 of *LNCS*, pages 159–181. Springer, Heidelberg, March 2015.

[MPR07] Ueli M. Maurer, Krzysztof Pietrzak, and Renato Renner. Indistinguishability amplification. In Alfred Menezes, editor, *CRYPTO 2007*, volume 4622 of *LNCS*, pages 130–149. Springer, Heidelberg, August 2007.

[MR92] Silvio Micali and Phillip Rogaway. Secure computation (abstract). In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 392–404. Springer, Heidelberg, August 1992.

[MR11] Ueli Maurer and Renato Renner. Abstract cryptography. In *In Innovations in Computer Science*. Citeseer, 2011.

[MR16] Ueli Maurer and Renato Renner. From indifferentiability to constructive cryptography (and back). In Martin Hirt and Adam D. Smith, editors, *TCC 2016-B, Part I*, volume 9985 of *LNCS*, pages 3–24. Springer, Heidelberg, October / November 2016.

[MT88] Yoram Moses and Mark R Tuttle. Programming simultaneous actions using common knowledge. *Algorithmica*, 3(1):121–169, 1988.

[MT13] Daniele Micciancio and Stefano Tessaro. An equational approach to secure multi-party computation. In Robert D. Kleinberg, editor, *ITCS 2013*, pages 355–372. ACM, January 2013.

[MV17] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. Technical report, MIT, 2017.

[MXC+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 31–42. ACM Press, October 2016.

[Nie03] Jesper Buus Nielsen. *On protocol security in the cryptographic model*. BRICS, 2003.

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238. Springer, Heidelberg, May 1999.

[Pas03] Rafael Pass. Simulation in quasi-polynomial time, and its application to protocol composition. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 160–176. Springer, Heidelberg, May 2003.

[PCR08] Arpita Patra, Ashish Choudhury, and C. Pandu Rangan. Efficient asynchronous multiparty computation with optimal resilience. Cryptology ePrint Archive, Report 2008/425, 2008. `https://eprint.iacr.org/2008/425`.

[PCR09] Arpita Patra, Ashish Choudhary, and C Pandu Rangan. Communication efficient statistical asynchronous multiparty computation with optimal resilience. In *International Conference on Information Security and Cryptology*, pages 179–197. Springer, 2009.

[PR18] Arpita Patra and Divya Ravi. On the power of hybrid networks in multi-party computation. *IEEE Trans. Information Theory*, 2018.

[PS04] Manoj Prabhakaran and Amit Sahai. New notions of security: Achieving universal composability without trusted setup. In László Babai, editor, *36th ACM STOC*, pages 242–251. ACM Press, June 2004.

[PS17a] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 91. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[PS17b] Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT 2017, Part II*, volume 10625 of *LNCS*, pages 380–409. Springer, Heidelberg, December 2017.

[PS18] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33. Springer, Heidelberg, April / May 2018.

[PSR02] B. Prabhu, K. Srinathan, and C. Pandu Rangan. Asynchronous unconditionally secure computation: An efficiency improvement. In Alfred Menezes and Palash Sarkar, editors, *INDOCRYPT 2002*, volume 2551 of *LNCS*, pages 93–107. Springer, Heidelberg, December 2002.

[PW00] Birgit Pfitzmann and Michael Waidner. *Composition and integrity preservation of secure reactive systems*. IBM Thomas J. Watson Research Division, 2000.

[Rab83] Michael O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409. IEEE Computer Society Press, November 1983.

[Ram20] Matthieu Rambaud. Lower bounds for authenticated randomized Byzantine consensus under (partial) synchrony: The limits of standalone digital signatures, 2020. Available at `https://perso.telecom-paristech.fr/rambaud/articles/lower.pdf`.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[RC88] MK Reiter and RA Crowds. Anonymity for web transaction. *ACM Transactions on Information and System Security*, pages 66–92, 1988.

[SGR97] Paul F Syverson, David M Goldschlag, and Michael G Reed. Anonymous connections and onion routing. In *Security and Privacy, 1997. Proceedings., 1997 IEEE Symposium on*, pages 44–54. IEEE, 1997.

[Sho00] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Heidelberg, May 2000.

[SR00] K. Srinathan and C. Pandu Rangan. Efficient asynchronous secure multiparty distributed computation. In Bimal K. Roy and Eiji Okamoto, editors, *INDOCRYPT 2000*, volume 1977 of *LNCS*, pages 117–129. Springer, Heidelberg, December 2000.

[TC84] Russell Turpin and Brian A Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[vGHV10] Marten van Dijk, Craig Gentry, Shai Halevi, and Vinod Vaikuntanathan. Fully homomorphic encryption over the integers. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 24–43. Springer, Heidelberg, May / June 2010.

[Wik16] Douglas Wikström. Simplified universal composability framework. In Eyal Kushilevitz and Tal Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 566–595. Springer, Heidelberg, January 2016.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, October 1986.