Studies on Full Security in Multiparty Computation

Ran Cohen

Department of Computer Science

Ph.D. Thesis

Submitted to the Senate of Bar-Ilan University

Ramat-Gan, Israel June, 2016

This work was carried out under the supervision of

Prof. Yehuda Lindell

Department of Computer Science, Bar-Ilan University

Acknowledgements

First and foremost, I would like to thank my advisor, Prof. Yehuda Lindell, who has been an inspiration before, throughout and even after the conclusion of my graduate studies. As a researcher and a teacher Yehuda is an example of striving for excellence while keeping true to his internal integrity. It is always a tremendous pleasure to talk with Yehuda and discover his curiosity and knowledgability. I have been very fortunate to have worked with Yehuda and learned so much from him. However, an advisor is much more than a scholar, and indeed one of the great beauties of Yehuda is that his non-compromising strive for excellence never comes at the expanse of his kindness and patience. The path for a Ph.D. is filled with ups and downs – Yehuda always knows when to say a kind word that puts the winds in the sails again. I will always be grateful to Yehuda for his endless support and continuous faith. Thank you very much.

Any work becomes easier when working in a nice environment. I was privileged to be a part of the Cryptography Research Group at Bar-Ilan University. I would like to thank the past and present members of the group for a very enjoyable time. Benny Pinkas who is with no doubt the kindest cryptographer I've met, Carmit Hazay with the perpetual smile and endless energy. The (no longer) post-docs: Rafi Chen, Eran Omri, Claudio Orlandi, and Ben Riva. My fellow graduate students and office mates: Gilad Asharov, Ariel Nof, Omer Shlomovits, Erez Waisbard, Avishay Yanai, and Hila Zarosim. The master students: Asaf Cohen, Tali Oberman, Eli Oxman, and Or Weinstein. The software team: Assi Barak, Yael Ejgenberg, Moriya Farbstein, Asaf Kleinbort, Lior Koskas, and Meital Levy. Tal Malkin and Kobbi Nissim who visited us. Moti Geva and Nehora Krolzig. And most importantly Yonit Homburger who is the living force behind the group. Thank you all for most entertaining talks on cryptography, mathematics, life and good places to have lunch.

During my studies I was also very fortunate to join a very warm and welcoming research community. I would like to especially thank Ran Canetti, Iftach Haitner, Yuval Ishai, Juan Garay, Tal Malkin, Benny Pinkas, and Tal Rabin – each of you has inspired me and helped me throughout the way both in your willingness to share your knowledge and more importantly in giving me good advice and guidance. You are true role models.

One of the nicest things about research is the ability to collaborate with other colleagues; a special thanks to my co-authors for most fruitful discussions: Sandro Coretti, Juan Garay, Iftach Haitner, Eran Omri, Chris Peikert, Lior Rotem, and Vassilis Zikas. I have learned a lot from each and every one of you.

Looking back, when I started my studies I was a happy single man, and now, when the journey has completed, I am even happier with my wonderful family. My deepest gratitude goes to my wife Elise, for the endless love and encouragement, for taking the burden in many sleepless nights before deadlines, and for being my partner throughout these special times. My

graduate studies are divided into two parts: before and after my son, Daniel, was born. It has been great pride and joy to see Daniel grow up. My growth as a scholar is intertwined with your growth as a person, as apparently, I only know how to publish papers with you in my life. Finally, my daughter, Noa, and her smile have been a boost of joy over the last three months. I dedicate this thesis to my family.

Ran Cohen June, 2016

Contents

Abstract

In the setting of secure multiparty computation, a set of mutually distrusting parties wish to jointly compute a function of their inputs in a secure manner. The computation should preserve security properties such as privacy (no party should learn anything more than its prescribed output), correctness (each party is guaranteed to receive a correct output), independence of inputs (the corrupted parties must choose their inputs independently of the honest parties' inputs), fairness (if one party receives output then all parties receive output), and guaranteed output delivery (corrupted parties should not be able to prevent honest parties from receiving their output). Loosely speaking, a protocol has full security if it satisfies all of the above requirements.

Secure computation provides a general and strong notion of security, and essentially every distributed task can be computed securely. Impressive feasibility results have been established (under appropriate assumptions): If a majority of the parties are honest, every function can be computed with full security, when the parties can communicate over a broadcast channel. In the special case where more than two-thirds of the parties are honest, broadcast is not required. If an honest majority is not assumed, every function can be computed with privacy, correctness and independence of inputs, but without fairness and guaranteed output delivery.

The aim of this thesis is to study full security in multiparty computations, both in terms of *feasibility*, i.e., when full security can be achieved and when it is impossible, and in terms of *efficiency*, i.e., how many resources, such as round complexity, are needed in order to achieve full security.

The first part of this thesis explores the feasibility of full security. The main results presented are as follows:

1. We analyze the relation between fairness and guaranteed output delivery and show that in the multiparty setting these properties are the same if parties can communicate over a broadcast channel, but distinct in general. More specifically, we show the existence of non-trivial functions for which complete fairness is achievable (without an honest majority) but guaranteed output delivery is not, and the existence of non-trivial functions for which complete fairness and guaranteed output delivery are achievable (without broadcast).

2. We provide a complete characterization of public-output functionalities that can be computed with full security, when all parties communicate over a point-to-point network, but no broadcast channel (nor a secure setup phase) is available:

   - An n-party functionality can be computed with full security in the presence of $n/3 \leq t < n/2$ corruptions (i.e., honest majority) if and only if every subset of $(n - 2t)$ parties can choose input values that determine the output of the function.

- Assuming the existence of one-way functions, an n-party functionality can be securely computed as above in the presence of $t \geq n/2$ corruptions (i.e., no honest majority) if and only if every party can choose an input value that determines the output and, in addition, the function can be securely computed with broadcast.

The second part of this thesis focuses on the efficiency cost, in terms of round complexity, of achieving full security. It is well known that every function can be computed with perfect and full security (i.e., unconditionally and without any error probability) in the presence of $t < n/3$ corruptions using secure point-to-point channels, and moreover, in the broadcast model, the round complexity is linear only in the depth of the circuit representing the function. An analogue result was not known in the point-to-point model, since instantiating a broadcast channel using "fast" protocols results with probabilistic and non-simultaneous termination rounds, and introduces numerous composition problems.

We put forth the first simulation-based treatment of multiparty protocols with probabilistic termination. We define (fully) secure multiparty computation with probabilistic termination in the UC framework, and prove a universal composition theorem for probabilistic-termination protocols. We showcase our definitions and compiler by providing the first composable protocols (with simulation-based security proofs) for the following primitives, relying on point-to-point channels:

1. Expected-constant-round perfect Byzantine agreement.
2. Expected-constant-round perfect parallel broadcast.
3. Multiparty computation with perfect and full security, with round complexity independent of the number of parties.

Chapter 1

Introduction

1.1 Background

There is a new terrorist threat over the world. Several intelligence agencies try to prevent the attack. Each of the agencies holds some secret information (the time of the attack, the location, which terror organization is planing the attack, and so on) but cannot stop the attack on its own. It is essential that the information each intelligence agency managed to gain will remain private, as any unnecessary leakage might risk the sources of the data. Moreover, it is known that the terrorists have double agents inside some of the agencies, and so each intelligence agency cannot trust the rest. Can the agencies join forces and stop the attack?

The scenario described above demonstrates the power of *secure multiparty computation*. By using secure protocols the intelligence agencies can perform computations over their private data without exposing any information but the outcome of the computation. The aim of this dissertation is to study a strong security notion, called full security, which informally means, in the example above, that the terrorists cannot prevent the intelligence agencies from stopping the attack nor gain any information about the sources of the data.

1.1.1 Secure Multiparty Computation

In the setting of secure multiparty computation, a set of mutually distrusting parties wish to jointly compute a function of their inputs in a "secure" manner. In order to carry out this joint computation, the parties execute an interactive protocol amongst themselves. In this sense, security means that even when a subset of dishonest parties collude, they will not be able to interfere with the computation: the colluding parties will not be able to learn the honest parties' inputs (beyond what is revealed by the output), nor to affect the result of the computation, thus causing the honest parties to obtain an invalid output. This threat is normally modeled by a central adversarial entity, which may corrupt a subset of parties and control them.

Secure computation is a very general notion, and in a sense, anything that can be computed can also be computed in a secure manner. Examples where secure protocols are used range from basic distributed-computing tasks such as coin flipping (where all parties agree on a common unbiased bit) and broadcast (where a sender distributes its message to all the parties) and more involved computations such as online auctions (where each party bids and the highest bidder

wins the auction) and the example described above of information sharing.

Intuitively, a secure protocol should provide several security properties. Below we list some of the security requirements, and illustrate them by using the example of an online auction, where the input of each party is its bid and the joint output is the highest bid. Loosely speaking, we say that a protocol has full security if it satisfies all of the requirements described below.

Privacy: No party should learn any additional information other than its prescribed output. In particular, the adversary should not learn any information about the honest parties' inputs other than what can be derived from corrupted parties' inputs and the output of the computation. In the auction example, the meaning of privacy is that although each party learns the highest bid, and thus an upper bound on all the inputs, no other information is learned on individual inputs.

Correctness: It is guaranteed that if a party receives an output, then the output is correct. In other words, the only way the adversary can affect the output of the computation is by choosing the input values for the corrupted parties. Continuing with the example, correctness means that a party cannot win the auction without providing the highest bid.

Independence of Inputs: Corrupted parties must choose their inputs independently of the honest parties' inputs. The meaning in the auction example is that corrupted parties cannot choose their bids as a function of the honest parties' bids, e.g., the highest honest bid plus 1. Note that despite the similarity of this requirement to privacy, there is a significant difference, which can be illustrated in the following scenario.

Say party A encrypts its bid x using some homomorphic encryption scheme (where ciphertexts can be "added" to obtain an encryption of the sum of the plaintexts). Party B cannot learn anything about x (as long as the encryption scheme is semantically secure), nonetheless, B can generate an encryption of $x + 1$, and so claim to have a higher bid than A without breaching privacy, i.e., without obtaining any additional information on A's actual bid.

Fairness: If one party obtains output then all parties obtain the output. In particular, if the corrupted parties get output, they cannot prevent the honest parties from receiving their output. Note that fairness does not rule out scenarios where the adversary prevents honest parties from receiving output, as long as corrupted parties also do not get output. In the auction example, fairness means that the adversary cannot first learn the highest bid (i.e., the output), and decide based on this information whether to allow the computation to complete or to prematurely abort the protocol.

Guaranteed Output Delivery: The honest parties are guaranteed to receive their output, regardless of the actions taken by the adversary. This requirement is very similar to fairness, as in both cases corrupted parties do not gain any advantage over honest parties. However, if only fairness is required the adversary can carry out "denial of service" attacks, in which the computation is aborted repeatedly, without any party learning the output. If guaranteed output delivery is required, such attacks cannot take place, and honest parties are promised to receive output eventually. In the auction example, guaranteed output delivery means that the adversary cannot prevent the honest parties from executing the auction, and receiving their output.

The above list does not constitute a security definition but merely a "check list" of security requirements. Using the simulation paradigm a central security definition can simultaneously capture all of these requirements (and more).

1.1.2 The Simulation Paradigm

Standard security definitions for multiparty computation are based on the *simulation paradigm*. Consider an ideal world, where an incorruptible trusted party is willing to carry out the computation for the parties. Each party hands its input to the trusted party (over a perfectly secure channel), the trusted party computes the function and sends the corresponding output to each party. Now consider the real world (in which no such trusted party exist), where parties execute a protocol in order to compute the function. The execution of the protocol in the real world is compared to the ideal-world computation. The requirement is that the computation in the real world behaves just like the computation in the ideal world, i.e., that no external distinguisher, that provides inputs to the parties and receives their outputs, can tell if it is interacting with parties in the real world or in the ideal world.

The ideal computation is secure by definition, as there is no interaction whatsoever between the parties. When an adversary corrupts a subset of parties in the ideal world, all it can do is to alter the inputs and outputs of these parties and nothing more. A protocol for computing a function in the real world is considered secure if whatever an adversary, controlling some subset of the parties, can obtain in a real execution of the protocol, can also be obtained in the ideal world where the parties communicate with a trusted party. The underlying idea is that if every attack in the real-world execution of the protocol can be simulated in the ideal world, and no meaningful attack can take place in the ideal world, then the conclusion is that no meaningful attack can be carried out on the protocol in the real world.

1.1.3 Adversarial Power

Normally, there is a clear tradeoff between security of protocols on the one hand and their feasibility/efficiency on the other hand. Therefore, a key ingredient in a design of any cryptographic protocol is defining the power of the adversary. The weaker the adversary is assumed to be, the stronger the feasibility (and the efficiency) of computation becomes, and vice versa, the stronger the adversary is, the weaker becomes the feasibility of computation.

There are several aspects to consider regarding the adversary's power:

- The adversary's complexity: this parameter relates to the running time of the adversary. Two possibilities are normally considered:
  - **All-powerful adversaries**: this is the so called *information-theoretic model*, where the running time of the adversary is not limited. No cryptographic hardness assumptions can be used in this model. Traditionally, in this setting the parties communicate over perfectly secure channels, where the adversary cannot eavesdrop nor tamper with the communication between honest parties.

    In this setting, it is customary to distinguish between *perfect security* where the error probability is zero, and *statistical security*, where a negligible error probability is allowed.

  - **Efficient adversaries**: this is the so called *computational model*, where the adversary is restricted to run in probabilistic polynomial time. Protocols in this setting are normally based on certain cryptographic assumptions, such as the existence of oblivious transfer. There is no need for secure channels in this model but merely of authenticated channels, as the communication between honest parties can be implemented using public-key encryption.

- The adversarial behaviour: this parameter relates to the behaviour that the adversary is allowed to do once corrupting a party.

  - **Semi-honest adversaries**: the corrupted parties follow the prescribed protocol specification as honest parties. However, the adversary can collect the internal states of all corrupted parties in order to try and learn additional information.
  - **Fail-Stop adversaries**: the corrupted parties follow the prescribed protocol specification as honest parties. The difference from the semi-honest setting is that the adversary may instruct a corrupted party to "crash" and stop sending messages.
  - **Malicious adversaries**: in this case there are no restrictions on the behaviour of the adversary, and it can instruct the corrupted parties to deviate from the protocol in any arbitrary way it chooses.

- Corruption strategy: this parameter relates to the strategy in which the adversary can corrupt the parties.

  - **Static adversaries**: the adversary decides which parties to corrupt before the execution of the protocol begins. In particular, honest parties remain honest throughout the course of protocol's execution.
  - **Adaptive adversaries**: the adversary can dynamically corrupt any party it wants during the execution of the protocol. The decision as to *which* party to corrupt and *when* is based on information gathered by the adversary, i.e., messages communicated between parties (over insecure channels) and internal states of parties that have been corrupted thus far.

- Number of corruptions: this parameter relates to (the upper bound of) the number of parties the adversary can corrupt. Normally, there is a difference between the case where strictly less than a third of the parties might be corrupted, the case where an honest majority is assumed and the case where the adversary can corrupt an arbitrary number of parties.

- The environment in which the protocol is executed: in the stand-alone setting, a single protocol is executed at any point in time, and the adversary can attack the protocol using only information it gathers from the protocol's execution. When considering composition of protocols, the adversary can use information that it collects from executions of different protocols by different sets of parties.

  Various types of protocol composition are considered: *sequential composition* (where protocols are executed one after the other), *parallel composition* (where all protocols are executed simultaneously round-by-round) and *concurrent composition* (where the adversary can start new protocols at any time, and proceed at its own pace, in order to attack a

specific protocol execution). Another classification of protocol composition considers *self composition* (where a single set of parties execute several independent copies of a single protocol) and *general composition* (where many sets of parties carry out many executions of arbitrary protocols).

Looking ahead, throughout this thesis we will use various combinations of the above properties. In general, it is preferred to consider *weak* adversaries when proving lower bounds, and *strong* adversaries when constructing protocols. For example, in Part I we present impossibility results of full security, and we consider static, malicious, polynomial-time adversaries attacking stand-alone protocols communicating over secure channels (the reason we consider malicious adversaries is that, as we show, these impossibility results do not hold for the weaker notion of fail-stop adversaries). In Part II, we construct perfect and fully secure protocols and prove their security facing very strong adversaries, namely, adaptive, malicious, computationally unbounded adversaries in the universal composition (UC) framework, in which security holds even under concurrent general composition.

1.1.4 Feasibility of Secure Computation

Impressive feasibility results have been established for secure protocols. We informally list the main feasibility results below.

- When more than two-thirds of the parties are honest, every function can be computed with full security in a point-to-point network.
  - In the computational setting, where the adversary runs in polynomial-time and the communication channels are authenticated, this holds under appropriate cryptographic assumptions (e.g., the existence of oblivious transfer), in the stand-alone setting facing static, malicious adversaries; see Goldreich et al. [53]. Under stronger cryptographic assumptions, this result holds under concurrent general composition and facing adaptive, malicious adversaries; see Canetti et al. [23].
  - In the information-theoretic setting, where the adversary is computationally unbounded and the communication channels are ideally secure, perfect and full security can be achieved facing adaptive, malicious adversaries; see Ben-Or et al. [12] and Chaum et al. [25]. Perfect and full security under concurrent general composition was presented by Canetti [20].
- When an honest majority is assumed, every function can be computed with full security, assuming the parties have access to a broadcast channel, or if the parties share some correlated randomness that is generated in a trusted setup phase.
  - In the computational setting, this holds, under appropriate cryptographic assumptions, in the stand-alone setting facing static, malicious adversaries, see Goldreich et al. [53], and under concurrent general composition facing adaptive, malicious adversaries, see Canetti et al. [23].
  - In the information-theoretic setting, where ideally secure channels are available in addition to the broadcast channel, this holds unconditionally but with a negligible non-zero error probability, facing static corruptions, see Rabin and Ben-Or [81], and facing adaptive corruptions, Cramer et al. [35].

- When considering any number of corrupted parties, every function can be securely computed with abort (meaning that all security properties but fairness and guaranteed output delivery are fulfilled), in the computational setting, given access to a broadcast channel. This holds, under appropriate cryptographic assumptions, in the stand-alone setting, facing static corruptions, see Yao [84] and Goldreich et al. [53], and under concurrent general composition facing adaptive corruptions; see Canetti et al. [23].

1.2 Feasibility of Full Security

Full security provides very strong security guarantees. As discussed in Section 1.1, full security ensures many security properties such as privacy, correctness, independence of inputs, fairness, guaranteed output delivery and more.

When an honest majority is assumed, every function can be computed with full security when a broadcast channel is available to the parties. Feasibility results were shown both in the computational setting assuming authenticated channels and the existence of oblivious transfer [53] and in the information-theoretic setting assuming secure channels [81].

When an honest majority is not assumed, Cleve [27] showed that full security cannot be obtained in general, more specifically, that it is impossible to obtain generic protocols for secure multiparty computation that guarantee output delivery and fairness. The security requirements are therefore typically relaxed when no honest majority is assumed. In particular, under certain circumstances, honest parties may not receive any output, and fairness is not always guaranteed. Recently, Gordon et al. [57] showed that some (in fact many [2]) two-party functionalities can be computed with full security (a characterization of full security for two-party Boolean functions was given in [5]). In addition, Gordon and Katz [56] showed that is possible to even compute some multiparty functionalities with full security, for any number of corrupted parties; in particular, the *majority* function may be securely computed with three parties, and the *Boolean OR* function may be securely computed for any number of parties.

1.2.1 Fairness Versus Guaranteed Output Delivery

The two notions of fairness and of guaranteed output delivery (i.e., full security) are quite similar and are often interchanged. However, there is a fundamental difference between them. If a protocol guarantees output delivery, then the parties always obtain output and cannot abort. In contrast, if a protocol is fair, then it is only guaranteed that *if* one party receives output then all parties receive output. Thus, it is possible that all parties abort. In order to emphasize the difference between the notions, we note that every protocol that provides guaranteed output delivery can be transformed into a protocol that provides fairness but *not* guaranteed output delivery, as follows. At the beginning every party broadcasts OK; if one of the parties did not send OK then all the parties output $\perp$; otherwise the parties execute the original protocol (that ensures guaranteed output delivery). Clearly every party can cause the protocol to abort. However, it can only do so before any information has been obtained. Thus, the resulting protocol is fair, but does not guarantee output delivery.

It is immediate to see that guaranteed output delivery implies fairness, since if all parties

must receive output then it is not possible for the corrupted parties to receive output while the honest do not. However, the opposite direction is not clear. In the two-party case, guaranteed output delivery is indeed implied by fairness since upon receiving abort, the honest party can just compute the function on its own input and a default input for the other party. However, when there are many parties involved, it is not possible to replace inputs with default inputs since the honest parties do not necessarily know who is corrupted (and security mandates that honest parties' inputs cannot be changed; otherwise, this could be disastrous in an election-type setting). This leads us to the following fundamental questions, which until now have not been considered at all (indeed, fairness and guaranteed output delivery are typically used synonymously):

> *Does fairness imply guaranteed output delivery? Do there exist functionalities that can be securely computed with fairness but not with guaranteed output delivery? Are there conditions on the function/network model for which fairness implies guaranteed output delivery?*

The starting point of our work is the observation that the *broadcast functionality* does actually separate guaranteed output delivery and fairness. Specifically, let n denote the overall number of parties, and let t denote an upper bound on the number of corrupted parties. Then, it is well known that secure broadcast can be achieved if and only if $t < n/3$ [78, 73].[1] However, it is also possible to achieve *detectable* broadcast (which means that either all parties abort and no one receives output, or all parties receive and agree upon the broadcasted value) for any $t < n$ [47]. In our terms, this is a secure computation of the broadcast functionality with *fairness* but *no guaranteed output delivery*. Thus, we see that for $t \geq n/3$ there exist functionalities that can be securely computed with fairness but not with guaranteed output delivery (the fact that broadcast cannot be securely computed with guaranteed output delivery for $t \geq n/3$ follows directly from the bounds on Byzantine Generals [78, 73]). Although broadcast does provide a separation, it is an atypical function. Specifically, there is no notion of privacy, and the functionality can be computed information theoretically for any $t < n$ given a secure setup phase [79]. Thus, broadcast is a trivial functionality.[2] This leaves the question of whether fairness and guaranteed output delivery are distinct still holds for more "standard" secure computation tasks.

It is well known that for $t < n/2$ any multiparty functionality can be securely computed with guaranteed output delivery given a broadcast channel [53, 81]. Fitzi et al. [46] used detectable broadcast in the protocols of [53, 81] and showed that *any* functionality can be securely computed with fairness for $t < n/2$. This leaves open the question as to whether there exist functionalities (apart from broadcast) that *cannot* be securely computed with guaranteed output delivery for $n/3 \leq t < n/2$.

Gordon and Katz [56] showed that the three-party majority function and multiparty Boolean OR function can be securely computed with guaranteed output delivery for *any* number of

[1] The impossibility of broadcast for $t \geq n/3$ holds in the plain model, where no trusted setup is available to the parties (which is the model considered in this work). Indeed, if the parties have access to some correlated randomness (e.g., a public-key infrastructure) broadcast can be computed facing any number of corrupted parties both in the computational setting, assuming one-way function exist [39] and in the information-theoretic setting [79].

[2] We stress that "trivial" does not mean easy to achieve or uninteresting. Rather, it means that cryptographic hardness is not needed to achieve it in the setting of no honest majority [70].

corrupted parties (in particular, with an honest minority). However, the constructions of [56] use a broadcast channel. This leads us to the following questions for the range of $t \geq n/3$:

1. Can the three-party majority function and multiparty Boolean OR function be securely computed with guaranteed output delivery without a broadcast channel?

2. Can the three-party majority function and multiparty Boolean OR function be securely computed with fairness without a broadcast channel?

3. Does the existence of broadcast make a difference with respect to fairness and/or guaranteed output delivery *in general*?

We remark that conceptually guaranteed output delivery is a stronger notion of security and that it is what is required in some applications. Consider the application of "mental poker"; if guaranteed output delivery is not achieved, then a corrupted party can cause the execution to abort in case it is dealt a bad hand. This is clearly undesirable.

1.2.1.1 Our Results

Separating fairness and guaranteed output delivery. We show that the three-party majority function, which can be securely computed with fairness [56], *cannot* be securely computed with guaranteed output delivery. Thus, there exist non-trivial functionalities (i.e., functionalities that cannot be securely computed in the information-theoretic setting without an honest majority) for which fairness can be achieved but guaranteed output delivery cannot. Technically, we show this by proving that the three-party majority function can be used to achieve broadcast, implying that it cannot be securely computed with guaranteed output delivery.

Theorem 1.2.1 (informal)**.** *Consider a model without a broadcast channel and consider any $t \geq n/3$. Then, there exist non-trivial functionalities f (e.g., the majority function) such that f can be securely computed with fairness but f cannot be securely computed with guaranteed output delivery.*

This proves that fairness and guaranteed output delivery are distinct, at least in a model without a broadcast channel.

Feasibility of guaranteed output delivery without broadcast. The protocols of [56] for majority and Boolean OR both use a broadcast channel to achieve guaranteed output delivery. As shown in Theorem 1.2.1, this is essential for achieving their result for the majority function. However, is this also the case for the Boolean OR function? In general, do there exist non-trivial functionalities for which guaranteed output delivery is achievable without a broadcast channel and for any number of corrupted parties?

Theorem 1.2.2 (informal)**.** *Consider a model without a broadcast channel and consider any number of corruptions. Then, there exist non-trivial functionalities f (e.g., the Boolean OR function) such that f can be securely computed with guaranteed output delivery.*

On the role of broadcast. We show that the existence or non-existence of broadcast is meaningless with respect to fairness, but of great significance with respect to guaranteed output delivery. Specifically, we show the following:

Theorem 1.2.3 (informal)**.** *Let f be a multiparty functionality. Then:*

1. *There exists a protocol for securely computing f with fairness with a broadcast channel* if and only if *there exists a protocol for securely computing f with fairness without a broadcast channel.*

2. *If there exists a protocol for securely computing f with fairness (with or without a broadcast channel), then there exists a protocol for securely computing f with guaranteed output delivery with a broadcast channel.*

Thus, fairness and guaranteed output delivery are *equivalent* in a model with a broadcast channel and *distinct* without a broadcast channel. In contrast, by Theorem 1.2.1 we already know that without broadcast it does not hold that fairness implies guaranteed output delivery (otherwise, the separation in Theorem 1.2.1 would not be possible). We also show that under black-box reductions, fairness *never* helps to achieve guaranteed output delivery. That is:

Theorem 1.2.4 (informal)**.** *Let f be a multiparty functionality and consider a hybrid model where a trusted party computes f fairly for the parties (i.e., either all parties receive output or none do). Then, there exists a protocol for securely computing f with guaranteed output delivery in this hybrid model if and only if there exists a protocol for securely computing f with guaranteed output delivery in the real model with no trusted party.*

Intuitively, Theorem 1.2.4 follows from the fact that an adversary can always cause the result of calls to f to be abort in which case they are of no help. This does not contradict item (2) of Theorem 1.2.3 since given a broadcast channel and non-black-box access to the protocol that computes f with fairness, it is possible to apply a variant of the GMW compiler [53] and detect which party cheated and caused the abort to occur.

Conditions under which fairness implies guaranteed output delivery. We have already seen that fairness implies guaranteed output delivery given broadcast. We also consider additional scenarios in which fairness implies guaranteed output delivery. We prove that if a functionality can be securely computed with fairness and *identifiable abort* (meaning that the identity of the cheating party is detected), then the functionality can be securely computed with guaranteed output delivery. Finally, we show that in the fail-stop model (where the only thing an adversary can do is instruct a corrupted party to halt prematurely), fairness is always equivalent to guaranteed output delivery. This follows from the fact that broadcast is trivial in the fail-stop model.

Identifiable abort and broadcast. In the model of identifiable abort, the identity of the cheating party is revealed to the honest parties. This definition was explicitly used by Aumann and Lindell [6], who remarked that it is met by most protocols (e.g., [53]), but not all (e.g., [55]). This model has the advantage that a cheating adversary who runs a "denial of service" attack and causes the protocol to abort cannot go undetected. Thus, it cannot repeatedly prevent the parties from obtaining output. An interesting corollary that comes out of our work—albeit

not related to fairness and guaranteed output delivery—is that security with identifiable abort *cannot* be achieved in general for $t \geq n/3$ without broadcast. This follows from the fact that if identifiable abort can be achieved in general (even without fairness), then it is possible to achieve broadcast. Thus, we conclude:

Corollary 1.2.5 (informal)**.** *Consider a model without a broadcast channel and consider any $t \geq n/3$. Then, there exist functionalities f that cannot be securely computed with identifiable abort.*

Summary of feasibility results. Table 1.1 summarizes the state of affairs regarding feasibility for secure computation with fairness and guaranteed output delivery, for different ranges regarding the number of corrupted parties.

Number of Corrupted	**With Broadcast**	**Without Broadcast**
$t < n/3$	All f can be securely computed with guaranteed output delivery	
$n/3 \leq t < n/2$	All f can be computed with guaranteed output delivery	f_{or} can be computed with guaranteed output delivery
$t \geq n/2$	Fairness implies guaranteed output delivery	f_{maj} *cannot* be computed with guaranteed output delivery
$t < n$	f can be securely computed fairly with broadcast iff f can be securely computed fairly without broadcast	

Table 1.1: Feasibility of fairness and guaranteed output delivery

1.2.2 Characterization of Multiparty Computation without Broadcast

Broadcast (introduced by Lamport et al. [73] as the Byzantine Generals problem) allows any party to deliver a message of its choice to all parties, such that all honest parties will receive the same message even if the broadcasting party is corrupted. Broadcast is an important resource for implementing secure multiparty computation. Indeed, much can be achieved when broadcast is available; in the computational setting, assuming the existence of oblivious transfer, every efficient functionality can be securely computed *with abort*, facing an arbitrary number of corruptions [83, 53]. Some functionalities can be computed with *full security*, e.g., Boolean OR and three-party majority [56], or with $1/p$-security,[3] e.g., coin-flipping protocols [76, 60]. In the information-theoretic setting, considering ideally secure communication lines between the parties, every efficient functionality can be computed with full security against unbounded adversaries, facing any minority of corrupted parties [81].

The above drastically changes when broadcast or a secure setup phase are not available,[4] specifically, when considering multiparty protocols (involving more than two parties), in which

[3]The success probability of any efficient distinguisher between the real model and an "ideal computation" without abort is bounded from above by $1/p$.

[4]In case a secure setup phase is available, *authenticated broadcast* can be computed facing $t < n$ corrupted parties; Authenticated broadcast exists in the computational setting over authenticated channels assuming the existence of one-way functions [39] and in the information-theoretic setting over secure channels given a correlated-randomness setup [79].

the parties are connected only via a point-to-point network and one third of the parties, or more, might be corrupted.[5] Considering authenticated channels and assuming the existence of oblivious transfer, every efficient functionality can be securely computed with abort, facing an arbitrary number of corruptions [47]. In the full-security model, some important functionalities *cannot* be securely computed when a third of the parties might be corrupted (e.g., Byzantine agreement [78] and three-party majority, see Theorem 1.2.1), whereas other functionalities can be securely computed facing an arbitrary number of corruptions (e.g., *weak* Byzantine agreement [47] and Boolean OR, see Theorem 1.2.2). The characterization of many other functionalities, however, was unknown. For instance, it was unknown whether the coin-flipping functionality or the Boolean XOR functionality can be computed with full securely, even when assuming an honest majority.

1.2.2.1 Our Result

A protocol is *t-consistent*, if in any execution of the protocol, in which at most t parties are corrupted, *all* honest parties output the same value. Our main technical result is the following attack on consistent protocols.

Lemma 1.2.6 (main lemma, informal)**.** *Let $n \geq 3$, $t \geq n/3$, and let $s = n - 2t$ if $t < n/2$ and $s = 1$ otherwise. Let π be an n-party, t-consistent protocol in the point-to-point model with secure channels. Then, there exists an adversary that by corrupting any s-size subset $\mathcal{I}$ of the parties can do the following: first, before the execution of π, output a value $y^* = y^*(\mathcal{I})$. Second, during the execution of π, force the remaining honest parties to output y^*. The running time of the adversary is proportional to executing polynomially many instances of π.*

The lemma extends to expected polynomial-time protocols, and to protocols that only guarantee consistency to hold with high probability. We prove the lemma by extending the well-known hexagon argument of Fischer et al. [44], originally used for proving the impossibility of reaching (strong and weak) Byzantine agreement in the point-to-point model.

A corollary of Lemma 1.2.6 is the following lower bound on public-output functionalities, where all parties receive the same output value. (To give a stronger lower bound, we state the result in the secure-channels model rather than in the authenticated-channels model, since any functionality that can be computed with authenticated channels can also be computed with secure channels.) A functionality is *k-dominated*, if there exists a value y^* such that *any* k-size subset of the functionality input variables, can be manipulated to make the output of the functionality be y^* (e.g., the Boolean OR functionality is 1-dominated with value $y^* = 1$).

Corollary 1.2.7 (Informal)**.** *Let $n \geq 3$, $t \geq n/3$, and let $s = n - 2t$ if $t < n/2$ and $s = 1$ otherwise. A public-output n-party functionality that can be computed with full security in the point-to-point model with secure channels, facing up to t corruptions, is s-dominated.*

Interestingly, the above lower bound is tight. Theorem 1.2.2 shows that assuming one-way functions exist, any 1-dominated functionality (e.g., Boolean OR) that can be securely computed

[5] For two-party protocols, the broadcast model is equivalent to the point-to-point model (and thus all the results mentioned in the broadcast model hold also in the point-to-point model). If less than a third of the parties are corrupted, broadcast can be implemented using a protocol, and every functionality can be computed with information-theoretic security [12, 25].

in the broadcast model with authenticated channels can be securely computed in the point-to-point model with authenticated channels. This shows tightness when an honest majority is not assumed. We generalize this approach, using the two-threshold detectable precomputation of Fitzi et al. [48], to get the following upper bound (in all our positive results, the efficiency of the protocol is considered to be polynomial in the circuit size of the functionality).

Proposition 1.2.8 (Informal)**.** *Let $n \geq 3$ and $n/3 \leq t < n/2$. Assuming up to t corruptions, any public-output n-party functionality that is $(n-2t)$-dominated, can be computed in the secure-channels point-to-point model with information-theoretic security.*

Combining Corollary 1.2.7, Proposition 1.2.8 and Theorem 1.2.2, yields the following characterization of public-output functionalities.

Theorem 1.2.9 (main theorem, informal)**.** *Let $n \geq 3$, let $t \geq n/3$, and let f be a public-output n-party functionality.*

1. *For $t < n/2$, f can be t-securely computed (with information-theoretic security) in the secure-channels point-to-point model if and only if f is $(n-2t)$-dominated.*
2. *For $t \geq n/2$, assuming one-way functions exist, f can be t-securely computed (with computational security) in the authenticated-channels point-to-point model if and only if f is 1-dominated and can be t-securely computed (with computational security) in the authenticated-channels broadcast model.*

Another application of Lemma 1.2.6 regards coin-flipping protocols. A coin-flipping protocol [15] allows the honest parties to jointly flip an unbiased coin, where even a coalition of (efficient) cheating parties cannot bias the outcome of the protocol by too much. We focus on protocols in which honest parties must output the same bit. Although Theorem 1.2.9 shows that fully secure coin flipping cannot be achieved facing one-third corruptions, we provide a stronger impossibility result under a weaker security requirement that only assumes $n/3$-consistency and a non-trivial bias. In particular, we show that $1/p$-secure coin flipping cannot be achieved using consistent protocols in case a third of the parties might be corrupted.

Corollary 1.2.10 (impossibility of multiparty coin flipping in the point-to-point model, informal)**.** *In the secure-channels point-to-point model, there exists no $(n \geq 3)$-party coin-flipping protocol that guarantees a non-trivial bias (i.e., smaller than $1/2$) against an efficient adversary controlling one third of the parties.*

The above is in contrast to the broadcast model, in which coin flipping can be computed with full security if an honest majority exists [17, 26], and with $1/p$-security when no honest majority is assumed [27, 9, 60, 1, 18, 34].

1.2.2.2 Our Technique

We present the ideas underlying our main technical result, showing that the following holds in the point-to-point model. For any consistent protocol involving more than two parties, if one third of the parties (or more) might be corrupted, then there exists an adversary that can make the honest parties output a predetermined value. Furthermore, the adversary is efficient and its running time is proportional to executing a polynomial number of instances of the original

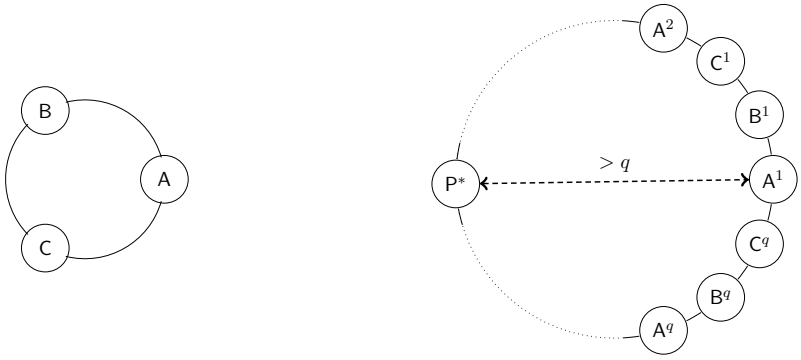

Figure 1.1: The original three-party protocol $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ is on the left. On the right is the $3q$-Ring — q copies of π concatenated. Communication time between parties of opposite sides is larger than $3q/2 > q$.

protocol. In the following discussion we focus on three-party protocols with a single corrupted party.

Let $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ be a 1-consistent, three-party protocol, and let q be its round complexity on inputs of fixed length κ. (For the sake of clarity and by abuse of notation, we consider the joined bit-length of the deterministic inputs and random coins to be κ; see Section 4.1 for a formal analysis.) Consider the following $3q$-party protocol $\mathsf{R} = (\mathsf{A}^1, \mathsf{B}^1, \mathsf{C}^1, \ldots, \mathsf{A}^q, \mathsf{B}^q, \mathsf{C}^q)$, where the parties are connected in a ring network such that each two consecutive parties, as well as the first and last, are connected via a secure channel, and party P^j, for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, has the code of P (see Figure 1.1).

Consider an execution of R on input $\boldsymbol{w} = (w^1_{\mathsf{A}}, w^1_{\mathsf{B}}, w^1_{\mathsf{C}}, \ldots, w^q_{\mathsf{A}}, w^q_{\mathsf{B}}, w^q_{\mathsf{C}}) \in (\{0,1\}^\kappa)^{3q}$ (i.e., party P^i has input w^i_{P}, containing its actual input and random coins – the actual inputs are arbitrarily chosen whereas the random coins are uniformly distributed). A key observation is that the view of party A^j, for instance, in this execution, is a *valid* view of the party A on input w^j_{A} in an interaction of π in which B acts honestly on input w^j_{B}. It is also a valid view of A, on input w^j_{A}, in an interaction of π in which C acts honestly on input $w^{j-1 \pmod q}_{\mathsf{C}}$. Hence, the consistency of π yields that any two consecutive parties in R output the same value, and thus *all* parties of R output the *same* value.

Consider for concreteness an attack on the parties $\{\mathsf{A}, \mathsf{B}\}$. The adversary $\mathcal{A}$ first selects a value $\boldsymbol{w} \in (\{0,1\}^\kappa)^{3q}$, emulates (in its head) an execution of R on $\boldsymbol{w}$, and sets y^* to be the output of the party $\mathsf{P}^* = \mathsf{A}^{q/2}$ in this execution. To interact with the parties $\{\mathsf{A}, \mathsf{B}\}$ in π, the adversary $\mathcal{A}$ corrupts party C and emulates an execution of R, in which all but $\{\mathsf{A}^1, \mathsf{B}^1\}$ have their inputs according to $\boldsymbol{w}$ (the roles of all parties but $\{\mathsf{A}^1, \mathsf{B}^1\}$ are played by the corrupted C), and $\{\mathsf{A}, \mathsf{B}\}$ take (without knowing it) the roles of $\{\mathsf{A}^1, \mathsf{B}^1\}$.

We claim that the output of the parties $\{\mathsf{A}, \mathsf{B}\}$ under the above attack is y^*. Observe that the emulation of R, induced by the interaction of $\mathcal{A}$ with $\{\mathsf{A}, \mathsf{B}\}$, is just a valid execution of R on some input $\boldsymbol{w}'$ (not completely known to the adversary). Hence, by the above observation, all parties in R (including $\{\mathsf{A}, \mathsf{B}\}$) output the same value at the end of this emulation. Since the execution of R ends after at most q rounds, and since the number of communication links between parties $\{\mathsf{A}^1, \mathsf{B}^1\}$ and the designated party P^* is $\approx 3q/2 > q$, the actions of $\{\mathsf{A}^1, \mathsf{B}^1\}$ have *no effect* on the view of P^*. In particular, the output of P^* in the attack is also y^*, and by the above this is also the output of parties $\{\mathsf{A}, \mathsf{B}\}$.

Extension to expected-polynomial-round protocols. The above attack works perfectly if the round complexity of π is (strictly) polynomial. For expected-polynomial-round protocols, one has to work slightly harder to come up with an attack that is (almost) as good.

Let q be the expected round complexity of π. That is, an honest party of π halts after q rounds in expectation, regardless of what the other parties do, where the expectation is over its random coins. Consider the m-party protocol $\mathsf{R} = (\mathsf{A}^1, \mathsf{B}^1, \mathsf{C}^1, \ldots, \mathsf{A}^m, \mathsf{B}^m, \mathsf{C}^m)$, for $m = 2q$, connected in a ring topology as before. By Markov bound, in a random execution of R, a party halts after m rounds with probability at least $1/2$.

The adversary $\mathcal{A}$ attacking the honest parties $\{\mathsf{A}, \mathsf{B}\}$ is defined as follows. For choosing a value for y^*, $\mathcal{A}$ emulates an execution of R on arbitrary inputs and uniformly distributed random coins. If the party $\mathsf{P}^* = \mathsf{A}^{m/2}$ halts in at most m rounds, $\mathcal{A}$ sets y^* to be P^*'s output, and continues to the second stage of the attack. Otherwise, it emulates R on new inputs and random coins. Note that in k attempts, $\mathcal{A}$ finds a good execution with probably (at least) $1 - 2^{-k}$. After finding y^*, the adversary $\mathcal{A}$ continues as in the strict polynomial case discussed above.

The key observation here is that in the emulated execution of R, induced by the interaction of $\mathcal{A}$ with parties $\{\mathsf{A}, \mathsf{B}\}$, the party P^* *never* interacts in more than m communication rounds. Therefore, again, being far from the parties $\{\mathsf{A}, \mathsf{B}\}$, their actions do not affect P^* in the first m rounds, and so do not affect it at all. Hence, P^* outputs y^* also in the induced execution, and so do the parties $\{\mathsf{A}, \mathsf{B}\}$.

Relation of our attack to the impossibility of Byzantine agreement. Our attack is based on the hexagon argument that was used to rule out three-party Byzantine agreement tolerating one corrupted party [44].[6] Assuming such a protocol exists, two copies are composed into a hexagonal system, where every pair of adjacent parties must output the same bit (by agreement) and upon starting with the same bit b, must output b, independently of the rest of the system (by validity). The adversary, controlling one party in the three-party protocol, can force the output of the remaining parties to be 0, even upon starting with input 1, by emulating towards both honest parties the hexagonal system where all virtual parties have input 0. This argument does not extend to arbitrary three-party consistent protocols, since the output of two adjacent parties might be influenced by the entire hexagonal system. We overcome this barrier by using a larger ring system, and by running the system twice: once to learn the output value and second to force the output for the honest parties in the three-party protocol.

1.2.2.3 Additional Related Work

Negative results. In their seminal work, Lamport et al. [73] defined the problem of simulating a broadcast channel in the point-to-point model in terms of the Byzantine agreement problem. They showed that a broadcast protocol exists if and only if more than two-thirds of the parties are honest. Lamport [72] defined the weak Byzantine agreement problem, and showed that even this weak variant of agreement cannot be computed, using deterministic protocols, facing one-third corruptions. Fischer et al. [44] presented simpler proofs to the above impossibility results using the so-called hexagon argument, which is also the basis of our lower bound (see

[6] In a Byzantine agreement protocol all honest parties must output the same bit (agreement), and if all honest parties have the same input bit b, then the common output should be b (validity).

Section 1.2.2.2). They assumed a protocol exists for the three-party case, and composed multiple copies of this protocol into a ring system that contains an internal conflict. Since the ring system cannot exist, it follows that the three-party protocol does not exist. We remark that the result of [44] easily extends to public-coins protocols, where parties have access to a common random string, and even when new common random coins are revealed in each round, i.e., if the three-party protocol is defined in the coin-flipping hybrid model; this is achieved by simulating the same random coins to all instances of the three-party protocol in the ring system. It follows that coin flipping is not sufficient for solving Byzantine agreement, and thus the impossibility result for coin flipping stated in Corollary 1.2.10 is not implied by the aforementioned impossibility of Byzantine agreement.

Positive results. If the model is augmented with a trusted setup phase, e.g., a public-key infrastructure (PKI), then Byzantine agreement can be computed facing any number of corrupted parties [73]. Pfitzmann and Waidner [79] presented an information-theoretic broadcast protocol given a correlated-randomness setup. Fitzi et al. [47] presented a probabilistic protocol that securely computes weak Byzantine agreement facing an arbitrary number of corrupted parties.

Graham and Yao [59] showed that if agreement is only required to be achieved with a constant probability (at most $(\sqrt{5}-1)/2$), then broadcast protocols exist for $n = 3$ and $t = 1$. Indeed, our attack does not apply to protocols with small consistency guarantees, and our impossibility results consider protocols that are consistent with all but negligible probability.

Goldwasser and Lindell [55] presented a weaker definition for MPC without agreement, in which non-unanimous abort is permitted, i.e., some honest parties may receive output while others might abort. Using this definition, they utilized non-consistent protocols to construct secure protocols in the point-to-point model, assuming an arbitrary number of corruptions.

1.3 Round-Efficient MPC with Full Security

As discussed in Section 1.1, the original security definitions of multiparty computation had a property-based flavor (i.e., the protocols were required to satisfy correctness and privacy, and potentially, as in the case of full security, other security properties such as independence of inputs, fairness and guaranteed output delivery). However, it is by now widely accepted that security of multiparty cryptographic protocols should be argued in a simulation-based manner. Informally, the protocol execution is compared to an ideal world where the parties have access to a trusted third party (aka the "ideal functionality") that captures the security properties we want the protocol to achieve. The trusted party takes the parties' inputs and performs the computation on their behalf. A protocol is regarded as secure if for any adversary attacking it, there exists an ideal adversary (the simulator) attacking the execution in the ideal world, such that no external distinguisher (environment) can tell the real and the ideal executions apart.

There are several advantages in proving a protocol secure in this way. For starters, the definition of the functionality captures all security properties the protocol is supposed to have, and therefore its design process along with the security proof often exposes potential design flaws or issues that have been overlooked in the protocol design. A very important feature of many simulation-based security definitions is *composability*, which ensures that a protocol can

be composed with other protocols without compromising its security. Intuitively, composability ensures that if a protocol $\pi^{\mathcal{G}}$ which uses a "hybrid" $\mathcal{G}$ (a broadcast channel, for example) securely realizes functionality $\mathcal{F}$, and protocol ρ securely realizes the functionality $\mathcal{G}$, then the protocol $\pi^{\rho/\mathcal{G}}$, which results by replacing in π calls to $\mathcal{G}$ by invocations of ρ, securely realizes $\mathcal{F}$. In fact, simulation-based security is the one and only way we know to ensure that a protocol can be generically used to implement its specification within an arbitrary environment.

Round complexity. The prevalent model for the design of MPC protocols is the synchronous model, where the protocol proceeds in rounds and all messages sent in any given round are received by the beginning of the next round. When executing such synchronous protocols over large networks, one needs to impose a long round duration in order to account for potential delay at the network level, since if the duration of the rounds is too short, then it is likely that some of the messages that arrive late will be ignored, or worse, assigned to a later round. Thus, the round complexity (i.e., the number of rounds it takes for a protocol to deliver outputs) is an important efficiency metric for such protocols, and depending on the network parameters, can play a dominant role in the protocol's running time.

An issue often overlooked in the analysis of the round complexity of protocols is that the relation between a protocol's round complexity and its actual running time is sensitive to the "hybrids" (e.g., network primitives) that the protocol is assumed to have access to. For example, starting with the seminal MPC works [83, 53, 12, 25, 81], a common assumption is that the parties have access to a broadcast channel which they invoke possibly in every round. In reality, however, such a broadcast channel might not be available and would have to be implemented by a broadcast protocol designed for a point-to-point network. Using a standard (deterministic) broadcast protocol for this job incurs a linear blowup (in n, the number of parties[7]) on the round complexity of the MPC protocol, as no deterministic broadcast protocol can tolerate a linear number of corruptions in a sub-linear number of rounds; see Fischer and Lynch [43] and Dolev and Strong [39]. Thus, even though the round complexity of these protocols is usually considered to be linear in the multiplicative depth d of the computed circuit, in reality their running time could become linear in nd (which can be improved to $O(n+d)$ [66]) when executed over point-to-point channels.[8]

In fact, all so-called constant-round multiparty protocols (e.g., [69, 8, 37, 64, 4, 50, 58, 77]) rely on broadcast rounds (rounds in which parties make calls to a broadcast channel) and therefore their running time when broadcast is implemented by a standard protocol would explode to be linear in n instead of constant.[9] As the results from [43, 39] imply, this is not a consequence of the specific choice of protocol but a limitation of any protocol in which there is a round such that all parties are guaranteed to have received their output; consistently with the literature on fault-tolerant distributed computing, we shall refer to protocols satisfying this property as *deterministic-termination* protocols. In fact, to the best of our knowledge, even if we allow a negligible chance for the broadcast to fail, the fastest known solutions tolerating a

[7] In fact, in the number of corruptions a protocol can tolerate, which is a constant fraction of n.

[8] Throughout this work we will consider fully secure protocols, in which all honest parties receive their output. If one relaxes this requirement (i.e., allow that some parties do not receive their output and give up on fairness) then the techniques of Goldwasser and Lindell [55] allow for replacing broadcast with a constant-round multi-cast primitive.

[9] We remark that even though those protocols are for the computational setting, the lower bound on broadcast round complexity also applies.

constant fraction of corruptions follow the paradigm from Feldman and Micali [42] (see below), which requires a poly-logarithmic (in n) number of rounds.

Protocols with probabilistic termination. A major breakthrough in fault-tolerant distributed algorithms (recently honored with the 2015 Dijkstra Prize in Distributed Computing), was the introduction of randomization to the field by Ben-Or [10] and Rabin [80], which, effectively, showed how to circumvent the above limitation by using randomization. Most relevant to this work, Rabin [80] showed that linearly resilient *Byzantine agreement (BA)* protocols [78, 73] in expected *constant* rounds were possible, provided that all parties have access to a "common coin" (i.e., a common source of randomness).[10] This line of research culminated with the work of Feldman and Micali [42], who showed how to obtain a shared random coin with constant probability from "scratch," yielding a probabilistic BA protocol tolerating the maximal number of misbehaving parties ($t < n/3$) that runs in expected constant number of rounds. The randomized BA protocol in [42] works in the information-theoretic setting; these results were later extended to the computational setting by Fitzi and Garay [45] and by Katz and Koo [65] who showed that, assuming digital signatures, there exists an (expected-)constant-round protocol for BA tolerating $t < n/2$ corruptions. The speedup on the running time in all these protocols, however, comes at the cost of uncertainty, as now they need to give up on guaranteed (eventual) termination (no fixed upper bound on their running time[11]) as well as on *simultaneous* termination (a party that terminates cannot be sure that other parties have also terminated[12]); see Dolev et al. [40]. These issues make the simulation-based proof of these protocols a very delicate task which is the motivation for the current work.

What made the simulation-based approach a more accessible technique in security proofs was the introduction of simulation-based security frameworks. The ones that stand out in this development, and most often used in the literature, are Canetti's modular composition (aka stand-alone security) [19] and the universal composition (UC) frameworks [20, 21]. The former defines security of synchronous protocols executed in isolation (i.e., only a single protocol is run at a time and whenever a subroutine-protocol is called, it is run until its completion); the latter allows protocols to be executed alongside arbitrary (other) protocols and be interleaved in an arbitrary manner. We remark that although the UC framework is inherently asynchronous, several mechanisms have been proposed to allow for a synchronous execution within it (e.g., [21, 71, 68]).

Despite the widespread use of the simulation-based paradigm to prove security of protocols with deterministic termination, the situation has been quite different when probabilistic-termination protocols are considered. Here, despite the existence of round-efficient BA protocols as mentioned above [42, 65], to our knowledge, no formal treatment of the problem in a simulation-based model exists, which would allow us to apply the ingenious ideas of Rabin and Ben-Or in order to speed up cryptographic protocols. We note that Katz and Koo [65] even provide an expected-constant-round MPC protocol using their fast BA protocol as a subroutine, employing several techniques to ensure proper use of randomized BA. However, in lack of a

[10] Essentially, the value of the coin can be adopted by the honest parties in case disagreement at any given round is detected, a process that is repeated multiple times.

[11] Throughout this section we use running time and round complexity interchangeably.

[12] It should be noted, however, that in many of these protocols there is a known (constant) "slack" of c rounds, such that if a party terminates in round r, then it can be sure that every honest party will have terminated by round $r + c$.

formal treatment, existing constructions are usually proved secure in a property-based manner or rely on ad hoc, less studied security frameworks [74].[13]

A simulation-based and composable treatment of such probabilistic-termination (PT for short) protocols would naturally allow, for example, to replace the commonly used broadcast channel with a broadcast protocol, so that the expected running time of the resulting protocol is asymptotically the same as the one of the original (broadcast-hybrid) protocol. However, a closer look at this replacement exposes several issues that have to do not only with the lack of simulation-based security but also with other inherent limitations. Concretely, it is usually the case in an MPC protocol that the broadcast channel is accessed by several (in many cases by all) parties in the same (broadcast) round in parallel. Ben-Or and El-Yaniv [11] observed that if we naïvely replace each such invocation with a PT broadcast protocol with expected constant running-time, then the expected number of rounds until *all* broadcasts terminate is no longer constant; in fact, it is not hard to see that in the case of [42] the expected round complexity would be logarithmic in the number of instances (We expand on the reason for this blowup in the round complexity in Section 5.3.) Nevertheless, Ben-Or and El-Yaniv [11] proposed a mechanism for implementing such parallel calls to broadcast so that the total number of rounds remains constant in expectation.

The difficulties arising with parallel composition are not the only issue with probabilistic-termination protocols. As observed by Lindell et al. [74], composing such protocols in sequence is also problematic. The main issue here is that, as already mentioned, PT protocols do not have simultaneous termination and therefore a party cannot be sure how long after he receives his output from a call to such a PT protocol he can safely carry on with the execution of the calling protocol. Although PT protocols usually guarantee a constant "slack" of rounds (say, c) in the output of any two honest parties, the naïve approach of using this property to synchronize the parties (i.e., wait c rounds after the first call, $2c$ rounds after the second call, and so on) imposes an exponential blowup on the round complexity of the calling protocol. To resolve this, Lindell et al. [74] proposed using fixed points in time at which a re-synchronization subroutine is executed, allowing the parties to ensure that they never get too far out-of-sync. An alternative approach for solving this issue was also proposed by Katz and Koo [65], but again, with a restricted (property-based) proof.

Despite their novel aspects, the aforementioned results on composition of PT protocols do not use simulation-based security, and therefore it is unclear how (or if) they could be used to, for example, instantiate broadcast within a higher-level cryptographic protocol. In addition, they do not deal with other important features of modern security definitions, such as adaptive security and *strict* polynomial time execution. In fact, this lack of a formal cryptographic treatment places some of their claims at odds with the state-of-the-art cryptographic definitions. Somewhat pointedly, Ben-Or and El-Yaniv [11] claimed adaptive security, which, although can be shown to hold in a property-based definition, is not achieved by the specified construction when simulation-based security is considered (cf. Chapter 8).

[13] As we discuss below, the protocol of Katz and Koo [65] has an additional issue with adaptive security in the rushing-adversary model, as defined in the UC framework, similar to the issue exploited by Hirt and Zikas [61].

Our Contributions

We provide the first formal simulation-based (and composable) treatment of MPC with probabilistic termination. Our treatment builds on the universal composition (UC) framework of Canetti [20]. In order to take advantage of the fast termination of PT protocols, parties typically proceed at different paces and therefore protocols might need to be run in an interleaved manner, e.g., in an MPC protocol a party might initiate the protocol for broadcasting his r-round message before other parties have received output from the broadcasting of messages for round $r-1$. This inherent concurrency along with its support for synchrony makes the UC framework the natural candidate for our treatment.

Our motivating goal, which we achieve, is to provide a generic compiler that allows us to transform any UC protocol π, even one that cannot be realized in the real world, making calls to deterministic-termination UC protocols ρ_i in a "stand-alone fashion" (similar to [19], i.e., the protocols ρ_i are invoked sequentially and in each round exactly one protocol is being executed by all the parties) into a (probabilistic-termination) protocol π' (where parties are no longer synchronized and the hybrids are invoked concurrently) that can be realized in the real world, and in which each ρ_i is replaced by a PT protocol ρ'_i. The compiled protocol π' achieves the same security as π and has (expected) round complexity proportional to $\sum_i d_i r_i$, where d_i is the expected number of calls π makes to ρ_i and r_i is the expected round complexity of ρ_i.

Toward this goal, the first step is to define what it means for a protocol to UC-securely realize a functionality with probabilistic termination in a simulation-based manner, by proposing an explicit formulation of the functionality that captures this important protocol aspect. The high-level idea is to parameterize the functionality with an efficiently sampleable distribution D that provides an upper bound on the protocol's running time (i.e., number of rounds), so that the adversary cannot delay outputs beyond this point (but is allowed to deliver the output to honest parties earlier, and even in different rounds).

Next, we prove our universal composability result. Informally, our result provides a generic compiler that takes as input a "stand-alone" protocol π, UC-realizing a probabilistic-termination functionality $\mathcal{F}^D$ (for a given distribution D) while making sequential calls to (deterministic-termination) secure function evaluation (SFE)-like functionalities, and compiles it into a new protocol π', in which the calls to the SFEs are replaced by probabilistic-termination protocols realizing them. The important feature of our compiler is that in the compiled protocol, the parties do not need to wait for every party to terminate their emulation of each SFE to proceed to the emulation of the next SFE. Rather, shortly after a party (locally) receives its output from one emulation, it proceeds to the next one. This yields an (at most) multiplicative blowup on the expected round complexity as discussed above. In particular, if the protocols used to emulate the SFEs are expected constant round, then the expected round complexity of π' is the same (asymptotically) as that of π.

We then showcase our definition and composition theorem by providing simulation-based (therefore composable) probabilistic-termination protocols and security proofs for several primitives relying on point-to-point channels: expected-constant-round perfect Byzantine agreement, expected-constant-round perfect parallel broadcast, and an MPC protocol with perfect and full security, with round complexity independent of the number of parties. Not surprisingly, the simulation-based treatment reveals several issues, both at the formal and at the intuitive levels, that are not present in a property-based analysis, and which we discuss along the way. We now

elaborate on each application in turn.

Regarding Byzantine agreement, we present a protocol that perfectly UC-realizes the Byzantine agreement functionality with probabilistic termination for $t < n/3$ with expected constant number of rounds. (We will use RBA to denote probabilistic-termination BA, as it is often referred to as "randomized BA."[14]) Our protocol follows the structure of the protocol in Feldman and Micali [42], with a modification inspired by Goldreich and Petrank [52], to make it run in strict polynomial time (see the discussion below). In a sense our protocol can be viewed as the analogue for RBA of the well-known "CLOS" protocol for MPC [23]. Indeed, similarly to how Canetti et al. [23] converted (and proved) the "GMW" protocol [53] from static security in the stand-alone setting into an adaptively secure UC version, our work transforms the broadcast and BA protocols from [42] into adaptively UC-secure randomized broadcast and RBA protocols.[15]

Theorem 1.3.1 (expected-constant-round RBA, informal)**.** *The randomized Byzantine agreement functionality can be UC-realized with perfect security using an expected-constant-round protocol in the secure-channels hybrid model, in the presence of an adaptive malicious adversary corrupting at most $t < n/3$ parties.*

Our first construction above serves as a good showcase of the power of our composition theorem, demonstrating how UC-secure RBA is built in a modular manner: First, we de-compose the subroutines that are invoked in [42] and describe simple(r) (SFE-like) functionalities corresponding to these subroutines; this provides us with a simple "backbone" of the protocol in [42] making calls to these hybrids, which can be easily proved to implement expected-constant-round RBA. Next, we feed this simplified protocol to our compiler which outputs a protocol that implements RBA from point-to-point secure channels; our composition theorem ensures that the resulting protocol is also expected constant round.

There is a sticky issue in proving Theorem 1.3.1 that we need to resolve for the above to work: the protocol in [42] does not have guaranteed termination and therefore the distribution of the terminating round is not sampleable by a strict probabilistic polynomial-time (PPT) machine.[16] A way around this issue would be to modify the UC model of execution so that the corresponding ITMs are *expected* PPT machines. Such a modification, however, would impact the UC model of computation, and would therefore require a new proof of the composition theorem, a trickier task than one might expect, as the shift to expected polynomial-time simulation is known to introduce additional conceptual and technical difficulties (cf. [67]), whose resolution is beyond the scope of this work. Instead, here we take a different approach which preserves full compatibility with the UC framework: We adapt the protocol from [42] using ideas from [52] so that it implements a functionality which samples the terminating round with almost the same probability distribution as in [42], but from a finite (linear-size) domain; as we show, this distribution is sampleable in *strict* polynomial time and can therefore be used by a standard UC functionality.

Next, using our composition theorem we derive the first simulation-based and adaptively secure parallel broadcast protocol, which guarantees that all broadcast values are received within

[14] BA is a deterministic output primitive and it should be clear that the term "randomized" can only refer to the actual number of rounds; however, to avoid confusion we will abstain from using this term for functionalities other than BA whose output might also be probabilistic.

[15] As we show, the protocol in [42] does not satisfy input independence, and therefore is not adaptively secure in a simulation-based manner (cf. [61]).

[16] All entities in UC, and in particular ideal functionalities, are strict interactive PPT Turing machines, and the UC composition theorem is proved for such PPT ITMs.

expected constant number of rounds. This extends the results of [11, 65] in several ways: first, our protocol is UC-secure, meaning that it can now be used within a UC-secure SFE protocol to implement a broadcast channel; second, it is adaptively secure against a rushing adversary.[17]

Theorem 1.3.2 (expected-constant-round parallel broadcast, informal)**.** *The parallel-broadcast functionality can be UC-realized with perfect security using an expected-constant-round protocol in the secure-channels hybrid model, in the presence of an adaptive malicious adversary corrupting at most $t < n/3$ parties.*

Finally, by applying once again our compiler to replace calls to the broadcast channel in the SFE protocol by Ben-Or, Goldwasser, and Wigderson [12] (which provides perfect and full security for $t < n/3$ corruptions in the broadcast-hybrid model [3]) by invocations to our adaptively secure UC parallel broadcast protocol, we obtain the first UC-secure probabilistic-termination fully secure MPC protocol in the point-to-point secure-channels model with (expected) round complexity $O(d)$, independently of the number of parties, where d is the multiplicative depth of the circuit being computed. As with RBA, this result can be seen as the first analogue of the UC compiler by Canetti et al. [23] for SFE protocols with probabilistic termination.

Theorem 1.3.3 (fully secure SFE with round complexity independent of n, informal)**.** *Let f be an n-party function. The secure function evaluation of f can be UC-realized with perfect and full security using a protocol whose round complexity is linear in the depth of the circuit representing f, in the secure-channels hybrid model, in the presence of an adaptive malicious adversary corrupting at most $t < n/3$ parties.*

We stress that the use of perfect security to showcase our composition theorem is just our choice and not a restriction of our composition theorem. In fact, our theorem can be also applied to statistically or computationally secure protocols. Moreover, if one is interested in achieving better constants in the (expected) round complexity, then one can use SFE protocols that attempt to minimize the use of the broadcast channel (e.g., [66]). Our composition theorem will give a direct methodology for this replacement and will, as before, eliminate the dependency of the round complexity from the number of parties.[18]

1.4 Organization

The thesis consists of two parts.

Part I considers the feasibility of full security. Chapter 3 analyzes the relation between fairness and guaranteed output delivery (i.e., full security) and Chapter 4 presents a characterization of full security without broadcast. The results in this part are based on [29, 32].

Part II considers round-efficient MPC with full security in the UC framework, and presents feasibility of secure computation without broadcast with the same round complexity as secure computation with broadcast, in expectation. The results in this part are based on [31].

[17] Although security against a "dynamic" adversary is also claimed in [11], the protocol does not implement the natural parallel broadcast functionality in the presence of an adaptive adversary (see Chapter 8).

[18] Note that even a single round of broadcast is enough to create the issues with parallel composition and non-simultaneous termination discussed above.

Each part is self contained and includes the necessary background and definitions, despite the risk of a small amount of repetitions. We also provide an appendix containing abstracts of additional and followup work done by the author during graduate studies at Bar-Ilan University.

Part I

Feasibility of Full Security

Chapter 2

Definitions

Notations We let $\kappa \in \mathbb{N}$ denote the security parameter. For $n \in \mathbb{N}$, let $[n] = \{1, \cdots, n\}$. Let poly denote the set all positive polynomials and let PPT denote a probabilistic interactive Turing machine that runs in *strictly* polynomial time. A function $\mu \colon \mathbb{N} \to [0,1]$ is *negligible*, denoted $\mu(\kappa) = \text{neg}(\kappa)$, if $\mu(\kappa) < 1/p(\kappa)$ for every $p \in \text{poly}$ and large enough κ. The statistical distance between two random variables X and Y over a finite set $\mathcal{U}$, denoted $\text{SD}(X, Y)$, is defined as $\frac{1}{2} \cdot \sum_{u \in \mathcal{U}} |\Pr[X = u] - \Pr[Y = u]|$.

Two distribution ensembles $X = \{X(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ and $Y = \{Y(a,\kappa)\}_{a \in \{0,1\}^*, \kappa \in \mathbb{N}}$ are computationally indistinguishable (denoted $X \overset{\text{c}}{\equiv} Y$) if for every non-uniform polynomial-time distinguisher $\mathcal{D}$ there exists a function $\mu(\kappa) = \text{neg}(\kappa)$, such that for every $a \in \{0,1\}^*$ and κ,

$$|\Pr[\mathcal{D}(X(a,\kappa), 1^\kappa) = 1] - \Pr[\mathcal{D}(Y(a,\kappa), 1^\kappa) = 1]| \leq \mu(\kappa).$$

The distribution ensembles X and Y are δ-close if for every $a \in \{0,1\}^*$ and every κ it holds that $\text{SD}(X(a,\kappa), Y(a,\kappa)) \leq \delta(\kappa)$, and statistically close (denoted $X \overset{\text{s}}{\equiv} Y$) is they are δ-close and δ is negligible.

2.1 Security of Protocols

In this section, we present definitions for secure multiparty computation. Out definitions follow Goldreich [51], which in turn follows [54, 7, 75, 19]. We consider several definitions of security: full security (i.e., security with guaranteed output delivery), security with complete fairness, security with complete fairness and identifiable abort, security with abort and security with identifiable abort.

All of these security definitions are based on the *real/ideal paradigm*, i.e., comparing what an adversary can do in the real execution of the protocol to what it can do in an ideal model, where an uncorrupted trusted party assists the parties. In an ideal-model execution, each party sends its input to the trusted party over a perfectly secure channel, the trusted party computes the function based on these inputs and sends to each party its corresponding output. Informally, a protocol is secure if whatever an adversary can do in the real protocol (where no trusted party exists) can be done in the above-described ideal computation. The difference between the

various security definitions is related to whether the ideal-model adversary is allowed to abort the ideal execution, and if so at what stage and under which conditions.

Functionalities: An n-party functionality is a random process that maps vectors of n inputs to vectors of n outputs, denoted as $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where $f = (f_1, \ldots, f_n)$. That is, for a vector of inputs $\boldsymbol{x} = (x_1, \ldots, x_n)$, the output-vector is a random variable $(f_1(\boldsymbol{x}), \ldots, f_n(\boldsymbol{x}))$ ranging over vectors of strings. The output for the ith party (with input x_i) is defined to be $f_i(\boldsymbol{x})$. We denote an empty input by λ. A functionality f is public-output, if the output values of all parties are the same, i.e., for every $\boldsymbol{x} \in (\{0,1\}^*)^n$, $f_1(\boldsymbol{x}) = f_2(\boldsymbol{x}) = \ldots = f_n(\boldsymbol{x})$; by abuse of notation we refer to the functionality f as f_1. Otherwise, f is a private-output functionality.

Adversarial behaviour: Loosely speaking, the aim of a secure multiparty protocol is to protect the honest parties against dishonest behavior from the corrupted parties. This is normally modeled using a central adversarial entity, which controls the set of corrupted parties and instructs them how to operate. That is, the adversary obtains the views of the corrupted parties, consisting of their inputs, random tapes and incoming messages, and provides them with the messages that they are to send in the execution of the protocol.

We differentiate between three types of adversaries:

- **Semi-honest adversaries:** a semi-honest adversary always instructs the corrupted parties to follow the protocol. Semi-honest adversaries model "honest but curious" behaviour, where the adversary tries to learn additional information other than the output, based on the internal states of the corrupted parties.

- **Fail-stop adversaries:** a fail-stop adversary instructs the corrupted parties to follow the protocol as a semi-honest adversary, but it may also instruct a corrupted party to halt early (only sending some of its messages in a round).

- **Malicious adversaries:** a malicious adversary can instruct the corrupted parties to deviate from the protocol in any arbitrary way it chooses. There are no restrictions on the behaviour of malicious adversaries.

Unless stated otherwise, we consider malicious adversaries who may arbitrarily deviate from the protocol specification. When considering malicious adversaries, there are certain undesirable actions that cannot be prevented. Specifically, parties may refuse to participate in the protocol, may substitute their local input (and enter with a different input) and may cease participating in the protocol before it terminates. Essentially, secure protocols limit the adversary to such behaviour only.

We further assume that the adversary is *static*, i.e., the adversary is given a set $\mathcal{I}$ of corrupted parties which it controls at the beginning of the execution.[19]

[19] In Part II we consider adaptive adversaries that can corrupt parties during the coarse of the protocol.

2.2 Execution in the Real World

We first define a real-model execution. An n-party protocol $\pi = (\mathsf{P}_1, \ldots, \mathsf{P}_n)$ is an n-tuple of probabilistic interactive Turing Machines. The term *party* P_i refers to the ith interactive Turing Machine. Each party P_i starts with input $x_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$ and security parameter $\kappa \in \mathbb{N}$.[20] An *adversary* $\mathcal{A}$ is another probabilistic interactive Turing Machine, describing the behavior of the corrupted parties. We assume that the adversary is *static*, and so $\mathcal{A}$ starts the execution with input that contains the identities of the corrupted parties and their private inputs, and possibly an additional auxiliary input. The parties execute the protocol in a synchronous network with rushing. That is, the execution proceeds in rounds: each round consists of a *send phase* (where parties send their message from this round) followed by a *receive phase* (where they receive messages from other parties). The adversary is *rushing* which means that it can see the messages the honest parties send in a round, before determining the messages that the corrupted parties send in that round.

We assume the parties are connected via a fully connected point-to-point network; we refer to this model as the point-to-point model. We consider two models for the communication lines between the parties: In the *authenticated-channels* model, the communication lines are assumed to be ideally authenticated but not private (and thus the adversary cannot modify messages sent between two honest parties but can read them).[21] In the *secure-channels* model, the communication lines are assumed to be ideally private (and thus the adversary cannot read or modify messages sent between two honest parties). We sometimes assume that the parties are given access to a physical broadcast channel in addition to the point-to-point network; we refer to this model as the broadcast model. In each section it will be explicitly clarified whether the existence of a broadcast channel is assumed or not. Furthermore, the delivery of messages between honest parties is guaranteed. Finally, we note that in both models, we do not assume any trusted preprocessing phase (that can be used to setup correlated randomness, for example, a public-key infrastructure).

Throughout the execution of the protocol, all the honest parties follow the instructions of the prescribed protocol, whereas the corrupted parties receive their instructions from the adversary. At the conclusion of the execution, the honest parties output their prescribed output from the protocol, the corrupted parties output nothing and the adversary outputs an (arbitrary) function of its view of the computation (containing the views of the corrupted parties). The view of a party in a given execution of the protocol consists of its input, its random coins, and the messages it sees throughout this execution.

Definition 2.2.1 (real-model execution)**.** *Let f be an n-party functionality, let π be a multiparty protocol for computing f, and let κ be the security parameter. Denote by $\mathcal{I} \subseteq [n]$ the set of indices of the parties corrupted by $\mathcal{A}$. Then, the* joint execution of π under $(\mathcal{A}, \mathcal{I})$ in the real model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{A}$ resulting from the protocol interaction, where for every $i \in \mathcal{I}$, party P_i computes its messages according to $\mathcal{A}$,*

[20] Without loss of generality, the input length of each party is sometimes considered to be the security parameter.

[21] If private channels are needed in the computational model, then privacy can be achieved over authenticated channels by simply using public-key encryption. This works for static corruptions and efficient adversaries.

and for every $j \notin \mathcal{I}$, party P_j computes its messages according to π.

2.2.0.1 Time and Round Complexity

We start by defining both *strict* and *expected* bounds on the time complexity of interactive Turing machines (ITMs).

Definition 2.2.2 (time complexity of interactive Turing machines)**.** *An ITM P has* running-time *T, if for every input $x \in \{0,1\}^*$ and any choice of its random coins, when interacting with arbitrary (possibly unbounded) ITMs, P's running time is at most $T(|x|)$. If $T \in \text{poly}$, then P is of* (strict) polynomial time*.*

The machine P has an expected running time *T, if for every input $x \in \{0,1\}^*$, when interacting with arbitrary (possibly unbounded) ITMs, P's expected running time, over its* own *random coins, is at most $T(|x|)$. If $T \in \text{poly}$, then P has* expected polynomial running time*.*

Next, we define both *strict* and *expected* bounds on the time and round complexities of protocols.

Definition 2.2.3 (time complexity of a protocol)**.** *Protocol $\pi = (\mathsf{P}_1, \ldots, \mathsf{P}_n)$ is a T-*time protocol*, if for every $i \in [n]$, party P_i has running-time T. If $T \in \text{poly}$, then π is of* (strict) polynomial time*.*

Protocol π has an expected running time *T, if for every $i \in [n]$, party P_i has an expected running time T. If $T \in \text{poly}$, then π has* expected polynomial running time*.*

Definition 2.2.4 (round complexity)**.** *Protocol $\pi = (\mathsf{P}_1, \ldots, \mathsf{P}_n)$ is a q-*round protocol*, if for every $i \in [n]$, and every input $x_i \in \{0,1\}^*$ and random coins $r_i \in \{0,1\}^*$, the round number in which an honest party P_i stops being active (i.e., stops sending and receiving messages) is at most $q(|x_i|)$, when interacting with arbitrary (possibly unbounded) interactive Turing machines $(\mathsf{P}_1^*, \ldots, \mathsf{P}_{i-1}^*, \mathsf{P}_{i+1}^*, \ldots, \mathsf{P}_n^*)$. If $q \in \text{poly}$, then π has* (strict) polynomial round complexity*.*

Protocol π has an expected round complexity *q, if for every $i \in [n]$, and every input value $x_i \in \{0,1\}^*$, the expected round number in which an honest party P_i stops being active, over its random coins r_i, when interacting with arbitrary (possibly unbounded) interactive Turing machines $(\mathsf{P}_1^*, \ldots, \mathsf{P}_{i-1}^*, \mathsf{P}_{i+1}^*, \ldots, \mathsf{P}_n^*)$ is at most $q(|x_i|)$. If $q \in \text{poly}$, then π has* expected polynomial round complexity*.*

Definitions 2.2.3 and 2.2.4 are fairly strong in the sense that they capture time and round complexities as local properties of every party in the protocol. Although all of our positive results readily meet these definitions, we note that weaker notions of time and round complexities can be defined, in which the running time is required to hold only when interacting with up to t corrupted parties. Our attack in Section 4.1 can be applied also to protocols with the weaker guarantees whenever $t \geq n/3$.

2.3 Execution in the Ideal World

We consider several ideal worlds, each provides a different notion of security.

2.3.1 Secure Computation with Guaranteed Output Delivery

This definition provides the strongest notion of security we consider, and so we also refer to this security notion as full security. According to this definition, the protocol can terminate only when all parties receive their prescribed output. Recall that a malicious party can always substitute its input or refuse to participate. Therefore, the ideal model takes this inherent adversarial behavior into account by giving the adversary the ability to do this also in the ideal model. Since the adversary cannot abort the execution of the protocol in this model, fail-stop adversaries are equivalent to semi-honest adversaries (in particular, they cannot substitute their input).

Ideal-model execution. An ideal computation of an n-party functionality f on input $\boldsymbol{x} = (x_1, \ldots, x_n)$ for parties $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$ in the presence of an ideal-model adversary $\mathcal{A}$ controlling the parties indexed by $\mathcal{I} \subseteq [n]$, proceeds via the following steps.

Send inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. The adversary may send to the trusted party arbitrary inputs for the corrupted parties. Let x_i' be the value actually sent as the input of party P_i. In case the adversary is semi-honest or fail-stop, we require that $x_i' = x_i$.

Trusted party answers the parties: If x_i' is outside of the domain for P_i, for some index i, or if no input was sent for P_i, then the trusted party sets x_i' to be some predetermined default value. Next, the trusted party computes $f(x_1', \ldots, x_n') = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every i.

Outputs: Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary $\mathcal{A}$ outputs an arbitrary function of the initial inputs $\{x_i\}_{i\in\mathcal{I}}$, the messages received by the corrupted parties from the trusted party $\{y_i\}_{i\in\mathcal{I}}$ and its auxiliary input.

Definition 2.3.1 (ideal-model computation with full security)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{full}}_{f,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{A}$ resulting from the above-described ideal process.*

2.3.2 Secure Computation with Complete Fairness

This definition is similar to the previous one, except that the execution can terminate in two possible ways: the first is when all parties receive their prescribed output (as in the previous case) and the second is when all parties (including the corrupted parties) abort without receiving output. This is "fair" since in both cases the adversary receives no more information than the honest parties. In this definition, when sending the inputs to the trusted party, the adversary is allowed to send a special abort command. In this case, the trusted party sends a special failure symbol $\perp$ as the output to all parties. Without loss of generality, we assume that a malicious party always sends an input which is either in the corresponding input domain or abort, since in case the trusted party receives a value outside of the domain, it can proceed

as if abort was sent. In this definition, fail-stop adversaries have the additional capability over semi-honest adversaries to abort the computation without anyone receiving output. An ideal execution proceeds as follows:

Send inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. The adversary may send to the trusted party arbitrary inputs for the corrupted parties, in particular it can send the special abort input. Let x'_i be the value actually sent as the input of party P_i. We require that in the case of a semi-honest adversary $x'_i = x_i$, whereas in the case of a fail-stop adversary $x'_i \in \{x_i, \mathsf{abort}\}$.

Trusted party answers the parties: If $x'_i = \mathsf{abort}$ for P_i, for some index i, the trusted party sends $\perp$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every $i \in [n]$.

Outputs: As in Definition 2.3.1.

Definition 2.3.2 (ideal-model computation with fairness)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{fair}}_{f,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{A}$ resulting from the above-described ideal process.*

2.3.3 Secure Computation with Complete Fairness and Identifiable Abort

This definition is identical to the previous definition of secure computation with complete fairness, except that if the adversary aborts the computation, all honest parties learn the identity of one of the corrupted parties.

Send inputs to trusted party: Each honest party P_i sends its input x_i to the trusted party. The adversary may send to the trusted party arbitrary inputs for the corrupted parties. Let x'_i be the value actually sent as the input of party P_i. In case the adversary instructs P_i to send abort, it chooses an index of a corrupted party $i^* \in \mathcal{I}$ and sets $x'_i = (\mathsf{abort}, i^*)$. We require that in the case of a semi-honest adversary, $x'_i = x_i$, whereas in the case of a fail-stop adversary $x'_i \in \{x_i, (\mathsf{abort}, i^*)\}$ for some $i^* \in \mathcal{I}$.

Trusted party answers the parties: If $x'_i = (\mathsf{abort}, i^*)$ for some $i \in [n]$ and $i^* \in \mathcal{I}$, the trusted party sends $(\perp, i^*)$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every $i \in [n]$.

Outputs: As in Definition 2.3.1.

Definition 2.3.3 (ideal-model computation with complete fairness and identifiable abort)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{id\text{-}fair}}_{f,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{A}$ resulting from the above-described ideal process.*

2.3.4 Secure Computation with Abort

This definition is similar to secure computation with fairness; however the protocol can also terminate when corrupted parties receive output yet honest parties do not. However, if one honest party receives output, then so do all honest parties. Thus, this is the notion of *unanimous abort* (cf. [55].

Send inputs to trusted party: As in Definition 2.3.2.

Trusted party answers adversary: If $x'_i = \mathsf{abort}$ for P_i, for some index i, the trusted party sends $\perp$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every $i \in \mathcal{I}$.

Trusted party answers remaining parties: The adversary, depending on the views of all the corrupted parties, sends the trusted party either $\mathsf{continue}$ or abort. In case of $\mathsf{continue}$, the trusted party sends y_i to P_i for every $i \notin \mathcal{I}$, whereas in case of abort the trusted party sends $\perp$ to P_i for every $i \notin \mathcal{I}$.

Outputs: As in Definition 2.3.1.

Definition 2.3.4 (ideal-model computation with abort)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^{\mathsf{abort}}_{f,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{A}$ resulting from the above-described ideal process.*

2.3.5 Secure Computation with Identifiable Abort

This definition is identical to the previous definition of secure computation with abort, except that if the adversary aborts the computation, all honest parties learn the identity of one of the corrupted parties.

Send inputs to trusted party: As in Definition 2.3.3.

Trusted party answers adversary: If $x'_i = (\mathsf{abort}, i^*)$ for P_i, for some index $i \in [n]$ and $i^* \in \mathcal{I}$, the trusted party sends $(\perp, i^*)$ to all the parties. Otherwise, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to every *corrupted* party P_i (for every $i \in \mathcal{I}$).

Trusted party answers remaining parties: The adversary, depending on the views of all the corrupted parties, sends the trusted party either $\mathsf{continue}$ or (abort, i^*), where $i^* \in \mathcal{I}$. In case of (abort, i^*) with $i^* \in \mathcal{I}$ the trusted party sends $(\perp, i^*)$ to P_i for every $i \notin \mathcal{I}$; in case of $\mathsf{continue}$, the trusted party sends y_i to P_i for every $i \notin \mathcal{I}$.

Outputs: As in Definition 2.3.1.

Definition 2.3.5 (ideal computation with identifiable abort)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, let $\mathcal{I} \subseteq [n]$ be the set of indices of the corrupted parties, and let κ be the security parameter. Then, the* joint execution of f under $(\mathcal{A}, I)$ in the ideal model*,*

on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$*, auxiliary input* z *to* $\mathcal{A}$ *and security parameter* κ*, denoted* $\text{IDEAL}^{\mathsf{id\text{-}abort}}_{f,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$*, is defined as the output vector of* $\mathsf{P}_1, \ldots, \mathsf{P}_n$ *and* $\mathcal{A}$ *resulting from the above-described ideal process.*

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

Having defined the ideal and real models, we can now define security of protocols. The underlying idea of the definition is that the adversary can do no more harm in a real protocol execution than in the ideal model (where security trivially holds). This is formulated by saying that adversaries in the ideal model are able to simulate adversaries in an execution of a protocol in the real model. When the adversary is all-powerful, i.e., computationally unbounded, we require that the real and ideal models are statistically close,[22] and define information-theoretic security. When the adversary is efficient, i.e., runs in probabilistic polynomial time, we require that the real and ideal models are computationally indistinguishable and define computational security.

Definition 2.4.1. *Let* $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ *be an* n*-party functionality, and let* π *be a probabilistic polynomial-time protocol computing* f*. The* protocol π t-securely computes f (with computational security)*, if for every probabilistic polynomial-time adversary* $\mathcal{A}$ *for the real model, there exists a probabilistic polynomial-time adversary* $\mathcal{S}$ *for the ideal model, such that for every* $\mathcal{I} \subseteq [n]$ *of size at most* t*, it holds that*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \overset{\text{c}}{\equiv} \left\{\text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathcal{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

Definition 2.4.2. *Let* $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ *be an* n*-party functionality, and let* π *be a probabilistic polynomial-time protocol computing* f*. The* protocol π t-securely computes f (with information-theoretic security)*, if for every adversary* $\mathcal{A}$ *for the real model, there exists an adversary* $\mathcal{S}$ *for the ideal model, whose running time is polynomial in the running time of* $\mathcal{A}$*, such that for every* $\mathcal{I} \subseteq [n]$ *of size at most* t*,*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \overset{\text{s}}{\equiv} \left\{\text{IDEAL}^{\mathsf{type}}_{f,\mathcal{I},\mathcal{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

2.5 The Hybrid Model

The *hybrid model* is a model that extends the real model with a trusted party that provides ideal computation for specific functionalities. The parties communicate with this trusted party in exactly the same way as in the ideal models described above. The question of which ideal model is considered must be specified. Specifically, the trusted party may work according to any of the ideal models that we have defined above.

[22] In particular, perfect security means that the real and ideal models are identically distributed.

Let f be a functionality. Then, an execution of a protocol π computing a functionality g in the f-hybrid model involves the parties sending normal messages to each other (as in the real model) and in addition, having access to a trusted party computing f. It is essential that the invocations of f are done sequentially, meaning that before an invocation of f begins, the preceding invocation of f must finish.[23] In particular, there is at most a single call to f per round, and no other messages are sent during any round in which f is called. This is especially important for reactive functionalities, where the calls to f are carried out in phases, and a new invocation of f cannot take place before all phases of the previous invocation complete. In addition, no other messages in π can be sent before f is completed. For example, if f computes the commitment functionality, then after the first call to f, computing the commit phase, another invocation of f cannot take place until the decommit phase of the first invocation is completed. (In this specific example, it typically won't be useful unless other messages can be sent between the commit and decommit phase. This can be overcome by not modeling the commitment as an ideal functionality. Alternatively, if the functionality allows for multiple commitments, then ordinary messages can be sent between the commit and decommit phase of a specific message by repeatedly committing and decommitting. This is an annoying technicality, but is nevertheless an inherent limitation of the sequential composition theorem of Canetti [19].)

Let $\mathsf{type} \in \{\mathsf{full}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$. Let $\mathcal{A}$ be a probabilistic polynomial time machine with auxiliary input z and let $\mathcal{I} \subseteq [n]$ be the set of corrupted parties. We denote by $\text{HYBRID}^{f,\mathsf{type}}_{\pi,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$ the random variable consisting of the view of the adversary and the output of the honest parties, following an execution of π with ideal calls to a trusted party computing f according to the ideal model "type", on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$, and security parameter κ. We call this the (f, type)-hybrid model.

The sequential composition theorem of Canetti [19] states the following. Let ρ be a protocol that securely computes f in the ideal model "type". Then, if a protocol π computes g in the (f, type)-hybrid model, then the protocol π^ρ, that is obtained from π by replacing all ideal calls to the trusted party computing f with the protocol ρ, securely computes g in the real model.

Proposition 2.5.1 ([19])**.** *Let* $\mathsf{type}_1, \mathsf{type}_2 \in \{\mathsf{full}, \mathsf{fair}, \mathsf{id\text{-}fair}, \mathsf{abort}, \mathsf{id\text{-}abort}\}$*. Let f be an n-party functionality. Let ρ be a protocol that t-securely computes f with* type_1*, and let π be a protocol that t-securely computes g with* type_2 *in the (f, type_1)-hybrid model. Then protocol π^ρ t-securely computes g with* type_2 *in the real model.*

[23] In Part II we discuss concurrent composition of protocols.

Chapter 3

Fairness Versus Guaranteed Output Delivery

In this chapter, we analyze the relation between fairness and guaranteed output delivery. In Section 3.1 we provide a non-trivial separation between these notions. In Section 3.2 we show a family of functions for which both notions are the same. In Section 3.3 we discuss the meaning of broadcast, and show that although it plays no role for feasibility of fairness, it is essential for guaranteed output delivery. In fact, we show that in the broadcast model, every fair protocol can be compiled into a protocol that allows to identify corrupted parties while preserving fairness, and that this property is sufficient for guaranteed output delivery. In Section 3.4 we show that fairness does not help in a black-box way for achieving guaranteed output delivery. In Section 3.5 we present additional results.

3.1 Separating Fairness from Guaranteed Output Delivery

In this section, we prove Theorem 1.2.1. As we have mentioned in the Introduction, it is known that secure broadcast can be t-securely computed with guaranteed output delivery if and only if $t < n/3$. In addition, secure broadcast can be computed with fairness, for any $t \leq n$, using the protocol of Fitzi et al. [47]. Thus, broadcast already constitutes a separation of fairness from guaranteed output delivery; however, since broadcast can be information-theoretically computed (and is trivial in the technical sense; see Footnote 2), we ask whether or not such a separation also exists for more standard secure computation tasks.

In order to show a separation, we need to take a function for which fairness in the multiparty setting is feasible. Very few such functions are known, and the focus of this work is not the construction of new protocols. Fortunately, Gordon and Katz [56] showed that the three-party majority function can be securely computed with fairness. (In [56] a broadcast channel is used. However, as we show in Section 3.3.1, this implies the result also without a broadcast channel.) We stress that the three-party majority function is not trivial, and in fact the ability to securely compute it with any number of corruptions implies the existence of oblivious transfer (this is shown by reducing the two-party greater-than functionality to it and applying [70]).

We show that the three-party majority function f_{maj} *cannot* be securely computed with guaranteed output delivery and any number of corrupted parties in the point-to-point network

model by showing that it actually implies broadcast. The key observation is that there exists an input $(1,1,1)$ for which the output of f_{maj} will be 1, even if a single corrupted party changes its input to 0. Similarly, there exists an input $(0,0,0)$ for which the output of f_{maj} will be 0, even if a single corrupt party changes its input to 1. Using this property, we show that if f_{maj} can be computed with guaranteed output delivery, then there exists a broadcast protocol for three parties that is secure against a single corruption. Given an input bit β, the sender sends β to each other party, and all parties compute f_{maj} on the input they received. This works since a corrupted dealer cannot make two honest parties output inconsistent values, since f_{maj} provides the same output to all parties. Likewise, if there is one corrupted receiver, then it cannot change the majority value (as described above). Finally, if there are two corrupted receivers, then it makes no difference what they output anyway.

Theorem 3.1.1. *Let $t \leq 3$ be a parameter and let $f_{\mathsf{maj}} : \{0,1\}^3 \to \{0,1\}^3$ be the majority functionality for three parties $f_{\mathsf{maj}}(x_1,x_2,x_3) = (y,y,y)$ where $y = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_2 \wedge x_3)$. If f_{maj} can be t-securely computed with guaranteed output delivery in a point-to-point network, then there exists a protocol that t-securely computes the three-party broadcast functionality.*

Proof. We construct a protocol π for securely computing the three-party broadcast functionality $f_{\mathsf{bc}}(x,\lambda,\lambda) = (x,x,x)$ in the $(f_{\mathsf{maj}}, \mathsf{full})$-hybrid model (i.e., in a hybrid model where a trusted party computes the f_{maj} functionality with guaranteed output delivery). Protocol π works as follows:

1. The sender P_1 with input $x \in \{0,1\}$ sends x to P_2 and P_3.
2. Party P_1 sends x to the trusted party computing f_{maj}. Each party P_i ($i \in \{2,3\}$) sends the value it received from P_1 to f_{maj}.
3. Party P_1 always outputs x. The parties P_2 and P_3 output whatever they receive from the trusted party computing f_{maj}.

Let $\mathcal{A}$ be an adversary attacking the execution of π in the $(f_{\mathsf{maj}}, \mathsf{full})$-hybrid model; we construct an ideal-model adversary $\mathcal{S}$ in the ideal model for f_{bc} with guaranteed output delivery. $\mathcal{S}$ invokes $\mathcal{A}$ and simulates the interaction of $\mathcal{A}$ with the honest parties and with the trusted party computing f_{maj}. $\mathcal{S}$ proceeds based on the following corruption cases:

- P_1 *alone is corrupted:* $\mathcal{S}$ receives from $\mathcal{A}$ the values $x_2, x_3 \in \{0,1\}$ that it sends to parties P_2 and P_3, respectively. Next, $\mathcal{S}$ receives the value $x_1 \in \{0,1\}$ that $\mathcal{A}$ sends to f_{maj}. $\mathcal{S}$ computes $x = f_{\mathsf{maj}}(x_1,x_2,x_3)$ and sends x to the trusted party computing f_{bc}. $\mathcal{S}$ simulates $\mathcal{A}$ receiving x back from f_{maj}, and outputs whatever $\mathcal{A}$ outputs.
- P_1 *and one of* P_2 *or* P_3 *are corrupted:* the simulation is the same as in the previous case except that if P_2 is corrupted then the value x_2 is taken from what $\mathcal{A}$ sends in the name of P_2 to f_{maj} (and not the value that $\mathcal{A}$ sends first to P_2); likewise for P_3. Everything else is the same.
- P_1 *is honest:* $\mathcal{S}$ sends an empty input λ to the trusted party for every corrupted party, and receives back some $x \in \{0,1\}$. Next, $\mathcal{S}$ simulates P_1 sending x to both P_2 and P_3. If both P_2 and P_3 are corrupted, then $\mathcal{S}$ obtains from $\mathcal{A}$ the values x_2 and x_3 that they send

to f_{maj}, computes $x' = f_{\mathsf{maj}}(x, x_2, x_3)$ and simulates the trusted party sending x' back to all parties. If only one of P_2 and P_3 are corrupted, then $\mathcal{S}$ simulates the trusted party sending x back to all parties. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

The fact that the simulation is good is straightforward. If P_1 is corrupted, then only consistency is important, and $\mathcal{S}$ ensures that the value sent to f_{bc} is the one that the honest party/parties would output. If P_1 is not corrupted, and both P_2 and P_3 are corrupted, then P_1 always outputs the correct x as required, and the outputs of P_2 and P_3 are not important. Finally, if P_1 and P_2 are corrupted, then $\mathcal{S}$ sends f_{bc} the value that P_3 would output in the real protocol as required; likewise for P_1 and P_3 corrupted. □

Theorem 3.1.1 implies that f_{maj} cannot be securely computed with guaranteed output delivery for any $t < 3$ in a point-to-point network; this follows immediately from the fact that the broadcast functionality can be securely computed if and only if $t < n/3$. Furthermore, by [56], f_{maj} *can* be securely computed fairly given oblivious transfer (and as shown in Section 3.3.1 this also holds in a point-to-point network). Thus, we have:

Corollary 3.1.2. *Let $t \geq n/3$ and assume that oblivious transfer exists. Then, there exist non-trivial functionalities f such that f can be t-securely computed with fairness but cannot be t-securely computed with guaranteed output delivery, in a point-to-point network.*

Three-party functionalities that imply broadcast. It is possible to generalize the property that we used to show that f_{maj} implies broadcast. Specifically, consider a functionality f with the property that there exist inputs (x_1, x_2, x_3) and (x'_1, x'_2, x'_3) such that $f(x_1, x_2, x_3) = 0$ and $f(x'_1, x'_2, x'_3) = 1$, and such that if either of x_2 or x_3 (resp., x'_2 or x'_3) are changed arbitrarily, then the output of f remains the same. Then, this function can be used to achieve broadcast. We describe the required property formally inside the proof of the theorem below. We show that out of the 256 functions over 3-bit inputs, there are 110 of them with this property. It follows that none of these can be securely computed with guaranteed output delivery in the presence of one or two corrupted parties. We prove the following:

Theorem 3.1.3. *There are* 110 *functions from the family of all three-party Boolean functions $\{f\colon \{0,1\} \times \{0,1\} \times \{0,1\} \to \{0,1\}\}$ that cannot be securely computed with guaranteed output delivery in a point-to-point network with $t = 1$ or $t = 2$.*

Proof. We provide a combinatorial proof of the theorem, by counting how many functions have the property that arbitrarily changing one of the inputs does not affect the output, and there are inputs that yield output 0 and inputs that yield output 1. As we have seen in the proof of Theorem 3.1.1, it is possible to securely realize the broadcast functionality given a protocol that securely computes any such functionality with guaranteed output delivery.

We prove that there are 110 functions $f\colon \{0,1\}^3 \to \{0,1\}$ in the union of the following sets F_1, F_2, F_3:

1. Let F_1 be the set of all functions for which there exist $(a, b, c), (a', b', c') \in \{0,1\}^3$ such that $f(a, b, \cdot) = f(a, \cdot, c) = 1$ and $f(a', b', \cdot) = f(a', \cdot, c') = 0$.

2. Let F_2 be the set of all functions for which there exist $(a, b, c), (a', b', c') \in \{0,1\}^3$ such that $f(a, b, \cdot) = f(\cdot, b, c) = 1$ and $f(a', b', \cdot) = f(\cdot, b', c') = 0$.

3. Let F_3 be the set of all functions for which there exist $(a,b,c),(a',b',c') \in \{0,1\}^3$ such that $f(\cdot,b,c) = f(a,\cdot,c) = 1$ and $f(\cdot,b',c') = f(a',\cdot,c') = 0$.

Observe that any function in one of these sets can be used to achieve broadcast, as described above. Based on the inclusion-exclusion principle and using Lemma 3.1.5 proven below, it follows that:

$$|F_1 \cup F_2 \cup F_3| = 3 \cdot 50 - 3 \cdot 16 + 8 = 110,$$

as required. We first prove the following lemma:

Lemma 3.1.4. *If $f \in F_1$, then $a \neq a'$, if $f \in F_2$ then $b \neq b'$ and if $f \in F_3$ then $c \neq c'$.*

Proof. Let $f \in F_1$ (the proof for F_2, F_3 is similar) and let $a, a', b, b', c, c' \in \{0,1\}$ be inputs fulfilling the condition for the set F_1. Then,

$$f(a,b,c) = f(a,\bar{b},c) = f(a,b,\bar{c}) = 1 \quad \text{and} \quad f(a',b',c') = f(a',\bar{b}',c') = f(a',b',\bar{c}') = 0.$$

On the one hand, $f(a,b',c) = 1$ because $f(a,\cdot,c) = 1$. On the other hand, if $a = a'$ then $f(a,b',c) = f(a',b',c) = 0$, because $f(a',b',\cdot) = 0$. □

It remains to prove the following lemma, to derive the theorem.

Lemma 3.1.5.

1. $|F_1| = |F_2| = |F_3| = 50$.
2. $|F_1 \cap F_2| = |F_1 \cap F_3| = |F_2 \cap F_3| = 16$.
3. $|F_1 \cap F_2 \cap F_3| = 8$.

Proof. Let $f\colon \{0,1\}^3 \to \{0,1\}$ be a function, and consider the representation of f using a binary string $(\beta_0\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7)$ as shown in Table 3.1:

0	0	0	β_0
0	0	1	β_1
0	1	0	β_2
0	1	1	β_3
1	0	0	β_4
1	0	1	β_5
1	1	0	β_6
1	1	1	β_7

Table 3.1: Representation of a Boolean function $\{0,1\}^3 \to \{0,1\}$

1. Assume $f \in F_1$ (the proof for F_2, F_3 is similar). The first quadruple $(\beta_0\beta_1\beta_2\beta_3)$ corresponds to $a = 0$ and the second quadruple $(\beta_4\beta_5\beta_6\beta_7)$ corresponds to $a = 1$. There exists b, c such that $f(a,b,c) = f(a,\bar{b},c) = f(a,b,\bar{c})$ and b', c' such that $f(\bar{a},b',c') = f(\bar{a},\bar{b}',c') = f(\bar{a},b',\bar{c}')$, in addition, $f(a,b,c) \neq f(\bar{a},b',c')$. Therefore, in each such quadruple there must be a triplet of 3 identical bits, and the two triplets have opposite values.

   Denote $\beta = f(a,b,c)$, there are 5 options for $(\beta_0\beta_1\beta_2\beta_3)$ in which 3 of the bits equal β:

$$(\beta\beta\beta\beta), (\beta\beta\beta\bar{\beta}), (\beta\beta\bar{\beta}\beta), (\beta\bar{\beta}\beta\beta), (\bar{\beta}\beta\beta\beta).$$

For each such option, there are 5 options for $(\beta_4\beta_5\beta_6\beta_7)$ in which 3 of the bits equal $\bar{\beta}$:

$$(\bar{\beta}\bar{\beta}\bar{\beta}\bar{\beta}), (\bar{\beta}\bar{\beta}\bar{\beta}\beta), (\bar{\beta}\bar{\beta}\beta\bar{\beta}), (\bar{\beta}\beta\bar{\beta}\bar{\beta}), (\beta\bar{\beta}\bar{\beta}\bar{\beta}).$$

There are 2 options for the value of β, so in total $|F_1| = 2 \cdot 5 \cdot 5 = 50$.

2. Assume $f \in F_1 \cap F_2$ (the proof for $F_1 \cap F_3, F_2 \cap F_3$ is similar). In this case $a' = \bar{a}$ and $b' = \bar{b}$ and the constraints are

$$f(a,b,c) = f(\bar{a},b,c) = f(a,\bar{b},c) = f(a,b,\bar{c}) \neq f(\bar{a},\bar{b},c') = f(a,\bar{b},c') = f(\bar{a},b,c') = f(\bar{a},\bar{b},\bar{c}').$$

Therefore, the string is balanced (there are 4 zeros and 4 ones), where 3 of the bits $(\beta_0\beta_1\beta_2\beta_3)$ are equal to β and one to $\bar{\beta}$, and 3 of the bits $(\beta_4\beta_5\beta_6\beta_7)$ are equal to $\bar{\beta}$ and one to β.

There are 4 options to select 3 bits in $(\beta_0\beta_1\beta_2\beta_3)$, and 2 options to select one bit in $(\beta_4\beta_5\beta_6\beta_7)$. These two options correspond either to $(\bar{a},b,c)$ or $(\bar{a},\bar{b},\bar{c})$. Hence, $|F_1 \cap F_2| = 2 \cdot 4 \cdot 2 = 16$.

3. Assume $f \in F_1 \cap F_2 \cap F_3$. In this case $a' = \bar{a}$, $b' = \bar{b}$ and $c' = \bar{c}$ and the constraints are

$$f(a,b,c) = f(\bar{a},b,c) = f(a,\bar{b},c) = f(a,b,\bar{c}) \neq f(\bar{a},\bar{b},\bar{c}) = f(a,\bar{b},\bar{c}) = f(\bar{a},b,\bar{c}) = f(\bar{a},\bar{b},c).$$

Therefore, the string is of the form $(\beta_0\beta_1\beta_2\beta_3\bar{\beta}_0\bar{\beta}_1\bar{\beta}_2\bar{\beta}_3)$, where 3 of the bits $(\beta_0\beta_1\beta_2\beta_3)$ are equal to β and one to $\bar{\beta}$.

There are 4 options to select 3 bits in $(\beta_0\beta_1\beta_2\beta_3)$, and setting them to the same value determines the rest of the string. Hence, $|F_1 \cap F_2 \cap F_3| = 2 \cdot 4 = 8$.

□

This completes the proof of Theorem 3.1.3. □

As we have mentioned in the Introduction, in the case that $t = 1$ (i.e., when there is an honest majority), all functions can be securely computed with fairness in a point-to-point network. Thus, we have that all 110 functions of Theorem 3.1.3 constitute a *separation* of fairness from guaranteed output delivery. That is, in the case of $n/3 \leq t < n/2$, we have that *many functions* can be securely computed with fairness but not with guaranteed output delivery. In addition, 8 out of these 110 functions reduce to three-party majority and so can be computed fairly for any $t \leq n$. Thus, these 8 functions form a separation for the range of $t \geq n/2$.

3.2 Fairness Implies Guaranteed Output Delivery for Default-Output Functionalities

In this section, we prove Theorem 1.2.2. In fact, we prove a stronger theorem, stating that fairness implies guaranteed output delivery for functions with the property that there exists a

"default value" such that any single party can fully determine the output to that value. For example, the multiparty Boolean AND and OR functionalities both have this property (for the AND functionality any party can always force the output to be 0, and for the OR functionality any party can always force the output to be 1). We call such a function a default-output functionality.[24] Intuitively, such a function can be securely computed with guaranteed output delivery if it can be securely computed fairly, since the parties can first try to compute it fairly. If they succeed, then they are done. Otherwise, they all received abort and can just output their respective default-output value for the functionality. This can be simulated since any single corrupted party in the ideal model can choose an input that results in the default-output value.

Definition 3.2.1. *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality. f is called a* default-output functionality *with default output $(\tilde{y}_1,\ldots,\tilde{y}_n)$, if for every $i \in [n]$ there exists a special input $\tilde{x}_i$ such that for every x_j with $j \neq i$ it holds that $f(x_1,\ldots,\tilde{x}_i,\ldots,x_n) = (\tilde{y}_1,\ldots,\tilde{y}_n)$.*

As an example for a default-output functionality, consider the n-party Boolean OR functionality, $f_{\mathsf{OR}}\colon \{0,1\}^n \to \{0,1\}^n$, where each party P_i has an input bit $x_i \in \{0,1\}$, and the output of each party is the OR of all the inputs, i.e.,

$$f_{\mathsf{OR}}(x_1,\ldots,x_n) = (x,\ldots,x) \quad \text{where} \quad x = x_1 \vee \ldots \vee x_n.$$

Observe that $(1,\ldots,1)$ is a default output for the Boolean OR function (and similarly, $(0,\ldots,0)$ is a default output for the Boolean AND function). We now prove that if a functionality f has a default-output value, then the existence of a fair protocol for f implies the existence of a protocol with guaranteed output delivery for f.

Theorem 3.2.2. *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be a default-output functionality and let $t < n$. If f can be t-securely computed with fairness in the point-to-point model with authenticated channels (with or without a broadcast channel), then f can be t-securely computed with guaranteed output delivery, in the point-to-point model with authenticated channels.*

Proof. Let f be as in the theorem statement, and let the default output be $(\tilde{y}_1,\ldots,\tilde{y}_n)$. Assume that f can be t-securely computed with fairness with or without a broadcast channel. By Theorem 3.3.1, f can be securely computed with fairness without a broadcast channel. We now construct a protocol π that t-securely computes f with guaranteed output delivery in the (f,fair)-hybrid model:

1. Each P_i sends its input x_i to the trusted party computing f.
2. Denote by y_i the value received by P_i from the trusted party.
3. If $y_i \neq \perp$, P_i outputs y_i, otherwise P_i outputs $\tilde{y}_i$.

Let $\mathcal{A}$ be an adversary attacking the execution of π in the (f,fair)-hybrid model. We construct an ideal-model adversary $\mathcal{S}$ in the ideal model with guaranteed output delivery. Let $\mathcal{I}$ be the set of corrupted parties, let $i \in \mathcal{I}$ be one of the corrupted parties (if no parties are corrupted then there is nothing to simulate), and let $\tilde{x}_i$ be the input guaranteed to exist by Definition 3.2.1. Then, $\mathcal{S}$ invokes $\mathcal{A}$ and simulates the interaction of $\mathcal{A}$ with the trusted party computing f (note

[24] In Chapter 4 we denote public-output functionalities satisfying this property as 1-dominated functionalities.

that there is no interaction between $\mathcal{A}$ and honest parties). $\mathcal{S}$ receives the inputs that $\mathcal{A}$ sends to f. If any of the inputs equal abort then $\mathcal{S}$ sends $\tilde{x}_i$ as P_i's input to its own trusted party computing f (with guaranteed output delivery), and arbitrary inputs for the other parties. Then, $\mathcal{S}$ simulates the corrupted parties receiving $\perp$ as output from the trusted party in π, and outputs whatever $\mathcal{A}$ outputs. Else, if none of the inputs equal abort, then $\mathcal{S}$ sends its trusted party the inputs that $\mathcal{A}$ sent. $\mathcal{S}$ then receives the outputs of the corrupted parties from its trusted party, and internally sends these to $\mathcal{A}$ as the corrupted parties' outputs from the trusted party computing f in π. Finally, $\mathcal{S}$ outputs whatever $\mathcal{A}$ outputs.

If $\mathcal{A}$ sends abort, then in the real execution every honest party P_j outputs $\tilde{y}_j$. However, since $\mathcal{S}$ sends the input $\tilde{x}_i$ to the trusted party computing f, by Definition 3.2.1 we have that the output of every honest party P_j in the ideal execution is also $\tilde{y}_j$. Furthermore, if $\mathcal{A}$ does not send abort, then $\mathcal{S}$ just uses exactly the same inputs that $\mathcal{A}$ sent. It is clear that the view of $\mathcal{A}$ is identical in the execution of π and the simulation with $\mathcal{S}$. We therefore conclude that π t-securely computes f with guaranteed output delivery, as required. □

We have proven that fairness implies guaranteed output delivery for default-output functionalities; it remains to show the existence of fair protocols for some default-output functionalities. Fortunately, this was already proven in [56]. The only difference is that [56] uses a broadcast channel. Noting that the multiparty Boolean OR functionality is non-trivial (in the sense of Footnote 2) and that it has default output $(1, \ldots, 1)$ as mentioned above, we have the following corollary.

Corollary 3.2.3. *Assume that oblivious transfer exists. Then, there exist non-trivial functionalities f that can be t-securely computed with guaranteed output delivery in a point-to-point network, for any $t < n$.*

Feasibility of guaranteed output delivery. In Theorem 3.2.4, we prove that 16 non-trivial functionalities can be securely computed with guaranteed output delivery in a point-to-point network (by showing that they are default-output functionalities). Thus, guaranteed output delivery can be achieved for a significant number of functions.

Theorem 3.2.4. *Assume that oblivious transfer exists. There are* 16 *non-trivial functions from the family of all three-party Boolean functions $\{f\colon \{0,1\} \times \{0,1\} \times \{0,1\} \to \{0,1\}\}$ that can be securely computed with guaranteed output delivery in a point-to-point network for any number of corrupted parties.*

Proof. When represented using its truth table as a binary string (see Table 3.1), the three-party Boolean OR function is (01111111), similarly, the Boolean AND function is (00000001). Every function $(\beta_0\beta_1\beta_2\beta_3\beta_4\beta_5\beta_6\beta_7)$ such that there exists i for which $\beta_i = \beta$ and for every $j \neq i$ $\beta_j = \bar{\beta}$ can be reduced to computing Boolean OR. Since there are 8 ways to choose i and 2 ways to choose β, we conclude that there are 16 such functions. □

3.3 The Role of Broadcast

In this section, we prove Theorem 1.2.3 and show that a functionality can be securely computed fairly with broadcast if and only if it can be securely computed fairly without broadcast. In

addition, we show that if a functionality can be securely computed with fairness, then given a broadcast channel, it can be securely computed with guaranteed output delivery.

We start by defining the n-party broadcast functionality, $f_{\mathsf{bc}}\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$, where the sender P_1 has an input $x \in \{0,1\}^*$ while all other parties have the empty input λ (in plain English, this means that only the first party P_1 has input). The output of each party is x.

$$f_{\mathsf{bc}}(x, \lambda, \ldots, \lambda) = (x, \ldots, x).$$

3.3.1 Fairness is Invariant to Broadcast

Gordon and Katz [56] constructed two fair multiparty protocols, both of them require a broadcast channel. In this section, we show that fairness holds for both even without a broadcast channel. More generally, fairness can be achieved with a broadcast channel if and only if it can be achieved without a broadcast channel.

It is immediate that fairness without broadcast implies fairness with broadcast. The other direction follows by using the protocol of Fitzi et al. [47] for detectable broadcast. In the first stage, the parties execute a protocol that establishes a public-key infrastructure. This protocol is independent of the parties' inputs and is computed with abort. If the adversary aborts during this phase, it learns nothing about the output and fairness is retained. If the adversary does not abort, the parties can use the public-key infrastructure and execute multiple (sequential) instances of authenticated broadcast, and so can run the original protocol with broadcast that is fair.

One subtlety arises since the composition theorem replaces every ideal call to the broadcast functionality with a protocol computing broadcast. However, in this case, each authenticated-broadcast protocol relies on the same public-key infrastructure that is generated using a protocol with abort. We therefore define a reactive ideal functionality which allows abort only in the first "setup" call. If no abort was sent in this call, then the functionality provides a fully secure broadcast (with guaranteed output delivery) from there on. The protocol of [47] securely computes this functionality with guaranteed output delivery, and thus, constitutes a sound replacement of the broadcast channel (unless an abort took place).

Theorem 3.3.1. *Let f be an n-party functionality and let $t \leq n$. Then, assuming the existence of one-way functions, f can be t-securely computed with fairness assuming a broadcast channel if and only if f can be t-securely computed with fairness in a point-to-point model with authenticated channels.*

Proof sketch. If f can be t-securely computed with fairness in a point-to-point network, then it can be t-securely computed with fairness with a broadcast channel by just ignoring the broadcast channel.

Next, assume that f can be t-securely computed with fairness assuming a broadcast channel. We now show that it can be t-securely computed with fairness in a point-to-point network. We define the reactive functionality for *conditional broadcast* f_{condbc}. In the first call to f_{condbc}, the functionality computes the AND function, i.e., each party has an input bit b_i and the functionality returns $b = b_1 \wedge \ldots \wedge b_n$ to each party. In addition, the functionality stores the bit b as its internal state for all future calls. In all future calls to f_{condbc}, if $b = 1$ it behaves exactly like f_{bc}, whereas if $b = 0$ it returns $\perp$ to all the parties in the first call and halts. By

inspection, it is immediate that the protocol of [47] securely computes f_{condbc} with guaranteed output delivery, for any $t \leq n$ in a point-to-point network.

Let π be the protocol that t-securely computes f assuming a broadcast channel; stated differently, π t-securely computes f in the $(f_{\mathsf{bc}}, \mathsf{full})$-hybrid model. We construct a protocol π' for t-securely computing f in the $(f_{\mathsf{condbc}}, \mathsf{full})$-hybrid model. π' begins by all parties sending the bit 1 to f_{condbc} and receiving back output. If a party receives back $b = 0$, it aborts and outputs $\perp$. Else, it runs π with the only difference that all broadcast messages are sent to f_{condbc} instead of to f_{bc}. Since f_{condbc} behaves exactly like f_{bc} as long $b = 1$ is returned from the first call, we have that in this case the output of π and π' is identical. Furthermore, π' is easily simulated by first invoking the adversary $\mathcal{A}'$ for π' and obtaining the corrupted parties' inputs to f_{condbc} in the first call. If any 0 bit is sent, then the simulator $\mathcal{S}'$ for π' sends abort to the trusted party, outputs whatever $\mathcal{A}'$ outputs and halts. Otherwise, it invokes the simulator $\mathcal{S}$ that is guaranteed to exist for π on the residual adversary $\mathcal{A}$ that is obtained by running $\mathcal{A}'$ until the end of the first call to f_{condbc} (including $\mathcal{A}'$ receiving the corrupted parties' output bits from this call). Then, $\mathcal{S}'$ sends whatever $\mathcal{S}$ wishes to send to the trusted party, and outputs whatever $\mathcal{S}$ outputs. Since f_{condbc} behaves exactly like f_{bc} when $b = 1$ in the first phase, we have that the output distribution generated by $\mathcal{S}'$ is identical to that of $\mathcal{S}$ when $b = 1$. Furthermore, when $b = 0$, it is clear that the simulation is perfect. □

3.3.2 Fairness with Identifiable Abort Implies Guaranteed Output Delivery

Before proceeding to prove that fairness implies guaranteed output delivery in a model with a broadcast channel, we first show that fairness with identifiable abort implies guaranteed output delivery. Recall that a protocol securely computes a functionality f with identifiable abort, if when the adversary causes an abort all honest parties receive $\perp$ as output along with the identity of a corrupted party. If a protocol securely computes f with fairness and identifiable abort, then it is guaranteed that if the adversary aborts, it learns nothing about the output and all honest parties learn an identity of a corrupted party. In this situation, the parties can eliminate the identified corrupted party and execute the protocol again, where an arbitrary party emulates the operations of the eliminated party using a default input. Since nothing was learned by the adversary when an abort occurs, the parties can rerun the protocol from scratch (without the identified corrupted party) and nothing more than a single output will be revealed to the adversary. Specifically, given a protocol π that computes f with fairness and identifiable abort, we can construct a new protocol π' that computes f with guaranteed output delivery. In the protocol π', the parties iteratively execute π, where in each iteration, either the adversary does not abort and all honest parties receive consistent output, or the adversary aborts without learning anything and the parties identify a corrupted party, who is eliminated from the next iteration.

Theorem 3.3.2. *Let f be an n-party functionality and let $t \leq n$. If f can be t-securely computed with fairness and identifiable abort, then f can be t-securely computed with guaranteed output delivery.*

Proof. We prove the theorem by constructing a protocol π that t-securely computes f with guaranteed output delivery in the $(f, \mathsf{id\text{-}fair})$-hybrid model. For every party P_i, we assign a default input value $\tilde{x}_i$ and construct the protocol π as follows:

Protocol 3.3.3. *(id-fairness to full security)*

1. *Let $\mathcal{P}_1 = \{1, \ldots, n\}$ denote the set of indices of all participating parties.*
2. *For $i = 1, \ldots, t+1$*
   (a) *All parties in $\mathcal{P}_i$ send their inputs to the trusted party computing f, where the party with the lowest index in $\mathcal{P}_i$ simulates all parties in $\mathcal{P}_1 \setminus \mathcal{P}_i$, using their predetermined default input values.*
   For each $j \in \mathcal{P}_i$, denote the output of P_j from f by y_j.
   (b) *For every $j \in \mathcal{P}_i$, party P_j checks whether y_j is a valid output, if so P_j outputs y_j and halts. Otherwise, all parties received $(\perp, i^*)$ as output, where i^* is an index of a corrupted party. If $i^* \notin \mathcal{P}_i$ (and so i^* is a previously identified corrupted party), then all parties set i^* to be the party with the lowest index in $\mathcal{P}_i$.*
   (c) *Set $\mathcal{P}_{i+1} = \mathcal{P}_i \setminus \{i^*\}$.*

First note that there are at most $t+1$ iterations; therefore, π terminates in polynomial time. Let $\mathcal{A}$ be an adversary attacking π and let $\mathcal{I}$ be the set of corrupted parties. We construct a simulator $\mathcal{S}$ for the ideal model with f and guaranteed output delivery, as follows. $\mathcal{S}$ invokes $\mathcal{A}$ and receives its inputs to f in every iteration. If an iteration contains an abort, then $\mathcal{S}$ simulates sending the response $(\perp, i^*)$ to all parties, and proceeds to the next iteration. In the first iteration in which no abort is sent (and such an iteration must exist since there are $t+1$ iterations and in every iteration except for the last one corrupted party is removed), $\mathcal{S}$ sends the inputs of the corrupted parties that $\mathcal{A}$ sent to the trusted party computing f. In addition, $\mathcal{S}$ sends the values for any corrupted parties that were identified in previous iterations: if the lowest index remaining is honest, then $\mathcal{S}$ sets these values to be the default values; else, it sets these values to be the values sent by $\mathcal{A}$ for these parties. Upon receiving the output from its trusted party, $\mathcal{S}$ hands it to $\mathcal{A}$ as if it were the output of the corrupted parties in the iteration of π, and outputs whatever $\mathcal{A}$ outputs.

The simulation in the $(f, \mathsf{id\text{-}fair})$-hybrid model is perfect since $\mathcal{S}$ can perfectly simulate the trusted party for all iterations in which an abort is sent. Furthermore, in the first iteration for which an abort is not sent, $\mathcal{S}$ sends f the exact inputs upon which the function f is computed in the protocol. Thus, the view of $\mathcal{A}$ and the output of the honest parties in the simulation with $\mathcal{S}$ are identical to their view and output in an execution of π in the $(f, \mathsf{id\text{-}fair})$-hybrid model. □

3.3.3 Fairness with Broadcast Implies Guaranteed Output Delivery

In Section 3.3.2, we saw that if a functionality can be securely computed with fairness and identifiable abort, then it can be securely computed with guaranteed output delivery. In this section, we show that assuming the existence of a broadcast channel, there is a protocol compiler that given a protocol computing a functionality f with fairness, outputs a protocol computing f with fairness and identifiable abort. Therefore, assuming broadcast, fairness implies guaranteed output delivery.

The protocol compiler we present is a modification of the GMW compiler, which relies on the code of the underlying fair protocol and requires non-black-box access to the protocol. (Therefore, this result does not contradict the proof in Section 3.4 that black-box access to an

ideal functionality that computes f with fairness does not help to achieve guaranteed output delivery.) The underlying idea is to use the GMW compiler [53, 51]. However, instead of enforcing semi-honest behaviour, the compiler is used in order to achieve security with identifiable abort. This is accomplished by tweaking the GMW compiler so that first only public-coin zero-knowledge proofs are used, and second if an honest party detects dishonest behaviour—i.e., if some party does not send a message or fails to provide a zero-knowledge proof for a message it sent—the honest parties record the identity i^* of the cheating party. We stress that the parties do *not* abort the protocol at this point, but rather continue until the end to see if they received $\perp$ or not. If they received $\perp$, then they output $(\perp, i^*)$ and halt. Else, if they received proper output, then they output it. Note that if the parties were to halt as soon as they detected a cheating party, then this would not be secure since it is possible that some of the corrupted parties already received output by that point. Thus, they conclude the protocol to determine whether they should abort or not.

The soundness of this method holds because in the GMW compiler with public-coin zero-knowledge proofs, a corrupted party cannot make an honest party fail, and all parties can verify if the zero-knowledge proof was successful or not. A brief description of the GMW compiler appears in Section 3.6.1. We prove the following:

Theorem 3.3.4. *Assume the existence of one-way functions and let $t \leq n$. If a functionality f can be t-securely computed with fairness in the broadcast model (where all the communication is over the broadcast channel), then f can be t-securely computed with guaranteed output delivery.*

Proof. We begin by proving that fairness with a broadcast channel implies fairness with identifiable abort.

Lemma 3.3.5. *Assume the existence of one-way functions and let $t \leq n$. Then, there exists a polynomial-time protocol compiler that receives any protocol π, running over a broadcast channel, and outputs a protocol π', such that if π t-securely computes a functionality f with fairness then π' t-securely computes f with fairness and identifiable abort.*

Proof sketch. Since the protocol is run over a single broadcasts channel, if at any point a party does not broadcast a message when it is supposed to, then all the parties detect it and can identify this party as corrupted.

We consider a tweaked version of the GMW compiler. The input-commitment phase and the coin-generation phase are kept the same, with the sole exception that if a party is identified as corrupted at this stage (e.g., if it does not send any value) then all the parties hard-wire to the function the default input value corresponding to this party. In the protocol-emulation phase, when a sender transmits a message to a receiver, they execute a strong zero-knowledge proof of knowledge with perfect completeness, in which the sender acts as the prover and the receiver as the verifier. The statement is that the message was constructed by the next-message function, based on the sender's input, random coins and the history of all the messages the sender received in the protocol. However, if the prover fails to prove the statement, unlike in the GMW compiler, the verifier does not immediately broadcast the verification coins, but stores the verification coins along with the identity of the sender in memory, and resumes the protocol.

At the end of the protocol emulation, each party checks whether it received an output, if so it outputs it and halts. If a party did not receive an output and it received a message for

which the corresponding zero-knowledge proof failed, it broadcasts the verification coins it used during the zero-knowledge proof. In this case, the other parties verify whether this is a justified reject, and if so they output $\perp$ along with the identity of the prover. If the reject is not justified, the parties output $\perp$ along with the identity of the party that sent the false verification coins.

Since the zero-knowledge proof has perfect completeness, a corrupted party cannot produce verification coins that will falsely reject an honest party. Hence, only parties that deviate from the protocol can be identified as corrupted.

It case each honest party finishes the execution of the compiled protocol with some output, the compiled protocol remains secure, based on the security of the underlying protocol and of the zero-knowledge proof.

In case one of the honest parties did not get an output, there must be at least one message that does not meet the protocol's specification, hence at least one honest party received a message without a valid proof. Therefore, all the honest parties output $\perp$ along with an identity of a corrupted party. However, in this situation, the adversary does not learn anything about the output, since otherwise there exists an attack violating the fairness of the underlying protocol π. Hence, the compiled protocol retains fairness. □

Applying Theorem 3.3.2 to Lemma 3.3.5 we have that f can be t-securely computed with guaranteed output delivery, completing the proof of the theorem. □

3.4 Black-Box Fairness does not Help for Guaranteed Output Delivery

In this section, we show that the ability to securely compute a functionality with complete fairness does not assist in computing the functionality with guaranteed output delivery, at least in a black-box manner. More precisely, a functionality f can be securely computed with guaranteed output delivery in the (f, fair)-hybrid model if and only if f can be securely computed with guaranteed output delivery in the plain model.

The idea is simply that any protocol that provides guaranteed output delivery in the (f, fair)-hybrid model has to work even if the output of every call to the trusted party computing f fairly concludes with an abort. This is because a corrupted party can always send abort to the trusted party in every such call.

Proposition 3.4.1. *Let f be an n-party functionality and let $t \leq n$. Then, f can be t-securely computed in the (f, fair)-hybrid model with guaranteed output delivery if and only if f can be t-securely computed in the real model with guaranteed output delivery.*

Proof sketch. If f can be t-securely computed in the real model with guaranteed output delivery, then clearly it can be t-securely computed in the (f, fair)-hybrid model with guaranteed output delivery by simply not sending anything to the trusted party.

For the other direction, let π be a protocol that t-securely computes f in the (f, fair)-hybrid model with guaranteed output delivery. We construct a protocol π' in the real model which operates exactly like π, except that whenever there is a call in π to the ideal functionality f, the parties in π' emulate receiving $\perp$ as output. It is immediate that for every adversary $\mathcal{A}'$ for

π', there exists an adversary $\mathcal{A}$ for π so that the output distributions of the two executions are identical ($\mathcal{A}$ just sends abort to every ideal call in π, and otherwise sends the same messages that $\mathcal{A}'$ sends). By the assumption that π is secure, there exists a simulator $\mathcal{S}$ for the ideal model for f with guaranteed output delivery. This implies that $\mathcal{S}$ is also a good simulator for $\mathcal{A}'$ in π', and so π' t-securely computes f with guaranteed output delivery in the real model. □

3.5 Additional Results

In this section, we prove two additional results. First, there exist functionalities for which identifiable abort cannot be achieved (irrespective of fairness), and fairness and guaranteed output delivery are equivalent for fail-stop adversaries.

3.5.1 Broadcast is Necessary for Identifiable Abort

We show that security with identifiable abort cannot be achieved in general without assuming a broadcast channel.

Proposition 3.5.1. *Assume the existence of one-way functions and let $t \geq n/3$. There exist functionalities that cannot be t-securely computed with identifiable abort, in the point-to-point network model.*

Proof sketch. Assume by contradiction that the PKI setup functionality defined by

$$f_{\mathsf{PKI}}(\lambda, \ldots, \lambda) = ((\vec{pk}, sk_1), \ldots, (\vec{pk}, sk_n)),$$

can be t-securely computed with identifiable abort for $t = n/3$, where $\vec{pk} = (pk_1, \ldots, pk_n)$ and each (pk_i, sk_i) are a public/private key pair for a secure digital-signature scheme (that exists if one-way function exists). Then, we can t-securely compute f_{bc} by running the protocol π that is assumed to exist for f_{PKI}, where π is t-secure with identifiable abort. As in the proof of Theorem 3.3.2, if π ends with abort, then the party who is identified as corrupted is removed. This continues iteratively until the π terminates without abort, in which case a valid PKI is established between all remaining parties. Given this PKI, the parties can run authenticated broadcast in order to securely compute f_{bc}. Since f_{bc} cannot be securely computed for $t = n/3$, we have a contradiction. □

3.5.2 Fairness Implies Guaranteed Output Delivery for Fail-Stop Adversaries

In the presence of malicious adversaries, fairness and guaranteed output delivery are different notions, since there exist functionalities that can be computed with complete fairness but cannot be computed with guaranteed output delivery. In the presence of semi-honest adversaries, it is immediate that both notions are equivalent, since the adversary cannot abort. In this section, we show that in the presence of the fail-stop adversaries, i.e., when the corrupted parties follow the protocol with the exception that the adversary is allowed to abort, fairness implies guaranteed output delivery.

The underlying idea is that if a corrupted party does not send a message to an honest party during the execution of a fair protocol, the honest party can inform all parties that it identified a corrupted party. Since the adversary is fail-stop, corrupted parties cannot lie and falsely incriminate an honest party. Similarly to the proof of Theorem 3.3.4, the parties do not halt if a party is detected cheating (i.e., halting early). Rather, the parties continue to the end of the protocol: if the protocol ended with output, then they take the output and halt; otherwise, they remove the cheating party and begin again. Since the original protocol is fair, this guarantees that nothing is learned by any party if anyone receives abort; thus, they can safely run the protocol again. As in the proof of Theorem 3.3.2, this process is repeated iteratively until no abort is received. We conclude that:

Theorem 3.5.2. *Let f be an n-party functionality and let $t \leq n$. Then, f can be t-securely computed with fairness in the presence of fail-stop adversaries, if and only if f can be t-securely computed with guaranteed output delivery in the presence of fail-stop adversaries.*

3.6 Overview of Related Protocols

In this section, we present a high-level overview of the protocols that were used above.

3.6.1 The GMW Compiler

The GMW compiler [53] consists of a pre-compiler and an authenticated-computation compiler:

- The pre-compiler produces a protocol that behaves as the original protocol, but instead of using a point-to-point network, all the communication is sent over a single broadcast channel. Each party generates a pair of keys for a public-key encryption scheme and broadcasts the encryption key. Next, every message is encrypted under the public key of the receiver and sent over the broadcast channel.

- The authenticated-computation compiler produces a protocol which may abort, but otherwise is enforced to behave as the input protocol. This compiler consists of an input-commitment phase, a coin-generation phase and a protocol-emulation phase.

    1. In the input-commitment phase, every party commits to its input towards all other parties.
    2. In the coin-generation phase, the parties jointly generate random tapes for each party. Each party receives its random tape and commitments for the random tapes of all other parties.
    3. In the protocol-emulation phase, the parties emulate the input protocol, where for each message, the sending party and the receiving party execute a zero-knowledge proof, proving that the message is produced by the next-message function based on the input, random tape and all prior messages.
    Note that a malicious party may abort the execution during this phase by not sending a message or by providing an invalid proof. However, when using a public-coin proof,

all parties can publicly verify if the proof is valid or not, and a corrupted party cannot cause an honest prover to fail.

The first part of the GMW compiler transforms any protocol running over a point-to-point communication network into a protocol running over a single broadcast channel, under the assumption that collections of trapdoor permutations exist (in order to obtain public-key encryption). Since we begin with a protocol that works over a broadcast channel, we can ignore this step.

3.6.2 The Detectable Broadcast Protocol of Fitzi et al.

Fitzi et al. [47] constructed protocols for detectable broadcast, i.e., protocols in which either all parties abort and no one receives output, or all parties receive and agree upon the broadcasted value. Two protocols are provided in [47]: the first protocol is in the computational setting, where the channels are authenticated, and is secure facing polynomial-time adversaries, assuming the existence of one-way functions. The second protocol is in the information-theoretic setting, where the channels are ideally secure, and is secure facing computationally unbounded adversaries. Both protocols can tolerate an arbitrary number of corruptions.

More precisely, the protocols in [47] provide detectable precomputation for broadcast, i.e., they compute correlated randomness that can later be used for authenticated broadcast protocols. We will describe the protocol in the computational setting, which computes a public-key infrastructure (PKI).

Initially, every party P_i generates a pair of signing and verification keys of a digital signature scheme and sends the verification key to all the parties. Each party echoes all the verification keys it received to all other parties and locally assigns a grade g_j for each party P_j: 1 if all verification keys it received for P_j are consistent with each other and 0 otherwise. Next, each party computes the logical AND of the grades and invokes an authenticated broadcast protocol (e.g., the protocol from [39]) to distribute its result (the verification keys from the first round are used as the PKI for the authenticated broadcast). Finally, each party computes the logical AND of all n values that were received by the authenticated broadcast protocols. If the result is 0, it aborts, and if the result is 1, it outputs all n verification keys along with its signing key.

If the protocol successfully completes, then the parties establish a PKI that can be used within any MPC protocol to replace broadcast calls with authenticated broadcast protocols, whereas if the protocol aborts then clearly fairness is retained, since the computation is independent of the inputs to the MPC protocols.

3.6.3 The Protocols of Gordon and Katz

Gordon and Katz [56] constructed two fair protocols in the broadcast model, tolerating any number of corruptions, assuming that oblivious transfer exists.

Multiparty Boolean OR. Initially, every party commits to its input bit and broadcasts the commitment; if some party did not broadcast, then all parties output 1. Next, the parties iteratively compute the committed OR functionality (described below) with identifiable abort and eliminate identified corrupted party (or parties) in each iteration until a binary output is obtained.

The committed OR functionality receives from each party its input bit, decommitment information and the vector of commitments from the first round. If all parties provided consistent commitments and valid decommitments, then the functionality computes the Boolean OR on all the input bits and outputs the result. Otherwise, the functionality outputs to each party P_i the set of parties that are not consistent with him, where a party is not consistent with P_i if it provided a different vector of commitments or if it did not provide a valid decommitment.

Correctness follows since all honest parties are always consistent with each other and so will always proceed together to the next iteration in case the committed OR functionality outputs to each party the (local) set of non-consistent parties. Privacy follows since if the adversary learned the value of the committed OR functionality and decided to abort, then it learns useful information about honest parties' inputs only if all corrupted parties use input 0 (indeed if some corrupted party uses input 1 the output will always be 1).

Three-party majority. On input $x_1, x_2, x_3 \in \{0,1\}$ for the parties, the protocol consists of a share-generation phase followed by $m = \omega(\log \kappa)$ rounds. Initially, all the parties compute with abort the share-generation functionality that selects a number i^* from the geometric distribution (with parameter 1/5) and prepares $3(m+1)$ values as follows. For $0 \leq i < i^*$ and $j \in \{1.2.3\}$, select a random bit $\tilde{x}_j$ and compute $b_{i,j} = f_{\mathsf{maj}}(x_{j-1}, \tilde{x}_j, x_{j+1})$, and for $i^* \leq i \leq m$, compute $b_{i,j} = f_{\mathsf{maj}}(x_1, x_2, x_3)$. Next, the functionality prepares authenticated 3-out-of-3 secret shares $(b^1_{i,j}, b^2_{i,j}, b^3_{i,j})$ for each value $b_{i,j}$, and outputs $b^{j'}_{i,j}$ to party $\mathsf{P}_{j'}$. In addition, output $b^j_{0,j}$ to parties P_{j-1} and P_{j+1}.

In case this phase aborts and a corrupted party is identified, the remaining pair compute the Boolean OR of their inputs using the protocol from [57]. Otherwise, the parties run m iterations, where in iteration i each party P_j broadcasts $b^j_{i,j}$ (and $b^j_{m,1}$ in the m'th iteration). If P_j aborts in iteration i then parties P_{j-1} and P_{j+1} exchange $b^{j-1}_{i-1,j}$ and $b^{j+1}_{i-1,j}$ and reconstruct (using the value $b^j_{i-1,j}$) and output the value $b_{i-1,j}$. If two parties abort, the remaining party outputs its own input value. If all iterations completed successfully, the parties reconstruct $b_{m,1}$ and output it.

The key observation used in the proof is that the adversary can either determine the output by choosing identical inputs for two corrupted parties (in which case it does not learn anything about the honest party's input) or learn the honest party's input by choosing opposite inputs for the corrupted parties (in which case it cannot determine the output).

Chapter 4

Characterization of Secure Multiparty Computation without Broadcast

In this chapter we present the characterization of public-output functionalities in the point-to-point model. We start in Section 4.1 by presenting an attack on consistent protocols. In Section 4.2 we use this attack to show impossibility results on secure protocols, and in Section 4.3 we present matching upper bounds, showing that the results are tight.

4.1 Attacking Consistent Protocols

In this section, we present a lower bound for secure protocols in the secure-channels point-to-point model. Protocols in consideration are only assumed to have a very mild security property (discussing the more standard notion of security is deferred to Section 4.2). Specifically, we only require the protocol to be consistent – all honest parties output the same value. We emphasize that in a consistent protocol, a party may output the special error symbol $\perp$ (i.e., abort), but it can only do so if all honest parties output $\perp$ as well.

Definition 4.1.1 (consistent protocols)**.** *A protocol π is (δ, t)-*consistent *against C-class (e.g., polynomial-time, expected polynomial-time) adversaries, if the following holds. Consider an execution of π with any vector of inputs of length κ for the parties, in which a C-class adversary controls at most t parties. Then with probability at least $\delta(\kappa)$, all honest parties output the* same *value, where the probability is taken over the random coins of the adversary and of the honest parties.*

In Section 4.1.1 we present an attack on consistent protocols whose round complexity is strictly bounded, and in Section 4.1.2 we extend the attack to consistent protocols with a bound on their *expected* number of rounds.

4.1.1 Protocols of Strict Running-Time Guarantee

Lemma 4.1.2 (restating Lemma 1.2.6)**.** *Let $n \geq 3$, let $t \geq n/3$, and let $s = n - 2t$ if $t < n/2$ and $s = 1$ otherwise. Let π be an n-party, T-time, q-round protocol in the secure-channels point-to-point model that is $(1 - \delta, t)$-consistent against $(T_{\mathcal{A}} = 2nqT)$-time adversaries. Then,*

there exists a $T_{\mathcal{A}}$-time adversary $\mathcal{A}$ such that given t control over any s-size subset $\mathcal{I}$ of parties, the following holds: on security parameter κ, $\mathcal{A}$ first outputs a value $y^* = y^*(\mathcal{I})$. Next, $\mathcal{A}$ interacts with the remaining honest parties of π on arbitrary inputs of length κ unknown to $\mathcal{A}$, and except for probability at most $\left(\frac{3}{2} \cdot q(\kappa) + 1\right) \cdot \delta(\kappa)$, the output of every *honest party in this execution is y^*.*

For a polynomial-time protocol that is $(1 - \mathrm{neg}, t)$-consistent against PPT adversaries and assuming an honest majority, Lemma 4.1.2 yields a PPT adversary that by controlling $n - 2t$ of the parties can manipulate the outputs of the honest parties (i.e., forcing them all to be y^*) with all but a negligible probability. If an honest majority is not assumed, the adversary can manipulate the outputs of the honest parties, by controlling any single party, except for a negligible probability. We remark that we would get slightly better parameters using an attack in which at least one honest party (but not necessarily all) outputs y^*.

We start by proving the lemma for three-party protocols, and later prove the multiparty case using a reduction to the three-party case. We actually prove a stronger statement for the three-party case, where the value y^* is independent of the set of corrupted parties. In the following lemma, we denote by T the combined running-time of the parties. As opposed to T-time 3-party protocols, this more general measure captures asymmetry between the running time of the parties, and will turn out to be useful for proving Lemma 4.1.2.

Lemma 4.1.3 (attack on three-party protocols)**.** *Let π be a three-party, q-round protocol in the secure-channels point-to-point model, and let T be the combined running-time of all three parties. If π is $(1 - \delta, 1)$-consistent against $(T_{\mathcal{A}} = 2qT)$-time adversaries, then there exists a $T_{\mathcal{A}}$-time adversary $\mathcal{A}$ such that the following holds. On security parameter κ, $\mathcal{A}$ first outputs a value y^*. Next, given control over* any *non-empty set of parties, $\mathcal{A}$ interacts with the remaining honest parties of π on arbitrary inputs of length κ unknown to $\mathcal{A}$, and except for probability at most $\frac{3}{2} \cdot q(\kappa) \cdot \delta(\kappa)$, the output of* every *honest party in this execution is y^*.*

Proof. We fix the input-length parameter κ and omit it from the notation when its value is clear from the context. Let $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ and let $m = q$ (assume for ease of notation that m is even). Consider, without loss of generality, that a single party is corrupted (the case of two corrupted parties follows by letting the adversary simulate an honest party) and assume for concreteness that the corrupted party is C.

Consider the following $3q$-party protocol $\mathsf{R} = (\mathsf{A}^1, \mathsf{B}^1, \mathsf{C}^1, \ldots, \mathsf{A}^m, \mathsf{B}^m, \mathsf{C}^m)$, in which the parties are connected in a ring network such that each two consecutive parties, as well as the first and last, are connected via a secure channel, and party P^j, for $\mathsf{P} \in \{\mathsf{A}, \mathsf{B}, \mathsf{C}\}$, has the code of P. Let $v = \kappa + T(\kappa)$, and consider an execution of R with arbitrary inputs and uniformly distributed random coins for the parties being $\boldsymbol{w} = (w^1_{\mathsf{A}}, w^1_{\mathsf{B}}, w^1_{\mathsf{C}}, \ldots, w^m_{\mathsf{A}}, w^m_{\mathsf{B}}, w^m_{\mathsf{C}}) \in (\{0,1\}^v)^{3m}$ (i.e., party P^i has input w^i_{P}, containing its actual input and random coins).

A key observation is that the point of view of the party A^j, for instance, in such an execution, is a *valid* view of the party A on input w^j_{A} in an execution of π in which B acts honestly on input w^j_{B}. It is also a valid view of A, on input w^j_{A}, in an execution of π in which C acts honestly on input $w^{j-1 \pmod m}_{\mathsf{C}}$. This observation yields the following consistency property of R.

Claim 4.1.4. *Consider an execution of R on joint input $\boldsymbol{w} \in (\{0,1\}^v)^{3m}$, where the parties' coins in $\boldsymbol{w}$ are chosen uniformly at random, and the parties' (actual) inputs are chosen arbitrarily. Then parties of distance d in R, measured by the (minimal) number of communication links*

between them, as well as all $d-1$ parties between them, output the same value with probability at least $1-d\delta$.

Proof. Consider the pair of neighboring parties $\{\mathsf{A}^j, \mathsf{B}^j\}$ in the ring R (an analogous argument holds for any two neighboring parties). Let $\mathcal{A}$ be an adversary, controlling the party C of π that interacts with $\{\mathsf{A}, \mathsf{B}\}$ by emulating an execution of R on $\boldsymbol{w}$ (apart from the roles of $\{\mathsf{A}^j, \mathsf{B}^j\}$), and let $\{\mathsf{A}, \mathsf{B}\}$ take (without knowing that) the roles of $\{\mathsf{A}^j, \mathsf{B}^j\}$ in this execution. The joint view of $\{\mathsf{A}, \mathsf{B}\}$ in this emulation has the same distribution as the joint view of $\{\mathsf{A}^j, \mathsf{B}^j\}$ in an execution of R. Hence, the $(1-\delta)$-consistency of π yields that A^j and B^j output the same value in an execution of R on $\boldsymbol{w}$ with probability at least $1-\delta$. The proof follows by a union bound. This concludes the proof of Claim 4.1.4. □

The adversary $\mathcal{A}$ first selects a value for $\boldsymbol{w} \in (\{0,1\}^v)^{3m}$, consisting of arbitrary input values (e.g., zeros) and uniformly distributed random coins, and sets y^* to be the output of $\mathsf{P}^* = \mathsf{A}^{m/2}$ in the execution of R on $\boldsymbol{w}$. To interact with $\{\mathsf{A}, \mathsf{B}\}$ in π, the adversary $\mathcal{A}$ emulates an execution of R in which all but $\{\mathsf{A}^1, \mathsf{B}^1\}$ have their inputs according to $\boldsymbol{w}$, and $\{\mathsf{A}, \mathsf{B}\}$ take the roles of $\{\mathsf{A}^1, \mathsf{B}^1\}$. The key observation is that the view of party P^* in the emulation induced by the above attack, is the *same* as its view in the execution of R on $\boldsymbol{w}$ (regardless of the inputs of $\{\mathsf{A}, \mathsf{B}\}$). This is true since the execution of R ends after at most m communication rounds. Thus, the actions of $\{\mathsf{A}, \mathsf{B}\}$ have no effect on the view of P^*, and therefore the output of P^* is y^* also in the emulated execution of R. Finally, since all the parties in the emulated execution of R have uniformly distributed random coins, and since the distance between P^* and $\{\mathsf{A}, \mathsf{B}\}$ is (less than) $\frac{3m}{2}$, Claim 4.1.4 yields that with probability at least $1 - \frac{3m}{2} \cdot \delta$, the output of $\{\mathsf{A}, \mathsf{B}\}$ under the above attack is y^*.

Note that the value y^* does not depend on the identity of the corrupted party, since in the first step y^* is set independently of C, and in the second step the attack follows without any change when the honest parties play the roles of $\{\mathsf{B}^1, \mathsf{C}^1\}$ if A is corrupted or $\{\mathsf{A}^2, \mathsf{C}^1\}$ if B is corrupted. □

We now proceed to prove Lemma 4.1.2 in the multiparty case.

Proof. Let $\pi = (\mathsf{P}_1, \ldots, \mathsf{P}_n)$ be a T-time, q-round, n-party protocol that is $(1-\delta, t)$-consistent against $2nqT$-time adversaries. We will show an adversary that by controlling any s corrupted parties, manipulates all honest parties to output a predetermined value. We separately handle the case that $n/3 \leq t < n/2$ and the case that $n/2 \leq t < n$.

Case $n/3 \leq t < n/2$**.** Let $\mathcal{I} \subseteq [n]$ be a subset of size $s = n - 2t$, representing the indices of the corrupted parties in π. Consider the three-party protocol $\pi' = (\mathsf{A}', \mathsf{B}', \mathsf{C}')$, defined by partitioning the set $[n]$ into three subsets $\{\mathcal{I}_{\mathsf{A}'}, \mathcal{I}_{\mathsf{B}'}, \mathcal{I}\}$, where $\mathcal{I}_{\mathsf{A}'}$ and $\mathcal{I}_{\mathsf{B}'}$ are each of size t, and letting party A' run the parties $\{\mathsf{P}_i\}_{i \in \mathcal{I}_{\mathsf{A}'}}$, party B' run the parties $\{\mathsf{P}_i\}_{i \in \mathcal{I}_{\mathsf{B}'}}$ and party C' run the parties $\{\mathsf{P}_i\}_{i \in \mathcal{I}}$. Each of the parties in π' waits until all the virtual parties it is running halt, arbitrarily selects one of them and outputs the virtual party's output value.

Since the subsets $\mathcal{I}_{\mathsf{A}'}, \mathcal{I}_{\mathsf{B}'}, \mathcal{I}$ are of size at most t, the q-round, three-party protocol π' is $(1-\delta, 1)$-consistent against $2nqT$-adversaries (otherwise there exists a $2nqT$-time adversary against the consistency of π, corrupting at most t parties). In addition, since the combined time complexity of all three parties is nT, by Lemma 4.1.3 there exists a $2nqT$-time adversary

$\mathcal{A}'$ that first determines a value y^*, and later, given control over any party in π' (in particular C'), can force the two honest parties to output y^* with probability at least $1 - \frac{3q\delta}{2}$.

The attacker $\mathcal{A}$ for π, controlling the parties indexed by $\mathcal{I}$, is defined as follows: In the first step, $\mathcal{A}$ runs $\mathcal{A}'$ and outputs the value y^* that $\mathcal{A}'$ outputs. In the second step, $\mathcal{A}$ interacts with the honest parties in π by simulating the parties $\{\mathsf{A}', \mathsf{B}'\}$ to $\mathcal{A}'$, i.e., $\mathcal{A}$ runs $\mathcal{A}'$ and sends every message it receives from $\mathcal{A}'$ to the corresponding honest party in π, and similarly, whenever $\mathcal{A}$ receives a message from an honest party in π it forwards it to $\mathcal{A}'$. It is immediate that there exists $i \in \mathcal{I}_{\mathsf{A}'}$ such that P_i outputs y^* in the execution of π with the same probability that A' outputs y^* in the execution of π', i.e., with probability at least $1 - \frac{3q\delta}{2}$. From the consistency property of π, all honest parties output the same value with probability at least $1-\delta$, and using the union bound we conclude that the output of all honest parties in π under the above attack is y^* with probability at least $1 - (\frac{3q\delta}{2} + \delta)$.

Case $n/2 \leq t < n$**.** Let $i^* \in [n]$ be the index of the corrupted party in π and consider the three-party protocol $\pi' = (\mathsf{A}', \mathsf{B}', \mathsf{C}')$ defined by partitioning the set $[n]$ into three subsets $\{\mathcal{I}_{\mathsf{A}'}, \mathcal{I}_{\mathsf{B}'}, \{i^*\}\}$, for $|\mathcal{I}_{\mathsf{A}'}| = \left\lceil \frac{n-1}{2} \right\rceil$ and $|\mathcal{I}_{\mathsf{B}'}| = \left\lfloor \frac{n-1}{2} \right\rfloor$. As in the previous case, the size of each subset $\mathcal{I}_{\mathsf{A}'}, \mathcal{I}_{\mathsf{B}'}, \{i^*\}$ is at most t, and the proof proceeds as above. □

4.1.2 Protocols of Expected Running-Time Guarantee

In this section, we extend the attack presented above to consistent protocols with bound on their *expected* number of rounds.

Lemma 4.1.5. *Let $n \geq 3$, let $t \geq n/3$, let $s = n - 2t$ if $t < n/2$ and $s = 1$ otherwise, and let $z = z(\kappa)$ be an integer function. Let π be an n-party protocol of expected running time T and expected round complexity q in the secure-channels point-to-point model, that is $(1-\delta, t)$-consistent against adversaries with expected running time $T_{\mathcal{A}} = 2n(z+1)qT$. Then, there exists an adversary $\mathcal{A}$ with expected running-time $T_{\mathcal{A}}$ such that given control over any s-size subset $\mathcal{I}$ of parties, the following holds: on security parameter κ, $\mathcal{A}$ first outputs a value $y^* = y^*(\mathcal{I})$. Next, $\mathcal{A}$ interacts with the remaining honest parties of π on arbitrary inputs of length κ unknown to $\mathcal{A}$, and except for probability at most $2 \cdot (3 \cdot q(\kappa) + 1) \cdot \delta(\kappa) + 2^{-z(\kappa)}$, the output of* every *honest party in this execution is y^*.*

Proof. We prove the lemma for the three-party case, the proof for the general case is similar to the proof of Lemma 4.1.2. We fix the input-length parameter κ and omit it from the notation when clear from the context.

Let $\pi = (\mathsf{A}, \mathsf{B}, \mathsf{C})$ and let $m = 2q$. Similarly to the proof of Lemma 4.1.3, we consider the (now double size) ring $\mathsf{R} = (\mathsf{A}^1, \mathsf{B}^1, \mathsf{C}^1, \ldots, \mathsf{A}^m, \mathsf{B}^m, \mathsf{C}^m)$. The attacker $\mathcal{A}$ follows in similar lines to those used in the proof of Lemma 4.1.3. The main difference is that in order to select y^*, the adversary $\mathcal{A}$ iterates the following for z times. In each iteration, $\mathcal{A}$ emulates an execution of the ring R on arbitrary inputs and uniformly distributed random coins,[25] for m communication rounds. If during one of these iterations party $\mathsf{P}^* = \mathsf{A}^{m/2}$ halts, $\mathcal{A}$ sets y^* to be its output in this iteration. Otherwise, in case the value y^* was not set during all z iterations, $\mathcal{A}$ outputs $\perp$ and aborts. The attack continues as in the proof of Lemma 4.1.3.

[25] Note that now we have no a priori bound on the number of random coins used by the parties. Yet, the emulation can be done in expected time nmT.

To analyze the above attack, we first present an upper bound on the probability that $\mathcal{A}$ aborts.

Claim 4.1.6. $\Pr[\mathcal{A} \textit{ aborts }] \leq 2^{-z}$.

Proof. By Markov bound, the probability that in a single iteration of $\mathcal{A}$ the party P^* does not halt within $m = 2q$ rounds is at most $1/2$. Therefore, the probability that y^* is not set in all z iterations is at most 2^{-z}. □

Since P^* halts in the iteration that produced y^* within m rounds, its view in the emulated execution of R induced by the attack is the *same* as in this selected iteration (this holds even though some parties might run for more rounds in the emulated execution). In particular, P^* outputs y^* also in the emulated execution. The proof continues as in the proof of Lemma 4.1.3, where the only additional subtlety is that it is no longer true that the random coins of the parties in the emulated execution induced by the attack are uniformly distributed. Indeed, we have selected a value for $\boldsymbol{w}$ that causes P^* to halt within m rounds. Yet, since in a random execution of R, party P^* halts within m rounds with probability at least $1/2$, the method used to sample $\boldsymbol{w}$ at most doubles the probability of inconsistency in the ring. It follows that the attacked parties $\{\mathsf{A}, \mathsf{B}\}$ output y^* with probability at least $1 - 2 \cdot 3q\delta$ times the probability that $\mathcal{A}$ does not abort, and the proof of the lemma follows. □

4.2 Impossibility Results for Secure Computation

In this section, we present implications of the attack presented in Section 4.1 to secure multiparty computations in the secure-channels point-to-point model (note that a lower bound in the secure-channels model is stronger than in the authenticated-channels model). In Section 4.2.1, we show that the only public-output functionalities that can be securely realized, according to the real/ideal paradigm, in the presence of $n/3 \leq t < n/2$ corrupted parties (i.e., honest majority), are $(n - 2t)$-dominated functionalities. The only public-output functionalities that can be securely realized in the presence of $n/2 \leq t < n$ corrupted parties (i.e., no honest majority), are 1-dominated functionalities. In Section 4.2.2, we show that for $n \geq 3$, non-trivial n-party coin-flipping protocols, in which the honest parties must output a bit, are impossible when facing $t \geq n/3$ corrupted parties.

For concreteness, the focus in this section is on strict polynomial-time protocols secure against strict polynomial-time adversaries, but all the results readily extend to the expected polynomial-time regime.

4.2.1 Public-Output Functionalities

4.2.1.1 Dominated Functionalities

A special class of public-output functionalities are those with the property that every subset of a certain size can fully determine the output. For example, the multiparty Boolean AND and OR functionalities both have the property that every individual party can determine the output (for the AND functionality any party can always force the output to be 0, and for the

OR functionality any party can always force the output to be 1). We distinguish between the case where there exists a single value for which every large enough subset can force the output and the case where different subsets can force the output to be different values. Looking ahead, we will focus on the first variant where a unique output value can be forced by every subset. In Section 4.2.1.2, we will show that every function that can be computed without broadcast is in fact k-dominated (where the value of k depends on the maximal number of tolerable corruptions), and in Section 4.3, we will show that dominated functions can be computed without broadcast, where the unique value (that can be forced by every large enough subset) is used as a default output value by the honest parties in case they identify some misbehaviour in the protocol.

Definition 4.2.1 (dominated functionalities)**.** *A public-output n-party functionality f is* weakly k-dominated*, if for every k-size subset $\mathcal{I} \subseteq [n]$ there exists a value $y^*_{\mathcal{I}}$, for which there exist inputs $\{x_i\}_{i\in\mathcal{I}}$, such that $f(x_1,\ldots,x_n) = y^*_{\mathcal{I}}$ for* any *complementing subset of inputs $\{x_j\}_{j\notin\mathcal{I}}$. The functionality f is* k-dominated*, if there exists a value y^* such that for every k-size subset $\mathcal{I} \subseteq [n]$ there exist inputs $\{x_i\}_{i\in\mathcal{I}}$, for which $f(x_1,\ldots,x_n) = y^*$ for* any *subset of inputs $\{x_j\}_{j\notin\mathcal{I}}$.*

Example 4.2.2. *The function $f(x_1,x_2,x_3,x_4) = (x_1 \wedge x_2) \vee (x_3 \wedge x_4)$ is an example of a 4-party function that is weakly 2-dominated but not 2-dominated. Every pair of input variables can be set to determine the output value. However, there is no single output value that can be determined by all pairs, for example, $\{x_1,x_2\}$ can force the output to be 1 (by setting $x_1 = x_2 = 1$) whereas $\{x_1,x_3\}$ can force the output to be 0 (by setting $x_1 = x_3 = 0$). The function*

$$f_{2\text{-of-}4}(x_1,x_2,x_3,x_4) = (x_1 \wedge x_2) \vee (x_1 \wedge x_3) \vee (x_1 \wedge x_4) \vee (x_2 \wedge x_3) \vee (x_2 \wedge x_4) \vee (x_3 \wedge x_4)$$

is 2-dominated with value $y^* = 1$.

Note that in fact, 1-dominated functionalities are default-output functionalities with public output (cf. Definition 3.2.1).

Claim 4.2.3. *Let f be an n-party functionality and let $m \leq n/3$. If f is weakly m-dominated, then it is m-dominated.*

Proof. Let $\mathcal{I}_1,\mathcal{I}_2 \subseteq [n]$ be two subsets of size m. Denote by $\{x_i\}_{i\in\mathcal{I}_1}$ (resp., $\{x_i\}_{i\in\mathcal{I}_2}$) the input values for $\mathcal{I}_1$ (resp., $\mathcal{I}_2$) that force the output to be $y^*_{\mathcal{I}_1}$ (resp., $y^*_{\mathcal{I}_2}$). In case $\mathcal{I}_1$ and $\mathcal{I}_2$ are disjoint, fix an arbitrary complementing subset of inputs $\{x_j\}_{j\notin\mathcal{I}_1\cup\mathcal{I}_2}$. On the one hand it holds that $f(x_1,\ldots,x_n) = y^*_{\mathcal{I}_1}$ and on the other hand it holds that $f(x_1,\ldots,x_n) = y^*_{\mathcal{I}_2}$, hence $y^*_{\mathcal{I}_1} = y^*_{\mathcal{I}_2}$.

In case $\mathcal{I}_1$ and $\mathcal{I}_2$ are not disjoint, it holds that $|\mathcal{I}_1 \cup \mathcal{I}_2| < 2m \leq \frac{2n}{3}$ and since $m \leq n/3$, there exists a subset $\mathcal{I}_3 \subseteq [n] \setminus (\mathcal{I}_1 \cup \mathcal{I}_2)$ of size m. Denote by $y^*_{\mathcal{I}_3}$ the output value that can be determined by the input variables $\{x_i\}_{i\in\mathcal{I}_3}$ ($y^*_{\mathcal{I}_3}$ is guaranteed to exist since f is weakly m-dominated). $\mathcal{I}_3$ is disjoint from $\mathcal{I}_1$ and from $\mathcal{I}_2$, so it follows from the argument above that $y^*_{\mathcal{I}_1} = y^*_{\mathcal{I}_3}$ and $y^*_{\mathcal{I}_2} = y^*_{\mathcal{I}_3}$, therefore $y^*_{\mathcal{I}_1} = y^*_{\mathcal{I}_2}$. □

4.2.1.2 The Lower Bound

Lemma 4.2.4 (restating Corollary 1.2.7)**.** *Let $n \geq 3$, let $t \geq n/3$, and let f be a public-output n-party functionality that can be t-securely computed in the secure-channels point-to-point model with computational security.*

1. *If $n/3 \leq t < n/2$, then f is $(n-2t)$-dominated.*

2. *If $n/2 \leq t < n$, then f is 1-dominated.*

Proof. Assume that $n/3 \leq t < n/2$ (the proof for $n/2 \leq t < n$ is similar). Let π be a protocol that t-securely computes f in the point-to-point model with secure channels. Since f is public-output, all honest parties output the same value (except for a negligible probability), hence π is $(1-\text{neg}, t)$-consistent; let $\mathcal{A}$ be the PPT adversary guaranteed from Lemma 4.1.2 and let $\mathcal{I} \subseteq [n]$ be any subset of size $n-2t$. It follows that given control over $\{\mathsf{P}_i\}_{i\in\mathcal{I}}$, the adversary $\mathcal{A}$ can first fix a value $y^*_{\mathcal{I}}$, and later force the output of the honest parties to be $y^*_{\mathcal{I}}$ (except for a negligible probability). Since π is a protocol that t-securely computes f and $n-2t \leq t$, there exists an ideal-model adversary $\mathcal{S}$ that upon corrupting $\{\mathsf{P}_i\}_{i\in\mathcal{I}}$ can force the output of the honest parties in the ideal-model computation to be $y^*_{\mathcal{I}}$. All $\mathcal{S}$ can do is to select the input values of the corrupted parties, hence, there must exist input values $\{x_i\}_{i\in\mathcal{I}}$ that determine the output of the honest parties to be $y^*_{\mathcal{I}}$, i.e., f is weakly $(n-2t)$-dominated. Since $n-2t \leq n/3$ and following Claim 4.2.3 we conclude that f is $(n-2t)$-dominated. □

4.2.2 Coin-Flipping Protocols

A coin-flipping protocol [15] allows the honest parties to jointly flip an unbiased coin, where even a coalition of (efficient) cheating parties cannot bias the outcome of the protocol by much. Our focus is on coin flipping, where the honest parties *must* output a bit. Although Lemma 4.2.4 immediately shows that coin flipping cannot be securely computed according to the real/ideal paradigm, we present a stronger impossibility result by considering weaker security requirements. For simplicity, we consider coin-flipping protocols with perfect consistency, however, our negative result readily extends to protocols where consistency is only guaranteed to hold with high probability.

Definition 4.2.5 (coin-flipping protocol)**.** *A polynomial-time n-party protocol π is a (γ, t)-*bias *coin-flipping protocol, if the following holds.*

1. *π is $(1,t)$-consistent against* PPT *adversaries.*

2. *When interacting on security parameter κ with a* PPT *adversary controlling at most t corrupted parties, the common output of the honest parties is $\gamma(\kappa)$-close to being a uniform bit. (In particular, the honest parties are allowed to output $\perp$, or values other than $\{0,1\}$, with probability at most γ.)*

The following is a straightforward application of Lemma 4.1.2.

Lemma 4.2.6 (restating Corollary 1.2.10)**.** *In the secure-channels point-to-point model, for $n \geq 3$ and $\gamma(\kappa) < 1/2 - 2^{-\kappa}$, there exists no n-party, $(\gamma, \lceil n/3 \rceil)$-bias coin-flipping protocol.*

Proof. Let π be a point-to-point n-party $(\gamma, \lceil n/3 \rceil)$-bias coin-flipping protocol. Let $\mathcal{A}$ be the PPT adversary that is guaranteed by Lemma 4.1.2 (since π is $(1, \lceil n/3 \rceil)$-consistent against PPT adversaries). Consider some fixed set of $\lceil n/3 \rceil$ corrupted parties of π and let $Y(\kappa)$ denote the random variable of $\mathcal{A}(\kappa)$'s output in the first step of the attack. Without loss of generality, for infinitely many values of κ it holds that $\Pr[Y(\kappa) = 0] \leq 1/2$. Consider the adversary $\mathcal{A}'$ that on security parameter κ, repeats the first step of $\mathcal{A}(\kappa)$ until the resulting value of y^* is non-zero or

κ failed attempts have been reached, where if the latter happens $\mathcal{A}'$ aborts. Next, $\mathcal{A}'$ continues the non-zero execution of $\mathcal{A}$ to make the honest parties of π output y^*. It is immediate that for infinitely many values of κ, the common output of the honest parties under the above attack is 0 with probability at most $2^{-\kappa}$, and hence the common output of the honest parties is $1/2 - 2^{-\kappa}$ far from uniform. Thus, π is not a $(\gamma, \lceil n/3 \rceil)$-bias coin-flipping protocol. □

4.3 Characterizing Secure Computation without Broadcast

In this section, we show that the lower bounds presented in Lemma 4.2.4 are tight. We treat separately the case where an honest majority is assumed and the case where no honest majority is assumed.

4.3.1 No Honest Majority

Theorem 3.2.2 states that, assuming the existence of one-way functions, any 1-dominated functionality that can be t-securely computed in the broadcast model with authenticated channels, can also be t-securely computed in the point-to-point model with authenticated channels.[26] Combining with Lemma 4.2.4, we establish the following result.

Theorem 4.3.1 (restating second part of Theorem 1.2.9)**.** *Let $n \geq 3$, let $n/2 \leq t < n$ and assume that one-way functions exist. A public-output n-party functionality can be t-securely computed in the authenticated-channels point-to-point model, if and only if it is 1-dominated and can be t-securely computed in the authenticated-channels broadcast model.*

Proof. Immediately by Lemma 4.2.4 and Theorem 3.2.2. □

4.3.2 Honest Majority

To prove the matching upper bound in the honest-majority setting (Proposition 4.3.5 below) we use the *two-threshold multiparty protocol* of Fitzi et al. [48, Thm. 6]. This protocol with parameters t_1, t_2 runs in the point-to-point model with secure channels, and whenever $t_1 \leq t_2$ and $t_1 + 2t_2 < n$, the following holds. Let $\mathcal{I}$ be the set of parties that the (computationally unbounded) adversary corrupts. If $|\mathcal{I}| \leq t_1$, then the protocol computes f with full security. If $t_1 < |\mathcal{I}| \leq t_2$, then the protocol securely computes f with fairness (i.e., the adversary may force *all* honest parties to output $\perp$, provided that it learns no new information). In Section 4.3.2.1, we formally define the notion of two-threshold security that captures the security achieved by the protocol of Fitzi et al. [48]. In Section 4.3.2.2, we present the fully secure protocol for $(n - 2t)$-dominated functionalities.

Theorem 4.3.2 ([48, Thm. 6])**.** *Let $n \geq 3$, let t_1, t_2 be parameters such that $t_1 \leq t_2$ and $t_1 + 2t_2 < n$, and let f be an n-party functionality. Then, f can be (t_1, t_2)-securely computed in the secure-channels point-to-point model with information-theoretic security.*

[26] The result in Theorem 3.2.2 is based on the computationally secure protocol in [47, Thm. 2]. In the authenticated-channels point-to-point model, this protocol requires one-way functions for constructing a consistent public-key infrastructure between the parties, to be used for authenticated broadcast.

4.3.2.1 Defining Two-Threshold Security

We present a weaker variant of the ideal model that allows for a premature (and fair) abort, in case sufficiently many parties are corrupted. Next, we define two-threshold security of protocols.

Threshold ideal-model execution. A t-threshold ideal computation of an n-party functionality f on input $\boldsymbol{x} = (x_1, \ldots, x_n)$ for parties $(\mathsf{P}_1, \ldots, \mathsf{P}_n)$, in the presence of an ideal-model adversary $\mathcal{A}$ controlling the parties indexed by $\mathcal{I} \subseteq [n]$, proceeds via the following steps.

Sending inputs to trusted party: An honest party P_i sends its input x_i to the trusted party. The adversary may send to the trusted party arbitrary inputs for the corrupted parties. If $|\mathcal{I}| > t$, then the adversary may send a special abort command to the trusted party. Let x'_i be the value actually sent as the input of party P_i.

Trusted party answers the parties: In case the adversary sends the special abort command (specifically, $|\mathcal{I}| > t$), then the trusted party sends $\perp$ to all the parties. Otherwise, if x'_i is outside of the domain for P_i, for some index i, or if no input is sent for P_i, then the trusted party sets x'_i to be some predetermined default value. Next, the trusted party computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ and sends y_i to party P_i for every i.

Outputs: Honest parties always output the message received from the trusted party and the corrupted parties output nothing. The adversary $\mathcal{A}$ outputs an arbitrary function of the initial inputs $\{x_i\}_{i\in\mathcal{I}}$, the messages received by the corrupted parties from the trusted party $\{y_i\}_{i\in\mathcal{I}}$ and its auxiliary input.

Definition 4.3.3 (Threshold ideal-model computation)**.** *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality and let $\mathcal{I} \subseteq [n]$. The* joint execution of f under $(\mathcal{A}, I)$ in the t-threshold ideal model*, on input vector $\boldsymbol{x} = (x_1, \ldots, x_n)$, auxiliary input z to $\mathcal{A}$ and security parameter κ, denoted $\text{IDEAL}^t_{f,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)$, is defined as the output vector of $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and $\mathcal{A}(z)$ resulting from the above-described ideal process.*

Definition 4.3.4. *Let $f\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n$ be an n-party functionality, and let π be a probabilistic polynomial-time protocol computing f. The* protocol π (t_1, t_2)-securely computes f (with information-theoretic security)*, if for every real-model adversary $\mathcal{A}$, there exists an adversary $\mathcal{S}$ for the t_1-threshold ideal model, whose running time is polynomial in the running time of $\mathcal{A}$, such that for every $\mathcal{I} \subseteq [n]$ of size at most t_2*

$$\left\{\text{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \overset{\text{s}}{\equiv} \left\{\text{IDEAL}^{t_1}_{f,\mathcal{I},\mathcal{S}(z)}(\boldsymbol{x}, \kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

4.3.2.2 Full Security with an Honest Majority

Proposition 4.3.5. *Let $n \geq 3$, let $n/3 \leq t < n/2$, and let f be a public-output n-party functionality. If f is $(n-2t)$-dominated, then it can be t-securely computed in the secure-channels point-to-point model with information-theoretic security.*

Proof. Let f be an $(n-2t)$-dominated functionality with default-output value y^*. If $n-2t = 1$, then f is 1-dominated, and since $t < n/2$, f can be t-securely computed with information-

theoretic security in the secure-channels broadcast model (e.g., using Rabin and Ben-Or [81]). Hence, the proposition follows from Theorem 3.2.2.[27]

For $n - 2t \geq 2$, set $t_1 = n - 2t - 1$ and $t_2 = t$, and let π' be the n-party protocol, guaranteed to exist by Theorem 4.3.2, that (t_1, t_2)-securely computes f. We define π to be the following n-party protocol for computing f in the point-to-point model with secure channels (Protocol 4.3.6).

Protocol 4.3.6 (full security for $(n-2t)$-dominated functionalities)**.**

1. *The parties run the protocol π'. Let y_i be the output of P_i at the end of the execution.*
2. *If $y_i \neq \perp$, party P_i outputs y_i, otherwise it outputs y^*.*

Let $\mathcal{A}$ be an adversary attacking the execution of π and let $\mathcal{I} \subseteq [n]$ be a subset of size at most t. It follows from Theorem 4.3.2 that there exists a (possibly aborting) adversary $\mathcal{S}'$ for $\mathcal{A}$ in the t_1-threshold ideal model such that

$$\left\{\mathrm{REAL}_{\pi',\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\equiv} \left\{\mathrm{IDEAL}^{t_1}_{f,\mathcal{I},\mathcal{S}'(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

Using $\mathcal{S}'$, we construct the following non-aborting adversary $\mathcal{S}$ for the full-security ideal model. On inputs $\{x_i\}_{i\in\mathcal{I}}$ and auxiliary input z, $\mathcal{S}$ starts by emulating $\mathcal{S}'$ on these inputs, playing the role of the trusted party (in the t_1-threshold ideal model). If $\mathcal{S}'$ sends an abort command, it is guaranteed that $|\mathcal{I}| \geq n-2t$ and since f is $(n-2t)$-dominated, there exist input values $\{x'_i\}_{i\in\mathcal{I}}$ that determine the output of f to be y^*. So in this case, $\mathcal{S}$ sends these $\{x'_i\}_{i\in\mathcal{I}}$ to the trusted party (in the full-security ideal model) and returns $\perp$ to $\mathcal{S}'$. Otherwise, $\mathcal{S}'$ does not abort and $\mathcal{S}$ forwards the message from $\mathcal{S}'$ to the trusted party and the answer from the trusted party back to $\mathcal{S}'$. In both cases $\mathcal{S}$ outputs whatever $\mathcal{S}'$ outputs and halts.

A main observation is that the views of the adversary $\mathcal{A}$ in an execution of π and in an execution of π' (with the same inputs and random coins) are identical. This holds since the only difference between π and π' is in the second step of π that does not involve any interaction. It follows that in case the output of the parties in Step 1 of π is not $\perp$, the joint distribution of the honest parties' output and the output of $\mathcal{A}$ in π is statistically close to the output of the honest parties and of $\mathcal{S}$ in the full-security ideal model (since the latter is exactly the output of the honest parties and of $\mathcal{S}'$ in the t_1-threshold ideal model). If the output in Step 1 of π is $\perp$, then all honest parties in π output y^*. In this case $\mathcal{S}'$ sends abort (except for a negligible probability) and since $\mathcal{S}$ sends to the trusted party the input values $\{x'_i\}_{i\in\mathcal{I}}$ that determine the output of f to be y^*, the honest parties' output is y^* also in the ideal computation. We conclude that

$$\left\{\mathrm{REAL}_{\pi,\mathcal{I},\mathcal{A}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}} \overset{\mathrm{s}}{\equiv} \left\{\mathrm{IDEAL}^{\mathsf{full}}_{f,\mathcal{I},\mathcal{S}(z)}(\boldsymbol{x},\kappa)\right\}_{(\boldsymbol{x},z)\in(\{0,1\}^*)^{n+1},\kappa\in\mathbb{N}}.$$

□

[27] When an honest majority is assumed, the result in Theorem 3.2.2 can be adjusted to use the information-theoretically secure protocol of Fitzi et al. [47, Thm. 3]. In the secure-channels point-to-point model, this protocol uses information-theoretically pseudo-signatures [79] for computing a setup, to be used for authenticated broadcast.

Theorem 4.3.7 (restating the first part of Theorem 1.2.9)**.** *Let $n \geq 3$ and $n/3 \leq t < n/2$. A public-output n-party functionality can be t-securely computed in the secure-channels point-to-point model, if and only if it is $(n-2t)$-dominated.*

Proof. Immediately follows by Lemma 4.2.4 and Proposition 4.3.5. □

Part II

Round-Efficient MPC with Full Security

Chapter 5

Preliminaries

5.1 The UC Framework

In this section, we describe the universal composition framework, for more details see [20].

5.1.1 The Real Model

An execution of a protocol π in the real model consists of n PPT *interactive Turing machines* (ITMs) $\mathsf{P}_1, \ldots, \mathsf{P}_n$ representing the parties, along with two additional ITMs, an *adversary* $\mathcal{A}$, describing the behavior of the corrupted parties and an *environment* $\mathcal{Z}$, representing the external network environment in which the protocol operates. The environment gives inputs to the honest parties, receives their outputs, and can communicate with the adversary at any point during the execution. The adversary controls the operations of the corrupted parties and the delivery of messages between the parties.

In more details, each ITM is initialized with the security parameter κ and random coins, where the environment may receive an additional auxiliary input. The protocol proceeds by a sequence of *activations*, where the environment is activated first and at each point a single ITM is active. When the environment is activated it can read the output tapes of all honest parties and of the adversary, and it can activate one of the parties or the adversary by writing on its input tape. Once a party is activated it can perform a local computation, write on its output tape or send messages to other parties by writing on its outgoing communication tapes. After the party completes its operations the control is returned to the environment. Once the adversary is activated it can send messages on behalf of the corrupted parties or send a message to the environment by writing on its output tape. In addition, $\mathcal{A}$ controls the communication between the parties, and so it can read the contents of the messages on outgoing tapes of honest parties and write messages on their incoming tapes. We assume that only messages that were sent in the past by some party can be delivered, and each message can be delivered at most once. The adversary $\mathcal{A}$ is *adaptive*, and so it can also corrupt an honest party, gain access to all its tapes and control all its actions. Whenever a party is corrupted the environment is notified. If $\mathcal{A}$ wrote on the incoming tape of an honest party, this party is activated next, otherwise the environment is activated. The protocol completes once $\mathcal{Z}$ stops activating other parties and outputs a single bit. We consider *malicious* adversaries, that may instruct the corrupted parties

to deviate from the protocol arbitrarily.

Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting with adversary $\mathcal{A}$ and parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ running protocol π with random tapes $\boldsymbol{r} = (r_1, \ldots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$ as described above. Let $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote the random variable $\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$, when the vector $\boldsymbol{r}$ is uniformly chosen.

5.1.2 The Ideal Model

A computation in the ideal model consists of n *dummy* parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, an *ideal-process adversary* (simulator) $\mathcal{S}$, an *environment* $\mathcal{Z}$, and an *ideal functionality* $\mathcal{F}$. As in the real model, the environment gives inputs to the honest (dummy) parties, receives their outputs, and can communicate with the ideal-process adversary at any point during the execution. The dummy parties act as channels between the environment and the ideal functionality, meaning that they send the inputs received from $\mathcal{Z}$ to $\mathcal{F}$ and vice-versa. The ideal functionality $\mathcal{F}$ defines the desired behaviour of the computation. $\mathcal{F}$ receives the inputs from the dummy parties, executes the desired computation and sends the output to the parties. The ideal-process adversary does not see the communication between the parties and the ideal functionality, however, $\mathcal{S}$ can communicate with $\mathcal{F}$.

Hiding the communication between the ideal functionality and the parties from the adversary may be too restrictive, and indeed, in the standard UC framework the adversary is often given the power to determine *when* a party will receive the message. We say that the ideal functionality $\mathcal{F}$ sends a *delayed output* v to a party P if $\mathcal{F}$ first sends to the adversary a message that it is ready to generate an output to P. In case the output is public $\mathcal{F}$ sends v to the adversary. When the adversary replies to the message, $\mathcal{F}$ outputs the value v to P.[28]

Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting with ideal-process adversary $\mathcal{S}$ and dummy parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ that interact with ideal functionality $\mathcal{F}$ using random tapes $\boldsymbol{r} = (r_{\mathcal{S}}, r_{\mathcal{Z}})$ as described above. Let $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)$ denote the random variable $\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z, \boldsymbol{r})$, when the vector $\boldsymbol{r}$ is uniformly chosen.

Definition 5.1.1. *We say that a protocol π UC-realizes an ideal functionality $\mathcal{F}$ with perfect security in the presence of adaptive malicious adversaries, if for any adaptive malicious adversary $\mathcal{A}$ and any environment $\mathcal{Z}$, there exists an ideal-process adversary $\mathcal{S}$, whose running time is polynomial in the running time of $\mathcal{A}$, such that the following two distribution ensembles are computationally indistinguishable*

$$\left\{\text{REAL}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \equiv \left\{\text{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

Canetti [20, Claim 10] provided a simplification of the above definition, and showed that instead of quantifying over all adversaries, it is sufficient to prove security facing the *dummy adversary*, which acts as a channel between the environment and the protocol. Namely, the

[28] The ideal-process adversary may never release messages from the ideal functionality to the dummy parties and so termination of the computation is not guaranteed. In order to rule out trivial protocols that never produce output, Canetti et al. [23] defined *non-trivial protocols* that have the following property: if the real-model adversary delivers all messages and does not corrupt any parties, then the ideal-process adversary also delivers all messages and does not corrupt any parties. We note that using techniques from [68] guaranteed termination can be enforced.

dummy adversary forwards any information it gathers directly to the environment, and forwards any instruction it receives from the environment to the corresponding corrupted party.

Claim 5.1.2. *A protocol π UC-realizes a functionality $\mathcal{F}$ according to Definition 5.1.1 if and only if π UC-realizes $\mathcal{F}$ with respect to the dummy adversary.*

5.1.3 The Hybrid Model

The *$\mathcal{F}$-hybrid model* is a combination of the real and ideal models, it extends the real model with an ideal functionality $\mathcal{F}$. The parties communicate with each other in exactly the same way as in the real model described above, however, they can interact with $\mathcal{F}$ as in the ideal model. An important property of the UC framework is that the ideal functionality $\mathcal{F}$ in a $\mathcal{F}$-hybrid model can be replaced with a protocol that UC-realizes $\mathcal{F}$.

Let the global output $\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{A},\mathcal{Z}}(\kappa, z)$ denote $\mathcal{Z}$'s output on input z and security parameter κ, after interacting in a $\mathcal{F}$-hybrid model with adversary $\mathcal{A}$ and parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ with uniformly distributed random tapes $\boldsymbol{r} = (r_1, \ldots, r_n, r_{\mathcal{A}}, r_{\mathcal{Z}})$ running protocol π.

Theorem 5.1.3 (Canetti [20])**.** *Let $\mathcal{F}$ be an ideal functionality and let ρ be a protocol that UC-realizes $\mathcal{F}$ in the presence of adaptive malicious adversaries, and let π be a protocol that UC-realizes $\mathcal{G}$ in the $\mathcal{F}$-hybrid model in the presence of adaptive malicious adversaries. Then for any adaptive malicious real-model adversary $\mathcal{A}$ and any environment $\mathcal{Z}$, there exists an adaptive malicious adversary $\mathcal{S}$ in the $\mathcal{F}$-hybrid model such that*

$$\left\{\text{REAL}_{\pi^\rho,\mathcal{A},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*} \equiv \left\{\text{HYBRID}^{\mathcal{F}}_{\pi,\mathcal{S},\mathcal{Z}}(\kappa, z)\right\}_{\kappa\in\mathbb{N},z\in\{0,1\}^*}.$$

5.2 Synchronous Communication in the UC Framework

The UC framework as described in Section 5.1 is inherently asynchronous, since the adversary has full control over the delivery of messages between honest parties. It follows that full security cannot be defined in the UC framework, since guaranteed termination can never be enforced. Several definition of synchronous communication over the UC framework have been proposed, e.g., [20, 21, 71, 68]. We follow that model of Katz et al. [68], which ensures guaranteed termination, and can be used to define full security over the UC framework. However, we note that the remainder of Part II also applies to the synchronous setting without guaranteed termination.

Concretely, Katz et al. [68] introduced a framework for universally composable synchronous computation. For self-containment we describe here the basics of the model and introduce some terminology that simplifies the description of corresponding functionalities.

Synchronous protocols can be cast as UC protocols which have access to a special clock functionality $\mathcal{F}_{\text{CLOCK}}$, which allows them to coordinate round switches as described below, and communicate over bounded-delay channels.[29] In a nutshell, the clock-functionality works as follows: It stores a bit b which is initially set to 0 and it accepts from each party two types

[29] As argued in [68], bounded-delay channels are essential as they allow parties to detect whether or not a message was sent within a round.

of messages: CLOCK-UPDATE and CLOCK-READ. The response to CLOCK-READ is the value of the bit b to the requestor. Each CLOCK-UPDATE is forwarded to the adversary, but it is also recorded, and upon receiving such a CLOCK-UPDATE message from all honest parties, the clock functionality updates b to $b \oplus 1$. It then keeps working as above, until it receives again a CLOCK-UPDATE message from all honest parties, in which case it resets b to $b \oplus 1$ and so on.

Such a clock can be used as follows to ensure that honest parties remain synchronized, i.e., no honest party proceeds to the next round before all (honest) parties have finished the current round: Every party stores a local variable where it keeps (its view of) the current value of the clock indicator b. At the beginning of the protocol execution this variable is 0 for all parties. In every round, every party uses all its activations (i.e., messages it receives) to complete all its current-round instructions and only then sends CLOCK-UPDATE to the clock signaling to the clock that it has completed its round; following CLOCK-UPDATE, all future activations result in the party sending CLOCK-READ to the clock until its bit b is flipped; once the party observes that the bit b has flipped, it starts its next round. For clarity, we refrain from writing this clock functionality in our theorem statement; however, all our results assume access to such a clock functionality.

In the communication network of [68], parties have access to bounded-delay secure channels. These channels work in a so-called "fetch" mode, i.e., in order to receive his output the receiver issues a `fetch-output` command. This allows to capture the property of a channel between a sender P_s and a receiver P_r, delaying the delivery of a message by an amount δ: as soon as the sender P_s submits an input y (message to be sent to the receiver) the channel functionality starts counting how many times the receiver requests it.[30] The first $\delta - 1$ such `fetch-output` requests (plus all such requests that are sent before the sender submits input) are ignored (and the adversary is notified about them); the δ'th `fetch-output` request following a submitted input y from the sender results in the channel sending $(\texttt{output}, y)$ to P_r. In this work we take an alternative approach and model secure channels as special simple SFE-like functionalities[31] that take as input from the sender the message he wishes to send (and a default input from other parties) and deliver the output to the receiver in a fetch mode.[32] Such a simple secure-channel SFE can be realized in a straightforward manner from bounded-delay channels and a clock $\mathcal{F}_{\text{CLOCK}}$.

As is common in the synchronous protocols literature, throughout this work we will assume that protocols have the following structure: In each round every party sends/receives a (potentially empty) message to all parties and hybrid functionalities. Such a protocol can be described in UC in a regular form using the methodology from [68] as follows: Let $\mu \in \mathbb{N}$ denote the maximum number of messages that any party P_i might send to all its hybrids during some round.[33] Every party in the protocol uses exactly μ activations in each round. That is, once a party P_i observes that the round has changed, i.e., the indicator-bit b of the clock has being flipped, P_i starts its next round as described above. However, this round finishes only after

[30] Following the simplifying approach of [68], we assume that communication channels are single use, thus, each message transmission uses an independent instance of the channel.

[31] In Chapter 6 we introduce a more liberal variant of the UC SFE functionality that we call *canonical synchronous functionality* (in short, CSF) that allows us to abstract several (even more complicated) tasks such as Byzantine agreement.

[32] In fact, for simplicity we assume that they deliver output on the first "fetch".

[33] In the simple case where the parties only use point-to-point channels, $\mu = 2(n-1)$, since each party uses $n-1$ channels as sender and $n-1$ as receiver to exchange his messages for each round with all other n parties.

P_i receives μ additional activations. Note that P_i uses these activations to execute his current round instructions; since μ is a bound on the number of hybrids used in any round by any party, μ activations are enough for the party to complete its round, (If P_i finishes the round early, i.e., in less than μ activations, it simply "does nothing" until the μ activations are received, i.e., forward the activation from the environment to its hybrid functionalities.) Once μ activations are received in the current round, P_i sends CLOCK-UPDATE to the clock and then keeps sending CLOCK-READ message, as described above, until it observes a flip of b indicating that P_i can go to the next round.

In addition to the regular form of protocol execution, Katz et al. [68] described a way of capturing in UC the property that a protocol is guaranteed to terminate in a given number of rounds.[34] The idea is that a synchronous protocol in regular form which terminates after r rounds realizes the following functionality $\mathcal{F}$. The functionality $\mathcal{F}$ keeps track of the number of times every honest party sends μ activations/messages and delivers output as soon as this has happened r times. More concretely, imitating an r-round synchronous protocol with μ activations per party per round, upon being instantiated, $\mathcal{F}$ initiates a global round-counter $\lambda = 0$ and an indicator variable $\lambda_i := 0$ for each $\mathsf{P}_i \in \mathcal{P}$; as soon as some party P_i sends μ messages to $\mathcal{F}$, while the round-counter λ is the same, $\mathcal{F}$ sets $\lambda_i := 1$ and does the following check:[35] if $\lambda_i = 1$ for all honest P_i then increase $\lambda := \lambda + 1$ and reset $\lambda_i = 0$ for all $\mathsf{P}_i \in \mathcal{P}$. As soon as $\lambda = r$, $\mathcal{F}$ enters a "delivery" mode. In this mode, whenever a message `fetch-output` is received by some party P_i, the functionality $\mathcal{F}$ outputs to P_i its output. (If $\mathcal{F}$ has no output to P_i is outputs $\perp$.)

We refer to a functionality that has the above structure, i.e., which keeps track of the current round λ by counting how many times every honest party has sent a certain number μ of messages, as a *synchronous functionality*. To simplify the description of our functionalities, we introduce the following terminology. We say that *a synchronous functionality $\mathcal{F}$ is in round ρ* if the current value of the above internal counter in $\mathcal{F}$ is $\lambda = \rho$. All synchronous functionalities considered in this work have the following format: They can receive any message as input from the parties, however, they ignore all messages until the first message of the special form $(\texttt{input}, \cdot)$; as soon as an honest party sends its input message, any future message by this party is treated as a $(\texttt{fetch-output}, \cdot)$ message. Without loss of generality, whenever clear from the context we will describe functionalities for $\mu = 1$, i.e., once a functionality receives a message from every party it proceeds to the simulation of the next protocol round. We stress that this is done to simplify the description, and in an actual evaluation, as in the synchronous setting of [68], in order to give the simulator sufficiently many activations to perform its simulation, functionalities typically have to wait for $\mu > 1$ messages from each party where the last $\mu - 1$ of these messages are typically "dummy" activations (usually of the type $(\texttt{fetch-output}, \cdot)$).

We note that protocols in the synchronous model of [68] enjoy the strong composition properties of the UC framework. However, in order to have protocols being executed in a lock-step mode, i.e., where all protocols complete their round within the same clock-tick, Katz et al. [68] make use of the composition with joint-state (JUC) [22]. The idea is the parties use an $\mathcal{F}_{\text{CLOCK}}$-hybrid protocol $\hat{\pi}$ that emulates towards each of the protocols, subclocks and assigns

[34] The wrappers presented in this work generalize the notion of guaranteed termination to capture randomized number of rounds. Concretely, one can view the functionality for SFE with guarantee termination from [68] as a wrapped version of the standard SFE functionality with our wrapper with a deterministic round distribution.

[35] To make sure that the simulator can keep track of the round index, $\mathcal{F}$ notifies $\mathcal{S}$ about each received input, unless it has reached its delivery state defined below.

to each subclock a unique sub-session ID (ssid). Each of these subclocks is local to its calling protocol, but $\hat{\pi}$ makes sure that it gives a CLOCK-UPDATE to the actual (joint) clock functionality $\mathcal{F}_{\text{CLOCK}}$, only when all subclocks have received such a CLOCK-UPDATE message. This ensures that all clocks will switch their internal bits at the same time with the bigger clock, which means that the protocols using them will be mutually synchronized. This property can be formally proved by direct application of the JUC theorem. For further details the interested reader is referred to [68, 22].

A final delicate point that needs to be addressed is with respect to whether/when parties can stop sending messages to their functionalities. Concretely, since the parties do not a priori know the termination round, a party cannot halt as soon as it produces output, since some other party might still need several rounds to produce its own output. Rather, every protocol in our setting continues sending (dummy) messages to all its hybrids even after having generated its output—as long as it keeps being activated by the environment—to enable also the other (slower) parties compute their outputs. Note that this is a straightforward adaptation of classical UC protocols—instead of the protocol ignoring messages after it has produced output, it enters a special dummy stage. Most importantly, it does not yield inefficient protocols, as the input of a UC protocol consists of the concatenation of its inputs, and, therefore, as long as the environment provides new inputs/activations the protocol can continue executing dummy steps. (The UC experiment finishes when the environment decides to stop and make its guess.)

5.3 On Parallel (In)Composability of Protocols with Probabilistic Termination

Ben-Or and El-Yaniv [11] observed that when executing randomized protocols with probabilistic termination in parallel, then, in general, the expected running time of the composed protocol (i.e., the rounds its takes for all protocols to give output to all parties) is not preserved. We prove a formal example where this is the case. Concretely, consider a protocol realizing a particular ideal functionality such that the probability that all parties have completed the protocol by round k is p^k for some $0 < p < 1$. Then, the expected running time of the protocol is $1/p$ rounds, i.e., constant. (This is essentially the case in most randomized BA protocols starting with Feldman and Micali [42].) However, as implied by the following lemma, if m instances of the protocol are run in parallel, in a straightforward manner, the resulting protocol will have an expected running time of $\Theta(\log m)$, which is no longer constant.

In particular, running m parallel copies of the protocol of Feldman and Micali [42] results in a protocol that in expectation takes $\Theta(\log m)$ phases (and thus rounds) to complete.

Lemma 5.3.1. *Let $X_1, \ldots, X_m$ be independent, identically distributed (IID) geometric random variables, such that for every $i \in [m]$ it holds that $\Pr[X_i = 1] = p$ for some $0 < p < 1$. Then,*

$$E\left[\max_{1 \le i \le m} X_i\right] = \Theta(\log m).$$

Proof. As shown in Eisenberg [41], the expected value of the maximum of the random variables

is

$$\frac{1}{-\log(1-p)}\sum_{k=1}^{m}\frac{1}{k} \leq E\left[\max_{1\leq i\leq m} X_i\right] \leq 1 + \frac{1}{-\log(1-p)}\sum_{k=1}^{m}\frac{1}{k}.$$

The lemma follows immediately from the properties of the Harmonic numbers $H_m = \sum_{k=1}^{m}\frac{1}{k}$. In particular, denote $u_m = H_m - \log m$, then the series $(u_m)_m$ converges to Euler's constant γ, implying that

$$H_m = \Theta(\log m).$$

□

Chapter 6

Secure Computation with Probabilistic Termination

The work of Katz et al. [68] addresses (synchronous) cryptographic protocols that terminate in a fixed number of rounds for all honest parties. However, as mentioned in Chapter 1, Ben-Or [10] and Rabin [80] showed that in some cases, great asymptotic improvements on the *expected* termination of protocols can be achieved through the use of randomization. Recall, for example, that in the case of BA, even though a lower bound of $t + 1$ rounds of any deterministic BA protocol tolerating t corruptions exists [43, 39], Rabin's global-coin technique (fully realized later on in [42]) yields an expected constant round protocol. This speedup, however, comes at a price, namely, of relinquishing both *fixed* and *simultaneous* termination [40]: the round complexity of the corresponding protocols may depend on random choices made during the execution, and parties may obtain output from the protocol in different rounds.

In this section, we show how to capture protocols with such *probabilistic termination (PT)*, i.e., without fixed and without simultaneous termination, within the UC framework. To capture probabilistic termination, we first introduce a functionality template $\mathcal{F}_{\text{CSF}}$ called a *canonical synchronous functionality (CSF)*. $\mathcal{F}_{\text{CSF}}$ is a simple two-round functionality with explicit (one-round) input and (one-round) output phases. Computation with probabilistic termination is then defined by wrapping $\mathcal{F}_{\text{CSF}}$ with an appropriate functionality *wrapper* that enables non-fixed, non-simultaneous termination.

6.1 Canonical Synchronous Functionalities

At a high level, $\mathcal{F}_{\text{CSF}}$ corresponds to a generalization of the UC-secure function evaluation (SFE) functionality to allow for potential leakage on the inputs to the adversary and potential adversarial influence on the outputs.[36] In more detail, $\mathcal{F}_{\text{CSF}}$ has two parameters: (1) a (possibly) randomized function f that receives $n + 1$ inputs (n inputs from the parties and one additional input from the adversary) and (2) a leakage function l that leaks some information about the input values to the adversary.

[36] Looking ahead, this adversarial influence will allow us to describe BA-like functionalities as simple and intuitive CSFs.

$\mathcal{F}_{\text{CSF}}$ proceeds in two rounds: in the first round all the parties hand $\mathcal{F}_{\text{CSF}}$ their input values, and in the second round each party receives its output. This is very similar to the standard (UC) SFE functionality; the difference here is that whenever some input is submitted to $\mathcal{F}_{\text{CSF}}$, the adversary is handed some leakage function of this input—similarly, for example, to how UC-secure channels leak the message length to the adversary. The adversary can use this leakage when deciding the inputs of corrupted parties. Additionally, he is allowed to input an extra message, which—depending on the function f—might affect the output(s). The detailed description of $\mathcal{F}_{\text{CSF}}$ is given in Figure 6.1.

Functionality $\mathcal{F}_{\text{CSF}}^{f,l}(\mathcal{P})$

$\mathcal{F}_{\text{CSF}}$ proceeds as follows, parametrized by a function $f\colon (\{0,1\}^* \cup \{\bot\})^{n+1} \to (\{0,1\}^*)^n$ and a leakage function $l\colon (\{0,1\}^* \cup \{\bot\})^n \to \{0,1\}^*$, and running with parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ and an adversary $\mathcal{S}$.

- Initially, set the input values $x_1, \ldots, x_n$, the output values $y_1, \ldots, y_n$, and the adversary's value a to $\bot$.
- In round $\rho = 1$:
  - Upon receiving $(\texttt{adv-input}, \mathsf{sid}, v)$ from the adversary, set $a \leftarrow v$.
  - Upon receiving a message $(\texttt{input}, \mathsf{sid}, v)$ from some party $\mathsf{P}_i \in \mathcal{P}$, set $x_i \leftarrow v$ and send $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_i, l(x_1, \ldots, x_n))$ to the adversary.
- In round $\rho = 2$:
  - Upon receiving $(\texttt{adv-input}, \mathsf{sid}, v)$ from the adversary, if $y_1 = \ldots = y_n = \bot$, set $a \leftarrow v$. Otherwise, discard the message.
  - Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $\mathsf{P}_i \in \mathcal{P}$, if $y_1 = \ldots = y_n = \bot$ compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n, a)$. Next, send $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i and $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.

Figure 6.1: The canonical synchronous functionality

Next, we point out a few technical issues about the description of $\mathcal{F}_{\text{CSF}}$. Following the simplifications from Section 5.2, $\mathcal{F}_{\text{CSF}}$ advances its round as soon as it receives $\mu = 1$ message from each honest party. This ensures that the adversary cannot make the functionality stall indefinitely. Thus, formally speaking, the functionality $\mathcal{F}_{\text{CSF}}$ is not well-formed (cf. [23]), as its behavior depends on the identities of the corrupted parties.[37] We emphasize that the non-well-formedness relates only to advancing the rounds, and is unavoidable if we want to restrict the adversary not to block the evaluation indefinitely (cf. [68]).

Once an honest party sends its input, the adversary receives a leakage from the functionality, and based on this information can corrupt the party, replace its input value, and send an additional input message to the functionality. Note that the functionality will consider the latest input value received by a party in order to allow the adversary such a behavior.

We point out that as a generalization of the SFE functionality, CSFs are powerful enough to capture any deterministic well-formed functionality. In fact, all the basic (unwrapped) functionalities considered in this work will be CSFs. We now describe how standard functionalities from the MPC literature can be cast as CSFs:

[37] This is, in fact, also the case for the standard UC SFE functionality.

- Secure Message Transmission (aka Secure Channel). In the *secure message transmission* (SMT) functionality, a sender P_i with input x_i sends its input to P_j. Since $\mathcal{F}_{\text{CSF}}$ is an n-party functionality and involves receiving `input` messages from all n parties, we define the two-party task using an n-party function. The function to compute is $f^{i,j}_{\text{SMT}}(x_1,\ldots,x_n,a) = (\lambda,\ldots,x_i,\ldots,\lambda)$ (where x_i is the value of the j'th coordinate) and the leakage function is $l^{i,j}_{\text{SMT}}(x_1,\ldots,x_n) = y$, where $y = |x_i|$ in case P_j is honest and $y = x_i$ in case P_j is corrupted. We denote by $\mathcal{F}^{i,j}_{\text{SMT}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions $f^{i,j}_{\text{SMT}}$ and $l^{i,j}_{\text{SMT}}$, for sender P_i and receiver P_j.
- Broadcast. In the (standard) *broadcast* functionality, a sender P_i with input x_i distributes its input to all the parties, i.e., the function to compute is $f^{i}_{\text{BC}}(x_1,\ldots,x_n,a) = (x_i,\ldots,x_i)$. The adversary only learns the length of the message x_i before its distribution, i.e., the leakage function is $l^{i}_{\text{BC}}(x_1,\ldots,x_n) = |x_i|$. This means that the adversary does not gain new information about the input of an honest sender before the output value for all the parties is determined, and in particular, the adversary *cannot* corrupt an honest sender and change its input *after* learning the input message. We denote by $\mathcal{F}^{i}_{\text{BC}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f^{i}_{BC} and l^{i}_{BC}, for sender P_i.
- Secure Function Evaluation. In the *secure function evaluation* functionality, the set of parties compute a randomized function $g(x_1,\ldots,x_n)$, i.e., the function to compute is $f^{g}_{\text{SFE}}(x_1,\ldots,x_n,a) = g(x_1,\ldots,x_n)$. The adversary learns the length of the input values via the leakage function, i.e., the leakage function is $l_{\text{SFE}}(x_1,\ldots,x_n) = (|x_1|,\ldots,|x_n|)$. We denote by $\mathcal{F}^{g}_{\text{SFE}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f^{g}_{SFE} and l_{SFE}, for computing the n-party function g.
- Byzantine Agreement (aka Consensus). In the *Byzantine agreement* functionality, defined for the set V, each party P_i has input $x_i \in V$. The common output is computed such that if $n-t$ of the input values are the same, this will be the output; otherwise the adversary gets to decide on the output. The adversary is allowed to learn the content of each input value from the leakage (and so it can corrupt parties and change their inputs based on this information). The function to compute is $f_{\text{BA}}(x_1,\ldots,x_n,a) = (y,\ldots,y)$ such that $y = x$ if there exists a value x such that $x = x_i$ for at least $n-t$ input values x_i; otherwise $y = a$. The leakage function is $l_{\text{BA}}(x_1,\ldots,x_n) = (x_1,\ldots,x_n)$. We denote by $\mathcal{F}^{V}_{\text{BA}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f_{BA} and l_{BA}, defined for the set V.

6.2 Probabilistic Termination in UC

Having defined CSFs, we turn to the notion of (non-reactive) computation with probabilistic termination. This is achieved by defining the notion of an *output-round randomizing wrapper*. Such a wrapper is parametrized by an efficient probabilistic algorithm D, termed the *round sampler*, that may depend on a specific protocol implementing the functionality. The round sampler D samples a round number ρ_{term} by which all parties are guaranteed to receive their outputs no matter what the adversary strategy is. Moreover, since there are protocols in which all parties terminate in the same round and protocols in which they do not, we consider two wrappers: the first, denoted $\mathcal{W}_{\text{strict}}$, ensures in a strict manner that all (honest) parties terminate

in the same round, whereas the second, denoted $\mathcal{W}_{\text{flex}}$, is more flexible and allows the adversary to deliver outputs to individual parties at any time before round ρ_{term}.

A delicate issue that needs to be addressed is the following: While an ideal functionality can be used to abstractly describe a protocol's task, it cannot hide the protocol's round complexity. This phenomenon is inherent in the synchronous communication model: any environment can observe how many rounds the execution of a protocol takes, and, therefore, the execution of the corresponding ideal functionality must take the same number of rounds.[38]

As an illustration of this issue, let $\mathcal{F}$ be an arbitrary functionality realized by some protocol π. If $\mathcal{F}$ is to provide guaranteed termination (whether probabilistic or not), it must enforce an upper bound on the number of rounds that elapse until all parties receive their outputs. If the termination round of π is not fixed (but may depend on random choices made during its execution), this upper bound must be chosen according to the distribution induced by π.

Thus, in order to simulate correctly, the functionality $\mathcal{F}$ and π's simulator $\mathcal{S}$ must coordinate the termination round, and therefore, $\mathcal{F}$ must pass the upper bound it samples to $\mathcal{S}$. However, it is not sufficient to simply inform the simulator about the guaranteed-termination upper bound ρ_{term}. Intuitively, the reason is that protocol π may make probabilistic choices as to the order in which it calls its hybrids (and, even worse, these hybrids may even have probabilistic termination themselves). Thus, $\mathcal{F}$ needs to sample the upper bound based on π and the protocols realizing the hybrids called by π. As $\mathcal{S}$ needs to emulate the entire protocol execution, it is now left with the task of trying to sample the protocol's choices conditioned on the upper bound it receives from $\mathcal{F}$. In general, however, it is unclear whether such a reverse sampling can be performed in (strict) polynomial time.

To avoid this issue and allow for an efficient simulation, we have $\mathcal{F}$ output all the coins that were used for sampling round ρ_{term} to $\mathcal{S}$. Because $\mathcal{S}$ knows the round sampler algorithm, it can reproduce the entire computation of the sampler and use it in its simulation. In fact, as we discuss below, it suffices for our proofs to have $\mathcal{F}$ output a *trace* of its choices to the simulator instead of all the coins that were used to sample this trace. In the remainder of this section, we motivate and formally describe our formulation of such traces. The formal description of the wrappers, which in particular sample traces, can then be found at the end of this section.

Execution traces. As mentioned above, in the synchronous communication model, the execution of the ideal functionality must take the same number of rounds as the protocol. For example, suppose that the functionality $\mathcal{F}$ in our illustration above is used as a hybrid by a higher-level protocol π'. The functionality $\mathcal{G}$ realized by π' must, similarly to $\mathcal{F}$, choose an upper bound on the number of rounds that elapse before parties obtain their outputs. However, this upper bound now depends not only on π' itself but also on π (in particular, when π is a probabilistic-termination protocol).

Given the above, the round sampler of a functionality needs to keep track of how the functionality was realized. This can be achieved via the notion of *trace*. A trace basically records which hybrids were called by a protocol, and in a recursive way, for each hybrid, which hybrids would have been called by a protocol realizing that hybrid. The recursion ends with the hybrids that are "assumed" by the model, called *atomic* functionalities.[39]

[38] In particular, this means that most CSFs are not realizable, since they always guarantee output after two rounds.

[39] In this work, atomic functionalities are always *parallel SMT* CSFs $\mathcal{F}_{\text{PSMT}}$, defined in Section 7.3.

Building on our running illustration above, suppose protocol π' (realizing $\mathcal{G}$) makes ideal hybrid calls to $\mathcal{F}$ and to some atomic functionality $\mathcal{H}$. Assume that in an example execution, π' happens to make (sequential) calls to instances of $\mathcal{H}$ and $\mathcal{F}$ in the following order: $\mathcal{F}$, then $\mathcal{H}$, and finally $\mathcal{F}$ again. Moreover, assume that $\mathcal{F}$ is replaced by protocol π (realizing $\mathcal{F}$) and that π happens to make two (sequential) calls to $\mathcal{H}$ upon the first invocation by π', and three (sequential) calls to $\mathcal{H}$ the second time (we assume that both π and π' call exactly one hybrid in every round). Then, this would result in the trace depicted in Figure 6.2.

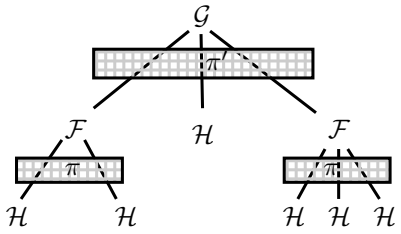

Figure 6.2: Example of an execution trace

Assume that π is a probabilistic-termination protocol and π' a deterministic-termination protocol. Consequently, this means that $\mathcal{F}$ is in fact a flexibly wrapped functionality of some CSF $\mathcal{F}'$, i.e., $\mathcal{F} = \mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$, where the distribution $D_{\mathcal{F}}$ samples (from a distribution induced by π) depth-1 traces with root $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ and leaves $\mathcal{H}$.[40] Similarly, $\mathcal{G}$ is a strictly wrapped functionality of some CSF $\mathcal{G}'$, i.e., $\mathcal{G} = \mathcal{W}_{\text{strict}}^{D_{\mathcal{G}}}(\mathcal{G}')$, where the distribution $D_{\mathcal{G}}$ first samples (from a distribution induced by π') a depth-1 trace with root $\mathcal{W}_{\text{strict}}^{D_{\mathcal{G}}}(\mathcal{G}')$ and leaves $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ as well as $\mathcal{H}$. Then, each leaf node $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ is replaced by a trace (independently) sampled from $D_{\mathcal{F}}$. Thus, the example trace from Figure 6.2 would look as in Figure 6.3.

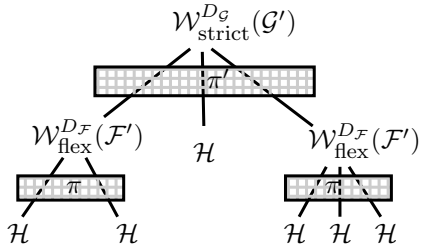

Figure 6.3: An execution trace with probabilistic-termination and deterministic-termination protocols

Formally, a trace is defined as follows:

Definition 6.2.1 (traces)**.** *A* trace *is a rooted tree of depth at least* 1*, in which all nodes are labeled by functionalities and where every node's children are* ordered*. The root and all internal nodes are labeled by wrapped CSFs (by either of the two wrappers), and the leaves are labeled by unwrapped CSFs. The* trace complexity *of a trace T, denoted $c_{\mathsf{tr}}(T)$, is the number of leaves in T. Moreover, denote by $\mathsf{flex}_{\mathsf{tr}}(T)$ the number of flexibly wrapped CSFs in T.*

[40] Note that the root node of the trace sampled from $D_{\mathcal{F}}$ is merely *labeled* by $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$, i.e., this is not a circular definition.

Remark. The actual protocol trace encodes its round complexity and the access pattern to its hybrids (i.e., when is each hybrid used). Clearly, this pattern might depend on the inputs of the parties and/or the adversary. For example, in the Byzantine agreement protocol of Feldman and Micali [42], if all honest parties start with the same input, then they get their output faster. For simplicity, in this work, the class of trace-distributions we define, and which our wrappers sample from, considers traces that are sampled independently of the honest parties' inputs or adversary. Nonetheless, our wrappers give the simulator the power to influence the simulated access-pattern and/or termination round. This allows us to use this simplified trace-distribution class to devise functionalities which, as we show, are implemented by known protocols with probabilistic termination.

Strict wrapper functionality. We now proceed to give the formal descriptions of the wrappers. The *strict wrapper functionality*, defined in Figure 6.4, is parametrized by (a sampler that induces) a distribution D over traces, and internally runs a copy of a CSF functionality $\mathcal{F}$. Initially, a trace T is sampled from D; this trace is given to the adversary once the first honest party provides its input. The trace T is used by the wrapper to define the termination round $\rho_{\mathsf{term}} \leftarrow c_{\mathsf{tr}}(T)$. In the first round, the wrapper forwards all the messages from the parties and the adversary to (and from) the functionality $\mathcal{F}$. Next, the wrapper essentially waits until round ρ_{term}, with the exception that the adversary is allowed to send $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages and change its input to the function computed by the CSF. Finally, when round ρ_{term} arrives, the wrapper provides the output generated by $\mathcal{F}$ to all parties.

Wrapper Functionality $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$

$\mathcal{W}_{\text{strict}}$, parametrized by an efficiently sampleable distribution D, internally runs a copy of $\mathcal{F}$ and proceeds as follows:

- Initially, sample a trace $T \leftarrow D$ and compute the output round $\rho_{\mathsf{term}} \leftarrow c_{\mathsf{tr}}(T)$. Send $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a]
- At all times, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- In round $\rho = 1$: Forward $(\texttt{input}, \mathsf{sid}, \cdot)$ messages from each party $\mathsf{P}_i \in \mathcal{P}$ to $\mathcal{F}$. In addition, forward $(\texttt{leakage}, \mathsf{sid}, \cdot)$ messages from $\mathcal{F}$ to the adversary.
- In rounds $\rho > 1$: Upon receiving a message $(\texttt{fetch-output}, \mathsf{sid})$ from some party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
  - If $\rho = \rho_{\mathsf{term}}$, forward the message to $\mathcal{F}$, and the response $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i.
  - Else, send $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.

[a] Technically, the trace is sent to the adversary at the first activation of the functionality along with the first message.

Figure 6.4: The strict-wrapper functionality

Flexible-wrapper functionality. The *flexible-wrapper functionality*, defined in Figure 6.5, follows in similar lines to the strict wrapper. The difference is that the adversary is allowed to instruct the wrapper to deliver the output to each party at any round. In order to accomplish this, the wrapper assigns a termination indicator term_i, initially set to 0, to each party. Once

the wrapper receives an `early-output` request from the adversary for P_i, it sets $\mathsf{term}_i \leftarrow 1$. Now, when a party P_i sends a `fetch-output` request, the wrapper checks if $\mathsf{term}_i = 1$, and lets the party receive its output in this case (by forwarding the `fetch-output` request to $\mathcal{F}$). When the guaranteed-termination round ρ_{term} arrives, the wrapper provides the output to all parties that did not receive it yet.

Wrapper Functionality $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F})$

$\mathcal{W}_{\text{flex}}$, parametrized by an efficiently sampleable distribution D, internally runs a copy of $\mathcal{F}$ and proceeds as follows:

- Initially, sample a trace $T \leftarrow D$ and compute the output round $\rho_{\mathsf{term}} \leftarrow c_{\mathsf{tr}}(T)$. Send $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a] In addition, initialize termination indicators $\mathsf{term}_1, \ldots, \mathsf{term}_n \leftarrow 0$.
- At all times, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- In round $\rho = 1$: Forward $(\texttt{input}, \mathsf{sid}, \cdot)$ messages from each party $\mathsf{P}_i \in \mathcal{P}$ to $\mathcal{F}$. In addition, forward $(\texttt{leakage}, \mathsf{sid}, \cdot)$ messages from $\mathcal{F}$ to the adversary.
- In rounds $\rho > 1$:
  - Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
    - If $\mathsf{term}_i = 1$ or $\rho = \rho_{\mathsf{term}}$ (and P_i did not receive output yet), forward the message to $\mathcal{F}$, and the output $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i.
    - Else, send $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.
  - Upon receiving $(\texttt{early-output}, \mathsf{sid}, \mathsf{P}_i)$ from the adversary, set $\mathsf{term}_i \leftarrow 1$.

[a] Technically, the trace is sent to the adversary at the first activation of the functionality along with the first message.

Figure 6.5: The flexible-wrapper functionality

Chapter 7

(Fast) Composition of Probabilistic-Termination Protocols

Canonical synchronous functionalities that are wrapped using the flexible wrapper (cf. Section 6.2), i.e., functionalities that correspond to protocols with non-simultaneous termination, are cumbersome to be used as hybrid functionalities for protocols. The reason is that the adversary can cause parties to finish in different rounds, and, as a result, after the execution of the first such functionality, the parties might be *out of sync*.

This "slack" can be reduced, however, only to a difference of one round, unless one is willing to pay a linear blowup (in the number of parties) in round complexity [43, 39]. Hence, all protocols must be modified to deal with a non-simultaneous start of (at least) one round, and protocols that introduce slack must be followed by a slack-reduction procedure. In this section, we provide general transformations to reduce the desired tasks to the simpler task of designing protocols in a "stand-alone" setting, where all parties remain synchronized throughout the protocol (and no slack and round-complexity issues arise), and all the hybrids are (unachievable) CSFs that are called in a strictly sequential manner.

Definition 7.0.1 (SNF)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities. A synchronous protocol π in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model is in* synchronous normal form (SNF) *if in every round exactly one ideal functionality $\mathcal{F}_i$ is invoked by all honest parties, and in addition, no honest party hands inputs to other CSFs before this instance halts.*

Clearly, designing and proving the security of SNF protocols, which only make calls to simple two-round CSFs is a much simpler task than dealing with protocols that invoke more complicated hybrids, potentially with probabilistic termination (see Chapter 8 for concrete examples).

SNF protocols are designed as an intermediate step, since the hybrid functionalities $\mathcal{F}_i$ are two-round CSFs, and, in general, cannot be realized by real-world protocols. To that end, we define a protocol compiler that transforms SNF protocols into (non-SNF) protocols making calls to wrapped CSFs that *can* be realized in the real world, while maintaining their security and asymptotic (expected) round complexity. At the same time, the compiler takes care of any potential slack that is introduced by the protocol and ensures that the protocol can be executed even if the parties do not start the protocol simultaneously.

In Section 7.1 we apply this approach to deterministic-termination protocols, and in Section 7.2 generalize it to the probabilistic-termination setting. Section 7.3 covers the base case

of realizing the wrapped *parallel secure message transmission* $\mathcal{F}_{\text{PSMT}}$ using only *(non-parallel) secure message transmission* $\mathcal{F}_{\text{SMT}}$.

7.1 Composition with Deterministic Termination

We start by defining a slack-tolerant variant of the strict wrapper (cf. Section 6.2), which can be used even when parties operate with a (known) slack. Then, we show how to compile an SNF protocol π realizing a strictly wrapped CSF $\mathcal{F}$ into a (non-SNF) protocol π' realizing a version of $\mathcal{F}$ wrapped with the slack-tolerant strict wrapper and making calls to wrapped hybrids.

Slack-tolerant strict wrapper. The *slack-tolerant strict wrapper* $\mathcal{W}_{\text{sl-strict}}^{D,c}$, formally defined in Figure 7.1, is parametrized by an integer $c \geq 0$, which denotes the amount of slack tolerance that is added, and a distribution D over traces. The wrapper $\mathcal{W}_{\text{sl-strict}}$ is similar to $\mathcal{W}_{\text{strict}}$ but allows parties to provide input within a window of $2c+1$ rounds and ensures that they obtain output with the same slack they started with. The wrapper essentially increases the termination round by a factor of $B_c = 3c+1$, which is due to the slack-tolerance technique used to implement the wrapped version of the atomic parallel SMT functionality (cf. Section 7.3).[41]

Wrapper Functionality $\mathcal{W}_{\text{sl-strict}}^{D,c}(\mathcal{F})$

$\mathcal{W}_{\text{sl-strict}}$, parametrized by an efficiently sampleable distribution D and a non-negative integer c, internally runs a copy of $\mathcal{F}$ and proceeds as follows:

- Initially, sample a trace $T \leftarrow D$ and compute the output round $\rho_{\mathsf{term}} \leftarrow B_c \cdot c_{\mathsf{tr}}(T)$, where $B_c = 3c+1$. Send $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a] Initialize slack counters $c_1, \ldots, c_n \leftarrow 0$.
- At all times, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- In rounds $\rho = 1, \ldots, 2c+1$: Upon receiving a message from some party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
  - If the message is $(\texttt{input}, \mathsf{sid}, \cdot)$, forward it to $\mathcal{F}$, forward the $(\texttt{leakage}, \mathsf{sid}, \cdot)$ message $\mathcal{F}$ subsequently outputs to the adversary, and set P_i's local slack $c_i \leftarrow \rho - 1$.
  - Else, send $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.
- In rounds $\rho > 2c+1$: Upon receiving a message $(\texttt{fetch-output}, \mathsf{sid})$ from some party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
  - If $\rho = \rho_{\mathsf{term}} + c_i$, send the message to $\mathcal{F}$, and the output $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i.
  - Else, send $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.

[a] Technically, the trace is sent to the adversary at the first activation of the functionality along with the first message.

Figure 7.1: The slack-tolerant strict wrapper functionality

[41] We note that the insufficiency of the blowup factor $2c+1$ rounds does not correspond to any particular attack, but it is merely a technicality of the wrapped-CSF definition, see Section 7.3.

Deterministic-termination compiler. Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let π an SNF protocol that UC-realizes the strictly wrapped functionality $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$, for some depth-1 distribution D, in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, assuming that all honest parties receive their inputs at the same round. We define a compiler $\mathsf{Comp}^{c}_{\text{DT}}$, parametrized with a slack parameter $c \geq 0$, that receives as input the protocol π and distributions $D_1, \ldots, D_m$ over traces and replaces every call to a CSF $\mathcal{F}_i$ with a call to the wrapped CSF $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$. We denote the output of the compiler by $\pi' = \mathsf{Comp}^{c}_{\text{DT}}(\pi, D_1, \ldots, D_m)$.[42]

As shown below, π' realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$, for a suitably adapted distribution D^{full}, assuming all parties start within $c+1$ consecutive rounds. Consequently, the compiled protocol π' can handle a slack of up to c rounds while using hybrids that are realizable themselves.

Calling the wrapped CSFs instead of the CSFs $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$ affects the trace corresponding to $\mathcal{F}$. The new trace $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$ is obtained as follows:

1. Sample a trace $T \leftarrow D$, which is a depth-1 tree with a root label $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ and leaves from the set $\{\mathcal{F}_1, \ldots, \mathcal{F}_m\}$.
2. Construct a new trace T' with a root label $\mathcal{W}^{D^{\mathsf{full}}}_{\text{strict}}(\mathcal{F})$.
3. For each leaf node $\mathcal{F}' = \mathcal{F}_i$, for some $i \in [m]$, sample a trace $T_i \leftarrow D_i$ and append the trace T_i to the first layer in T' (i.e., replace the node $\mathcal{F}'$ with T_i).
4. Output the resulting trace T'.

The following theorem states that the compiled protocol π' UC-realizes the wrapped functionality $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$.

Theorem 7.1.1. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, let $t < n/3$, and let π an SNF protocol that UC-realizes the functionality $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ with perfect security in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, in the presence of an adaptive, malicious t-adversary, and assuming that all honest parties receive their inputs at the same round. Let $D_1, \ldots, D_m$ be arbitrary distributions over traces, let $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$, and $c \geq 0$.*

Then, the compiled protocol $\pi' = \mathsf{Comp}^{c}_{\text{DT}}(\pi, D_1, \ldots, D_m)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ with perfect security in the $(\mathcal{W}^{D_1,c}_{\text{sl-strict}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl-strict}}(\mathcal{F}_m))$-hybrid model, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.

The expected round complexity of the compiled protocol π' is

$$B_c \cdot \sum_{i \in [m]} d_i \cdot E[c_{\mathsf{tr}}(T_i)],$$

where d_i is the expected number of calls in π to hybrid $\mathcal{F}_i$, T_i is a trace sampled from D_i, and $B_c = 3c + 1$ is the blowup factor.

We start by giving the intuition for the proof of Theorem 7.1.1. Loosely speaking, the main differences between the SNF protocol π implementing the functionality $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ and the

[42] The distributions D_i depend on the protocols realizing the strictly wrapped functionalities $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$. Note, however, that the composition theorems in Sections 7.1 and 7.2 actually work for arbitrary distributions D_i.

compiled protocol π' implementing $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ is that π uses CSF as hybrids, whereas π' uses wrapped CSFs, and in addition, parties might not start at the same round, but with a slack of c rounds. In order to ensure that any potential overlap between concurrent calls to different wrapped hybrids remain secure, the wrappers expand each round to $3c+1$ rounds.

Now, given a simulator $\mathcal{S}$ for the dummy adversary and the SNF protocol π, we construct a simulator $\mathcal{S}'$ for the dummy adversary and the compiled protocol π'. The new simulator acts as a proxy between $\mathcal{S}$ on the one hand and the environment and the ideal functionality on the other, with the exception that it must "synchronize" the round counters between them. Therefore, $\mathcal{S}'$ stores a local round counter ρ_i for every hybrid $\mathcal{H}_i$, and a "slack counter" c_j for every party P_j to ensure that its messages are delivered with the same initial slack it started the protocol.

Proof. Let $\mathcal{S}$ be the simulator for protocol π running with the dummy adversary.[43] Consider the following simulator $\mathcal{S}'$ for π', that internally runs a copy of $\mathcal{S}$. Initially, $\mathcal{S}'$ sets slack counters $c_1, \ldots, c_n \leftarrow 0$ and proceeds as follows.

- At any round forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{S}$ to $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$.
- In rounds $\rho = 1, \ldots, 2c+1$, upon receiving $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_j, \cdot)$ from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$, forward the message to $\mathcal{S}$ and in addition record the slack for party P_j as $c_j \leftarrow \rho - 1$.

  Along with the very first such message, $\mathcal{S}'$ receives a trace T^{full} from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$. $\mathcal{S}'$ constructs a new trace T with the root $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$, where the leaves are set as follows: Each node in the first layer of T^{full} is a root for a subtree labeled with $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ (for some $i \in [m]$); $\mathcal{S}'$ adds the leaf $\mathcal{F}_i$ to the first layer in T. Finally, $\mathcal{S}'$ passes T to $\mathcal{S}$.
- Simulate the execution of every wrapped hybrid $\mathcal{H}_i = \mathcal{W}^{D_j,c}_{\text{sl-strict}}(\mathcal{F}_j)$ (for some $j \in [m]$) in the order they appear in the first layer in T^{full} as follows[44] (the first such hybrid must be simulated as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap):
  - Let ρ_i be the (simulated) round counter of $\mathcal{H}_i$ and let T^{full}_i be the corresponding subtree in T^{full}.
  - In any round ρ_i forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.
  - For every party P_j, in round $\rho_i = c_j + 1$, obtain the simulated leakage for P_j from $\mathcal{S}$ and pass it to the environment;[45] add $(\texttt{trace}, \mathsf{sid}, T^{\mathsf{full}}_i)$ to the first such message.
  - In all other rounds ρ_i, forward the messages $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ to the environment (to simulate the advancement of $\mathcal{H}_i$).
  - The simulation for party P_j ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T^{\mathsf{full}}_i) + c_j$.

Let $\mathcal{Z}'$ be an environment that can distinguish between an execution of protocol π' in the $(\mathcal{W}^{D_1,c}_{\text{sl-strict}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl-strict}}(\mathcal{F}_m))$-hybrid model with the dummy adversary and the execution

[43] Recall that proving security with respect to the dummy adversary is sufficient (cf. Claim 5.1.2).
[44] Recall that the children at each node in a trace are ordered.
[45] $\mathcal{S}$ can be advanced by suitably sending it $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages.

in the ideal model with $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ and $\mathcal{S}'$. We construct the following environment $\mathcal{Z}$ distinguishing between an execution of π in the $(\mathcal{F}_1,\ldots,\mathcal{F}_m)$-hybrid model with the dummy adversary and the ideal model with $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ and $\mathcal{S}$:

- $\mathcal{Z}$ internally runs a copy of $\mathcal{Z}'$, emulating the parties and the adversary (either in a real execution of π or an ideal execution of $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$). Initialize slack counters $c_1,\ldots,c_n \leftarrow 0$ and a simulated round counter ρ (for $\mathcal{Z}'$).
- Whenever $\mathcal{Z}'$ sends a message $(\texttt{input},\mathsf{sid},\cdot)$ to P_j in rounds $\rho = 1,\ldots,2c+1$, $\mathcal{Z}$ forwards the message to P_j and records slack $c_j \leftarrow \rho - 1$.
- For each executed (resp. simulated) two-round CSF hybrid $\mathcal{F}_i$, proceed as follows to simulate an execution (resp. simulation) of $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ to $\mathcal{Z}'$ (the first such simulation takes place as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap):
  - Initialize a round counter $\rho_i \leftarrow 1$ and sample a trace T_i^{full} from D_i.
  - At any round, forward $(\texttt{adv-input},\mathsf{sid},\cdot)$ messages from $\mathcal{Z}'$ to the adversary.
  - For every party P_j, in round $\rho_i = c_j + 1$, obtain the leakage for P_j from the adversary and pass it to $\mathcal{Z}'$;[46] add $(\texttt{trace},\mathsf{sid},T_i^{\mathsf{full}})$ to the first such message.
  - In all other rounds ρ_i, upon receiving $(\texttt{fetch-output},\mathsf{sid},\cdot)$ messages from $\mathcal{Z}'$ for some party P_j, pass $(\texttt{fetch-output},\mathsf{sid},\mathsf{P}_j)$ to $\mathcal{Z}$ (to simulate the advancement of the execution).
  - The simulation for party P_j ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T_i^{\mathsf{full}}) + c_j$.
- At any round, forward $(\texttt{output},\mathsf{sid},\cdot)$ messages from a party to $\mathcal{Z}'$.
- Output whatever decision bit $\mathcal{Z}'$ outputs.

It can be seen by inspection that:

- When $\mathcal{Z}$ interacts with a real-world execution of π with hybrids $\mathcal{F}_i$, the view of $\mathcal{Z}'$ is exactly the view it would have when interacting with a real-world execution of π' with hybrids $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$, and
- When $\mathcal{Z}$ interacts with an ideal-world execution of $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ with simulator $\mathcal{S}$, the view of $\mathcal{Z}'$ is exactly the view it would have when interacting with an ideal-world execution of $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ with simulator $\mathcal{S}'$.

The expected round complexity follows by linearity of expectation and by noting that the expected number of times a hybrid $\mathcal{F}_i$ is called in π and the expected trace complexity of D_i are independent random variables. Indeed, the trace complexity needed to implement $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ is independent of the protocol π, and the number of calls to $\mathcal{F}_i$ in π depends on the CSF representation of $\mathcal{F}_i$, and not on its wrapped version. □

[46] The execution $\mathcal{Z}$ interacts with can be advanced by suitably sending $(\texttt{fetch-output},\mathsf{sid},\cdot)$ messages to the parties.

7.2 Composition with Probabilistic Termination

The composition theorem in Section 7.1 does not work if the protocol π itself introduces slack (e.g., the fast broadcast protocol by Feldman and Micali [42]) or if one of the hybrids needs to be replaced by a slack-introducing protocol (e.g., instantiating the broadcast hybrids with fast broadcast protocols in BGW [12]).

As in Section 7.1, we start by adjusting the flexible wrapper (cf. Section 6.2) to be slack-tolerant. In addition, the slack-tolerant flexible wrapper ensures that all parties will obtain their outputs within two consecutive rounds. Then, we show how to compile an SNF protocol π realizing a CSF $\mathcal{F}$, wrapped with the flexible wrapper, into a (non-SNF) protocol π' realizing a version of $\mathcal{F}$ wrapped with slack-tolerant flexible wrapper. The case where π implements a strictly wrapped CSF, but some of the hybrids are wrapped with the slack-tolerant flexible wrapper follows along similar lines.

Slack-tolerant flexible wrapper. The *slack-tolerant flexible wrapper* $\mathcal{W}_{\text{sl-flex}}^{D,c}$, formally defined in Figure 7.2, is parametrized by an integer $c \geq 0$, which denotes the amount of slack tolerance that is added, and a distribution D over traces. The wrapper $\mathcal{W}_{\text{sl-flex}}$ is similar to $\mathcal{W}_{\text{flex}}$ but allows parties to provide input within a window of $2c+1$ rounds and ensures that all honest parties will receive their output within two consecutive rounds. The wrapper essentially increases the termination round to

$$\rho_{\mathsf{term}} = B_c \cdot c_{\mathsf{tr}}(T) + 2 \cdot \mathsf{flex}_{\mathsf{tr}}(T) + c,$$

where the blowup factor B_c is as explained in Section 7.1, and the additional factor of 2 results from the termination protocol described below for every flexibly wrapped CSF, which increases the round complexity by at most two additional rounds (recall that $\mathsf{flex}_{\mathsf{tr}}(T)$ denotes the number of such CSFs), and c is due to the potential slack. $\mathcal{W}_{\text{sl-flex}}$ allows the adversary to deliver output at any round prior to ρ_{term} but ensures that all parties obtain output with a slack of at most one round. Moreover, it allows the adversary to obtain the output using the $(\texttt{get-output}, \mathsf{sid})$ command, which is necessary in order to simulate the termination protocol.

Probabilistic-termination compiler. Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let π be an SNF protocol that UC-realizes the flexibly wrapped functionality $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, assuming all parties start at the same round. Define the following compiler $\mathsf{Comp}_{\text{PTR}}^{c}$, parametrized by a slack parameter $c \geq 0$. The compiler receives as input the protocol π, distributions $D_1, \ldots, D_m$ over traces, and a subset $I \subseteq [m]$ indexing which CSFs $\mathcal{F}_i$ are to be wrapped with $\mathcal{W}_{\text{sl-flex}}$ and which with $\mathcal{W}_{\text{sl-strict}}$; every call in π to a CSF $\mathcal{F}_i$ is replaced with a call to the wrapped CSF $\mathcal{W}_{\text{sl-flex}}^{D_i,c}(\mathcal{F}_i)$ if $i \in I$ or to $\mathcal{W}_{\text{sl-strict}}^{D_i,c}(\mathcal{F}_i)$ if $i \notin I$.

In addition, the compiler adds the following termination procedure, based on an approach originally suggested by Bracha [16], which ensures all honest parties will terminate within two consecutive rounds:

- As soon as a party is ready to output a value y (according to the prescribed protocol) or

upon receiving at least $t+1$ messages $(\mathtt{end}, \mathsf{sid}, y)$ for the same value y (whichever happens first), it sends $(\mathtt{end}, \mathsf{sid}, y)$ to all parties.

- Upon receiving $n-t$ messages $(\mathtt{end}, \mathsf{sid}, y)$ for the same value y, a party outputs y as the result of the computation and halts.

Observe that this technique only works for public-output functionalities, and, therefore, only CSFs with public output can be wrapped by $\mathcal{W}_{\text{sl-flex}}$. We denote the output of the compiler by $\pi' = \mathsf{Comp}^{c}_{\text{PTR}}(\pi, D_1, \ldots, D_m, I)$.

The following theorem states that the compiled protocol π' UC-realizes the wrapped functionality $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$, for the adapted distribution $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$. Consequently, the compiled protocol π' can handle a slack of up to c rounds, while using hybrids that are realizable themselves, and ensuring that the output slack is at most one round (as opposed to π). Calling the wrapped hybrids instead of the CSFs affects the trace corresponding to $\mathcal{F}$ in exactly the same way as in the case with deterministic termination (cf. Section 7.1).[47]

Wrapper Functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F})$

$\mathcal{W}_{\text{sl-flex}}$, parametrized by an efficiently sampleable distribution D and a non-negative integer c, internally runs a copy of (the public-output functionality) $\mathcal{F}$ and proceeds as follows:

- Initially, sample a trace $T \leftarrow D$ and compute the output round $\rho_{\mathsf{term}} \leftarrow B_c \cdot c_{\mathsf{tr}}(T) + 2 \cdot \mathsf{flex}_{\mathsf{tr}}(T) + c$, where $B_c = 3c+1$. Send $(\mathtt{trace}, \mathsf{sid}, T)$ to the adversary.[a] Initialize termination indicators $\mathsf{term}_1, \ldots, \mathsf{term}_n \leftarrow 0$.
- At all times, forward $(\mathtt{adv\text{-}input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- In rounds $\rho = 1, \ldots, 2c+1$: Upon receiving a message from some party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
  - If the message is $(\mathtt{input}, \mathsf{sid}, \cdot)$, send it to $\mathcal{F}$ and forward the $(\mathtt{leakage}, \mathsf{sid}, \cdot)$ message $\mathcal{F}$ subsequently outputs to the adversary.
  - Else, send $(\mathtt{fetch\text{-}output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.
- In rounds $\rho > 2c+1$:
  - Upon receiving a message $(\mathtt{fetch\text{-}output}, \mathsf{sid})$ from some party $\mathsf{P}_i \in \mathcal{P}$, proceed as follows:
    - If $\mathsf{term}_i = 1$ or $\rho = \rho_{\mathsf{term}}$, forward the message to $\mathcal{F}$, and the output $(\mathtt{output}, \mathsf{sid}, y)$ to P_i. Record the output value y.
    - Else, output $(\mathtt{fetch\text{-}output}, \mathsf{sid}, \mathsf{P}_i)$ to the adversary.
  - Upon receiving $(\mathtt{get\text{-}output}, \mathsf{sid})$ from the adversary, if the output value y was not recorded yet, send $(\mathtt{fetch\text{-}output}, \mathsf{sid})$ to $\mathcal{F}$ on behalf of some party P_i. Next, send $(\mathtt{output}, \mathsf{sid}, y)$ to the adversary.
  - Upon receiving $(\mathtt{early\text{-}output}, \mathsf{sid}, \mathsf{P}_i)$ from the adversary, set $\mathsf{term}_i \leftarrow 1$ and $\rho_{\mathsf{term}} \leftarrow \min\{\rho_{\mathsf{term}}, \rho+1\}$.

[a] Technically, the trace is sent to the adversary at the first activation of the functionality along with the first message.

Figure 7.2: The slack-tolerant flexible wrapper functionality

[47] Of course, the root of the trace T sampled from D is a flexibly wrapped functionality $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$ in the probabilistic-termination case.

Theorem 7.2.1. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, let $t < n/3$, and let π an SNF protocol that UC-realizes $\mathcal{W}^D_{\text{flex}}(\mathcal{F})$ with perfect security in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, in the presence of an adaptive, malicious t-adversary, and assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$ and let $c \geq 0$. Assume that $\mathcal{F}$ and $\mathcal{F}_i$, for every $i \in I$, are public-output functionalities.*

Then, the compiled protocol $\pi' = \mathsf{Comp}^c_{\text{PTR}}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ with perfect security in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, where $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ in case $i \in I$ and $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ otherwise, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.

The expected round complexity of the compiled protocol π' is

$$B_c \cdot \sum_{i \in [m]} d_i \cdot E[c_{\mathsf{tr}}(T_i)] + 2 \cdot \sum_{i \in [m]} d_i \cdot E[\mathsf{flex}_{\mathsf{tr}}(T_i)] + 2,$$

where d_i is the expected number of calls in π to hybrid $\mathcal{F}_i$, T_i is a trace sampled from D_i, and $B_c = 3c + 1$ is the blowup factor.

The intuition for proving Theorem 7.2.1 is similar to that of proving Theorem 7.1.1. In addition to simply synchronizing between the simulator $\mathcal{S}$ and the ideal functionality and environment, $\mathcal{S}'$ must also address the following issues. First, some CSFs ($\mathcal{F}_i$ for $i \in I$) are wrapped using the flexible wrapper, whereas others ($\mathcal{F}_i$ for $i \notin I$) are wrapped using the strict wrapper. Second, $\mathcal{S}'$ must simulate the termination procedure at the end of every flexibly wrapped CSF and at the end of the simulation.

Proof. Let $\mathcal{S}$ be the simulator for protocol π running with the dummy adversary.[48] Consider the following simulator $\mathcal{S}'$ for π', that internally runs a copy of $\mathcal{S}$. Initially, $\mathcal{S}'$ sets slack counters $c_1, \ldots, c_n \leftarrow 0$ and proceeds as follows.

- At any round forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{S}$ to $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$.
- In rounds $\rho = 1, \ldots, 2c+1$, upon receiving $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_j, \cdot)$ from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$, forward the message to $\mathcal{S}$ and in addition record the slack for party P_j as $c_j \leftarrow \rho - 1$.

  Along with the very first such message, $\mathcal{S}'$ receives a trace T^{full} from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$. $\mathcal{S}'$ constructs a new trace T with the root $\mathcal{W}^D_{\text{flex}}(\mathcal{F})$, where the leaves are set as follows: Each node in the first layer of T^{full} is a root for a subtree labeled with $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ or $\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}_i)$ (for some $i \in [m]$); $\mathcal{S}'$ adds the leaf $\mathcal{F}_i$ to the first layer in T. Finally, $\mathcal{S}'$ passes T to $\mathcal{S}$.
- Simulate the execution of all wrapped hybrids $\mathcal{H}_i$ in the order they appear in T^{full} (the first such hybrid must be simulated as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap). If the hybrid $\mathcal{H}_i$ is of the form $\mathcal{W}^{D_j,c}_{\text{sl-strict}}(\mathcal{F}_j)$ (for some $j \in [m]$), i.e., if $j \notin I$, proceed as follows:
  - Let ρ_i be the (simulated) round counter of $\mathcal{H}_i$ and let T^{full}_i be the corresponding subtree in T^{full}.

[48] Recall that proving security with respect to the dummy adversary is sufficient, see Claim 5.1.2.

– In any round ρ_i forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.

– For every party P_j, in round $\rho_i = c_j + 1$, obtain the simulated leakage for P_j from $\mathcal{S}$ and pass it to the environment;[49] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.

– In all other rounds ρ_i, forward the messages $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ from $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$ to the environment (to simulate the advancement of $\mathcal{H}_i$' execution).

– The simulation for party P_j ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T_i^{\mathsf{full}}) + c_j$.

If the hybrid $\mathcal{H}_i$ is of the form $\mathcal{W}_{\text{sl-flex}}^{D_j,c}(\mathcal{F}_j)$ (for some $j \in [m]$), i.e., if $j \in I$, proceed as follows:

– Let ρ_i be the (simulated) round counter of $\mathcal{H}_i$ and let T_i^{full} be the corresponding subtree in T^{full}. Set $\rho_{\mathsf{term}} \leftarrow B_c \cdot c_{\mathsf{tr}}(T^{\mathsf{full}}) + 2 \cdot \mathsf{flex}_{\mathsf{tr}}(T^{\mathsf{full}}) + c$.

– In any round ρ_i, forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.

– For every party P_j, in round $\rho_i = c_j + 1$, obtain the simulated leakage for P_j from $\mathcal{S}$ and pass it to the environment;[50] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.

– In all other rounds ρ_i, forward the messages $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ from $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$ to the environment (to simulate the advancement of $\mathcal{H}_i$' execution).

– If the environment issues $(\texttt{early-output}, \mathsf{sid}, \cdot)$ commands, before round ρ_{term}, for certain parties $\mathsf{P}_j \in \mathcal{P}$, set $c_j \leftarrow 0$ for these parties and $c_j \leftarrow 1$ for the others and end the simulation of $\mathcal{H}_i$ one round later. If the environment does not issue such a command for any party, set $c_j \leftarrow 0$ for all parties and end the simulation of $\mathcal{H}_i$ in round ρ_{term}.

- When $\mathcal{S}$ wants to output $(\texttt{early-output}, \mathsf{sid}, \mathsf{P}_j)$ to $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$, proceed as follows:

  – Pass $(\texttt{get-output}, \mathsf{sid})$ to $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$, obtain $(\texttt{output}, \mathsf{sid}, y)$, and record y (the first time).

  – Simulate P_j sending $(\texttt{end}, \mathsf{sid}, y)$ to all parties.

  – For every party P_j, keep track of how many simulated $(\texttt{end}, \mathsf{sid}, \cdot)$ messages have been received by P_j (including those sent by corrupted parties).

    * When a party receives $t+1$ messages $(\texttt{end}, \mathsf{sid}, y)$ (for the same value y), simulate that party's sending of such a message of its own (unless already done so).

    * When a party P_j receives $n-t$ messages $(\texttt{end}, \mathsf{sid}, y)$ (for the same value y), send $(\texttt{early-output}, \mathsf{sid}, \mathsf{P}_j)$ to $\mathcal{W}_{\text{PT}}^{D^{\mathsf{full}}}(\mathcal{F})$.

Let $\mathcal{Z}'$ be an environment distinguishing between an execution of π' in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model and the ideal model with $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$ and $\mathcal{S}'$. We construct the following environment $\mathcal{Z}$ distinguishing between an execution of π in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model and the ideal model with $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F})$ and $\mathcal{S}$:

[49] $\mathcal{S}$ can be advanced by suitably sending it $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages.
[50] $\mathcal{S}$ can be advanced by suitably sending it $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages.

- $\mathcal{Z}$ internally runs a copy of $\mathcal{Z}'$ and emulates the parties and the adversary (either in a real execution of π or an ideal execution of $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$). Initialize slack counters $c_1, \ldots, c_n \leftarrow 0$ and a simulated round counter ρ (for $\mathcal{Z}'$).

- When $\mathcal{Z}'$ sends a message $(\texttt{input}, \mathsf{sid}, \cdot)$ for P_j in rounds $\rho = 1, \ldots, 2c+1$, $\mathcal{Z}$ forwards the message to P_j and records slack $c_j \leftarrow \rho - 1$.

- For each executed (resp. simulated) two-round CSF hybrid $\mathcal{F}_i$, simulate an execution (resp. simulation) of $\mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ or $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ to $\mathcal{Z}'$ (the first such simulation takes place as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap). If $i \notin I$, i.e., if the hybrid functionality is of the form $\mathcal{H}_i = \mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$, proceed as follows:

  - Initialize a round counter $\rho_i \leftarrow 1$ and sample a trace T_i^{full} from D_i.
  - At any round, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to the adversary.
  - For every party P_j, in round $\rho_i = c_j + 1$, obtain the leakage for P_j from the adversary and pass it to $\mathcal{Z}'$;[51] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.
  - In all other rounds ρ_i, upon receiving $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ for some party P_j, pass $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_j)$ to $\mathcal{Z}$ (to simulate the advancement of the execution).
  - The simulation for party P_j ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T_i^{\mathsf{full}}) + c_j$.

  If $i \in O$, i.e., if the hybrid is of the form $\mathcal{H}_i = \mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$, proceed as follows:

  - Initialize a round counter $\rho_i \leftarrow 1$ and sample a trace T_i^{full} from D_i. Set

    $$\rho_{\mathsf{term}} \leftarrow B_c \cdot c_{\mathsf{tr}}(T_i^{\mathsf{full}}) + 2 \cdot \mathsf{flex}_{\mathsf{tr}}(T_i^{\mathsf{full}}) + c.$$

  - At any round, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to the adversary.
  - For every party P_j, in round $\rho_i = c_j + 1$, obtain the leakage for P_j from the adversary and pass it to $\mathcal{Z}'$;[52] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.
  - In all other rounds ρ_i, upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ messages from $\mathcal{Z}'$ for some party P_j, pass $(\texttt{fetch-output}, \mathsf{sid}, \mathsf{P}_j)$ to $\mathcal{Z}$ (to simulate the advancement of the execution).
  - If $\mathcal{Z}'$ issues $(\texttt{early-output}, \mathsf{sid}, \cdot)$ commands for certain parties $\mathsf{P}_j \in \mathcal{P}$ before round ρ_{term}, set $c_j \leftarrow 0$ for these parties and $c_j \leftarrow 1$ for the others, and end the simulation of $\mathcal{H}_i$ one round later. If $\mathcal{Z}'$ did not issue such a command to any party by round ρ_{term}, set $c_j \leftarrow 0$ for all parties and end the simulation of $\mathcal{H}_i$ in round ρ_{term}.

- When a party wants to output $(\texttt{output}, \mathsf{sid}, y)$, proceed as follows:

  - Pass $(\texttt{get-output}, \mathsf{sid})$ to $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$, obtain $(\texttt{output}, \mathsf{sid}, y)$, and record y (only at the first time).

[51] The execution $\mathcal{Z}$ interacts with can be advanced by suitably sending $(\texttt{fetch-output}, \mathsf{sid})$ messages to the parties.

[52] The execution $\mathcal{Z}$ interacts with can be advanced by suitably sending $(\texttt{fetch-output}, \mathsf{sid})$ messages to the parties.

– Simulate (to $\mathcal{Z}'$) P_j sending $(\mathtt{end}, \mathsf{sid}, y)$ to all parties.

– For every party P_j, keep track of how many simulated $(\mathtt{end}, \mathsf{sid}, \cdot)$ messages it has received (including those sent by corrupted parties).

∗ When a party receives $t+1$ messages $(\mathtt{end}, \mathsf{sid}, y)$ (for the same y), simulate that party's sending such a message of its own (unless already done so).

∗ When a party P_j receives $n-t$ messages $(\mathtt{end}, \mathsf{sid}, y)$ (for the same y), pass $(\mathtt{output}, \mathsf{sid}, y)$ to $\mathcal{Z}'$ on behalf of P_j.

- Output whatever decision bit $\mathcal{Z}'$ outputs.

It can be seen by inspection that:

- When $\mathcal{Z}$ interacts with a real-world execution of π with hybrids $\mathcal{F}_i$, the view of $\mathcal{Z}'$ is exactly the view it would have when interacting with a real-world execution of π' with hybrids $\mathcal{W}(\mathcal{F}_i)$ and the dummy adversary.

- When $\mathcal{Z}$ interacts with an ideal-world execution of $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$ with simulator $\mathcal{S}$, the view of $\mathcal{Z}'$ is exactly the view it would have when interacting with an ideal-world execution of $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ with simulator $\mathcal{S}'$.

The expected round complexity follows by similar arguments as in Theorem 7.1.1. □

Consider now the scenario where an SNF protocol π realizes a strictly wrapped functionality, yet some of the CSF hybrids are to be wrapped by flexible wrappers. The corresponding compiler $\mathsf{Comp}_{\text{PT}}$ works as $\mathsf{Comp}_{\text{PTR}}$ with the exception that the slack-reduction protocol is not performed at the end. The proof of the following theorem follows that of Theorem 7.2.1.

Theorem 7.2.2. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, let $t < n/3$, and let π an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ with perfect security in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, in the presence of an adaptive, malicious t-adversary, and assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$ and let $c \geq 0$. Assume that $\mathcal{F}$ and $\mathcal{F}_i$, for every $i \in I$, are public-output functionalities.*

Then, the compiled protocol $\pi' = \mathsf{Comp}^{c}_{\text{PT}}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ with perfect security in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, where $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ in case $i \in I$ and $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ otherwise, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.

The expected round complexity of the compiled protocol π' is

$$B_c \cdot \sum_{i \in [m]} d_i \cdot E[c_{\mathsf{tr}}(T_i)] + 2 \cdot \sum_{i \in [m]} d_i \cdot E[\mathsf{flex}_{\mathsf{tr}}(T_i)],$$

where d_i is the expected number of calls in π to hybrid $\mathcal{F}_i$, T_i is a trace sampled from D_i, and $B_c = 3c+1$ is the blowup factor.

7.3 Wrapping Secure Channels

The basis of the top-down, inductive approach taken in this work consists of providing protocols realizing wrapped atomic functionalities, using merely secure channels, i.e., $\mathcal{F}_{\text{SMT}}$. Due to the restrictions to SNF protocols, which may only call a single CSF hybrid in any given round, a parallel variant $\mathcal{F}_{\text{PSMT}}$ of $\mathcal{F}_{\text{SMT}}$ (defined below) is used as an atomic functionality. This ensures that in SNF protocols parties can securely send messages to each other simultaneously.

Parallel SMT. The *parallel secure message transmission functionality* $\mathcal{F}_{\text{PSMT}}$ is a CSF for the following functions f_{PSMT} and l_{PSMT} Each party P_i has a vector of input values $(x_1^i, \ldots, x_n^i)$ such that x_j^i is sent from P_i to P_j. That is, the function to compute is

$$f_{\text{PSMT}}((x_1^1, \ldots, x_n^1), \ldots, (x_1^n, \ldots, x_n^n), a) = ((x_1^1, \ldots, x_1^n), \ldots, (x_n^1, \ldots, x_n^n)).$$

As we consider rushing adversaries, that can determine the messages sent by the corrupted parties *after* receiving the messages sent by the honest parties, the leakage function should leak the messages that are to be delivered from honest parties to corrupted parties. Therefore, the leakage function is $l_{\text{PSMT}}((x_1^1, \ldots, x_n^1), \ldots, (x_1^n, \ldots, x_n^n)) = (y_1^1, y_2^1, \ldots, y_{n-1}^n, y_n^n)$, where $y_j^i = |x_j^i|$ in case P_j is honest and $y_j^i = x_j^i$ in case P_j is corrupted.

Realizing wrapped parallel SMT. The remainder of this section deals with securely realizing $\mathcal{W}_{\text{sl-strict}}^{D,c}(\mathcal{F}_{\text{PSMT}})$ in the $\mathcal{F}_{\text{SMT}}$-hybrid model, for a particular distribution D and an arbitrary non-negative integer c. Note that the corresponding protocol π_{PSMT} is *not* an SNF protocol since it makes n^2 parallel calls to $\mathcal{F}_{\text{SMT}}$ in each round; this is of no concern since it directly realizes a wrapped functionality and therefore need not be compiled. There is a straightforward (non-SNF) protocol realizing $\mathcal{F}_{\text{PSMT}}$ in the $\mathcal{F}_{\text{SMT}}$-hybrid model, and therefore (due to the UC composition theorem) it suffices to describe protocol π_{PSMT} in the $\mathcal{F}_{\text{PSMT}}$-hybrid model.

A standard solution to overcome asynchrony by a constant number of rounds $c \geq 0$, introduced by Lindell et al. [74] and used by Katz and Koo [65], is to expand each communication round to $2c+1$ rounds. Each party listens for messages throughout all $2c+1$ rounds, and sends its own messages in round $c+1$. It is straightforward to verify that if the slack is c, i.e., the parties start within $c+1$ rounds from each other, round r-messages (in the original protocol, without round expansion) are sent, and delivered, before round $(r+1)$-messages and after round $(r-1)$-messages.

The solution described above does not immediately apply to our case, due to the nature of canonical synchronous functionalities. Recall that in a CSF the adversary can send an `adv-input` message (and affect the output) only before any honest party has received an output from the functionality. If only $2c+1$ rounds are used a subtle problem arises: Assume for simplicity that $c = 1$ and say that P_1 is a fast party and P_2 is a slow party. Initially, P_1 listens for one round. In the second round P_2 listens and P_1 send its messages to all the parties. In the third round P_2 sends its messages and P_1 receives its message, produces output, and completes the round. Now, P_2 listens for an additional round, and the adversary can send it messages on behalf of corrupted parties. In other words, the adversary can choose the value for P_2's output *after* P_1 has received its output – such a phenomena cannot be modeled using CSFs. For this

reason we add an additional round where each party is idle; if P_1 waits one more round (without listening) before it produces its output, then P_2 will receive all the messages that determine its output, and so once P_1 produces output and completes, the adversary cannot affect the output of P_2.

As a result, in the protocol presented in Figure 7.3, each round is expanded to $3c+1$ rounds, where during the final c rounds, parties are simply idle and ignore any messages they receive.

Denote by D_{PSMT} the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{PSMT}}}(\mathcal{F}_{\text{PSMT}})$ and $3c+1$ leaves $\mathcal{F}_{\text{PSMT}}$.

Lemma 7.3.1. *Let $c \geq 0$ and $t < n/3$. Protocol π_{PSMT} UC-realizes $\mathcal{W}_{\text{sl-strict}}^{D_{\text{PSMT}},c}(\mathcal{F}_{\text{PSMT}})$ with perfect security in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

Protocol π_{PSMT} (realizing wrapped $\mathcal{F}_{\text{PSMT}}$)

Each party $\mathsf{P}_i \in \mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ proceeds as follows:

- Initially, obtain input $(\texttt{input}, (x_1^{(i)}, \ldots, x_n^{(i)}))$ from the environment. Set $y_1, \ldots, y_n \leftarrow \perp$.
- In every round $\rho = 1, \ldots, c$:[a] Send $(\texttt{input}, \perp)$ to (a fresh instance of) $\mathcal{F}_{\text{PSMT}}$. Obtain output $(\texttt{output}, (u_1, \ldots, u_n))$ from $\mathcal{F}_{\text{PSMT}}$ with $u_i \in \{0,1\}^* \cup \{\perp\}$. For each i with $u_i \neq \perp$, set $y_i \leftarrow u_i$.
- In round $\rho = c+1$: Send the message $(\texttt{input}, (x_1^{(i)}, \ldots, x_n^{(i)}))$ to $\mathcal{F}_{\text{PSMT}}$, and obtain the output $(\texttt{output}, (u_1, \ldots, u_n))$ with $u_i \in \{0,1\}^* \cup \{\perp\}$. For each i with $u_i \neq \perp$, set $y_i \leftarrow u_i$.
- In every round $\rho = c+2, \ldots, 2c+1$: Send $(\texttt{input}, \perp)$ to $\mathcal{F}_{\text{PSMT}}$, and obtain the output $(\texttt{output}, (u_1, \ldots, u_n))$ with $u_i \in \{0,1\}^* \cup \{\perp\}$. For each i with $u_i \neq \perp$, set $y_i \leftarrow u_i$.
- In every round $\rho = 2c+2, \ldots, 3c$: Do nothing.
- In round $\rho = 3c+1$: Output $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_n))$.

[a] Note that ρ is the *local* round counter of party P_i.

Figure 7.3: The wrapped parallel SMT protocol, in the $\mathcal{F}_{\text{PSMT}}$-hybrid model

Proof. For simplicity, denote by $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$ the wrapped functionality $\mathcal{W}_{\text{sl-strict}}^{D_{\text{PSMT}},c}(\mathcal{F}_{\text{PSMT}})$. We construct the following simulator $\mathcal{S}$ running with $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$, simulating the dummy adversary.[53] Let $\mathcal{Z}$ be an environment. The main idea is to simulate each of the $3c+1$ instances of $\mathcal{F}_{\text{PSMT}}$ ro $\mathcal{Z}$. Initially, $\mathcal{S}$ receives the message $(\texttt{trace}, \mathsf{sid}, T)$, where T is a depth-1 trace consisting of $3c+1$ leaves $\mathcal{F}_{\text{PSMT}}$. Next, $\mathcal{S}$ simulates $3c+1$ sequential instances of $\mathcal{F}_{\text{PSMT}}$, by interacting with $\mathcal{Z}$ and $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$. In the first $2c+1$ instances of $\mathcal{F}_{\text{PSMT}}$, the simulator $\mathcal{S}$ proceeds as follows for every instance of $\mathcal{F}_{\text{PSMT}}$:

- In the first ("input") round of this $\mathcal{F}_{\text{PSMT}}$ instance, upon receiving an input message $(\texttt{input}, \mathsf{sid}, x_i)$ from $\mathcal{Z}$, where $x_i \neq \perp$ is a vector of messages to be sent by a corrupted P_i, the simulator $\mathcal{S}$ forwards the message to $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$.
- If $\mathcal{S}$ receives a leakage message $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_i, l_i)$ from $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$, where l_i is a length-n vector, consisting of the messages sent by P_i to each corrupted party (and the length of

[53] Recall that proving security with respect to the dummy adversary is sufficient (cf. Claim 5.1.2).

messages P_i sends to honest parties), $\mathcal{S}$ forwards the message to $\mathcal{Z}$. If no leakage message arrived during this round, $\mathcal{S}$ sends the message $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_i, \perp)$ to $\mathcal{Z}$, on behalf of every party.

- In the second ("output") round of this $\mathcal{F}_{\text{PSMT}}$ instance, $\mathcal{S}$ sends $(\texttt{output}, \mathsf{sid}, y_i)$ to $\mathcal{Z}$ for every corrupted P_i, where y_i is a vector consisting of the messages sent to P_i in the "input" round (if some party did not send a message to P_i the value λ is used).

In the last c instances of $\mathcal{F}_{\text{PSMT}}$, the simulator $\mathcal{S}$ does not forward the input messages from $\mathcal{Z}$ to $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$, and outputs $(\texttt{output}, \mathsf{sid}, \lambda)$ for every corrupted party.

Since the view of every party in the protocol π_{PSMT} is simply the messages sent and received to $fpsmt$, and no random coins are used, upon a corruption request of a party P_i, the simulator simply hands the internal state of P_i to $\mathcal{Z}$, and resumes the simulation as above.

By inspection, it can be seen that the view of $\mathcal{Z}$ is identically distributed when interacting with $\mathcal{S}$ in an ideal computation of $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$, or when interacting with the dummy adversary in an execution of π_{PSMT}. □

The corollary follows since $\mathcal{F}_{\text{PSMT}}$ can be realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model in a straightforward way, by calling $\mathcal{F}_{\text{SMT}}$ in parallel n^2 times.

Corollary 7.3.2. *Let $c \geq 0$ and $t < n/3$. The functionality $\mathcal{W}^{D_{\text{PSMT}},c}_{\text{sl-strict}}(\mathcal{F}_{\text{PSMT}})$ can be UC-realized with perfect security in the $\mathcal{F}_{\text{SMT}}$-hybrid model, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

Chapter 8

Applications of Our Fast Composition Theorem

In this section, we demonstrate the power of our framework by providing some concrete applications. All of the protocols we present in this section enjoy perfect security facing adaptive adversaries corrupting less than a third of the parties. We start in Section 8.1 by presenting expected-constant-round protocols for Byzantine agreement. Next, in Section 8.2 we present an expected-constant-round protocol for parallel broadcast. Finally, in Section 8.3 we present a secure function evaluation protocol whose round complexity is $O(d)$ in expectation, where d is the depth of the circuit representing the function.

8.1 Fast and Perfectly Secure Byzantine Agreement

We start by describing the binary and multi-valued randomized Byzantine agreement protocols (the definition of $\mathcal{F}_{\text{BA}}$ appears in Section 6.1). These protocols are based on techniques due to Feldman and Micali [42] (which are in turn based on Ben-Or [10] and Rabin [80]) and Turpin and Coan [82], with modifications to work in the UC framework. We provide simulation-based proofs for these protocols.

A binary Byzantine agreement protocol. We now describe a UC protocol for randomized binary Byzantine agreement, that is based on the protocol of Feldman and Micali [42]. For simplicity, we work in a hybrid model, where parties have access to the *oblivious common coin* functionality; we first present this functionality as a canonical synchronous functionality.

Oblivious common coin. In the *oblivious common coin* ideal functionality (introduced in [42]) every honest party P_i outputs a bit $y_i \in \{0,1\}$ such that the following holds: with probability $p > 0$ all honest parties will agree on a uniformly distributed bit, and with probability $1 - p$ the output for each honest party is determined by the adversary. The meaning of *obliviousness* here is that the parties are unaware of whether agreement on the coin is achieved or not.

In more detail, each honest party P_i sends an empty string $x_i = \lambda$ as input, and the leakage function is $l_{\text{oc}}(x_1, \ldots, x_n) = \perp$. The function to compute, $f_{\text{oc}}(x_1, \ldots, x_n, a) = (y_1, \ldots, y_n)$,

is parametrized by an efficiently sampleable distribution D over $\{0,1\}$, that outputs 1 with probability p and 0 with probability $1-p$, and works as follows:

- Initially, sample a "fairness bit" $b \leftarrow D$.
- If $b = 1$ or if $a = \perp$ (i.e., if the adversary did not send an `adv-input` message) sample a uniformly distributed bit $y \leftarrow \{0,1\}$ and set $y_i \leftarrow y$ for every $i \in [n]$.
- If $b = 0$ and $a \neq \perp$, parse the adversarial input a as a vector of n values $(a_1, \ldots, a_n)$, and set $y_i \leftarrow a_i$ for every $i \in [n]$.

We denote by $\mathcal{F}_{\text{OC}}$ the CSF functionality parametrized with the above functions f_{oc} and l_{oc}. Feldman and Micali [42, Thm. 3] showed a constant-round oblivious common coin protocol for $p = 0.35$. Denote by D_{OC} the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{OC}}}(\mathcal{F}_{\text{OC}})$ and 32 leaves $\mathcal{F}_{\text{PSMT}}$.

Theorem 8.1.1 ([42])**.** *Let $t < n/3$, then, assuming all honest parties receive their inputs at the same round, $\mathcal{W}_{\text{strict}}^{D_{\text{OC}}}(\mathcal{F}_{\text{OC}})$ can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary.*

Overview of the protocol. The binary BA functionality, realized by the protocol, is the wrapped functionality $\mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ (the distribution D_{RBA} is formally defined in Lemma 8.1.2), denoted $\mathcal{F}_{\text{RBA}}$ for short. The protocol π_{RBA}, described in Figure 8.1, is based on the protocol from [42] modified using the "best-of-both-worlds" technique due to Goldreich and Petrank [52]. Recall that following Chapter 7, it is sufficient to describe the protocol using CSFs as hybrids rather than wrapped CSFs (even though such a description might be overly ideal, and cannot be instantiated in the real world), and the same level of security is automatically achieved in a compiled protocol (that can be instantiated) where the underlying CSFs are properly wrapped. Therefore, the protocol is defined in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model.

At first sight, it may seem odd that the binary Byzantine agreement functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ is used in order to implement the randomized binary Byzantine agreement functionality $\mathcal{F}_{\text{RBA}}$. However, the functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ will only be invoked in the event (which occurs with a negligible probability) that the protocol does not terminate within a poly-log number of rounds. Once the protocol is compiled, the CSF functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ will be wrapped using a strict wrapper, such that the wrapped functionality $\mathcal{W}_{\text{strict}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ can be instantiated using any linear-round deterministic Byzantine agreement protocol (e.g., the protocol in [61]).

At a high level, protocol π_{RBA} proceeds as follows. Initially, each party sends its input to all other parties over a point-to-pint channel using $\mathcal{F}_{\text{PSMT}}$, and sets its vote to be its input bit. Next, the parties proceed in phases, where each phase consists of invoking the functionality $\mathcal{F}_{\text{OC}}$ followed by a voting process consisting of three rounds of sending messages via $\mathcal{F}_{\text{PSMT}}$. The voting ensures that (1) if all honest parties agree on their votes at the beginning of the phase, they will terminate at the end of the phase, (2) in each phase, all honest parties will agree on their votes at the end of each phase with probability at least p, and (3) if an honest party terminates in some phase then all honest parties will terminate with the same value by the end of the next phase. In the negligible event that the parties do not terminate after $\tau = \log^{1.5}(\kappa) + 1$ phases, the parties use the Byzantine agreement functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ in order to ensure termination.

To avoid confusion in π_{RBA} between the different calls to $\mathcal{F}_{\text{OC}}$, the α'th invocation will use the session identifier $\mathsf{sid}_\alpha = \mathsf{sid} \circ \alpha$, obtained by concatenating α to sid.

Denote by D_{RBA} the distribution that outputs a depth-1 trace, with root is $\mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, and leaves that are set as follows: initially sample an integer r from the geometric distribution with parameter $p = 0.35$ and support $\{1\ldots,\tau+1\}$ (representing the phase where $\mathcal{F}_{\text{OC}}$ samples a fairness bit 1, plus the option that $\mathcal{F}_{\text{OC}}$ samples 0 in all τ phases). The first leaf in the trace is $\mathcal{F}_{\text{PSMT}}$, followed by $\min(r,\tau)$ sequences of $(\mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}})$. Finally, if $r \geq \tau$ add the leaf $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ to the trace.

Protocol π_{RBA}

Each party $\mathsf{P}_i \in \mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ proceeds as follows:

- Initially, P_i sets the phase counter $\alpha \leftarrow 0$ and the termination indicator $\mathsf{term} \leftarrow 0$. For every other party $\mathsf{P}_j \in \mathcal{P}$ set a value $B_j \leftarrow 0$ for storing the last bit value received from P_j. In addition, denote $\tau = \log^{1.5}(\kappa) + 1$.
- In the first round, upon receiving $(\mathtt{input}, \mathsf{sid}, v)$ with $v \in \{0,1\}$ from the environment, party P_i sets $b_i \leftarrow v$ (note that the value b_i will change during the protocol) and sends (sid, b_i) to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Upon receiving (sid, b_j) from P_j (via $\mathcal{F}_{\text{PSMT}}$) with $b_j \in \{0,1\}$, set $B_j \leftarrow b_j$. If no message was received from P_j, set $b_j \leftarrow B_j$.
- While $\mathsf{term} = 0$ and $\alpha \leq \tau$, do the following:
  1. Set $\alpha \leftarrow \alpha + 1$ and send $(\mathtt{input}, \mathsf{sid}_\alpha, \lambda)$ to $\mathcal{F}_{\text{OC}}$. Let $(\mathtt{output}, \mathsf{sid}_\alpha, \beta)$, with $\beta \in \{0,1\}$, be the output received from $\mathcal{F}_{\text{OC}}$.
  2. Compute $c \leftarrow \sum_{j=1}^{n} b_j$.
     If $c < n/3$ set $b_i \leftarrow 0$; If $n/3 \leq c < 2n/3$ set $b_i \leftarrow \beta$; If $2n/3 \leq c \leq n$ set $b_i \leftarrow 1$.
     Send (sid, b_i) to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Upon receiving (sid, b_j) from P_j (via $\mathcal{F}_{\text{PSMT}}$) with $b_j \in \{0,1\}$, set $B_j \leftarrow b_j$; if no message was received from P_j, set $b_j \leftarrow B_j$.
  3. Compute $c \leftarrow \sum_{j=1}^{n} b_j$.
     If $c < n/3$ set $b_i \leftarrow 0$ and $\mathsf{term} \leftarrow \alpha$; If $n/3 \leq c < 2n/3$ set $b_i \leftarrow 0$; If $2n/3 \leq c \leq n$ set $b_i \leftarrow 1$.
     Send (sid, b_i) to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Upon receiving (sid, b_j) from P_j (via $\mathcal{F}_{\text{PSMT}}$) with $b_j \in \{0,1\}$, set $B_j \leftarrow b_j$; if no message was received from P_j, set $b_j \leftarrow B_j$.
  4. Compute $c \leftarrow \sum_{j=1}^{n} b_j$.
     If $c < n/3$ set $b_i \leftarrow 0$; If $n/3 \leq c < 2n/3$ set $b_i \leftarrow 1$; If $2n/3 \leq c \leq n$ set $b_i \leftarrow 1$ and $\mathsf{term} \leftarrow \alpha$.
     Send (sid, b_i) to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Upon receiving (sid, b_j) from P_j (via $\mathcal{F}_{\text{PSMT}}$) with $b_j \in \{0,1\}$, set $B_j \leftarrow b_j$; if no message was received from P_j, set $b_j \leftarrow B_j$.
- If $0 < \mathsf{term} < \tau$, then output $(\mathtt{output}, \mathsf{sid}, b_i)$ and halt.
- Else (i.e., if $\mathsf{term} = 0$ or $\mathsf{term} = \tau$), send $(\mathtt{input}, \mathsf{sid}, b_i)$ to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ (note that b_i is the value that was set in phase τ). Upon receving $(\mathtt{output}, \mathsf{sid}, b)$, with $b \in \{0,1\}$, if $\mathsf{term} = 0$ output $(\mathtt{output}, \mathsf{sid}, b)$ and halt. Else, if $\mathsf{term} = \tau$, output $(\mathtt{output}, \mathsf{sid}, b_i)$ and halt.

Figure 8.1: The binary randomized BA protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model

Lemma 8.1.2. *Let $t < n/3$, then, assuming all honest parties receive their inputs at the same round, protocol π_{RBA} UC-realizes $\mathcal{F}_{\text{RBA}} = \mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary.*

Proof. We first claim correctness, i.e., that all honest parties output the same value and that if $n-t$ of the inputs are the same, this value will be the common output. The protocol π_{RBA} consists of two parts, the first is running (up to) τ phases of the Feldman-Micali protocol, and the second (which only occurs if there exists an honest party that did not receive output, i.e., has value $\mathsf{term} = 0$, in the first part, or if there exists an honest party that received output in phase τ, i.e., has value $\mathsf{term} = \tau$) consists of calling a BA functionality. As shown in [42, Thm. 4], the Feldman-Micali protocol satisfies the consistency and validity properties in the property-based definition of Byzantine agreement. In addition, if some honest party received output b in some phase α (i.e., if it sets $\mathsf{term} = \alpha$), then the value b_i of every honest party P_i equals b at the end of phase α. It follows that:

- In case $n-t$ honest parties (in particular if all honest parties) start with the same input, they will agree on this value as their output and terminate in the first phase. (In all other cases it remains only to show that all honest parties agree on the output.)
- In case the first honest party received output in phase $\alpha < \tau - 1$, it holds that by phase $\alpha + 1 < \tau$ all honest parties will receive the same output (i.e., $0 < \mathsf{term} < \tau$ for all honest parties), and so correctness follows from [42].
- In case no honest party received output in all τ phases (i.e., $\mathsf{term} = 0$ for all honest parties), all honest parties send their internal values to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ and output the result, hence, correctness follows from the $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ functionality.
- In case all honest parties receive their outputs in phase τ (i.e., $\mathsf{term} = \tau$ for all honest parties), then by [42] they receive the same value. In this case, this is the value they will output after calling $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ and so correctness is satisfied.
- In case some honest parties receive their outputs in phase τ (i.e., $\mathsf{term} = \tau$) and the other honest parties do not (i.e., $\mathsf{term} = 0$), then it holds that all honest parties send the same value to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$, and correctness is satisfied.
- In case some honest parties receive their outputs in phase $\tau - 1$ (i.e., $\mathsf{term} = \tau - 1$), they do not send any input to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$. However, the remaining honest parties will receive the same output in phase τ (i.e., $\mathsf{term} = \tau$), and will output this value, *regardless* of the output they receive from $\mathcal{F}_{\text{BA}}^{\{0,1\}}$. Therefore, correctness is satisfied.

Regarding termination, [42, Claim T4-4] showed that for any positive integer m, if all honest parties agree on the same bit at the beginning of the m'th phase, then they will all terminate at the end of the phase with probability at least p. It follows that in case all honest parties start with the same input value, they will terminate within the first iteration. Otherwise, the probability distribution of terminating in less than $\tau = \log^{1.5}(\kappa) + 1$ phases is geometric with parameter p. In the negligible probability that the parties did not receive output in less than τ phases, termination is guaranteed by $\mathcal{F}_{\text{BA}}^{\{0,1\}}$.

We now prove that π_{RBA} UC-realizes $\mathcal{F}_{\text{RBA}}$. We construct a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$ that simulates the honest parties in π_{RBA} and the ideal functionalities $\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}$ and $\mathcal{F}_{\text{BA}}^{\{0,1\}}$. Let $\mathcal{Z}$ be an environment

- $\mathcal{S}$ forwards all messages from the environment to $\mathcal{A}$ (and vice versa).

- $\mathcal{S}$ simulates every honest party by independently sampling random coins for the party and running the protocol according to the protocol's specification. Note that $\mathcal{S}$ learns the input for each honest party P_i as soon as P_i sends it to $\mathcal{F}_{\text{RBA}}$ by receiving the message $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_i, (x_1, \ldots, x_n))$. In addition, $\mathcal{S}$ learns the trace of the protocol by receiving the message $(\texttt{trace}, \mathsf{sid}, T)$ from $\mathcal{F}_{\text{RBA}}$, and can derive the terminating phase r_{out} by counting the number of sequences $(\mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}})$ in T (and setting $r_{\mathsf{out}} \leftarrow \tau+1$ if the last leaf is $\mathcal{F}_{\text{BA}}^{\{0,1\}}$).

- Whenever $\mathcal{A}$ sends a message (sid, b_j) on behalf of a corrupted party P_j to some honest party during the first round, $\mathcal{S}$ sends $(\texttt{input}, \mathsf{sid}, b_j)$ to $\mathcal{F}_{\text{RBA}}$ on behalf of P_j.

- Whenever $\mathcal{A}$ requests to corrupt some party $\mathsf{P}_i \in \mathcal{P}$, $\mathcal{S}$ corrupts P_i and sends the simulated internal state of P_i (consisting of P_i's input, randomness and incoming messages) to $\mathcal{A}$. Recall that in case $\mathcal{A}$ corrupts a party P_i after it sent its input to some corrupted party, during the first round, $\mathcal{A}$ may instruct P_i to send a different value as its input to all other parties. In this case, $\mathcal{S}$ sends $(\texttt{input}, \mathsf{sid}, b_i)$ to $\mathcal{F}_{\text{RBA}}$ on behalf of P_i.

- When simulating $\mathcal{F}_{\text{OC}}$ in the first $r_{\mathsf{out}} - 1$ phases, instead of sampling the fairness bit, $\mathcal{S}$ acts as if $b = 0$, i.e., it allows $\mathcal{A}$ to decide on the output values of the parties. In case some subset of simulated honest parties $\mathcal{P}'$ terminate in a phase r (prior to phase r_{out}) with value $y \in \{0,1\}$, $\mathcal{S}$ sends $(\texttt{adv-input}, \mathsf{sid}, y)$ to $\mathcal{F}_{\text{RBA}}$ followed by $(\texttt{early-output}, \mathsf{sid}, \mathsf{P}_i)$ for every $\mathsf{P}_i \in \mathcal{P}'$. In addition, $\mathcal{S}$ proceeds based on the following cases:

  - In case $r < \tau$, $\mathcal{S}$ sends $(\texttt{early-output}, \mathsf{sid}, \mathsf{P}_i)$ for every $\mathsf{P}_i \in \mathcal{P} \setminus \mathcal{P}'$ in the next phase, ensuring that all honest parties will terminate appropriately.
  - In case $r = \tau$, then the honest parties in $\mathcal{P} \setminus \mathcal{P}'$ proceed to the invocation of $\mathcal{F}_{\text{BA}}^{\{0,1\}}$, $\mathcal{S}$ simulates all honest parties in $\mathcal{P} \setminus \mathcal{P}'$ sending y as their input and receives input values from the adversary. Next, $\mathcal{S}$ computes the output just like $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ would, and sends to the adversary the output values. (Recall that the output value from $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ is not being used by the honest parties.)
  - Note that the case $r = \tau + 1$ can never happen.

- In case no honest party has terminated prior to phase r_{out}, then $\mathcal{S}$ proceeds as follows:

  - In case $r_{\mathsf{out}} \leq \tau$, $\mathcal{S}$ samples a random bit $y \in \{0,1\}$ in the r_{out}'th phase, sends $(\texttt{adv-input}, \mathsf{sid}, y)$ to $\mathcal{F}_{\text{RBA}}$, and simulates the next invocation of $\mathcal{F}_{\text{OC}}$ by setting the fairness bit $b = 1$ and with output y, i.e., ensuring that the honest parties will receive output y in the simulated protocol. Recall that if $r_{\mathsf{out}} < \tau$ then indeed all honest parties will terminate in the simulated protocol, however, if $r_{\mathsf{out}} = \tau$ the simulator must simulate $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ to $\mathcal{A}$. Note that $\mathcal{A}$ cannot affect the output value in this scenario (as all honest parties participate with input value y); $\mathcal{S}$ simulates all honest parties sending y as their input, and responds with y as the output for all corrupted parties.
  - In case $r_{\mathsf{out}} = \tau + 1$, i.e., in case no party received output in all τ phases, $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ to the adversary. Initially, $\mathcal{S}$ simulates all honest parties sending their local intermediate value as their input to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$, and receives the input values from the adversary on behalf of the corrupted parties. (Recall that the adversary may dynamically corrupt honest parties and change their input message.) Next,

$\mathcal{S}$ computes the result as in $\mathcal{F}_{\text{BA}}^{\{0,1\}}$, i.e., it checks whether there exists at least $n-t$ input values that all equal to some value y, and if so sets it as the output; otherwise, it sets the output based on the $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ message sent by the adversary.

It follows using a standard hybrid argument that the view of the environment $\mathcal{Z}$ is identically distributed when interacting with a real-world execution of π_{RBA} in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model and the dummy adversary, and when interacting with the simulator $\mathcal{S}$ and the ideal model computation of $\mathcal{F}_{\text{RBA}}$, i.e.,

$$\text{REAL}_{\pi_{\text{RBA}},\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F}_{\text{RBA}},\mathcal{S},\mathcal{Z}}.$$

□

We now use Theorem 7.2.1 to derive the main result of this section.

Theorem 8.1.3 (restating Theorem 1.3.1)**.** *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}_{\text{sl-flex}}^{D,c}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

Proof sketch. Denote by D_{BA} the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{BA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ and $t+1$ leaves $\mathcal{F}_{\text{PSMT}}$. Let $D_{\text{RBA}}^{\mathsf{full}} = \mathsf{full\text{-}trace}(D_{\text{RBA}}, D_{\text{OC}}, D_{\text{PSMT}}, D_{\text{BA}})$.

For simplicity, denote the functionalities $\mathcal{F}_{\text{BA}}^{\text{PT}} = \mathcal{W}_{\text{sl-flex}}^{D_{\text{RBA}}^{\mathsf{full}},c}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, $\mathcal{F}_{\text{PSMT}}^{\text{DT}} = \mathcal{W}_{\text{sl-strict}}^{D_{\text{PSMT}},c}(\mathcal{F}_{\text{PSMT}})$, $\mathcal{F}_{\text{OC}}^{\text{DT}} = \mathcal{W}_{\text{sl-strict}}^{D_{\text{OC}},c}(\mathcal{F}_{\text{OC}})$ and $\mathcal{F}_{\text{BA}}^{\text{DT}} = \mathcal{W}_{\text{sl-strict}}^{D_{\text{BA}},c}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$. In addition, denote $D_1 = D_{\text{PSMT}}$, $D_2 = D_{\text{OC}}$, $D_3 = D_{\text{BA}}$ and $I = \emptyset$.

From Lemma 8.1.2, π_{RBA} UC-realizes $\mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs at the same round. Following Theorem 7.2.1, the compiled protocol $\mathsf{Comp}_{\text{PTR}}^{c}(\pi_{\text{RBA}}, D_1, D_2, D_3, I)$ UC-realizes $\mathcal{F}_{\text{BA}}^{\text{PT}}$, in the $(\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{OC}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{DT}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs within $c+1$ consecutive rounds.

The proof follows since each of the functionalities $(\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{OC}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{DT}})$ can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model. This follows from Lemma 7.3.1, Theorem 8.1.1, and the protocol from [61]. □

Multi-valued Byzantine agreement protocol. As presented above, π_{RBA} is a binary BA protocol. Using a transformation due to Turpin and Coan [82], the decision domain can be extended without increasing the expected running time. Given a set $V \subseteq \{0,1\}^*$, denote by $D_{\text{MV-BA}}$ the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V})$ and three leaves $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$.

Lemma 8.1.4. *Let $t < n/3$ and $V \subseteq \{0,1\}^*$. Then, assuming all honest parties receive their inputs at the same round, the protocol $\pi_{\text{MV-BA}}$ UC-realizes $\mathcal{W}_{\text{strict}}^{D_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary.*

The proof of the Lemma is straightforward.

Theorem 8.1.5. *Let $c \geq 0$, $t < n/3$ and $V \subseteq \{0,1\}^*$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}^V_{\text{BA}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

Proof sketch. Let $D^{\mathsf{full}}_{\text{MV-BA}} = \mathsf{full\text{-}trace}(D_{\text{MV-BA}}, D_{\text{PSMT}}, D^{\mathsf{full}}_{\text{RBA}})$. For simplicity, denote $\mathcal{F}^{\text{PT,V}}_{\text{BA}} = \mathcal{W}^{D^{\mathsf{full}}_{\text{MV-BA}},c}_{\text{sl-flex}}(\mathcal{F}^V_{\text{BA}})$, $D_1 = D_{\text{PSMT}}$, $D_2 = D^{\mathsf{full}}_{\text{RBA}}$ and $I = \{2\}$.

From Lemma 8.1.4, $\pi_{\text{MV-BA}}$ UC-realizes $\mathcal{W}^{D_{\text{MV-BA}}}_{\text{strict}}(\mathcal{F}^V_{\text{BA}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}^{\{0,1\}}_{\text{BA}})$-hybrid model, in three rounds, assuming all parties receive their inputs at the same round. Following Theorem 7.2.2, the compiled protocol $\mathsf{Comp}^c_{\text{PT}}(\pi_{\text{MV-BA}}, D_1, D_2, I)$ UC-realizes $\mathcal{F}^{\text{PT,V}}_{\text{BA}}$, in the $(\mathcal{F}^{\text{DT}}_{\text{PSMT}}, \mathcal{F}^{\text{PT}}_{\text{BA}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs within $c+1$ consecutive rounds.

The proof follows since, following Lemma 7.3.1 and Theorem 8.1.3 the functionalities $\mathcal{F}^{\text{DT}}_{\text{PSMT}}$ and $\mathcal{F}^{\text{PT}}_{\text{BA}}$ can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, using expected-constant-round protocols. □

8.2 Fast and Perfectly Secure Parallel Broadcast

As discussed in Chapter 1 (and Section 5.3), composing protocols with probabilistic termination naïvely does not retain expected round complexity. Ben-Or and El-Yaniv [11] constructed an elegant protocol for probabilistic-termination parallel broadcast[54] with a constant round complexity in expectation, albeit under a property-based security definition. In this section, we adapt the [11] protocol to the UC framework and show that it does *not* realize the parallel broadcast functionality, but rather a weaker variant which we call *unfair parallel broadcast*. Next, we show how to use unfair parallel broadcast in order to compute (fair) parallel broadcast in constant excepted number of rounds.

8.2.1 Unfair Parallel Broadcast

In a standard broadcast functionality (cf. Section 6.1), the sender provides a message to the functionality which delivers it to the parties. Hirt and Zikas [61] defined the *unfair* version of the broadcast functionality, in which the functionality informs the adversary which message it received, and allows the adversary, based on this information, to corrupt the sender and replace the message. Following the spirit of [61], we now define the unfair parallel broadcast functionality, using the language of CSF.

– Unfair Parallel Broadcast. In the *unfair parallel broadcast* functionality, each party P_i with input x_i distributes its input to all the parties. The adversary is allowed to learn the content of each input value from the leakage function (and so it can corrupt parties and change their messages prior to their distribution, based on this information). The function to compute is $f_{\text{UPBC}}(x_1, \ldots, x_n, a) = ((x_1, \ldots, x_n), \ldots, (x_1, \ldots, x_n))$, and the leakage

[54] In [11] the problem is referred to as "interactive consistency."

function is $l_{\text{UPBC}}(x_1, \ldots, x_n) = (x_1, \ldots, x_n)$. We denote by $\mathcal{F}_{\text{UPBC}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f_{UPBC} and l_{UPBC}.

8.2.1.1 The Unfair Parallel Broadcast Protocol

In this section, we adjust the interactive-consistency protocol of Ben-Or and El-Yaniv [11] (with minor adjustments) to the UC framework. The protocol π_{UPBC} (see Figure 8.2 for a detailed description) is parametrized by two integers d and m. Initially, each party distributes its input to all other parties. The underlying idea of the protocol is to run $n \cdot m$ instances of the BA protocol π_{RBA} in parallel, such that for each P_i, a class of m instances of π_{RBA} are executed on the input of P_i. However, in order to avoid the blowup in the number of rounds, the parallel execution of the protocols is truncated after d phases. Once the first step concludes, each party checks for each of the n classes if it received output in at least one of the executions. If so, it arbitrarily selects one output for each class and distributes the vector of output values to all the parties.

Next, the parties run a leader-election protocol and once some party P_k is elected to be the leader, all parties run a BA protocol on the output vector that was distributed by the leader P_k earlier (which might be null). Each party checks whether the agreed output corresponds to the output values it received in the first step and sets a termination indicator accordingly. Finally, the parties run another BA protocol on the termination indicators and terminate in case the output is 1; otherwise another iteration is executed.

Ben-Or and El-Yaniv [11] showed that consistency and validity properties are satisfied, and furthermore, if $m = \log(n)$ and d is such that at least 5 phases of the truncated randomized BA protocol are executed, then the protocol will terminate in a constant expected number of rounds.

We analyze this protocol in a hybrid model, where parties have access to a leader-election functionality $\mathcal{F}_{\text{LE}}$ and a Byzantine agreement functionality $\mathcal{F}_{\text{BA}}$. We actually require two types of BA functionalities, the first is a standard BA functionality, whereas the second is a "truncated" BA, which runs for a specific number of rounds and halts even if no output is specified. We now describe these ideal functionalities as CSFs.

Leader election. In the *leader-election* functionality, the parties agree on a random value $k \in_R [n]$. This functionality can be cast as a special case of secure function evaluation (as defined in Section 6.1), where the parties compute the function $g_{\mathsf{le}}(\lambda, \ldots, \lambda) = (k, \ldots, k)$. We denote by $\mathcal{F}_{\text{LE}}$ the functionality $\mathcal{F}_{\text{SFE}}^{g_{\mathsf{le}}}$.

Ben-Or and El-Yaniv [11] showed how to implement the leader-election functionality by first using the oblivious common coin protocol from [42] to compute an *oblivious leader election*, and next run a (multi-valued) Byzantine agreement protocol on the result. The oblivious leader election functionality $\mathcal{F}_{\text{OLE}}$ is defined in a similar way to the oblivious common coin functionality ($\mathcal{F}_{\text{OC}}$, Section 8.1), with the exception that output value y is not a bit, but a uniformly distributed element in $[n]$. Denote by D_{LE} the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{LE}}}(\mathcal{F}_{\text{LE}})$ and 2 leaves $(\mathcal{F}_{\text{OLE}}, \mathcal{F}_{\text{BA}}^{[n]})$.

Lemma 8.2.1 ([11])**.** *Let $t < n/3$. Then, assuming all honest parties receive their inputs at the same round, $\mathcal{W}_{\text{strict}}^{D_{\text{LE}}}(\mathcal{F}_{\text{LE}})$, in the $(\mathcal{F}_{\text{OLE}}, \mathcal{F}_{\text{BA}}^{[n]})$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary.*

Truncated Byzantine agreement. The *truncated Byzantine agreement* functionality, is a CSF whose function is parametrized by a set V, an efficiently sampleable distribution D, and a non-negative integer d. Each party P_i has input $x_i \in V$, and receives two output values (y_1^i, y_2^i). The adversary is allowed to learn all the input values as the honest parties send them, i.e., the leakage function is $l_{\text{T-RBA}}(x_1, \ldots, x_n) = (x_1, \ldots, x_n)$. The function to compute is $f_{\text{T-RBA}}(x_1, \ldots, x_n, a) = ((y_1^i, y_2^i), \ldots, (y_1^i, y_2^i))$, which operates as follows:

- If there exists a value y such that $y = x_i$ for at least $n - t$ input values x_i, then set $(y_1^i, y_2^i) \leftarrow (y, \perp)$ for every $i \in [n]$.
- Else, sample a number $r \leftarrow D$. The adversarial input a is parsed as a vector of $n + 1$ integer values $(a_0, a_1 \ldots, a_n)$. The first coordinate a_0 represents the output value, i.e., set $y \leftarrow a_0$. Next, for each party P_i, set a value $d_i \leftarrow \min(a_i, r)$. Finally, the output values for each party P_i is defined as follows:
  - If $d_i < d$ then set $(y_1^i, y_2^i) \leftarrow (y, \perp)$.
  - If $d_i = d$ then set $(y_1^i, y_2^i) \leftarrow (\perp, y)$.
  - If $d_i > d$ then set $(y_1^i, y_2^i) \leftarrow (\perp, \perp)$.

In fact, in the protocol π_{UPBC}, a parallel version (of s instances, for some s) of the above described functionality is required. That is, each party P_i has a vector of input values $\boldsymbol{x}_i = (x_1^i, \ldots, x_s^i)$, and receives a vector of s output values $(y_1^i, \ldots, y_s^i)$ where each y_j^i is a pair of values as above. The leakage function reveals all the input values to the adversary, and the function to compute is essentially s instances of the above function f, where for each instance the value r is sampled from D using independent random coins. In addition, the adversarial input a is parsed as a vector of $s(n + 1)$ integer values, where for each instance, the adversary specifies a different vector $(a_0, a_1 \ldots, a_n)$. Note, however, that the value d is the same in all s instances.

We denote by $\mathcal{F}_{\text{T-RBA}}$ the functionality $\mathcal{F}_{\text{CSF}}$ describing the parallel version of truncated randomized BA, as described above.

The protocol. We first describe a version of the protocol by [11] augmented with (a simpler version of) the technique from [52], where all hybrids used are CSFs;[55] using Theorem 7.2.1 we then obtain our result. Recall that the unfair parallel broadcast functionality $\mathcal{F}_{\text{UPBC}}$ is defined in Section 8.2.

Let $d \geq 5$, let $m = \log(n)$, and let $\tau = \log^{1.5}(\kappa) + 1$. Denote by D the geometric distribution with parameter $2q/3$ and support $\{1 \ldots, \tau + 1\}$, where q is the probability that when independently sampling nm "terminating phases" $(r_1, \ldots, r_{nm})$ from the distribution D_{RBA}, then for every $j \in [n]$ it holds that at least one of the values $(r_{(j-1)m+1}, \ldots, r_{(j-1)m+m})$ is smaller than d. (The distribution D outputs the phase in which the event where $\mathcal{F}_{\text{LE}}$ returned a party that was honest before the $\mathcal{F}_{\text{LE}}$ invocation and received output in each BA occurs, plus the option that this event did not occur in all τ phases.)

Denote by D_{UPBC} the distribution that outputs a depth-1 trace with a root $\mathcal{W}_{\text{flex}}^{D_{\text{UPBC}}}(\mathcal{F}_{\text{UPBC}})$ and where the leaves are set as follows: initially sample an integer $r \leftarrow D$. The first leaf is $\mathcal{F}_{\text{PSMT}}$, followed by $\min(r, \tau)$ sequences of $(\mathcal{F}_{\text{T-RBA}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{BA}})$. Finally, if $r = \tau + 1$ add the leaf $\mathcal{F}_{\text{UPBC}}$.

[55] Note that although the hybrids are CSFs, and all honest parties terminate at the same round, the protocol has probabilistic termination.

Lemma 8.2.2. *Let $t < n/3$, and assume that all honest parties receive their inputs at the same round. Then, the protocol π_{UPBC} UC-realizes $\mathcal{F}_{\text{PT-UPBC}} = \mathcal{W}_{\text{flex}}^{D_{\text{UPBC}}}(\mathcal{F}_{\text{UPBC}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{T-RBA}}, \mathcal{F}_{\text{UPBC}})$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary.*

Protocol π_{UPBC}

Protocol π_{UPBC}, parametrized by positive integers d (number of phases to run the truncated BA functionality) and m (how many instances of truncated BA to compute for each input value). The functionality $\mathcal{F}_{\text{T-RBA}}$ runs nm instances in parallel, and is parametrized by the distribution D_{RBA} and integer d.

1. Initially, P_i sets the phase index $\alpha \leftarrow 0$, and the termination indicator $\mathsf{term} \leftarrow 0$. In addition, denote $\tau = \log^{1.5}(\kappa) + 1$.
2. In the first round, upon receiving $(\mathtt{input}, \mathsf{sid}, x_i)$ with $x_i \in V$ from the environment, P_i sends (sid, x_i) to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Denote by x_j the value received from P_j.
3. While $\mathsf{term} = 0$ and $\alpha \leq \tau$, do the following:
   (a) Set $\alpha \leftarrow \alpha+1$ and send values to $\mathcal{F}_{\text{T-RBA}}$, such that the value x_j is sent to the m instances corresponding to the j'th value. Formally, prepare the vector $\boldsymbol{z} = (z_1, \ldots, z_{nm})$ such that for every $j \in [n]$ and every $l \in [m]$ set $z_{(j-1)m+l} = x_j$. Send $(\mathtt{input}, \mathsf{sid}_1^\alpha, \boldsymbol{z})$ to $\mathcal{F}_{\text{T-RBA}}$.
   Let $(\mathtt{output}, \mathsf{sid}_1^\alpha, \boldsymbol{v})$ be the output from $\mathcal{F}_{\text{T-RBA}}$, where $\boldsymbol{v}$ is a vector of nm pairs $((v_1^1, v_2^1), \ldots, (v_1^{nm}, v_2^{nm}))$ with $v_1^j, v_2^j \in V \cup \{\perp\}$.
   (b) For every $j \in [n]$, set $S_1^j \leftarrow \{v_1^{(j-1)m+1}, \ldots, v_1^{jm}\}$ (corresponding to output values before phase d) and $S_2^j \leftarrow \{v_2^{(j-1)m+1}, \ldots, v_2^{jm}\}$ (corresponding to output values at phase d).
   (c) If $S_1^j \neq \{\perp\}$ for every $j \in [n]$ (i.e., if for every class of BAs there was at least one output), then for every $j \in [n]$ choose $c_j \in S_1^j$ (arbitrarily), set $\boldsymbol{c}_i = (c_1, \ldots, c_n)$ and send $(\mathsf{sid}, \boldsymbol{c}_i)$ to all the parties (via $\mathcal{F}_{\text{PSMT}}$).
   Denote by $\boldsymbol{c}_j$ the tuple received from P_j; if no message was received, set $\boldsymbol{c}_j = \emptyset$.
   (d) Send $(\mathtt{input}, \mathsf{sid}_2^\alpha, \lambda)$ to the functionality $\mathcal{F}_{\text{LE}}$. Let $(\mathtt{output}, \mathsf{sid}_2^\alpha, k)$, with $k \in [n]$, be the output received from $\mathcal{F}_{\text{LE}}$.
   (e) Send $(\mathtt{input}, \mathsf{sid}_3^\alpha, \boldsymbol{c}_k)$ to $\mathcal{F}_{\text{BA}}$, parametrized by the set $V^n \cup \{\emptyset\}$. Let $(\mathtt{output}, \mathsf{sid}_3^\alpha, \boldsymbol{c})$ be the output received from $\mathcal{F}_{\text{BA}}$ (with $\boldsymbol{c} = (c_1, \ldots, c_n) \in V^n$ or $\boldsymbol{c} = \emptyset$).
   (f) If $\boldsymbol{c} \neq \emptyset$ and for every $j \in [n]$, $c_j \in S_1^j \cup S_2^j$ then set $b \leftarrow 1$; otherwise set $b \leftarrow 0$.
   (g) Send $(\mathtt{input}, \mathsf{sid}_4^\alpha, b)$ to $\mathcal{F}_{\text{BA}}$, parametrized by the set $\{0, 1\}$. Let $(\mathtt{output}, \mathsf{sid}_4^\alpha, \beta)$, with $\beta \in \{0, 1\}$, be the output received from $\mathcal{F}_{\text{BA}}$. If $\beta = 1$ then set $\mathsf{term} \leftarrow 1$.
4. If $\mathsf{term} = 1$, then output $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ and halt.
5. Else, set the vector $\boldsymbol{x}_i = (\lambda, \ldots, \lambda, x_i, \lambda, \ldots, \lambda)$ (the vector of length n whose ith coordinate is x_i and all other cooridinates are the empty string λ) and send $(\mathtt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ to $\mathcal{F}_{\text{UPBC}}$. Let $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ be the output received from $\mathcal{F}_{\text{UPBC}}$. Output $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ and halt.

Figure 8.2: The unfair parallel broadcast protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{T-RBA}}, \mathcal{F}_{\text{UPBC}})$-hybrid model

Proof. We first claim correctness. The protocol π_{UPBC} consists of two parts, the first is running (up to) τ phases of the Ben-Or and El-Yaniv [11] protocol, and the second (which only occurs if

no output was generated in the first part, i.e., if all honest parties have value $\mathsf{term} = 0$) consists of calling an unfair parallel broadcast functionality. As shown in [11, Thm. 5], the protocol satisfies the consistency and validity properties in the property-based definition of interactive consistency (i.e., parallel Byzantine agreement). In addition, since the last step in each phase is invoking the BA functionality in order to agree whether all honest parties received output and can safely terminate, or whether an additional phase should be executed, it follows that if one honest party has received output in some phase, then so do the rest of the honest parties.

It follows that:

- In case some honest party received output in phase $\alpha \leq \tau$, then all honest parties also receive the same output at this phase (i.e., $\mathsf{term} = 1$ for all honest parties), and so correctness follows from [11].

- In case no honest party received output in all τ phases (i.e., $\mathsf{term} = 0$ for all honest parties), all honest parties send their initial values to $\mathcal{F}_{\text{UPBC}}$ and output the result; hence, correctness follows from the $\mathcal{F}_{\text{UPBC}}$ functionality.

Regarding termination, Ben-Or and El-Yaniv showed that for $d \geq 5$ and $m = \log(n)$, all honest parties receive their outputs within a constant number of phases in expectation. In the negligible probability that the parties did not receive output in less than τ phases, termination is guaranteed by $\mathcal{F}_{\text{UPBC}}$.

We now prove that π_{UPBC} UC-realizes $\mathcal{F}_{\text{PT-UPBC}}$. We construct a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$ that simulates the honest parties in π_{UPBC} and the ideal functionalities $\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{T-PRBA}}$ and $\mathcal{F}_{\text{UPBC}}$, as follows. Let $\mathcal{Z}$ be an environment.

- $\mathcal{S}$ forwards all messages from the environment to $\mathcal{A}$ (and vice versa).

- $\mathcal{S}$ simulates every honest party by independently sampling random coins for the party and running the protocol according to the protocol's specification. Note that $\mathcal{S}$ learns the input for each honest party P_i as soon as P_i sends it to $\mathcal{F}_{\text{PT-UPBC}}$ by receiving the message $(\texttt{leakage}, \mathsf{sid}, \mathsf{P}_i, (x_1, \ldots, x_n))$. In addition, $\mathcal{S}$ learns the trace of the protocol by receiving the message $(\texttt{trace}, \mathsf{sid}, T)$ from $\mathcal{F}_{\text{PT-UPBC}}$, and can derive the guaranteed-termination phase r_{out} by counting the number of sequences $(\mathcal{F}_{\text{T-PRBA}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{BA}})$ in T (and setting $r_{\mathsf{out}} \leftarrow \tau + 1$ if the last CSF is $\mathcal{F}_{\text{UPBC}}$).

- Whenever $\mathcal{A}$ sends a message (sid, x_j) on behalf of a corrupted party P_j to some honest party during the first round, $\mathcal{S}$ sends $(\texttt{input}, \mathsf{sid}, x_j)$ to $\mathcal{F}_{\text{PT-UPBC}}$ on behalf of P_j. (Note that x_j is in fact a vector.)

- Whenever $\mathcal{A}$ requests to corrupt some $\mathsf{P}_i \in \mathcal{P}$, the simulator $\mathcal{S}$ corrupts P_i and sends the simulated internal state of P_i (consisting of P_i's input, randomness and incoming messages) to $\mathcal{A}$. Recall that in case $\mathcal{A}$ corrupts a party P_i after it sent its input to some corrupted party, during the first round, $\mathcal{A}$ may instruct P_i to send a different value x_i as its input to all other parties. In this case, $\mathcal{S}$ sends $(\texttt{input}, \mathsf{sid}, x_i)$ to $\mathcal{F}_{\text{PT-UPBC}}$ on behalf of P_i.

- In the first $r_{\mathsf{out}} - 1$ phases, $\mathcal{S}$ simulates $\mathcal{F}_{\text{T-RBA}}$ according to the behavior of the ideal functionality, i.e., by independently sampling nm values from D_{RBA}. Next, when simulating

the functionality $\mathcal{F}_{\text{LE}}$, instead of sampling a random index $k \in [n]$, the simulator $\mathcal{S}$ samples k such that in case $\mathcal{F}_{\text{T-RBA}}$ was successful (i.e., if the honest parties received output) k is uniformly distributed conditioned on P_k is corrupted, i.e., $\mathcal{S}$ allows $\mathcal{A}$ to decide whether the protocol will successfully terminate or not in this phase. In case $\mathcal{A}$ instructs P_k to follow the protocol, then all honest parties will terminate in this phase (prior to phase r_{out}) with value $\boldsymbol{c}$; $\mathcal{S}$ sends $(\texttt{adv-input}, \mathsf{sid}, \boldsymbol{c})$ to $\mathcal{F}_{\text{PT-UPBC}}$ followed by $(\texttt{early-output}, \mathsf{sid}, \mathsf{P}_i)$ for every $\mathsf{P}_i \in \mathcal{P}$.

- In case no honest party has terminated prior to phase r_{out}, then $\mathcal{S}$ proceeds as follows:
  - In case $r_{\mathsf{out}} \leq \tau$, when simulating $\mathcal{F}_{\text{T-RBA}}$ in the r_{out}'th phase, $\mathcal{S}$ ensures that honest parties will receive output, and when simulating $\mathcal{F}_{\text{LE}}$, $\mathcal{S}$ uniformly selects an index k such that P_k was honest before the simulation of $\mathcal{F}_{\text{LE}}$. Next, $\mathcal{S}$ sends $(\texttt{adv-input}, \mathsf{sid}, \boldsymbol{c}_k)$ to $\mathcal{F}_{\text{PT-UPBC}}$, and continues simulating the protocol. Since P_k was honest when distributing $\boldsymbol{c}_k$, this ensures that the honest parties will receive output $\boldsymbol{c}_k$ in the simulated protocol.
  - In case $r_{\mathsf{out}} = \tau + 1$, i.e., in case no party received output in all τ phases, $\mathcal{S}$ simulates the functionality $\mathcal{F}_{\text{UPBC}}$ to the adversary. Initially, $\mathcal{S}$ simulates all honest parties sending their initial inputs as their input to $\mathcal{F}_{\text{UPBC}}$, and receives the input values from the adversary on behalf of the corrupted parties. (Recall that the adversary may dynamically corrupt honest parties and change their input message.) Next, $\mathcal{S}$ computes the result as in $\mathcal{F}_{\text{UPBC}}$, i.e., it provides the output $(x_1, \ldots, x_n)$ to each party.

It follows using a standard hybrid argument that the view of the environment $\mathcal{Z}$ is identically distributed when interacting with a real-world execution of π_{UPBC} and the dummy adversary, and when interacting with the simulator $\mathcal{S}$ and the ideal model computation of $\mathcal{F}_{\text{PT-UPBC}}$, i.e.,

$$\text{REAL}_{\pi_{\text{UPBC}},\mathcal{A},\mathcal{Z}} \equiv \text{IDEAL}_{\mathcal{F}_{\text{PT-UPBC}},\mathcal{S},\mathcal{Z}}.$$

□

Using Theorem 7.2.1 we obtain the following as a result.

Theorem 8.2.3. *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{UPBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

Proof sketch. Denote by $D_{\text{T-RBA}}$ the deterministic distribution that outputs a trace consisting of a root $\mathcal{W}^{D_{\text{T-RBA}}}_{\text{strict}}(\mathcal{F}_{\text{T-RBA}})$ and a constant number of leaves $\mathcal{F}_{\text{PSMT}}$ (corresponding to d phases of π_{RBA}). Denote by $D_{\text{DT-UPBC}}$ the deterministic distribution that outputs a trace consisting of a root $\mathcal{W}^{D_{\text{DT-UPBC}}}_{\text{strict}}(\mathcal{F}_{\text{UPBC}})$ and $t+1$ leaves $\mathcal{F}_{\text{PSMT}}$. Denote by D_{OLE} the deterministic distribution that outputs a trace consisting of a root $\mathcal{W}^{D_{\text{OLE}}}_{\text{strict}}(\mathcal{F}_{\text{OLE}})$ and 32 leaves $\mathcal{F}_{\text{PSMT}}$.

Let $D^{\mathsf{full}}_{\text{LE}} = \mathsf{full\text{-}trace}(D_{\text{LE}}, D_{\text{PSMT}}, D^{\mathsf{full}}_{\text{MV-BA}})$. For simplicity, denote the functionalities $\mathcal{F}^{\text{PT}}_{\text{UPBC}} = \mathcal{W}^{D^{\mathsf{full}}_{\text{UPBC}},c}_{\text{sl-flex}}(\mathcal{F}_{\text{UPBC}})$, $\mathcal{F}^{\text{DT}}_{\text{T-RBA}} = \mathcal{W}^{D_{\text{T-RBA}},c}_{\text{sl-strict}}(\mathcal{F}_{\text{T-RBA}})$, $\mathcal{F}^{\text{PT}}_{\text{LE}} = \mathcal{W}^{D^{\mathsf{full}}_{\text{LE}},c}_{\text{sl-flex}}(\mathcal{F}_{\text{LE}})$, $\mathcal{F}^{\text{DT}}_{\text{UPBC}} = \mathcal{W}^{D_{\text{DT-UPBC}},c}_{\text{sl-strict}}(\mathcal{F}_{\text{UPBC}})$. In addition, denote $D_1 = D_{\text{PSMT}}$, $D_2 = D^{\mathsf{full}}_{\text{RBA}}$, $D_3 = D_{\text{LE}}$, $D_4 = D_{\text{T-RBA}}$, $D_5 = D_{\text{DT-UPBC}}$ and $I = \{2, 3\}$.

Following Lemma 8.2.2, protocol π_{UPBC} UC-realizes the functionality $\mathcal{W}_{\text{flex}}^{D_{\text{UPBC}}}(\mathcal{F}_{\text{UPBC}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{T-RBA}}, \mathcal{F}_{\text{UPBC}})$–hybrid model, using an expected constant number of rounds, assuming that all the parties receive their inputs at the same round. By applying Theorem 7.2.1, the compiled protocol $\mathsf{Comp}_{\text{PTR}}^{c}(\pi_{\text{UPBC}}, D_1, D_2, D_3, D_4, D_5, I)$ UC-realizes $\mathcal{F}_{\text{UPBC}}^{\text{PT}}$, in the $(\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{PT}}, \mathcal{F}_{\text{LE}}^{\text{PT}}, \mathcal{F}_{\text{T-RBA}}^{\text{DT}}, \mathcal{F}_{\text{UPBC}}^{\text{DT}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs within $c+1$ consecutive rounds.

The proof follows since each of the functionalities $\{\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{PT}}, \mathcal{F}_{\text{LE}}^{\text{PT}}, \mathcal{F}_{\text{T-RBA}}^{\text{DT}}, \mathcal{F}_{\text{UPBC}}^{\text{DT}}\}$ can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model with expected constant round complexity. □

8.2.2 Parallel Broadcast

We now turn to define the (fair) parallel broadcast functionality.

- Parallel Broadcast. In the *parallel broadcast* functionality, each party P_i with input x_i distributes its input to all the parties. Unlike the unfair version, the adversary only learns the length of the honest parties' messages before their distribution, i.e., the leakage function is $l_{\text{PBC}}(x_1, \ldots, x_n) = (|x_1|, \ldots, |x_n|)$. It follows that the adversary cannot use the leaked information in a meaningful way when deciding which parties to corrupt. The function to compute is identical to the unfair version, i.e., $f_{\text{PBC}}(x_1, \ldots, x_n, a) = ((x_1, \ldots, x_n), \ldots, (x_1, \ldots, x_n))$. We denote by $\mathcal{F}_{\text{PBC}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f_{PBC} and l_{PBC}.

Unfortunately, the unfair parallel broadcast protocol π_{UPBC} (cf. Figure 8.2) fails to realize (a wrapped version of) the standard parallel broadcast functionality $\mathcal{F}_{\text{PBC}}$. The reason is similar to the argument presented in [61]: in the first round of the protocol, each party distributes its input, and since we consider a rushing adversary, the adversary learns the messages *before* the honest parties do. It follows that the adversary can corrupt a party *before* the honest parties receive the message and replace the message to be delivered. This attack cannot be simulated in the ideal world where the parties interact with $\mathcal{F}_{\text{PBC}}$, since by the time the simulator learns the broadcast message in the ideal world, the functionality does not allow to change it.

Although protocol π_{UPBC} does not realize $\mathcal{F}_{\text{PBC}}$, it can be used in order to construct a protocol that does. Each party commits to its input value before any party learns any new information, as follows. Each party, in parallel, first secret shares its input using a $(t+1)$-out-of-n secret-sharing protocol.[56] In the second step, every party, in parallel, broadcast a vector with all the shares he received, by the use of the above unfair parallel broadcast functionality $\mathcal{F}_{\text{UPBC}}$, and each share is reconstructed based on the announced values. The reason this modification achieves fair broadcast is the following: If a sender P_i is not corrupted until he distributes his shares, then a t-adversary has no way of modifying the reconstructed output of P_i's input, since he can at most affect $t < n/3$ shares, which can be self-correct by the reconstruction algorithm (e.g., using Shamir's scheme). Thus, the only way the adversary can affect any of the broadcast messages is by corrupting the sender independently of his input, an attack which is easily simulated. In case a malicious sender generates shares that do not correspond to a degree t polynomial, all honest parties identify the misbehavior (since all shares are publicly transmitted over the unfair broadcast channel) and agree on a default value for the sender, e.g., zero. We describe this protocol, denoted π_{PBC}, in Figure 8.3.

[56] In [61] verifiable secret sharing (VSS) is used; however, as we argue, this is not necessary.

Protocol π_{PBC}

1. In the first round, upon receiving $(\texttt{input}, \mathsf{sid}, x_i)$ with $x_i \in V$ from the environment, P_i secret shares x_i using Shamir's $(t+1)$-out-of-n secret-sharing scheme, denoted by $(x_i^1, \ldots, x_i^n) \leftarrow \mathsf{share}(x_i)$. Next, P_i sends for every party P_j its share (sid, x_i^j) (via $\mathcal{F}_{\text{PSMT}}$). Denote by x_j^i the value received from P_j (replace invalid/missing values by zero).
2. In the second round, P_i broadcasts the values $\boldsymbol{x}_i = (x_1^i, \ldots, x_n^i)$ using the unfair parallel broadcast functionality, i.e., P_i sends $(\texttt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ to $\mathcal{F}_{\text{UPBC}}$. Denote by $\boldsymbol{y}_j = (y_1^j, \ldots, y_n^j)$ the value received from P_j (replace invalid/missing values by zero). Now, P_i reconstructs all the input values, i.e., for every $j \in [n]$ reconstructs $y_j = \mathsf{recon}(y_j^1, \ldots, y_j^n)$ (in case $y_j = \perp$ set $y_j \leftarrow 0$), and outputs $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_n))$.

Figure 8.3: The parallel broadcast protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{UPBC}})$-hybrid model

Theorem 8.2.4 (restating Theorem 1.3.2)**.** *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}_{\text{sl-flex}}^{D,c}(\mathcal{F}_{\text{PBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

Proof sketch. The simulator uses the adversary attacking π_{PBC} in a black-box straight-line manner. To simulate the first (secret-sharing) round, for honest senders the simulator simply hands the adversary random shares for all corrupted parties and for corrupted senders he follows the adversary's instructions. If during this step the adversary asks to corrupt new senders, the simulator learns their outputs and can easily complete the sharing to match this output. At the end of this phase, the simulator interacts with its hybrid until it produces output. Once this is the case, he uses this output to continue the simulation with its adversary. Clearly, for any sender P_i who is not corrupted until he distributes his shares, then a t-adversary has no way of modifying the reconstructed output of P_i's input, since he can at most affect $t < n/3$ shares (and Shamir's scheme can correct up to $n/3$ erroneous shares). Thus, the only way the adversary can affect any of the broadcasted message is by corrupting the sender independently of his input, an attack which is easily simulated. The fact that the running time is constant (expected) follows trivially from the fact that π_{PBC} executes only one round (namely the sharing round) more than the unfair protocol which is expected constant round (cf. Theorem 8.2.3). □

8.3 Fast and Perfectly Secure SFE

We conclude this section by showing how to construct a perfectly UC-secure SFE protocol which computes a given circuit in expected $O(d)$ rounds, independently of the number of parties, in the point-to-point channels model. The protocol is obtained by taking the protocol from [12],[57] denoted π_{BGW}. This protocol relies on (parallel) broadcast and (parallel) point-to-point channels, and therefore it can be described in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PBC}})$-hybrid model. It follows from Theorem 7.2.2, that the compiled protocol $\mathsf{Comp}_{\text{PT}}^{c}(\pi_{\text{BGW}}, D_1, D_2, I)$, for $D_1 = D_{\text{PSMT}}$, $D_2 = D_{\text{PBC}}^{\mathsf{full}}$

[57] A full simulation proof of the protocol with a black-box straight-line simulation was recently given by [3] and [38].

and $I = \{2\}$, UC-realizes the corresponding wrapped functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{SFE}})$ (for an appropriate distribution D), in the $(\mathcal{W}^{D_{\text{PSMT}},c}_{\text{sl-strict}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}^{D^{\text{full}}_{\text{PBC}},c}_{\text{sl-flex}}(\mathcal{F}_{\text{PBC}}))$-hybrid model, resulting in the following.

Theorem 8.3.1 (restating Theorem 1.3.3)**.** *Let f be an n-party function, C an arithmetic circuit with multiplicative depth d computing f, $c \geq 0$ and $t < n/3$. Then there exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}^{f}_{\text{SFE}})$ has round complexity $O(d)$ in expectation, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

Appendix A

Abstracts of Additional Results

In this appendix, we present abstracts of additional and followup work completed/initiated by the author during graduate studies at Bar-Ilan University.

Round-Preserving Parallel Composition of Probabilistic-Termination Cryptographic Protocols [33]. In Part II, we devised a framework for universal composition of PT protocols, and provided the first composable parallel-broadcast protocol with a simulation-based proof. This construction crucially relies on the fact that broadcast is "privacy free," and does not generalize to arbitrary protocols in a straightforward way. This raises the question of whether it is possible to execute arbitrary PT protocols in parallel, without increasing the round complexity.

In this paper, we tackle this question and provide both feasibility and infeasibility results. We construct a round-preserving protocol compiler, secure against a dishonest minority of actively corrupted parties, that compiles arbitrary protocols into a protocol realizing their parallel composition, while having a black-box access to the underlying *protocols*. Furthermore, we prove that the same cannot be achieved, using known techniques, given only black-box access to the *functionalities* realized by the protocols, unless merely security against semi-honest corruptions is required, for which case we provide a protocol.

Joint work with Sandro Coretti, Juan Garay, and Vassilis Zikas.

From Fairness to Full Security in Multiparty Computation [34]. In Theorem 3.3.4, we showed how to strengthen every fair protocol in the broadcast model to guarantee output delivery. This transformation, however, requires t invocations of the fair protocol.

In this paper, we present highly efficient transformations, assuming the fraction of honest parties is constant (e.g., 1% of the parties are honest). Compared to Theorem 3.3.4 that requires linear invocations (in the number of parties) of the fair computation, the new transformations require super-logarithmic, and sometimes even super-constant, such invocations. The main idea is to delegate the computation to chosen random committees that invoke the fair computation. Apart from the benefit of uplifting security, the reduction in the number of parties is also useful, since only committee members are required to work, whereas the remaining parties simply "listen" to the computation over a broadcast channel.

One application of these transformations is a new δ-bias coin-flipping protocol, whose round complexity has a super-logarithmic dependency on the number of parties, improving over the

protocol of Beimel, Omri, and Orlov [9] that has a linear dependency. A second application is a new fully secure protocol for computing the Boolean OR function, with a super-constant round complexity, improving over the protocol of Gordon and Katz [56] whose round complexity is linear in the number of parties.

Joint work with Iftach Haitner, Eran Omri, and Lior Rotem.

Asynchronous Secure Multiparty Computation in Constant Time [28]. In this dissertation, we focused on MPC over synchronous networks. It is well known that if the communication model is asynchronous, meaning that messages can be arbitrarily delayed by an unbounded (yet finite) amount of time, secure computation with guaranteed termination is feasible if and only if at least two-thirds of the parties are honest, as was shown by Ben-Or, Canetti, and Goldreich [13] and by Ben-Or, Kelmer, and Rabin [14]. The running time and communication complexity of all currently known protocols depend on the function to evaluate. In this work, we present the first asynchronous MPC protocol that runs in constant time and with communication complexity that is independent of the function.

Our starting point is the asynchronous MPC protocol of Hirt, Nielsen, and Przydatek [62, 63]. We integrate *threshold fully homomorphic encryption* in order to reduce the communication between the parties, thus completely removing the need for the expensive *king-slaves* approach taken by Hirt et al.. Initially, assuming an honest majority, we construct a constant-time protocol in the asynchronous Byzantine agreement (ABA) hybrid model. Using a concurrent ABA protocol that runs in constant expected time, we obtain a constant expected time asynchronous MPC protocol, secure facing static malicious adversaries, assuming $t < n/3$.

On Adaptively Secure Multiparty Computation with a Short CRS [30]. Recall that a protocol is adaptively secure if honest parties might get corrupted *after* the protocol has started. In TCC 2015, three constant-round adaptively secure protocols were presented [24, 36, 49]. All three constructions assume that the parties have access to a *common reference string* (CRS) whose size depends on the function to compute, even when facing semi-honest adversaries.

In this work, we study adaptively secure protocols which only rely on a short CRS that is independent on the function to compute. First, we raise a subtle issue relating to the usage of *non-interactive non-committing encryption* within security proofs in the UC framework, and explain how to overcome it. We demonstrate the problem in the security proof of the adaptively secure oblivious-transfer protocol from [23] and provide a complete proof of this protocol. Next, we present a new primitive called *non-committing indistinguishability obfuscation*, and show that this primitive is *complete* for constructing adaptively secure protocols with round complexity independent of the function.

Joint work with Chris Peikert.

Bibliography

[1] Bar Alon and Eran Omri. Almost-optimally fair multiparty coin-tossing with nearly three-quarters malicious. In *Proceedings of the 14th Theory of Cryptography Conference, TCC 2016-B, part I*, pages 307–335, 2016.

[2] Gilad Asharov. Towards Characterizing Complete Fairness in Secure Two-Party Computation. In *Proceedings of the 11th Theory of Cryptography Conference, TCC 2014*, pages 291–316, 2014.

[3] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[4] 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 *Advances in Cryptology – EUROCRYPT 2012*, pages 483–501, 2012.

[5] Gilad Asharov, Amos Beimel, Nikolaos Makriyannis, and Eran Omri. Complete Characterization of Fairness in Secure Two-Party Computation of Boolean Functions. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part I*, pages 199–228, 2015.

[6] Yonatan Aumann and Yehuda Lindell. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. *Journal of Cryptology*, 23(2):281–343, 2010.

[7] Donald Beaver. Foundations of Secure Interactive Computing. In *Advances in Cryptology – CRYPTO '91*, pages 377–391, 1991.

[8] Donald Beaver, Silvio Micali, and Phillip Rogaway. The Round Complexity of Secure Protocols (Extended Abstract). In *Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC)*, pages 503–513, 1990.

[9] Amos Beimel, Eran Omri, and Ilan Orlov. Protocols for Multiparty Coin Toss with a Dishonest Majority. *Journal of Cryptology*, 28(3):551–600, 2015.

[10] Michael Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols (Extended Abstract). In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 27–30, 1983.

[11] Michael Ben-Or and Ran El-Yaniv. Resilient-Optimal Interactive Consistency in Constant Time. *Distributed Computing*, 16(4):249–262, 2003.

[12] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation (Extended Abstract). In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 1988.

[13] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing (STOC)*, pages 52–61, 1993.

[14] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the 13th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 183–192, 1994.

[15] Manuel Blum. Coin Flipping by Telephone. In *Advances in Cryptology – CRYPTO '81*, pages 11–15, 1981.

[16] Gabriel Bracha. An Asynchronous $[(n-1)/3]$-Resilient Consensus Protocol. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 154–162, 1984.

[17] Andrei Z. Broder and Danny Dolev. Flipping Coins in Many Pockets (Byzantine Agreement on Uniformly Random Values). In *Proceedings of the 25th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 157–170, 1984.

[18] Niv Buchbinder, Iftach Haitner, Nissan Levi, and Eliad Tsfadia. Fair coin flipping: Tighter analysis and the many-party case. In *Proceedings of the 28th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2580–2600, 2017.

[19] Ran Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[20] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 136–145, 2001.

[21] Ran Canetti. Universally composable signature, certification, and authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17)*, pages 219–235, 2004.

[22] Ran Canetti and Tal Rabin. Universal Composition with Joint State. In *Advances in Cryptology – CRYPTO 2003*, pages 265–281, 2003.

[23] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally Composable Two-Party and Multi-Party Secure Computation. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC)*, pages 494–503, 2002.

[24] Ran Canetti, Shafi Goldwasser, and Oxana Poburinnaya. Adaptively secure two-party computation from indistinguishability obfuscation. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part II*, pages 557–585, 2015.

[25] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *Proceedings of the 10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 11–19, 1988.

[26] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults (Extended Abstract). In *Proceedings of the 26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 383–395, 1985.

[27] Richard Cleve. Limits on the Security of Coin Flips When Half the Processors are Faulty. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing (STOC)*, pages 364–369, 1986.

[28] Ran Cohen. Asynchronous secure multiparty computation in constant time. In *Proceedings of the 19th International Conference on the Theory and Practice of Public-Key Cryptography (PKC), part II*, pages 183–207, 2016.

[29] Ran Cohen and Yehuda Lindell. Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation. In *Advances in Cryptology – ASIACRYPT 2014, part II*, pages 466–485, 2014.

[30] Ran Cohen and Chris Peikert. On adaptively secure multiparty computation with a short CRS. In *Proceedings of the 10th Conference on Security and Cryptography for Networks (SCN)*, pages 129–146, 2016.

[31] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In *Advances in Cryptology – CRYPTO 2016, part III*, pages 240–269, 2016.

[32] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. Characterization of Secure Multiparty Computation Without Broadcast. In *Proceedings of the 13th Theory of Cryptography Conference, TCC 2016-A, part I*, pages 596–616, 2016.

[33] Ran Cohen, Sandro Coretti, Juan Garay, and Vassilis Zikas. Round-preserving parallel composition of probabilistic-termination cryptographic protocols. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 37:1–37:15, 2017.

[34] Ran Cohen, Iftach Haitner, Eran Omri, and Lior Rotem. From fairness to full security in multiparty computation. In *Proceedings of the 11th Conference on Security and Cryptography for Networks (SCN)*, pages 216–234, 2018.

[35] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient Multiparty Computations Secure Against an Adaptive Adversary. In *Advances in Cryptology – EUROCRYPT '99*, pages 311–326, 1999.

[36] Dana Dachman-Soled, Jonathan Katz, and Vanishree Rao. Adaptively secure, universally composable, multiparty computation in constant rounds. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part II*, pages 586–613, 2015.

[37] Ivan Damgård and Yuval Ishai. Constant-Round Multiparty Computation Using a Black-Box Pseudorandom Generator. In *Advances in Cryptology – CRYPTO 2005*, pages 378–394, 2005.

[38] Ivan Damgård and Jesper Buus Nielsen. Adaptive versus Static Security in the UC Model. In *Proceedings of the Eighth International Conference on Provable Security (ProvSec)*, pages 10–28, 2014.

[39] Danny Dolev and H. Raymond Strong. Authenticated Algorithms for Byzantine Agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[40] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early Stopping in Byzantine Agreement. *Journal of the ACM*, 37(4):720–741, 1990.

[41] Bennett Eisenberg. On the Expectation of the Maximum of IID Geometric Random Variables. *Statistics & Probability Letters*, 78(2):135–143, 2008.

[42] Pesech Feldman and Silvio Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[43] Michael J. Fischer and Nancy A. Lynch. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183–186, 1982.

[44] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy Impossibility Proofs for Distributed Consensus Problems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 59–70, 1985.

[45] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 211–220, 2003.

[46] Matthias Fitzi, Nicolas Gisin, Ueli M. Maurer, and Oliver von Rotz. Unconditional Byzantine Agreement and Multi-party Computation Secure against Dishonest Minorities from Scratch. In *Advances in Cryptology – EUROCRYPT 2002*, pages 482–501, 2002.

[47] Matthias Fitzi, Daniel Gottesman, Martin Hirt, Thomas Holenstein, and Adam Smith. Detectable Byzantine Agreement Secure Against Faulty Majorities. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 118–126, 2002.

[48] Matthias Fitzi, Martin Hirt, Thomas Holenstein, and Jürg Wullschleger. Two-Threshold Broadcast and Detectable Multi-party Computation. In *Advances in Cryptology – EUROCRYPT 2003*, pages 51–67, 2003.

[49] Sanjam Garg and Antigoni Polychroniadou. Two-round adaptively secure MPC from indistinguishability obfuscation. In *Proceedings of the 12th Theory of Cryptography Conference, TCC 2015, part II*, pages 614–637, 2015.

[50] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-Round Secure MPC from Indistinguishability Obfuscation. In *Proceedings of the 11th Theory of Cryptography Conference, TCC 2014*, pages 74–94, 2014.

[51] Oded Goldreich. *Foundations of Cryptography – Volume 2, Basic Applications*. Cambridge University Press, 2004. ISBN 0-521-83084-2.

[52] Oded Goldreich and Erez Petrank. The Best of Both Worlds: Guaranteeing Termination in Fast Randomized Byzantine Agreement Protocols. *Information Processing Letters*, 36 (1):45–49, 1990.

[53] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing (STOC)*, pages 218–229, 1987.

[54] Shafi Goldwasser and Leonid A. Levin. Fair Computation of General Functions in Presence of Immoral Majority. In *Advances in Cryptology – CRYPTO '90*, pages 77–93, 1990.

[55] Shafi Goldwasser and Yehuda Lindell. Secure Multi-Party Computation without Agreement. *Journal of Cryptology*, 18(3):247–287, 2005.

[56] Dov Gordon and Jonathan Katz. Complete Fairness in Multi-party Computation without an Honest Majority. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 19–35, 2009.

[57] Dov Gordon, Carmit Hazay, Jonathan Katz, and Yehuda Lindell. Complete Fairness in Secure Two-Party Computation. In *Proceedings of the 40th Annual ACM Symposium on Theory of Computing (STOC)*, pages 413–422, 2008.

[58] Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-Round MPC with Fairness and Guarantee of Output Delivery. In *Advances in Cryptology – CRYPTO 2015, part II*, pages 63–82, 2015.

[59] Ronald L. Graham and Andrew Chi-Chih Yao. On the improbability of reaching byzantine agreements (preliminary version). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)*, pages 467–478, 1989.

[60] Iftach Haitner and Eliad Tsfadia. An Almost-Optimally Fair Three-Party Coin-Flipping Protocol. In *Proceedings of the 46st Annual ACM Symposium on Theory of Computing (STOC)*, pages 817–836, 2014.

[61] Martin Hirt and Vassilis Zikas. Adaptively Secure Broadcast. In *Advances in Cryptology – EUROCRYPT 2010*, pages 466–485, 2010.

[62] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience (extended abstract). In *Advances in Cryptology – EUROCRYPT 2005*, pages 322–340, 2005.

[63] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Asynchronous multi-party computation with quadratic communication. In *Proceedings of the 35th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 473–485, 2008.

[64] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding Cryptography on Oblivious Transfer - Efficiently. In *Advances in Cryptology – CRYPTO 2008*, pages 572–591, 2008.

[65] Jonathan Katz and Chiu-Yuen Koo. On Expected Constant-Round Protocols for Byzantine Agreement. In *Advances in Cryptology – EUROCRYPT 2006*, pages 445–462, 2006.

[66] Jonathan Katz and Chiu-Yuen Koo. Round-Efficient Secure Computation in Point-to-Point Networks. In *Advances in Cryptology – EUROCRYPT 2007*, pages 311–328, 2007.

[67] Jonathan Katz and Yehuda Lindell. Handling Expected Polynomial-Time Strategies in Simulation-Based Security Proofs. In *Proceedings of the Second Theory of Cryptography Conference, TCC 2005*, pages 128–149, 2005.

[68] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally Composable Synchronous Computation. In *Proceedings of the 10th Theory of Cryptography Conference, TCC 2013*, pages 477–498, 2013.

[69] Joe Kilian. Zero-knowledge with Log-Space Verifiers. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–35, 1988.

[70] Joe Kilian. A General Completeness Theorem for Two-Party Games. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing (STOC)*, pages 553–560, 1991.

[71] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-Theoretically Secure Protocols and Security Under Composition. In *Proceedings of the 38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 109–118, 2006.

[72] Leslie Lamport. The Weak Byzantine Generals Problem. *Journal of the ACM*, 30(3): 668–676, 1983.

[73] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[74] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential Composition of Protocols Without Simultaneous Termination. In *Proceedings of the 21st Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 203–212, 2002.

[75] Silvio Micali and Phillip Rogaway. Secure Computation (Abstract). In *Advances in Cryptology – CRYPTO '91*, pages 392–404, 1991.

[76] Tal Moran, Moni Naor, and Gil Segev. An Optimally Fair Coin Toss. In *Proceedings of the 6th Theory of Cryptography Conference, TCC 2009*, pages 1–18, 2009.

[77] Pratyay Mukherjee and Daniel Wichs. Two Round Multiparty Computation via Multi-key FHE. In *Advances in Cryptology – EUROCRYPT 2016, part II*, pages 735–763, 2016.

[78] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, 1980.

[79] Birgit Pfitzmann and Michael Waidner. Unconditional Byzantine Agreement for any Number of Faulty Processors. In *Proceedings of the 9th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 339–350, 1992.

[80] Michael O. Rabin. Randomized Byzantine Generals. In *Proceedings of the 24th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 403–409, 1983.

[81] Tal Rabin and Michael Ben-Or. Verifiable Secret Sharing and Multiparty Protocols with Honest Majority (Extended Abstract). In *Proceedings of the 30th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 73–85, 1989.

[82] Russell Turpin and Brian A. Coan. Extending Binary Byzantine Agreement to Multivalued Byzantine Agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[83] Andrew C. Yao. Protocols for Secure Computations. In *Proceedings of the 23th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.

[84] Andrew C. Yao. How to Generate and Exchange Secrets. In *Proceedings of the 27th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 162–167, 1986.